LINUX

Příručka uživatele

Tato příručka obsahuje vše, co potřebujete znát pro práci s operačním systémem Linux, což je volně šiřitelná obdoba operačního systému Unix pro osobní počítače. Popisuje základní příkazy operačního systému Unix a rovněž se zabývá některými specifickými vlastnostmi Linuxu. Manuál je určen zejména pro začátečníky, avšak zkušenější uživatelé zde rovněž naleznou spoustu užitečných informací.

Kapitola

1

Úvod

1.1 Kdo by si měl přečíst tuto knihu

Patříte mezi ty, kteří by si měli přečíst tuto knihu? Pokud neznáte přímou odpověď, pokuste se odpovědět na jiné otázky. Získali jste někde operační systém Linux a nevíte, co máte dělat dál? Jste uživateli jiného operačního systému než Unix a uvažujete o používání systému Linux? Chcete se dozvědět, co lze od tohoto operačního systému očekávat?

Máte-li k dispozici tuto knihu a přitom jste na některou z předcházejících otázek odpověděli „ano“, pak byste si ji měli přečíst. Každý, kdo si nainstaloval na svůj osobní počítač operační systém Linux (což je volně šiřitelná obdoba operačního systému Unix vytvořená panem Linusem Torvaldsem) a kdo neví, jak jej má efektivně využívat, by si měl tuto knihu přečíst. Je v ní je popsána většina základních příkazů operačního systému Unix a rovněž jsou zde uvedeny pokročilejší techniky pro práci se systémem. Rovněž se zmíníme o výkonném editoru GNU Emacs a několika dalších důležitých aplikacích.

1.1.1 Co byste měli udělat, než začnete tuto knihu číst

V této knize se předpokládá, že máte zajištěn přístup k operačnímu systému typu Unix a že tímto operačním systémem je Linux, tedy operační systém běžící na osobních počítačích s procesorem Intel.* Posledně jmenovaný předpoklad však není nezbytný - budeme upozorňovat na případy, kdy se ostatní unixové operační systémy od systému Linux liší.

---

* Poznámka korektora: Operační systém Linux je v současné době k dispozici nejen pro procesory Intel, ale i SPARC, Alpha, ARM, PowerPC, pro stanice SGI a jiné.

Operační systém Linux je dostupný v mnoha formách, jež se nazývají distribuce. Předpokládá se, že máte nainstalovánu kompletní distribuci, jako je SlackWare, Redhat nebo MCC-Interim. Mezi jednotlivými distribucemi jsou jisté rozdíly, ale většinou nejsou důležité. Může se stát, že příklady uvedené v této knize se budou lišit od vaší distribuce. Ničeho se neobávejte, většinou půjde o odlišnosti zanedbatelné, se kterými si snadno poradíte. Pokud byste narazili na závažné rozdíly, neváhejte a napište o nich autorovi.

Pokud máte přístupová práva jako superuživatel (tedy například jako správce operačního systému nebo ten, kdo provedl instalaci), měli byste si pro sebe vytvořit normální uživatelský účet.

Potřebné informace najdete v instalační příručce. Jestliže nemáte přístupová práva jako super uživatel, pak požádejte o vytvoření účtu správce systému.

Na čtení této knihy byste měli mít dostatek času a trpělivosti. Naučit se pracovat s operačním systémem Linux není jednoduché - většina lidí považuje ostatní operační systémy za jednodušší (jako například Macintosh Operating System). Jakmile se však naučíte základům, budete pracovat velmi snadno a efektivně. Unix je výkonný operační systém a umožňuje snadno provádět i velmi složité úlohy.

Navíc se v této knize předpokládá, že jste seznámeni se základní počítačovou terminologií. I když není tento předpoklad nezbytným, bude se vám kniha číst snadněji. Měli byste například vědět, co je to spustitelný program nebo co je to textový soubor. Pokud to nevíte, budete potřebovat někoho, kdo vám při studiu tohoto operačního systému poradí a pomůže.

1.2 Kdy se můžete čtení této knihy vyhnout

Jakýkoliv počítačový program se nejlépe naučíte tak, že si jej budete zkoušet na počítači. Většina lidí přijde na to, že čtení manuálů bez možnosti bezprostředně program nebo operační systém používat není příliš užitečné. Operační systém Unix se nejlépe naučíte tak, že jej budete používat. Nevyhýbejte se experimentování - můžete sice leccos pokazit, ale stále máte možnost provést novou instalaci. Většině havárií způsobených vaší nezkušeností se můžete vyhnout důsledným zálohováním.

Operační systém Unix nelze používat tak intuitivně, jako většinu ostatních operačních systémů. Proto byste si měli přečíst alespoň kapitoly 4, 5 a 6.

Jednu z možností, jak se vyhnout čtení této knihy, představují tzv. manuálové stránky, jež poskytují dokumentaci on-line. Popisu manuálových stránek se budeme věnovat v oddílu 4.2.

1.3 Jak číst tuto knihu

Při čtení této knihy vám doporučujeme, abyste si bezprostředně vyzkoušeli to, co si právě přečtete. Pokud jste s některou tématikou již seznámeni, můžete příslušné odstavce přeskočit. Některá témata vám budou připadat zajímavá, jiná triviální, či dokonce nudná. Možná máte tolik sebejistoty, že budete některé příkazy zkoušet přesto, že přesně nevíte, jakou mají funkci. I to je způsob, jak se naučit pracovat s operačním systémem, i když poněkud riskantní.

Mnozí lidé ztotožňují operační systém Unix s příkazovým interpretem. Příkazový procesor je program, kterým lze řídit vše, co se v operačním systému Unix odehrává. Operační systém Unix je ve skutečnosti velmi složitý - většina jednoduše zadaných příkazů spustí spoustu procesů, o nichž uživatel zpravidla ani neví. V této knize probereme základní pravidla pro používání příkazového procesoru a také se zmíníme o důležitých programech, které jsou součástí operačního systému Unix nebo Linux.

Tato kapitola je spíše pseudokapitolou, která předkládá informace o obsahu knihy. Další kapitoly se zabývají tématy dle následujícího seznamu:

Druhá kapitola se zabývá vznikem operačních systému Unix a Linux. Dále předkládá informace o nadaci Free Software Foundation a o projektu GNU.

Ve **třetí kapitole** je vysvětleno, jak začít a skončit práci s počítačem, co se stane při startování a při ukončení činnosti operačního systému. Většina těchto informací není pro práci s operačním systémem Linux důležitá. Jsou to ale informace užitečné a zajímavé.

Kapitola 4 představuje úvod k práci s příkazovým procesorem operačního systému Unix. Jedná se o prostředí, v němž se provádějí příkazy a spouštějí programy. Dozvíte se zde o základních příkazech a programech, které musíte znát, pokud máte používat operační systém Unix.

V **kapitole 5** se budeme zabývat systémem X Window. Ten představuje primární grafické uživatelské rozhraní pro operační systémy typu Unix a některé distribuce jej používají jako základní uživatelské prostředí.*

V **kapitole 6** se budeme zabývat pokročilejšími technikami při práci s příkazovým procesorem, zejména těmi, které vaši práci zefektivní.

V **sedmé kapitole** jsou stručně popsány příkazy operačního systému Unix. Čím více nástrojů budete umět používat, tím efektivnější bude vaše práce.

* Poznámka korektora: V současné době používá systém X Window většina používaných distribucí operačního systému Linux.

Kapitola 8 popisuje věhlasný editor Emacs. Jedná se o velký program, který většinu nástrojů operačního systému Unix integruje do jediného prostředí.

Devátá kapitola je věnována konfiguraci operačního systému Unix. Pomocí této konfigurace si můžete nastavit vlastnosti systému tak, aby se vám co nejlépe pracovalo.

V **desáté kapitole** se zmíníme o možnostech, které má uživatel operačního systému Unix k dispozici, chce-li komunikovat s ostatními počítači, případně se sítí Internet. Najdete zde informace o elektronické poště a o systému World Wide Web.

Jedenáctá kapitola popisuje některé komplikovanější příkazy.

V **kapitole 12** se dozvíte, jak se při práci s operačním systémem Unix nebo Linux snadno vyhnout chybám.

1.4 Dokumentace k operačnímu systému Linux

Tato uživatelská příručka (původní název zní: „*The Linux Users' Guide*“) je určena začátečníkům. V rámci projektu „*Linux Documentation Project*“ se však připravují knihy pro pokročilejší uživatele.

1.4.1 Další knihy týkající se operačního systému Linux

Dokumentace k operačnímu systému Linux dále obsahuje tyto knihy: „*Installation and Getting Started*“ - příručka popisující instalaci systému Linux. „*The Linux System Administrator's Guide*“ - dokumentace týkající se správy operačního systému Linux. „*The Linux Kernel Hackers' Guide*“ - kniha popisující jádro systému Linux a způsoby jeho modifikace. „*The Linux Network Administration Guide*“ - příručka týkající se instalace, konfigurace a používání síťových spojení.

1.4.2 Soubory HOWTO

Kromě knih napsaných v rámci projektu „*Linux Documentation Project*“ byla vytvořena řada krátkých dokumentů ve formě souborů týkajících se jednotlivých témat kolem operačního systému Linux. Například soubor SCSI-HOWTO popisuje řešení možných komplikací s používáním rozhraní SCSI.

Soubory HOWTO jsou dostupné v několika formách - jako ucelené knihy (například „*The Linux Bible*“ nebo „*Dr. Linux*“), nebo je můžete získat prostřednictvím diskusní skupiny comp.os.linux.answers. Soubory se rovněž nacházejí na mnoha serverech FTP nebo WWW. Centrálním místem obsahujícím informace o operačním systému Linux je *http://www.linux.org*.

1.4.3 Co je to "Linux Documentation Project"?

Cílem projektu „*Linux Documentation Project*“ je shromáždit, utřídit a v jednotné formě prezentovat dokumentaci týkající se operačního systému Linux. Pracují na něm lidé po celém světě. Impuls k projektu dal Lars Wirzenius a dnes jej koordinuje Matt Welsh s pomocí pana Michaela K. Johnsona.

Předpokládá se, že v rámci projektu „*Linux Documentation Project*“ budou postupně vydány knihy týkající se všech témat kolem operačního systému Linux. Pokud budete mít jakékoliv náměty na zlepšení této dokumentace, dejte prosím vědět autorovi této knihy (prostřednictvím adresy *leg+@andrew.cmu.edu*) a/nebo panu Mattu Welshovi (prostřednictvím adresy *mdw@cs.cornell.edu*).

1.5 Operační systémy

Primární funkce operačního systému spočívá v tom, že poskytuje podporu pro realizaci počítačových programů. Proto například můžete používat svůj oblíbený editor a vytvářet dokumenty. Bez podpory operačního systému by editor nemohl pracovat - editor potřebuje ke své činnosti interakci s vaším terminálem, s vašimi soubory a dalším technickým vybavením počítače.

Nyní si položme otázku: Proč je nutné psát a číst knihy o operačních systémech, které jsou pouhou podporou vašich aplikací? Každý operační systém obsahuje spoustu programů pro správu, konfiguraci, řízení spojení s ostatními systémy, zálohování a podobně. Pokud jde o operační systém Linux, najdete zde řadu „miniaplikací“, jež vám pomohou pracovat efektivněji. Jestliže tedy nepoužíváte váš počítač pouze k práci s jednou nebo několika málo aplikacemi, je znalost operačního systému velmi důležitá.

Operační systém (nejčastěji označován zkratkou „OS“) může být jednoduchý a minimalizovaný (například DOS) nebo velký a složitý (například OS/2 nebo VMS). Operační systémy typu Unix patří ke středně velkým systémům. Poskytují o něco více zdrojů a prostředků než první jednoduché operační systémy, a to v takové formě, aby s jejich pomocí bylo možné řešit prakticky všechny úlohy.

Původně byl operační systém Unix navržen jako zjednodušení operačního systému Multics. Filosofie operačního systému Unix spočívá v tom, že by se veškeré funkce měly rozdělit do malých částí, tedy relativně jednoduchých programů.[1] Nové funkční vlastnosti lze získat vhodnou kombinací těchto jednoduchých programů. Samozřejmě se stále objevují nové a nové obslužné programy, které lze snadno integrovat do vašich nástrojů, a tak můžete váš operační systém neustále rozšiřovat. Když jsem například psal tento dokument, používal jsem následující programy: `fvwm` pro obsluhu a konfiguraci systému X Window, editor `Emacs` pro vytvoření textu, `xdvi` pro prohlížení textu před tiskem, `LATEX` pro formátování textu, `dvips` pro přípravu tisku a `lpr` pro vlastní tisk. Kdyby například existoval jiný program pro prohlížení textu před tiskem, pak bych jej mohl použít, aniž bych změnil ostatní programy. V tomto okamžiku na mém počítači běží současně třicet osm programů (většina z nich se nachází v neaktivním stavu „sleep“, dokud nebudou aktivovány k provedení nějakého úkolu).

Při používání operačního systému Unix budete jistě chtít minimalizovat množství práce potřebné k realizaci každého běžně prováděného úkolu. Operační systém Unix vám pro tento účel nabízí spoustu nástrojů. Abyste je mohli efektivně používat, budete muset poměrně přesně vědět, jakou funkci každý nástroj má. Pokud byste nad uskutečněním každého jednoduchého úkolu strávili hodinu nebo dokonce více, pak by vaše práce nebyla příliš produktivní. V této knize se dozvíte, jak a v které situaci využívat nástroje obsažené v operačním systému Unix a jak tyto nástroje kombinovat za účelem řešení složitějších úloh.

Klíčovou částí operačního systému je tzv. **jádro**. Ve většině operačních systémů, jako je Unix, OS/2 nebo VMS, plní jádro systému takové funkce, jako je spouštění programů, přidělování systémových zdrojů, přidělování času procesoru současně běžícím programům a podobně. Samozřejmě platí, že i jádro systému je program. Ten běží na vašem počítači jako první program po jeho nastartování, kdy realizuje všechny konfigurační funkce, a jako poslední program před vypnutím počítače, kdy realizuje všechny potřebné funkce k zastavení systému.

[1] Tato filosofie byla ve skutečnosti určena technickým vybavením počítačů, na kterých běžely první verze operačního systému Unix. Časem se ukázalo, že Unix dobře funguje i na počítačích s vyspělejším technickým vybavením. I dnes, po dvaceti pěti letech, je původní filosofie operačního systému Unix zcela vyhovující.

Kapitola

2

Co je to Unix

2.1 Historie operačního systému Unix

V roce 1965 pracovaly společnosti Bell Telephone Laboratories (divize AT&T) a General Electric na projektu „MAC of MIT“, jehož cílem bylo vytvořit operační systém Multics. Později se společnost Bell Telephone Laboratories rozhodla od spolupráce odstoupit, ale v důsledku toho neměla k dispozici kvalitní operační systém.

Pánové Ken Thompson a Dennis Ritchie se rozhodli navrhnout operační systém, který by společnosti Bell Telephone Laboratories vyhovoval. Ken Thompson tento návrh realizoval při vytváření vývojového prostředí na počítači PDP-7. Další výzkumný pracovník společnosti Bell Telephone Laboratories, pan Brian Kernighan, dal novému operačnímu systému název Unix.

Později zveřejnil pan Dennis Ritchie programovací jazyk C. V roce 1973 byl Unix kompletně přepsán do jazyka C (původní systém byl vytvořen v assembleru[1]). V roce 1977 byl operační systém Unix převeden z počítače PDP na nový počítač s použitím procesu, jež se nazývá „**porting**“. Tato akce byla uskutečnitelná právě proto, že byl operační systém Unix přepsán v jazyce C.

Koncem sedmdesátých let byla společnosti AT&T protimonopolním úřadem zakázána činnost v oblasti počítačového průmyslu. Proto se společnost rozhodla za velmi výhodných finančních podmínek převést licenci na operační systém Unix na některé university. Unix se tedy stal populárním především v akademických kruzích, avšak postupem času se začal prosazo-

[1] Assembler je základní programovací jazyk, který je úzce spjat s konkrétním typem počítače. Programy napsané v jazyce assembler zpravidla nejsou přenositelné mezi různými počítačovými platformami.

vat i v komerční sféře. Dnešní podoba Unixu se zcela liší od verze z roku 1970. Existují dvě základní varianty: System V od společnosti USL (Unix System Laboratories, dnes jej vlastní Novell[2]) a BSD (Berkeley Software Distribution).

Poslední verze USL má označení SVR4[3] (čtvrtá verze), zatímco poslední verze od BSD má označení 4.4. Kromě těchto základních verzí však existuje spousta dalších verzí operačního systému Unix. Komerční verze jsou zpravidla odvozeny od jedné z verzí USL nebo BSD. Existuje však spousta verzí operačního systému Unix, které kombinují vlastnosti obou základních verzí.

Ceny současných komerčních verzí operačního systému Unix pro počítače s procesorem Intel se pohybují od 500 do 2000 dolarů.

2.2 Historie operačního systému Linux

Autorem operačního systému Linux je pan Linus Torvalds. Jeho původní verze byla v průběhu asi deseti let zdokonalována bezpočtem lidí na celém světě. Linux představuje verzi operačního systému Unix pro osobní počítače s procesorem Intel, Alpha a další a byl kompletně vytvořen znovu - na jeho vývoji se společnosti Unix System Laboratories a Berkeley Software Distribution vůbec nepodílely. Zajímavé je, že se na vývoji operačního systému Linux podíleli lidé na celém světě - od Austrálie po Finsko a všichni doufali, že se Linux podaří uvést do podoby schopné konkurovat ostatním operačním systémům typu Unix.

Vlastní projekt operačního systému Linux začal výzkumem vlastností procesoru 386. Systém je navržen tak, aby maximálně využíval všech vlastností tohoto procesoru.

Na operační systém Linux se vztahují licenční podmínky GPL (GNU General Public Licence). Tato licence se vztahuje na veškeré programové vybavení produkované nadací Free Software Foundation a jejím cílem je zabránit komukoliv omezovat distribuční práva ostatních. Jinými slovy, podle licenčních podmínek GPL si můžete účtovat za distribuci programového vybavení GNU kolik chcete, ale nesmíte nikomu nařizovat, za jaký poplatek je má distribuovat dál. Dále musíte s každou distribucí programového vybavení GNU zpřístupnit zdrojové kódy[4], což je velmi užitečné pro programátory. Pak si totiž každý může například modifikovat operační systém Linux a dále distribuovat tuto modifikovanou verzi - opět za předpokladu, že ji bude distribuovat podle licenčních podmínek GPL.

---

2 Tato verze operačního systému Unix byla nedávno prodána společnosti Novell. Předtím byla verze USL vlastněna společností AT&T.

3 SVR4 je zkratkou pro „System V“ verze 4. (System V Release 4)

4 Zdrojový kód programu je soubor ASCII obsahující příkazy pro konkrétní programovací jazyk. Příslušným překladačem jej lze přeložit do strojového kódu pro konkrétní počítač. Strojový kód je soubor, který již není snadno čitelný a modifikovatelný tak jako zdrojový kód.

Operační systém Linux podporuje většinu programového vybavení napsaného pro Unix, včetně systému X Window. Tento systém byl vytvořen na Massachusetts Institute of Technology tak, aby umožňoval operačním systémům Unix vytvářet grafická okna a interaktivně spolupracovat mezi sebou. Dnes platí, že je systém X Window implementován pro každou verzi operačního systému Unix.

Kromě standardů, které představují verze System V a BSD, existuje sada standardizačních dokumentů publikovaná úřadem pro standardizaci IEEE, která má označení **POSIX**. Operační systém Linux splňuje většinu podmínek uvedených v dokumentech POSIX-1 a POSIX-2. Přitom má v mnoha aspektech vlastnosti podobné systému BSD, ale má také vlastnosti, které najdete u systému System V. Stručně řečeno, pokud jde o standard, představuje operační systém Linux kombinaci (a většina lidí se domnívá, že velmi dobrou) všech tří standardů.

Většina programového vybavení dodávaná s operačním systémem Linux pochází od nadace Free Software Foundation a je součástí projektu GNU. Hlavním cílem projektu GNU je jednak zpřístupnit programové vybavení původně napsané pro operační systém Unix uživatelům ostatních operačních systémů, jednak vytvořit přenositelný operační systém, který bude mít prakticky stejné vlastnosti jako Unix. Přenositelný znamená, že tento operační systém poběží na všech počítačových platformách (ať jsou vybaveny procesorem Intel, PowerPc, Alpha či jiným). Tento operační systém má označení Hurd. Hlavní rozdíl mezi operačním systémem Linux a operačním systémem Hurd nespočívá v uživatelském rozhraní, ale v programátorském rozhraní - Hurd představuje moderní operační systém, zatímco Linux je operačním systémem založeným na filosofii staříčkého Unixu.

Předcházející odstavce by mohly vzbudit dojem, že se o operační systém Linux stará pouze pan Linus Torvalds. V raných dobách existence tohoto systému se například o překladač GCC (základní překladač jazyka C pro Linux) a o knihovny Linux C staral H. J. Lu. V každé distribuci operačního systému Linux naleznete v adresáři `/usr/src/linux/` soubor `CREDITS` obsahující seznam všech lidí, kteří se významnou měrou podíleli na jeho vývoji.

2.2.1 Dnešní podoba operačního systému Linux

První číslo ve verzi operačního systému Linux představuje číslo hlavní revize. V době, kdy byla napsána tato kniha (únor 1996) je k dispozici teprve první verze. Druhé číslo představuje méně podstatné revize. Pokud je druhé číslo sudé, jedná se o stabilní verzi. Jestliže je liché, jedná se o vývojovou verzi. Ve vývojových verzích bývá spousta chyb, které „odvážní“ uživatelé postupně odhalují a programátoři postupně opravují. Jakmile jsou všechny závažné nedostatky z vývojové verze odstraněny, prohlásí se vývojová verze za stabilní a začne se pracovat na nové vývojové verzi. V únoru 1996 měla stabilní verze číslo 1.2.11 a poslední vývojová verze 1.3.61.*

* Poznámka korektora: V současné době (srpen 1998) je poslední stabilní verze 2.0.34 a vývojová verze 2.1.108.

Operační systém Linux je velkým systémem obsahujícím spoustu chyb, které se postupně odstraňují. Ve stabilních verzích se však vyskytují chyby velmi zřídka a souvisejí hlavně s nestandardním technickým vybavením počítače. Doposud jsme hovořili pouze o chybách jádra systému. Operační systém však zdaleka není jen jádro systému, ale obsahuje spoustu obslužných programů.

Ani velmi zkušený uživatel často není schopen určit, zda je původ chyby v obslužném programu nebo v jádře systému. Také je velmi nesnadné rozeznat, zda jsou některé „divné“ věci chybou operačního systému, nebo novou vlastností. Doufáme, že vám tato kniha pomůže orientovat se právě v takových situacích.

2.2.2 Několik málo otázek a odpovědí na ně

Než se pustíme do další kapitoly, odpovězme si na několik důležitých otázek.

Otázka: Jak se má vyslovovat Linux?

Odpověď: Podle autora operačního systému Linux se má samohláska „i“ vyslovovat krátce (jako například ve slově minimum). Linux by se měl rýmovat se slovem Minix, což je další verze operačního systému Unix. Samohláska „u“ by se měla vyslovovat ostře, jako například ve slově „rule“, a nikoliv měkce, jako například ve slově „ducks“. Obecně by se mělo slovo Linux rýmovat se slovem „cynics“. V našich zemích se Linux vyslovuje „linuks“.

Otázka: Proč by se mělo pracovat s operačním systémem Linux?

Odpověď: Proč ne. Linux je obecně levnější než ostatní operační systémy a je méně problematický než většina komerčních operačních systémů. Nemusí být pravda, že zrovna Linux je tím nejlepším operačním systémem pro vaši konkrétní aplikaci. Pokud má však někdo zájem o používání operačního systému typu Unix na osobním počítači a o aplikace vytvořené pro Unix, pak je Linux pravděpodobně nejlepším vysoce výkonným systémem.

2.2.3 Komerční programové vybavení pro operační systém Linux

Pro operační systém Linux existuje spousta komerčního programového vybavení. Nejdůležitější roli hraje systém Motif, což je uživatelské rozhraní pro systém X Window připomínající Microsoft Windows. Na základě systému Motif byla vytvořena řada komerčního programového vybavení. Dnes můžete koupit prakticky cokoliv ve verzi pro operační systém Linux - od oblíbeného textového procesoru Word Perfect až po Maple, univerzální nástroj pro matematické a fyzikální modelování.

Možná se budete divit, jak lze skloubit operační systém Linux, jenž je distribuován podle licenčních podmínek GPL (GNU General Public Licence, viz dodatek B), s komerčním programovým vybavením. Podmínky GPL platí pouze pro jádro systému, zatímco na ostatní aplikace vytvořené pro Linux se vztahují jiné podmínky GNU, „*Library General Public Licence*“ (viz dodatek B). Podle těchto podmínek smějí poskytovatelé programového vybavení prodávat své aplikace a přitom nemusejí distribuovat zdrojové kódy.

Uvědomte si prosím, že uvedené dva dokumenty představují něco jako autorská práva, ale v žádném případě nepředstavují licenci na užívání. Tyto dokumenty nikterak nevymezují způsoby, kterými můžete dané programové vybavení používat, ale vymezují pravidla, kterými se musíte řídit při distribuci tohoto programového vybavení. V tom spočívá hlavní myšlenka filosofie nadace Free Software Foundation a platí i pro Linux: na používání operačního systému Linux neexistuje žádná licence.

Kapitola

3

Začínáme pracovat s operačním systémem Linux

Pravděpodobně máte jisté zkušenosti s používáním jednouživatelského operačního systému, jako je DOS nebo OS/2. Chcete-li pracovat v těchto operačních systémech, nemusíte se identifikovat - předpokládá se, že jste jediným uživatelem operačního systému, který má přístup ke všem jeho zdrojům. Na druhé straně, operační systém Unix je víceuživatelským systémem. To znamená, že v daném okamžiku může mít k systému přístup více uživatelů najednou.

Aby mohl operační systém Unix komunikovat s každým uživatelem odděleně, musí jej identifikovat procesem, který se nazývá „**přihlášení se**“ (logging in). Když zapnete počítač, proběhne spousta inicializačních procesů a až potom je počítač schopen komunikovat s uživatelem. Protože je tato příručka věnována operačnímu systému Linux, vysvětlíme si, co se odehrává v průběhu zavádění tohoto operačního systému Linux.

Pokud nepoužíváte operační systém Linux na osobním počítači s procesorem Intel, pak se vás nebudou některé informace v této kapitole týkat. Většinu užitečných informací najdete v oddílu 3.1.

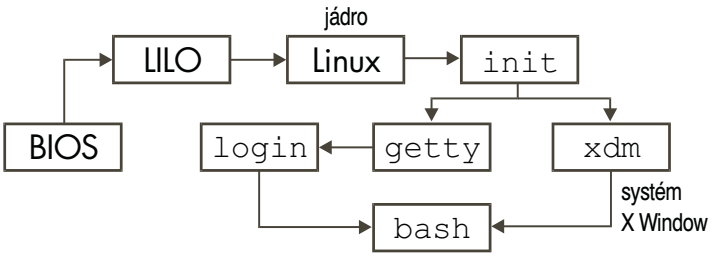
Jestliže se pouze zajímáte o používání vašeho počítače, pak kromě oddílu 3.3 nemusíte tuto kapitolu číst.

3.1 Zapnutí počítače

Po zapnutí počítače s procesorem Intel se jako první provede program zvaný BIOS. BIOS je zkratkou pro Basic Input/Output System. Tento program je trvale uložen ve zvláštní paměti počítače. Realizuje některé základní testy a identifikuje technické vybavení, jako je pevný

disk či disketová jednotka. Po identifikaci disku najde tzv. zaváděcí sektor (boot sector) a spustí kód, který v něm nalezne. Jestliže byl nalezen disk, ale žádný zaváděcí sektor, vypíše BIOS následující hlášení:

```
Non-system disk or disk error
```



Obrázek 3.1

Cesta, po které dospěje osobní počítač s procesorem Intel k příkazovému řádku. Proces `init` může, ale také nemusí realizovat zavedení systému X Window. Pokud systém X Window zavádí, spustí nejdříve program `xdm`. Jinak spustí příslušný počet procesů `getty`.

Pak ovšem stačí disk bez zaváděcího sektoru vyjmout, stisknout kteroukoliv klávesu a zaváděcí proces bude pokračovat.

Pokud není v disketové jednotce vložena disketa, BIOS vyhledá hlavní zaváděcí záznam MBR (master boot record) na pevném disku (závisí na nastavení BIOSu). Ten obsahuje kód (zaváděcí program), jehož úkolem je zavést operační systém. Pro operační systém Linux se tento program jmenuje LILO (LInux LOader). Linux Loader je nejpoužívanějším bootovacím programem pro Linux, ale je možno použít i GRUB Loader, SYSLinux nebo i NT Loader. To znamená, že program LILO je uložen právě v diskové oblasti MBR. Nyní budeme předpokládat, že se zavádí operační systém Linux. (U vaší konkrétní distribuce může proces zavádění operačního systému Linux probíhat poněkud odlišně - proto si pečlivě pročtěte dokumentaci k vaší distribuci. Další užitečné informace naleznete v dokumentaci k zaváděcímu programu LILO.)

3.2 Jak Linux přebírá kontrolu nad počítačem

Nejdříve tedy předá BIOS kontrolu zaváděcímu programu LILO a ten pak předá kontrolu **jádru** operačního systému Linux (Linux kernel). Jádro je centrálním programem celého systému a řídí všechny ostatní programy. V prvním kroku jádro realizuje přepnutí procesoru do tzv. chráněného módu. Procesor 80386[1], jež řídí váš počítač, může pracovat ve dvou módech:

[1] Pod pojmem procesor 80386 budeme dále rozumět všechny 32bitové procesory Intel, tedy i 80486, Pentium, Pentium Pro, Pentium MMX a Pentium II. Místo 80386 budeme také používat označení 386.

v tzv. „reálném módu“ a v tzv. „chráněném módu“. Operační systém DOS, stejně jako BIOS, funguje v reálném módu. Vyspělejší operační systémy však pracují v chráněném módu. Proto platí, že Linux po svém zavedení zcela nahradí BIOS.

Pokud jde o počítače s jinými procesory, probíhá tato fáze zavádění systému Linux poněkud jinak. Zpravidla nedochází k přepínání do chráněného módu a jen několik málo procesorů vyžaduje komplikované zavádění operačního systému podobné kombinaci BIOS - LILO. Po zavedení jádra pracuje Linux na všech počítačích stejně bez ohledu na procesor.

V dalším kroku se Linux pokouší identifikovat technické vybavení počítače, na kterém běží. Musí znát informace o pevném disku, zda je nebo není k dispozici myš, zda je nebo není váš počítač připojen k počítačové síti a podobně. Linux si není schopen po vypnutí počítače tyto údaje pamatovat, proto je pokaždé zjišťuje znovu. Naštěstí nemusíte údaje o technickém vybavení vašeho počítače zadávat - Linux si je zjistí sám.

V průběhu zavádění vypisuje operační systém Linux řadu hlášení. Podrobněji se jimi budeme zabývat v oddílu 3.4. Asi by se vám nelíbilo, pokud byste museli při každém startu odpovídat na spoustu dotazů týkajících se technického vybavení a konfigurace vašeho počítače. Na několik důležitých dotazů tohoto druhu však budete muset odpovědět při instalaci systému. Budete-li mít jakékoliv problémy, pročtěte si dokumentaci k vaší distribuci.

Jak jsme se již zmínili, jádro řídí všechny ostatní programy. Jestliže tedy úspěšně identifikuje veškeré technické vybavení počítače, spustí jiný program, který již začne dělat něco užitečného. Tento program má název `init`. (Všimněte si, že pro název programu jsme použili jiný font. `Neproporcionální font` budeme používat k označení programů, adresářů i obecných souborů.) Po nastartování programu `init` se jádro stane jakýmsi „manažerem“, avšak již není aktivním programem.

Jestliže chceme vědět, co dělá počítač po zavedení jádra, musíme studovat program `init`. Ten prochází mnoha komplikovanými stádii, která nejsou na všech počítačích stejná. Operační systém Linux má mnoho verzí programu `init` - každá z nich realizuje své cíle různými způsoby. Také záleží na tom, zda je váš počítač zapojen do sítě a jakou distribuci operačního systému Linux používáte. Nyní si uveďme některé základní procesy, jež program `init` realizuje:

- Zpravidla proběhne kontrola souborového systému. Co je to souborový systém? Je to systém je způsob organizace a ukládání souborů na pevném disku. Linux při kontrole souborového systému zjišťuje, které části disku jsou již použity a které jsou volné. Souborový systém se podobá rejstříku nebo štítkovému katalogu v knihovně. Bohužel se stává, že například při výpadku elektrické energie dojde k poškození některých souborů a pak

je používání některých částí pevného disku konfliktní. Program `init` proto spouští další program označený jako `fsck`, jenž se pokouší vadné soubory a konfliktně obsazené části pevného disku opravit.

- Pokud je váš počítač připojen k síti, spustí se několik speciálních programů, jež zajišťují komunikaci vašeho počítače s jinými počítači.

- Na pevném disku mohou zůstat některé dočasné soubory, které jsou jinak neužitečné. Ty mohou být programem `init` zrušeny.

- Zpravidla také dojde k aktualizaci systémového času. Operační systém Unix předpokládá, že je čas definován jako UCT (Universal Coordinated Time), známý také jako střední greenwichský čas. Přitom je čas ve vašem počítači pravděpodobně nastaven na lokální - to znamená, že některý program musí přečíst lokální čas nastavený ve vašem počítači a korigovat jej na čas UCT.

Jakmile program `init` ukončí všechny uvedené aktivity, přejde do stavu, jenž lze označit jako „rodič“ všech procesů v systému Unix. Pod pojmem proces si lze jednoduše představit běžící program. Protože jeden program může být spuštěn dvakrát nebo vícekrát, mohou existovat dva procesy nebo více procesů realizujících konkrétní program.

V operačním systému Unix tedy platí, že proces je instancí programu. Vytváří se systémovým voláním (službou poskytovanou jádrem systému) nazvaným „fork“ (rozvětvení). Pojem „fork“ se používá proto, že se původně jeden proces při spuštění dalšího procesu rozdělí na dva oddělené procesy. Typickým příkladem programu, který běží jako několikanásobný proces, je program `getty`. `getty` je důležitý program, jenž voláním programu `login` umožňuje uživateli přihlásit se do systému.

3.3 Činnost uživatele

3.3.1 Jak se přihlásit do systému

Než začnete používat operační systém Unix, musíte se identifikovat. Procesu identifikace se říká „**přihlášení do systému**“ (`login`). Jedná se o proces, ve kterém se operační systém přesvědčí, zda má uživatel oprávnění systém používat. Systém se vás zeptá na jméno účtu (account name) a heslo (password). Jméno účtu je zpravidla podobné vašemu jménu - pravděpodobně jste jej obdrželi od vašeho systémového správce nebo jste si vytvořili vlastní, pokud jste sami systémovým správcem. (Informace o vytváření jména účtu naleznete v příručce „*Příručka správce operačního systému Linux*“.)

Po dokončení všech zaváděcích procesů byste měli na svém monitoru spatřit hlášení podobné následujícímu. Nejnovější verze Linuxu se hlásí takto (první řádek je pouze uvítací zprávou – může zde být cokoliv jiného):

```
RedHat Linux release 5.1 (Manhattan) with all updates.
Kernel 4.1.108 on an Intel

mousehouse login:
```

Je však možné, že vás systém uvítá něčím úplně jiným. Místo nudné textové obrazovky se vám může představit nějaká grafická obrazovka. Avšak i v tomto případě budete vyzváni k přihlášení se do systému a proběhnou stejné procedury identifikace uživatele. Pokud se vám objevila zmíněná grafická obrazovka, pak jste zřejmě uživateli systému X Window. To znamená, že budete pracovat v systému oken, o kterém se podrobněji zmíníme v kapitole 5. Všimněte si, že veškerý výklad o systému X Window bude v této knize označován velkým písmenem X na levém okraji textu.

Jako jméno účtu budeme používat smyšlené jméno `larry`. Kdykoliv uvidíte v následujícím textu jméno `larry`, měli byste místo něj dosadit své vlastní jméno účtu. Jména účtu by měla být založena na skutečném jménu uživatele; ve větších operačních systémech typu Unix se zpravidla používají příjmení uživatelů, kombinace jména a příjmení nebo kombinace jména a číslic. Pro uživatele se jménem Larry Greenfield by se mohly použít například tyto kombinace: `larry`, `greenfie`, `lgreenfi` nebo `lg19`.

Pro bližší vysvětlení, jméno `mousehouse` je „jménem“ počítače, na kterém pracuji. Je pravděpodobné, že jste při instalaci operačního systému Linux použili jiné, podobně duchaplné jméno. Jméno počítače není důležité; v této knize budeme používat již uvedené jméno `mousehouse`, případně `lionsden`. Jestliže tedy zadáte své jméno účtu a stisknete klávesu Enter, objeví se výzva k zadání hesla:

```
mousehouse login: larry
Password:
```

Na druhém řádku očekává operační systém Linux zadání hesla (password). Při zadávání hesla neuvidíte na obrazovce, co píšete, proto zadávejte heslo pečlivě. Chybně zadaný znak můžete zrušit, ale opět nebudete vidět, který znak jste zrušili. Jestliže se na vás zrovna někdo dívá, nezadávejte heslo příliš pomalu. Mohlo by tak dojít k vyzrazení vašeho hesla a nepovolaná osoba by jej mohla zneužít. Jestliže zadáte heslo chybně, budete mít možnost zadat je znovu.

Pokud jste zadali heslo správně, objeví se zpravidla na vaší obrazovce nějaká zpráva, jež se většinou nazývá „zpráva dne“. Tato zpráva může obsahovat cokoliv - její znění zadává systémový správce.

Jako další se objeví tzv. výzva příkazového řádku (prompt). Výzva příkazového řádku je řetězec indikující skutečnost, že systém očekává zadání příkazu. Mohla by vypadat takto:

```
/home/larry#
```

Pokud jste uživateli systému X Window, pak zřejmě uvidíte výzvu podobnou výše uvedené v nějakém okně na obrazovce. Okno je obdélníková orámovaná oblast obrazovky, která v tomto případě imituje terminál. Pokud chcete zadat příkaz v příkazovém řádku v okně, musí být okno aktivní - stačí například do tohoto okna přesunout kurzor myši.

3.3.2 Jak ukončit práci s počítačem

Chcete-li skončit práci s počítačem, pak rozhodně nestačí pouze počítač vypnout. Pokud to uděláte v operačním systému Linux, s největší pravděpodobností ztratíte nějaká data nebo poškodíte nějaké důležité soubory.

Na rozdíl od operačního systému DOS, kde lze počítač vypnout vypínačem nebo restartovat tlačítkem Reset, je v operačním systému Linux nutné provést řadu kroků, které zajistí bezpečné ukončení systému. Uveďme si hlavní důvod. Operační systém Linux používá tzv. **paměť cache** pro diskové operace. To znamená, že jistá část souborů uložených na pevném disku je umístěna do paměti RAM[3]. K synchronizaci mezi pamětí RAM a pevným diskem dochází přibližně každých třicet sekund. Jestliže se má vypnout nebo restartovat počítač, pak je nutné, aby se část paměti RAM obsahující disková data uložila.

Máte-li tedy ukončit práci s počítačem a přitom jste normálně přihlášeni (t.j. zadali jste jméno uživatelského účtu a heslo), pak se musíte odhlásit. K tomuto účelu slouží příkaz `logout`. Napište v příkazovém řádku `logout` a stiskněte klávesu Enter. Klávesou Enter se odesílá každý příkaz. Než tuto klávesu stisknete, můžete chybně zadané znaky opravovat, případně celý příkaz zrušit a zadat nový.

[3] Rozdíl mezi pamětí RAM a pevným diskem lze přirovnat ke krátkodobé a dlouhodobé paměti. Po vypnutí počítače se veškeré informace v paměti RAM ztratí, zatímco informace uložené na pevném disku zůstávají uchovány. Na druhé straně platí, že přístup k informacím na pevném disku je nesrovnatelně pomalejší, než přístup k informacím v paměti RAM.

```
RedHat Linux release 5.1 (Manhattan) with all updates.
Kernel 4.1.108 on an Intel
mousehouse login:
```

Nyní se může do systému přihlásit další uživatel.

3.3.3 Vypnutí počítače

Jste-li jediným uživatelem počítače, pak po odhlášení můžete počítač odpojit od zdroje elektrické energie.[4] V takovém případě se přihlašujte jako uživatel se speciálním jménem účtu root. Účet root je účtem systémového správce, který má zajištěn přístup ke každému souboru v systému. Pokud hodláte vypnout počítač, pak musíte obdržet heslo od systémového správce. (Jste-li jediným uživatelem počítače, pak jste systémovým správcem právě vy! Proto nikdy nezapomeňte heslo!)

Přihlášení a odhlášení by mohlo vypadat takto:

```
mousehouse login: root
Password:
Last login: Sun 5 11:14:57 on tty1
/# shutdown -h now

Broadcast message from root (tty1) Sun 14:52:32 1998...

The system is going down for system halt NOW!
INIT:Switching to runlevel: 0
INIT: sending processes the TERM signal
The system is halted
System Halted
```

Po zadání příkazu `shutdown -h now` proběhnou jisté procedury, které připraví systém na vypnutí nebo reset.

[4] Chcete-li šetřit některé komponenty technického vybavení vašeho počítače, pak vypínejte počítač jen když nebudete na počítači pracovat delší dobu (například večer). Časté zapínání a vypínání počítače zbytečně namáhá některé jeho součásti.

Na závěr ještě uveďme krátkou poznámku pro pohodlnější uživatele. Místo procesu přihlašování a odhlašování jako uživatel root lze použít příkaz `su`. Jako běžný uživatel zadejte v příkazovém řádku příkaz `su` a stiskněte klávesu Enter. Systém si vyžádá heslo uživatele root a pak vám přidělí všechna jeho privilegia. Nyní můžete bez problémů systém ukončit příkazem `shutdown -h now`.

3.4 Hlášení jádra systému

Když poprvé startujete počítač, objeví se na obrazovce spousta hlášení popisující technické vybavení vašeho počítače. Uvedená hlášení vypisuje jádro operačního systému Linux. V tomto oddílu si některá důležitá hlášení podrobněji popíšeme.

Přirozeně platí, že se hlášení jádra systému budou lišit, což je dáno jednak typem počítače a jednat distribucí operačního systému Linux. V následujícím textu budou popsána hlášení platná pro můj počítač. Některá mají obecnou platnost, jiná jsou více specifická. Musím poznamenat, že můj počítač má minimální konfiguraci, pokud jde o technické vybavení, proto dále neuvidíte příliš mnoho informací týkajících se speciálního technického vybavení. Následující hlášení pocházejí z verze operačního systému Linux 1.3.55. Tato verze patří v době, kdy píši tuto knihu, k nejnovějším.

1. V prvním kroku operační systém Linux identifikuje grafickou kartu, aby mohl vybrat nejvhodnější font a jeho velikost. (Čím menší font je zvolen, tím větší počet hlášení se vejde na jednu obrazovku.) Linux se vás může zeptat, jaký font má používat, nebo použije standardní „kompilovaný“ font (compiled font).[5]

```
Console: 16 point font, 400 scans
Console: colour VGA+ 80x25, 1 virtual console (max 63)
```

V předcházejícím příkladu se vlastník počítače v době kompilace rozhodl pro větší, standardní font. Také si všimněte zastaralého tvaru slova „colour“. Linus se evidentně naučil špatnou verzi angličtiny.

2. V dalším kroku zobrazí jádro operačního systému zprávu o výkonnosti vašeho počítače. Výkonnost se měří v jednotkách „BogoMIPS“. Zkratka MIP (million instructions per second) označuje počet instrukcí, které je počítač schopen provést za sekundu. „BogoMIP“ je zkratkou „bogus MIP“ (bogus=podvodný, falešný) a označuje, kolikrát za sekundu neudělá počítač absolutně nic. Protože cyklus, kterým se výkonnost počítače měří, nedělá

[5] Kompilace je proces, při kterém se instrukce srozumitelné člověku přeloží do instrukcí srozumitelných počítači. Pojmem kompilovaný se zde rozumí „začleněný do programu“.

ve skutečnosti nic, nelze uvedené číslo považovat za objektivní ukazatel výkonnosti.* Operační systém Linux toto číslo používá v případě, kdy potřebuje vyčkat na odezvu nějakého zařízení.

```
Calibrating delay loop.. ok - 33.28 BogoMIPS
```

3. V třetím kroku vypíše jádro systému informace o použití paměti:

```
Memory: 23180k/24576k available (544 kernel code, 384k reserved,
  468k data)
```

Z této informace vyplývá, že má počítač 24 MB operační paměti. Část této paměti byla rezervována pro jádro. Zbytek může být využíván ostatními programy. Uvedené informace se týkají paměti **RAM**, která může být používána k uchovávání dat, jen když je počítač zapnut. Počítač však má k dispozici permanentní paměť zvanou **pevný disk**. Obsah pevného disku zůstává uchován i tehdy, když je váš počítač vypnut.

4. V průběhu zavádění operačního systému provádí Linux řadu testů technického vybavení a o výsledcích těchto testů vypisuje průběžné zprávy.

```
This processor honours the WP bit even when in supervisor
  mode. Good.
```

5. V dalším kroku se Linux zabývá konfigurací sítě. Následující zpráva by měla být popsána v manuálu „*Příručka správce sítě*“.

```
Swansea University Computer Society NET3.O33 for Linux 1.3.50
IP Protocols: ICMP, UDP, TCP
```

Popis konfigurace sítě přesahuje rámec této dokumentace, a proto se jím nebudeme zabývat.

6. Operační systém Linux podporuje jednotku pro výpočet v pohyblivé řádové čárce FPU (floating point unit). Jedná se o speciální integrovaný obvod (nebo přímo součást procesoru 80486DX), jenž realizuje aritmetické operace s neceločíselnými operandy. Některé z těchto integrovaných obvodů mohou být špatné a když se je operační systém Linux pokusí detekovat, dojde k jeho zhroucení - počítač přestane být funkční. Pokud nastane tento případ, uvidíte na obrazovce zprávu:

```
Checking 386/387 coupling...
```

* Poznámka korektora: Velikost tohoto čísla může být např. ovlivněna typem procesoru, jeho schopností předvídat cykly a také např. místem v paměti, kde se testovací cyklus provádí.

V případě, že používáte procesor 486DX, uvidíte tuto zprávu:

```
Checking 386/387 coupling... Ok, fpu using exception 16 error
  reporting.
```

Máte-li starší procesor 386 s matematickým koprocesorem 387, uvidíte zprávu:

```
Checking 386/387 coupling... Ok, fpu using irq13 error reporting.
```

7. Nyní proběhne další test, jenž testuje instrukci „halt“.*

```
Checking 'hlt' instruction... Ok.
```

8. Po dokončení počáteční konfigurace vypíše operační systém Linux řádek, kterým identifikuje sám sebe. Na tomto řádku je informace o verzi systému, verzi překladače GNU C Compiler, kterým bylo jádro přeloženo, a kdy byl překlad realizován.

```
Linux version 1.3.55 (root@mousehouse) (gcc version 2.7.0)
Sun Jan 7 14:56:26 EST 1996
```

9. Jako další se nastartuje ovladač sériového portu, který zjistí informace o příslušném technickém vybavení. **Ovladač** (driver) je součást jádra operačního systému, která řídí nějaké zařízení, zpravidla periferní. Ovladač je odpovědný za detaily komunikace mezi procesorem a periferním zařízením. Ovladače umožňují, aby se programátoři mohli při psaní aplikací soustředit na vlastní aplikaci a nemuseli se starat o takové problémy, jako je například tisk na tiskárně.

```
Serial driver version 4.11 no serial optins enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
tty02 at 0x03e8 (irq = 4) is a 16450
```

V tomto případě byly nalezeny tři sériové porty. Sériový port je ekvivalentní portu COM v operačním systému DOS. Jedná se o zařízení běžně používané ke komunikaci s modemy nebo myší.

Uvedený výpis sděluje, že sériový port s pořadovým číslem 0 (`COM1`) má adresu `0x03f8`. Když port přeruší činnost jádra (zpravidla proto, aby sdělil systému, že chce přenášet data), použije přerušení IRQ 4. Přerušení IRQ představuje další prostředek

* Poznámka korektora: V novějších jádrech Linuxu je obsažen i test na procesory, které obsahují i tzv. chybu „F00F“.

pro komunikaci mezi programovým vybavením a periferním zařízením. Každý sériový port má také svůj řídící integrovaný obvod. Často se používají obvody 16450, mohou se však také vyskytnou obvody 16550 nebo 820.*

10. Nyní přichází na řadu paralelní port. Paralelní port se nejčastěji připojuje k tiskárně a v operačním systému Linux začínají jména paralelních portů dvojicí písmen lp. lp je zkratkou pro Line Printer (řádková tiskárna), i když v poslední době by se mělo spíše jednat o zkratku pro Laser Printer (laserová tiskárna). Samozřejmě platí, že je operační systém Linux schopen komunikovat s každým typem paralelní tiskárny - jehličkovou, bublinkovou i laserovou.**

```
lp0 at 0x03bc, (polling)
```

Uvedená zpráva říká, že v systému byl nalezen jeden paralelní port a že pro něj bude použit standardní driver.

11. V dalším kroku identifikuje operační systém Linux ovladače pevných disků. Podle následující zprávy byly identifikovány dva pevné disky IDE:s

```
hda: WDC AC2340, 325MB, w/127KB Cache, CHS=1010/12/55
hda: WDC AC2850F, 814MB, w/64KB Cache, LBA, CHS=827/32/63
```

12. Dále provede jádro kontrolu disketových jednotek. Podle následujícího příkladu má počítač dvě disketové jednotky: jednotku „A“ pro diskety 5 1/4 palce a jednotku „B“ pro diskety 3 1/2 palce. Operační systém Linux přidělí disketové jednotce „A“ jméno `fd1` a disketové jednotce „B“ jméno `fd0`.

```
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
floppy: FDC 0 is a National Semiconductor PC87306
```

13. Dalším ovladačem v případě mého počítače je ovladač SLIP. Vypíše následující zprávu o své konfiguraci:

```
SLIP: version 0.8.3-NET3.019-NEWTTY (dynamic channels, max=256)
  (6 bit encapsulation enabled)
CSLIP: code copyright 1989 Regents of the University of
  California
```

* Poznámka korektora: V novějších počítačích je téměř vždy obvod 1655A na 16550A.

** Poznámka korektora: V poslední době se objevují i tzv. GDI-tiskárny, jejichž podpora není ještě zcela dokonalá, protože jejich výrobci nejsou ochotni poskytovat programátorům specifikaci komunikačního protokolu.

14. Dále jádro ověří všechny pevné disky, které byly identifikovány. Prohledá různé diskové oddíly a identifikuje ty části, na kterých se nachází operační systém, aby zabránil případné interferenci s jinými operačními systémy. V následujícím příkladě jsou identifikovány dva pevné disky: `hda` se čtyřmi oddíly a `hdb` s jedním oddílem:

```
Partition check:
hda: hda1 hda2 hda3 hda4
hdb: hdb1
```

15. V konečné fázi provede operační systém Linux připojení (mount) hlavního souborového systému platného pro kořenový oddíl (root partition). Kořenový oddíl je část pevného disku, ve které je umístěn operační systém. Když operační systém připojuje hlavní souborový systém, zpřístupní jej uživateli:

```
VFS: Mounted root (ext2 filesystem) readonly.
```

Kapitola

4

Příkazový interpret operačního systému Unix

4.1 Příkazy operačního systému Unix

Když se poprvé přihlásíte do operačního systému Unix, objeví se na obrazovce **výzva příkazového řádku** (prompt), která by mohla vypadat takto:

```
/home/larry#
```

Podle názvu „výzva příkazového řádku“ lze usuzovat, že systém na tomto místě očekává zadání příkazu. Každý příkaz operačního systému Unix představuje posloupnost písmen, čísel a znaků. Nepatří sem však mezery. Platnými příkazy operačního systému Unix jsou například: `mail`, `cat` nebo `CMU_is_Number-5`. Některé znaky není v příkazech povoleno používat - zmíníme se o nich později. Poznamenejme, že Unix má vlastnost označovanou jako „**case-sensitive**“ (doslova citlivý na malá a velká písmena).[1] To znamená, že `cat` a `Cat` jsou různé příkazy.

Příkazový řádek je zobrazován speciálním programem, který se nazývá **příkazový interpret** (shell). Příkazový interpret přijímá příkazy a realizuje je. Příkazové interprety mají svůj vlastní programovací jazyk a programy napsané v tomto jazyce se nazývají skripty příkazového interpretu (shell scripts).

Pro operační systémy Unix existují dva hlavní typy příkazových interpretů: Bourne shell a C shell. (V současnosti existuje více než desítka různých příkazových interpretů – např. v distribuci Red Hat Linux 5.1 je pro každého uživatele k dispozici pět příkazových interpre-

[1] Některé operační systémy, jako je OS/2 nebo Windows NT, sice zachovávají v názvech velká a malá písmena, ale jinak mezi nimi nerozlišují. V praxi platí, že se v operačních systémech Unix nepoužívají příkazy nebo obecná jména, která by se lišila pouze malými a velkými písmeny (například různé příkazy cat a Cat).

tů – `ash, bash, tcsh, zsh a pdksh` – ze kterých si může vybrat příkazem `chsh`). Příkazový procesor Bourne shell byl nazván podle svého autora, kterým je Steven Bourne. Ten vytvořil původní příkazový procesor pro operační systém Unix a většina příkazových procesorů vytvořených od té doby končí písmeny sh. Tím se indikuje, že další příkazové procesory jsou rozšířením původního příkazového interpretu. Dnes existuje spousta implementací původního příkazového interpretu Bourne shell. Jiný typ představují tzv. C shelly (původně je navrhl pan Bill Joy), jež se rovněž hojně používají. Obecně platí, že příkazové interprety typu Bourne shell se používají z důvodu kompatibility s originálním příkazovým interpretem, zatímco C shell se používá pro interaktivní zpracování příkazů.

Příkazové interprety typu C shell se vyznačují tím, že mají lepší vlastnosti pro interaktivní práci, ale mají poměrně nepružné vlastnosti z hlediska programátorského.

Operační systém Linux se distribuuje s příkazovým interpretem `bash`, který byl vytvořen nadací Free Software Foundation. `bash` je zkratkou pro Bourne Again Shell, což tradičně představuje spíše špatnou slovní hříčku typickou pro operační systém Unix. Jedná se o vylepšený příkazový procesor Bourne shell. Má podobné programátorské vlastnosti jako Bourne shell a také má spoustu dobrých vlastností převzatých z příkazového interpretu C shell. `bash` je implicitním příkazovým interpretem používaným v operačním systému Linux.

Když se přihlásíte do systému Linux, pak je to právě program `bash`, který zobrazuje výzvu příkazového řádku. Příkazový procesor `bash` vás bude provázet po celou dobu práce s počítačem.

4.1.1 Typické příkazy operačního systému Unix

Prvním příkazem, se kterým se seznámíme, je příkaz `cat`. Máte-li jej použít, napište `cat` a stiskněte klávesu Enter.

```
/home/larry# cat
```

Jestliže je nyní kurzor na následujícím řádku, pak jste zadali příkaz `cat` správně. Existuje několik způsobů, jak příkaz `cat` zadat. Některé jsou správné, jiné nikoliv.

- Předpokládejme, že jste se spletli:

```
/home/larry# ct
ct: command not found
/home/larry#
```

Tímto způsobem vás příkazový interpret upozornil na to, že nemůže najít program, který by se jmenoval „ct“. Nabídne vám další výzvu příkazového řádku a očekává další příkaz. Připomeňme, že operační systém Unix je citlivý na velká a malá písmena. Příkaz CAT je rovněž chybný.

- Před příkaz můžete uvést libovolný počet mezer, například:[2]

  ```
  /home/larry# _ cat
  ```

  Takto zadaný příkaz je v pořádku a program cat se spustí.

- V příkazovém řádku také můžete pouze stisknou klávesu Enter. Ničeho se neobávejte, nic se nestane.

Předpokládejme, že jste spustili program cat. Asi jste zvědavi, co dělá. Pokud si myslíte, že se jedná o nějakou hru, pak asi budete zklamáni. Tento program je velice užitečným nástrojem, i když se to na první pohled nezdá. Napište jakýkoliv text a stiskněte Enter. Pak na obrazovce uvidíte:

```
/home/larry# cat
Help! I'm stuck in a Linux program!
Help! I'm stuck in a Linux program!
```

(Skloněným písmem je označeno to, co jsem v programu cat napsal.) Zdá se, že program pouze kopíruje text. To je někdy užitečné, ale program cat toho umí mnohem více. Pro tento okamžik program cat ukončíme a později si uvedeme příklady, kdy je mnohem užitečnější.

K ukončení většiny příkazů operačního systému Unix se používá kombinace kláves Ctrl+D[3]. Touto kombinací odešlete znak konce souboru EOF (end-of-file). Alternativně lze znak EOF považovat za konec textu, avšak v této knize jej budeme považovat za konec souboru. Znak EOF signalizuje programům operačního systému Unix, že jste ukončili zadávání dat. Pro program cat to znamená, že další data nemá zpracovávat a ukončí se.

Také si můžete vyzkoušet program sort. Jak napovídá jeho název, jedná se o program pro třídění. Jestliže zapíšete několik řádků a pak stisknete Ctrl+D, zobrazí se vám tyto řádky uspořádány. Programy tohoto druhu se nazývají **filtry**. Fungují tím způsobem, že nejdříve akceptují nějaký text, provedou s ním nějaké operace (například třídění) a pak modifikovaný text vypíší na obrazovce (nebo na jiném výstupním zařízení). Programy cat a sort jsou poněkud neobvyklé filtry, cat je neobvyklý v tom, že přečte text a neprovádí v něm žádné změny. Pro-

2 _ indikuje mezeru.

3 Podržte stisknutou klávesu Ctrl a stiskněte klávesu D.

gram `sort` je neobvyklý v tom, že čte řádky a neprovádí nic, dokud nepřečte znak EOF. Mnoho filtrů zpracovává data řádek po řádku - přečtou řádek, provedou nějaké modifikace a zobrazí řádek modifikovaný.

4.2 Pomozte si sami

V operačních systémech Unix existuje příkaz `man`,[4] jenž zobrazuje tzv. manuálové stránky. Originální manuálové stránky jsou pouze v anglickém jazyce, ale na jejich překladu se již pracuje. Zadejte například příkaz:

```
   /home/larry# man cat


CAT(1)                                                    CAT(1)
NAME
   cat - concentrate files and print on the standard output
SYNOPSIS
   cat [-benstuvAET] [--number] [--number-nonblank]
   [--squeeye-blank] [--show-nonprinting] [--show-ends]
   [--show-tabs] [--show-all] [--help] [--version] [--file...]
DESCRIPTION
   This manual page documents the GNU version of cat.   cat
```

Uvedený výpis představuje asi jednu stránku informací o příkazu `cat`. Nepředpokládejte, že budete okamžitě všemu rozumět. Manuálové stránky zpravidla předpokládají, že má jejich čtenář jisté znalosti o operačním systému Unix, tedy znalosti, které vy zatím nemáte. Když čtete manuálovou stránku, pak se v dolním řádku objeví blok s textem jako „`--more--`“ nebo „`Line 1`“. Jedná se o nápovědu, že k danému tématu existuje více informací.

Man po zobrazení první stránky vyčkává, co se rozhodnete dělat dále. Chcete-li pokračovat dále v prohlížení manuálových stránek, stiskněte klávesu Mezerník – pak se vám zobrazí následující stránka. Pokud chcete program man ukončit, stiskněte klávesu Q. Pak se vrátíte do příkazového řádku příkazového procesoru, který bude očekávat zadání dalšího příkazu.

Programu man lze také předat jisté argumenty, jež řídí jeho funkce. Předpokládejme například, že si chcete přečíst informace o všem, co souvisí s formátem Postscript (Postscript je řídící jazyk pro tisk na tiskárnách od firmy Adobe). Pak zadejte příkaz `man -k ps` nebo `man`

[4] Program `man` také zobrazí informace o systémovém volání, podprogramu, formátu souboru a mnoho dalších informací. V originálních verzích operačního systému Unix obsahuje manuálová stránka stejné informace jako tištěná dokumentace. Pro tento okamžik se spokojíme s nápovědou k syntaxi příkazu.

`-k Postscript`. Zobrazí se vám seznam všech příkazů, systémových volání a další dokumentace pojednávající o tomto tématu. Tento postup je velmi užitečný v případě, kdy chcete nalézt nějaký nástroj, ale nevíte, kde jej hledat, nebo nevíte, zda vůbec existuje.

4.3 Ukládání informací

Programy patřící do kategorie filtrů jsou velmi užitečné, zejména pokud patříte mezi pokročilejší uživatele. Zůstává zde však jeden problém. Jak ukládat, uchovávat a aktualizovat informace? Za tímto účelem nabízí operační systém Unix **soubory** a **adresáře**.

Adresář je něco jako pořadač, který místo svazků papíru obsahuje soubory. Velké pořadače mohou obsahovat několik dalších menších pořadačů. V operačním systému Unix se systém adresářů a souborů nazývá souborový systém. Zpočátku je každý souborový systém tvořen jediným adresářem, který se nazývá kořenový nebo také hlavní adresář. Uvnitř tohoto adresáře se mohou vyskytovat další adresáře a uvnitř nich se opět mohou vyskytovat adresáře (ty se někdy nazývají podadresáře). V každém adresáři mohou být uloženy soubory.[5]

Každý soubor a každý adresář má své vlastní jméno. Jméno může být buď krátké a může se shodovat se jménem adresáře nebo souboru uloženého jinde, nebo dlouhé (tzv. jméno včetně cesty), které je v rámci souborového systému na daném disku unikátní. Příkladem krátkého jména může být `joe`, zatímco odpovídajícím dlouhým jménem (t.j. jménem včetně cesty) je `/home/larry/joe`. Posloupnost znaků v dlouhém jménu ukončená znakem `/` se nazývá cesta. Cesta může být dekódována do posloupnosti adresářů. Následující příklad ilustruje, jak systém dospěje k souboru `/home/larry/joe`:

```
/home/larry/joe
```

První znak `/` indikuje kořenový adresář.

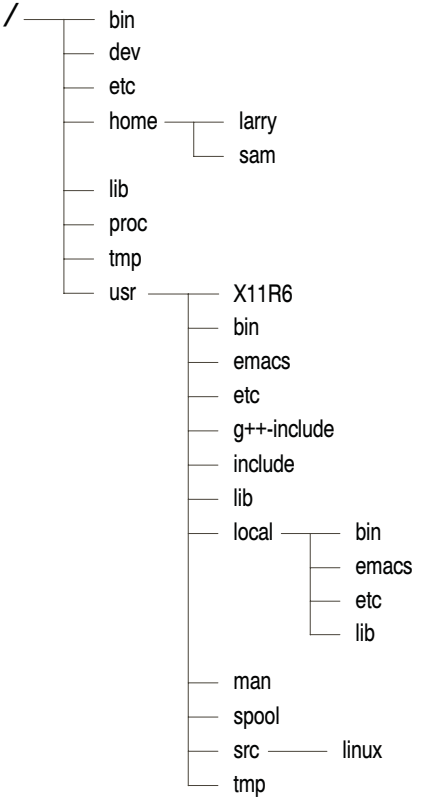
Následuje adresář se jménem `home`, který je podadresářem kořenového adresáře.

Další je adresář `larry`, který je podadresářem adresáře `home`.

Soubor `joe` je uvnitř adresáře `larry`. Cesta může odkazovat buď na adresář, nebo na soubor - `joe` tedy může být jak adresářem, tak i souborem. Všechny položky před krátkým jménem musejí být adresáře.

[5] Počet podadresářů může být omezen, ale v některých systémech nemusí. V operačním systému Linux jsem bez problémů vytvořil podadresáře až do úrovně 10.

Existuje jednoduchý způsob, jak strukturu adresářů vizualizovat. Tato vizualizace se nazývá stromový diagram a většina uživatelů asi zná stromové diagramy z operačního systému DOS (vytvářejí je známé programy pro správu souborů, jako je NORTON COMMANDER). Stromový diagram můžete vytvořit programem tree nebo v programu Midnight Commander, Linuxové obdobě Norton Commandera. Typickou stromovou strukturu adresářů používanou v operačním systému Linux znázorňuje obrázek 4.1. Poznamenejme, že zde uvedený diagram není kompletní. Úplná struktura operačního systému Linux obsahuje přes 8000 souborů! Zrovna tak platí, že zde mohou být uvedeny adresáře, jež se nevyskytují ve vaší instalaci a naopak, ve vaší instalaci mohou být adresáře, které v uvedeném obrázku uvedeny nejsou.



Obrázek 4.1

Typická (zkrácená) stromová struktura adresářů v operačním systému Unix

4.3.1 Prohledávání adresářů pomocí příkazu ls

Nyní máte jistě představu o tom, jak jsou v operačním systému Unix organizovány adresáře a soubory. Samozřejmě budete potřebovat nějaké nástroje, pomocí nichž budete moci se soubory a adresáři manipulovat. Prvním důležitým nástrojem je příkaz `ls`. Jedná se o příkaz, který zobrazuje seznam souborů. Nyní zadejte příkaz `ls` na úrovni příkazového řádku:

```
/home/larry# ls
/home/larry#
```

Žádný seznam se nezobrazil, což je v pořádku. Operační systém Unix pracuje přesně. Příkazu `ls` nebyly předány žádné parametry, proto se žádný seznam nezobrazil.

Jak jsme se již zmínili, při instalaci operačního systému Linux se nainstaluje více než 8000 souborů. Kde jsou tyto soubory uloženy? Abychom pochopili používání příkazu `ls`, musíme si vysvětlit pojem aktuální adresář. Podle nápovědy příkazového řádku poznáte, že aktuálním adresářem je `/home/larry`. Zde však nejsou uloženy žádné soubory, proto příkaz `ls` nezobrazil žádný seznam. Zkuste si zobrazit seznam souborů uložených v hlavním adresáři:

```
/home/larry# ls /
bin    etc    lostfound    proc   tmp
boot   home   misc         root   usr
dev    lib    mnt          sbin   var
/home/larry#
```

V předcházejícím příkazu „`ls /`“ jsme programu `ls` předali **parametr** „/“. Prvním slovem v příkazu je jméno příkazu a další slova jsou parametry. Obecně platí, že parametry modifikují funkce příkazů. V případě příkazu `ls` se jako první parametr uvádí jméno adresáře, jehož obsah má příkaz `ls` zobrazit. Některé příkazy mají speciální parametry, kterým se říká volby (options) nebo přepínače (switches). Vyzkoušejte si následující příkaz:

```
/home/larry# ls -F /
bin/   etc/   lostfound/   proc/  tmp/
boot/  home/  misc/        root/  usr/
dev/   lib/   mnt/         sbin/  var/
/home/larry#
```

Parametr `-F` se nazývá **volba**. Volba je speciální parametr, který začíná pomlčkou a modifikuje způsob provádění programu. V případě příkazu `ls` je parametr `-F` určen k rozlišení položek seznamu na adresáře, speciální soubory, programy a ostatní soubory. Každá položka končící znakem / je adresář. Později budeme hovořit o příkazu ls, který je opravdu velmi univerzální, podrobněji.

Nyní byste si měli udělat malé cvičení. Především se naučte, jak příkaz `ls` pracuje. Vyzkoušejte si zobrazit obsahy jiných adresářů podle obrázku 4.1. Některé z nich budou prázdné, jiné budou obsahovat spoustu souborů. Doporučuji, abyste si vyzkoušeli příkaz `ls` s parametrem `-F` i bez něj. Například adresář `/usr/local` by mohl vypadat takto:

```
/home/larry# ls /usr/local
Acrobat3   com   etc   include  lib   man       share      teTeX
bin              doc   games    info  libexec   sbin src
/home/larry#
```

Druhé cvičení bude více obecné. Spousta příkazů v operačním systému Unix se spouští podobně jako příkaz `ls`. Také mají nějaké parametry a nějaké volby, jež se zpravidla uvádějí jako jednoznakové řetězce za pomlčkou. Na rozdíl od příkazu `ls` však některé příkazy vyžadují parametry a/nebo volby. K vyjádření syntaxe příkazů budeme používat následující formu:

`ls [-aRF] [`*adresář*`]`

Když budeme popisovat nový příkaz, použijeme k jeho definici podobný syntaktický zápis. Prvním slovem je příkaz (v tomto případě `ls`). To, co následuje za příkazem, jsou parametry. Volitelné parametry jsou uzavřeny do hranatých závorek. Tzv. meta-proměnné jsou vyznačeny skloněným písmem - jsou to slova, která musejí být nahrazena skutečnými parametry. V uvedeném příkladu je meta-proměnnou *adresář*. Ta by měla být nahrazena jménem skutečného adresáře.

V případě voleb platí jiná pravidla. V syntaktické definici příkazu jsou uzavřeny v hranatých závorkách. V příkazu můžete použít kteroukoliv volbu nebo kteroukoliv jejich kombinaci. V případě příkazu `ls` máte tedy možnost použít osm kombinací voleb. Porovnejte výstup příkazů `ls -R` a `ls -F`.

4.3.2 Aktuální adresář a příkaz cd

`pwd`

Používání adresářů může být poněkud těžkopádné - asi by se vám nelíbilo, kdybyste museli pokaždé vypisovat celou cestu, kdykoliv byste si chtěli zpřístupnit nějaký soubor. V operačním systému Unix je zaveden pojem aktuálního adresáře. V příkladech, které jsme doposud použili, je aktuálním adresářem `/home/larry`. Pokud chcete zjistit, jaký je skutečný aktuální adresář, zadejte příkaz `pwd` (print working directory). V některých případech zobrazuje nápověda příkazového řádku i jméno počítače. To je užitečné pouze tehdy, když je počítač zapojen do sítě, ve které se vyskytuje více počítačů.

```
mousehouse> pwd
/home/larry
mousehouse>
```

`cd [`*adresář*`]`

Jak jste se mohli přesvědčit, příkaz `pwd` zobrazí aktuální adresář.[6] Příkaz `pwd` je tedy velmi jednoduchý. Většina příkazů v operačním systému Unix funguje implicitně v aktuálním adresáři, který lze změnit pomocí příkazu `cd`. Vyzkoušejte si například příkazy:

```
/home/larry# cd /home
/home# ls -F
larry/ sam/ shutdown/ steve/ user1/
/home#
```

Pokud vynecháte volitelný parametr *adresář*, pak se navrátíte do vašeho domovského adresáře, kterým je v našem případě `/home/larry`. Jinak se dostanete do adresáře specifikovaného prostřednictvím parametru. Například:

```
/home# cd
/home/larry# cd /
/# cd home
/home# cd /usr
/usr# cd local/bin
/usr/local/bin#
```

6 Někdy se také používá termín pracovní adresář. V této knize se budeme držet termínu aktuální adresář.

Jak jste si asi všimli, příkaz `cd` umožňuje specifikovat buď **absolutní cestu,** nebo **relativní cestu**. Absolutní cesta začíná znakem / a musí obsahovat všechny adresáře vedoucí k výslednému adresáři, do kterého se chcete přepnout. Relativní cesta se vztahuje k vašemu aktuálnímu adresáři. V předcházejícím příkladu jsme v adresáři `/usr` zadali relativní adresář `local/bin` - `local` je adresář pod adresářem `usr` a `bin` je adresář pod adresářem `local`. Podobně jsme použili relativní adresář v příkazu `cd home`.

V operačním systému Unix (podobně jako v operačním systému DOS) existují dva speciální adresáře, jež se používají pouze jako relativní. Jsou to adresáře „. “ a „. . “. Adresář „. “ specifikuje aktuální adresář a adresář „. . “ specifikuje nadřazený adresář. Jedná se tedy o zkratky, které existují v každém adresáři, ale ne vždy mají přesně ten význam, jak jsme jej popsali. Například hlavní adresář má jako nadřazený adresář sebe sama.

Soubor `./chapter-1` v aktuálním adresáři by měl být identický se souborem `chapter-1`. Některé příkazy vyžadují zadání typu `./chapter-1`, ale to není častý případ. Ve většině případů jsou `./chapter-1` a `chapter-1` identické specifikace souboru.

Adresář „. . “ je velmi užitečný při „pohybu“ v adresářové struktuře směrem nahoru:

```
/usr/local/bin# cd ..
/usr/local# ls -F
archives/    bin/   emacs@       etc/   ka9q/  lib/   tcl@
/usr/local# ls -F ../src
cweb/        linux/ xmris
/usr/local#
```

V tomto případě jsme se přepnuli do nadřazeného adresáře pomocí `cd ..` a pak jsme zobrazili obsah adresáře `/usr/src` z adresáře `/usr/local` pomocí příkazu `ls -F ../src`.

Pro domovský adresář existuje zkratka ~. Vyzkoušejte si následující příkaz:

```
/usr/local# ls -F ~/
/usr/local#
```

Opět jsme se přesvědčili, že v domovském adresáři nejsou žádné soubory. Užitečnost zkratky ~ pro domovský adresář vynikne zejména v případě, kdy začneme se soubory manipulovat.

4.3.3 Vytváření a odstraňování adresářů

`mkdir` *adresář1* [*adresář2* ... *adresářN*]

Vytváření vlastních adresářů je v operačním systému Unix opravdu jednoduché a představuje velmi užitečný nástroj pro organizaci a údržbu souborů. K vytvoření adresáře slouží příkaz `mkdir` (make directory).

Uveďme si jednoduchý příklad, abychom pochopili, jak příkaz `mkdir` funguje.

```
/home/larry# ls -F
/home/larry# mkdir report-1993
/home/larry# ls -F
report-1993/
/home/larry# cd report-1993
/home/larry/report-1993#
```

Příkaz `mkdir` může akceptovat více než jeden parametr. Předpokládá, že každý z nich představuje jméno adresáře, který se má vytvořit. Opět můžete specifikovat celou cestu nebo relativní adresář. V předcházejícím příkladu jsme použili relativní adresář `report-1993`.

```
/home/larry/report-1993# mkdir /home/larry/report-1993/chap1
  ~/report-1993/chap2
/home/larry/report-1993# ls -F
chap1/       chap2/
/home/larry/report-1993#
```

`rmdir` *adresář1* [*adresář2* ... *adresářN*]

Jakýmsi opakem k příkazu `mkdir` je příkaz `rmdir` (remove directory). `rmdir` funguje přesně jako `mkdir`. Podívejte se na následující příklad:

```
/home/larry/report-1993# rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report-1993# ls -F
chap2/
/home/larry/report-1993# cd ..
/home/larry# rmdir report-1993
rmdir: report-1993: Directory not empty
/home/larry#
```

Jak jste si zřejmě všimli, příkaz `rmdir` odmítnul odstranit adresář, který buď neexistuje nebo není prázdný. Uvědomte si, že adresář `report-1993` obsahuje podadresář `chap2`, proto nelze adresář `report-1993` odstranit.

4.4 Manipulace se soubory

Práce s adresáři je sice zajímavá, ale stále neřeší náš problém s ukládáním, aktualizací a udržováním informací. Tento problém se řeší prostřednictvím **souborů**.

Vytváření a editování souborů se budeme věnovat v několika následujících kapitolách.

Základními příkazy pro manipulaci se soubory jsou v operačním systému Unix příkazy `cp` (copy), `mv` (move) a `rm` (remove).

4.4.1 Příkaz pro kopírování souborů cp

`cp [-i]` *zdroj cíl*
`cp [-i]` *soubor1 soubor2 ... souborN cílový adresář*[7]

Příkaz `cp` je v operačním systému Unix velmi důležitým a výkonným příkazem. V jedné sekundě vám umožní přemístit tolik informací, kolik jich středověký mnich přemístil za celý rok.

Jestliže nemáte na svém pevném disku dostatek prostoru, buďte při používání příkazu `cp` opatrní. Nikdo nechce vidět zprávu „Disk full“, když pracuje s důležitými soubory. Příkaz `cp` je dále schopen přepsat důležité soubory bez jakéhokoliv varování - tomuto nebezpečí se budeme věnovat podrobněji.

Nejprve se zaměříme na první definici příkazu `cp`. Prvním parametrem příkazu `cp` je soubor, který se má kopírovat (tzv. zdrojový soubor, source file). Druhým parametrem je soubor, který má být kopií prvního (tzv. cílový soubor, destination file). Při kopírování můžete vytvářet soubory s novým jménem nebo můžete soubory se stejným jménem kopírovat do jiných adresářů. Vyzkoušejte si následující příklady:

```
/home/larry# ls -F /etc/passwd
/etc/passwd
/home/larry# cp /etc/passwd .
/home/larry# ls -F
```

[7] Příkaz `cp` má v předloze dva řádky, neboť význam druhého parametru může být rozdílný v závislosti na počtu parametrů.

```
passwd
/home/larry# cp passwd frog
/home/larry# ls -F
frog passwd
/home/larry#
```

Prvním příkazem `cp` se z adresáře `/etc` zkopíroval soubor `passwd` (jenž obsahuje jména všech uživatelů systému Unix spolu se zakódovanými hesly) do mého domovského adresáře. Toto platí pouze za předpokladu, že nepoužíváme tzv. shadow passwords, kdy jsou zakódovaná hesla uložena v souboru `/etc/shadow`. Příkaz `cp` neruší zdrojový soubor, proto jsem neudělal nic, co by nějak poškodilo systém. Nyní existují v mém systému dvě kopie souboru `passwd`, jedna v adresáři `/etc` a druhá v mém domovském adresáři `/home/larry` a obě kopie mají jméno `passwd`.

Třetí kopii jsem vytvořil příkazem `cp passwd frog`. Nyní jsou v mém systému tři kopie souboru: `/etc/passwd`, `/home/larry/passwd` a `/home/larry/frog`. Posledně vytvořená kopie má jiné jméno, ale obsahuje stejná data.

Příkaz `cp` je schopen kopírovat soubory mezi adresáři, a to v tom případě, kdy je prvním parametrem jméno souboru a druhým parametrem je jméno adresáře. Pak zůstane jméno cílového souboru shodné se jménem zdrojového souboru.

Kopírovat soubory a přitom měnit jejich jména lze jen tehdy, když se v příkazu `cp` jako parametry uvedou jména souborů. S příkazem `cp` je spojeno jedno nebezpečí. Když například zadáte příkaz `cp /etc/passwd /etc/group`, vytvoří příkaz `cp` nový soubor `group`, jehož obsah bude identický s obsahem souboru `/etc/passwd`. Pokud by však soubor `/etc/group` již existoval, pak jej přepíšete bez jakéhokoliv varování a bez možnosti starou verzi souboru `/etc/group` obnovit.

Podívejme se na další příklad použití příkazu `cp`:

```
/home/larry# ls -F
frog passwd
home/larry#  mkdir passwd_version
home/larry#  cp frog passwd passwd_version
home/larry#  ls -F
frog         passwd       passwd_version/
home/larry#  ls -F passwd_version
frog passwd
home/larry#
```

Jak jsem nyní použil příkaz `cp`? Je evidentní, že příkaz `cp` může akceptovat více než dva parametry (nyní jsem příkaz `cp` použil podle druhé definice). Příkaz `cp` v předcházejícím příkladu zkopíroval všechny uvedené soubory (`frog` a `passwd`) do adresáře `passwd_version`, který je uveden jako poslední parametr. V podstatě platí, že příkaz `cp` je schopen akceptovat jakýkoliv počet parametrů, přičemž prvních n-1 považuje za jména souborů a poslední považuje za jméno adresáře, do kterého se mají uvedené soubory kopírovat.

Jestliže kopírujete v daném okamžiku více než jeden soubor, pak jej nemůžete přejmenovat - kopírované soubory si uchovají svá původní jména. Co se stane, když zadáte příkaz `cp frog passwd toad`, kde `frog` a `passwd` jsou soubory a přitom `toad` není adresář? Vyzkoušejte si takový příkaz a uvidíte.

4.4.2 Odstranění souborů pomocí příkazu rm

`rm [-i]` *soubor1 soubor2 ... souborN*

Nyní, když jsme se naučili, jak kopírovat a vytvářet soubory (později si probereme jiné způsoby vytváření souborů), bude jistě užitečné vědět, jak soubory rušit. Ve skutečnosti je to velmi jednoduché. Příkaz pro odstraňování souborů má název `rm` a zruší všechny soubory, jejichž jména se uvedou jako parametry.

Například:

```
/home/larry# ls -F
frog passwd passwd_version/
/home/larry# rm frog toad passwd
rm: toad: No such file or directory
/home/larry# ls -F
passwd_version/
/home/larry#
```

Jak asi vidíte, příkaz `rm` je velmi nevlídný. Nejen, že se vás nezeptá, zda má uvedené soubory skutečně zrušit, ale provede zrušení existujících souborů, i když vlastně příkaz nebyl správně zadán (soubor `toad` neexistoval). Příkaz `rm` může být skutečně nebezpečný. Podívejme se na rozdíl mezi následujícími dvěma posloupnostmi příkazů:

```
/home/larry# ls -F
toad frog/
/home/larry# ls -F frog
toad
```

```
/home/larry# rm frog/toad
/home/larry#

/home/larry# rm frog toad
rm: frog is a directory
/home/larry# ls -F
frog/
/home/larry#
```

Pátý řádek v první posloupnosti a první řádek v druhé se liší jediným znakem. Uvedený příklad má ilustrovat, že opomenutí jediného znaku může v operačním systému Unix způsobit, že provedete něco úplně jiného, než jste původně chtěli. Proto je velmi důležité, abyste raději několikrát zkontrolovali zapsaný příkaz, než stisknete klávesu Entr.

4.4.3 Přesouvání souborů pomocí příkazu mv

`mv [-i]` *starý nový*
`mv [-i]` *soubor1 soubor2 ...souborN adresář*

Nakonec se podívejme na příkaz pro tzv. přesouvání souborů. Příkaz má název `mv` a je podobný příkazu `cp`. Na rozdíl od příkazu `cp` však zdrojové soubory po jejich zkopírování odstraňuje. Příkaz `mv` je tedy jakousi kombinací příkazů `cp` a `rm`. Uveďme si na následujícím příkladu, jak příkaz `mv` funguje:

```
/home/larry# cp /etc/passwd .
/home/larry# ls -F
passwd
/home/larry# mv passwd frog
/home/larry# ls -F
frog
/home/larry# mkdir report
/home/larry# mv frog report
/home/larry# ls -F
report/
/home/larry# ls -F report
frog
/home/larry#
```

Jak jste si pravděpodobně všimli, příkaz `mv` soubor přejmenuje, pokud je druhým parametrem jméno souboru. Jestliže je druhým parametrem jméno adresáře, pak se soubor přesune do nového adresáře a přitom zůstane uchováno jeho původní jméno.

Podobně jako v případě příkazu `cp` musíte být při používání příkazu `mv` velmi opatrní. Příkaz neprovádí kontrolu, zda cílový soubor existuje nebo ne. Kdyby například v mém adresáři `report` již existoval soubor `frog`, pak by se příkazem `mv frog report` nejdříve zrušil soubor `/report/frog` a pak by se nahradil obsahem souboru `~/frog`.

Ve skutečnosti existuje možnost, jak přimět příkazy `rm`, `cp` a `mv`, aby vás upozorňovali na možnost přepsání či zrušení souborů. Všechny tyto příkazy akceptují volbu `-i`. Po uvedení této volby bude uživatel před každým zrušením souboru upozorněn. Dokonce můžete použít tzv. **alias** (druhé jméno) a zařídit, aby vás například příkaz `rm` upozorňoval na zrušení souboru, aniž uvedete volbu `-i`. O tom však budeme diskutovat až v oddíle 9.1.3.

Kapitola

5

Systém X Window

Tato kapitola se týká pouze uživatelů systému X Window. Jestliže máte před sebou barevnou grafickou obrazovku s mnoha okny a kurzorem, kterým lze pohybovat pouze prostřednictvím myši, pak jste zřejmě uživateli systému X Window. Pokud však máte před sebou obrazovku s bílým textem na černém pozadí, pak momentálně systém X Window nemáte aktivován. Budete-li chtít systém X Window aktivovat, přečtěte si oddíl 5.1.

5.1 Spuštění a ukončení systému X Window

5.1.1 Spuštění systému X Window

Systém X Window můžete spustit i v případě, kdy se automaticky nenastartuje ihned po zavedení operačního systému. Existují dva příkazy, kterými lze nastartovat systém X Window: `startx` a `xinit`. Nejdříve vyzkoušejte příkaz `startx`. Pokud příkazový procesor ohlásí, že takový příkaz neexistuje, pak zkuste `xinit`. Pokud ani v tomto případě nedojde ke spuštění systému X Window, pak jej pravděpodobně nemáte nainstalován. Prostudujte si příslušnou dokumentaci.

Jestliže se příkaz spustí, ale po nějakém čase se opět vrátíte do příkazového řádku příkazového procesoru, pak je systém X Window sice instalován, ale není nakonfigurován. I v tomto případě si budete muset důkladně přečíst dokumentaci k instalaci systému X Window.

5.1.2 Ukončení systému X Window

V závislosti na tom, jak je váš systém X Window konfigurován, existují dva možné způsoby, jak systém X Window ukončit. První závisí na tom, zda váš systém X Window řídí správce oken či nikoliv. Pokud ano, můžete systém X Window ukončit prostřednictvím nabídky (viz oddíl 5.4.8). Chcete-li si zobrazit nabídku, klepněte myší na pozadí pracovní plochy.

Pak se vám mezi jinými objeví položka „Exit Window Manager“ nebo „Exit X“ nebo jiná položka obsahující slovo „Exit“. Pokuste se najít tuto položku (počítejte s tím, že zde může existovat více než jedna nabídka - vyzkoušejte různá tlačítka myši) a ukončete systém X Window.

Druhá metoda ukončení systému X Window spočívá v tom, že se k řízení systému X Window využije speciálního okna `xterm`. V takovém případě byste měli najít okno nazvané „login“ nebo „system `xterm`“. Přesuňte kurzor myši do tohoto okna a zadejte příkaz `exit`.

Jestliže se systém X Window automaticky spustil, když jste se přihlásili do systému, pak by vás jedna z výše uvedených metod ukončení systému X Window měla automaticky odhlásit. Jestliže jste však systém X Window nastartovali z příkazového řádku, pak se do tohoto příkazového řádku po ukončení systému X Window opět vrátíte (pokud se chcete odhlásit, zadejte příkaz `logout`).

5.2 Co je systém X Window?

Systém X Window je distribuované grafické uživatelské prostředí původně vyvinuté v akademické instituci Massachusetts Institute of Technology. Později byl předán skupině distributorů s názvem „The X Consortium“. Zde se dodnes udržuje a nadále vyvíjí.

Systém X Window (někdy zkracován jako „X“[1]) se každých několik málo let distribuuje v nové verzi, uváděné jako „release“. Poslední verze má označení X11R6, tedy „release“ 6. Číslo 11 značí hlavní verzi systému X Window, avšak toto označení je trvalé, protože se nová verze neplánuje.

V souvislosti se systémem X Window existují dva důležité pojmy, se kterými byste se měli seznámit. Program, jenž běží pod systémem X Window, se nazývá **klient**. Například `xterm` je klient, který se spustí v okamžiku, kdy se přihlašujete do systému. Program, který poskytuje služby ostatním programům typu klient, se nazývá **server**. Server například kreslí okno pro program `xterm` a zajišťuje komunikaci s uživatelem.

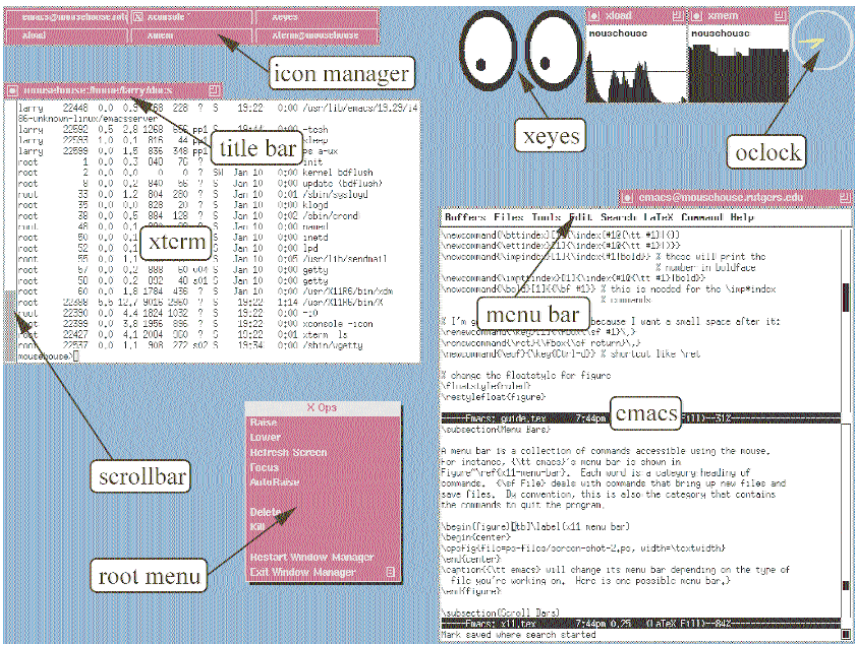
[1] Existuje několik přijatelných způsobů, jak nazývat systém X Window. Nejčastěji se vyskytuje zkratka „X“ nebo název „X Window“.

Protože jsou server a klient dva odlišné programy, je možné zajistit, aby server běžel na jednom počítači a klient na jiném. Vzhledem k tomu, že grafické uživatelské rozhraní dodržuje jistý standard, můžete v systému X Window spustit program na vzdáleném počítači (třeba na druhém konci světa) a přitom můžete grafické rozhraní tohoto programu vidět na svém počítači.

Dalším důležitým termínem, se kterým byste se měli seznámit, je **správce oken** (window manager). Jedná se o speciální program typu klient, který určuje pozici jednotlivých oken na pracovní ploše systému X Window a řídí způsoby, kterými uživatel například přemisťuje okna. Samotný server v podstatě pro uživatele nic nedělá. Pouze představuje rozhraní mezi uživatelem a klientem.

5.3 Co se nachází na pracovní ploše X Window

Když poprvé nastartujete systém X Window, zároveň se nastartuje řada dalších programů. Jako první se nastartuje server. Další v pořadí se nastartují některé programy typu klient - jejich pořadí a typ závisí na distribuci a není standardizován. Je však pravděpodobné, že mezi těmito klienty je správce oken (buď program `fvwm` nebo `twm`), terminálové okno `xterm` a hodiny `xclock`.



Obrázek 5.1

Příklad standardní obrazovky systému X Window. V tomto příkladu uživatel spustil program twm. Standardní hodiny byly nahrazeny jiným programem zvaným `oclock`.

5.3.1 Program XClock

```
xclock [-digital] [-analog] [-update seconds] [-hands color]
```

Program `xclock` (hodiny) funguje přesně tak, jak asi očekáváte. Zobrazuje ciferník s vteřinovou ručičkou a malou i velkou ručičkou v malém okně.

Prostřednictvím myši nemůžete vlastnosti okna s hodinami příliš modifikovat. Nanejvýš můžete měnit jeho velikost. Pokud však program xclock spustíte z příkazového řádku, pak můžete prostřednictvím jeho parametrů nastavit různé vlastnosti. Když zadáte příkaz `xclock -digital`, zobrazí se digitální hodiny. Pomocí parametru `-update` můžete nastavit, zda se má vteřinová ručička aktualizovat každou sekundu (`-update 1`) nebo například každých pět sekund (`update -5`).

Chcete-li vědět více o programu `xclock` a jeho parametrech, podívejte se do manuálových stránek. Stačí v příkazovém řádku zadat `man xclock`. Budete-li chtít spustit více programů `xclock` najednou, přečtěte si oddíl 6.4, ve kterém budeme diskutovat o provozování více úloh současně.

Jestliže máte program `xclock` spuštěn na popředí (což je běžný způsob v provozování programů) a chcete jej ukončit, stiskněte kombinaci kláves Ctrl-C.

5.3.2 Program xterm

Okno s příkazovým řádkem uvnitř (tedy v našem případě okno s výzvou `/home/larry#`) je řízeno programem, jenž se nazývá xterm. Což je zvláštní a přitom komplikovaný program. Na první pohled se zdá, že toho moc neumí, ale jeho prostřednictvím lze vykonat spoustu užitečné práce. Emuluje terminál, a proto v něm lze spouštět všechny textově orientované aplikace operačního systému Unix. Také obsahuje vyrovnávací paměť uchovávající jistou množinu příkazů - proto se můžete snadno vracet k dříve zadaným příkazům (podrobněji se tímto tématem budeme zabývat v oddílu 5.6.3).

Celá tato kniha je zaměřena na příkazy zadávané v příkazovém řádku terminálu, proto je program xterm v systému X Window tak důležitý. Chcete-li něco zadávat v okně tohoto programu xterm, musí být toto okno aktivní. To znamená, že musíte do okna terminálu přesunout kurzor myši. Způsob, kterým se nastavuje aktivní okno obecně, závisí na správci oken.

Program xterm představuje jednu z možností, jak v systému X Window spouštět více programů současně. Programy běžící v systému X Window jsou standardními programy operačního systému Unix, proto mohou být spuštěny z terminálového okna. Protože dlouhodobé progra-

my spouštěné z terminálového okna zablokují program `xterm` po celou dobu svého běhu, spouští se takové programy zpravidla na pozadí. Více informací o tomto tématu uvedeme v oddílu 6.4.

5.4 Správce oken

V operačním systému Linux existují dva běžně používané programy pro správu oken. První z nich, twm, je zkratkou pro „Tab Window Manager“. Druhý program je menší a má název fvwm („F(?) Virtual Window Manager“). Oba programy jsou konfigurovatelné, což znamená, že si uživatel může definovat význam ovládacích klíčů a způsob práce s myší.

Konfiguraci programu twm je věnován oddíl 9.2.1 a konfiguraci programu fvwm probereme v oddílu 9.2.2.

5.4.1 Jak se vytvářejí nová okna

Při vytvoření nového okna existují tři možné varianty, jež realizuje správce oken. V poslední době se objevuje spousta nových správců oken, např. `fvwm95`, `gnome` nebo `kwm`. Správce oken může být nakonfigurován tak, že se zobrazí rám nového okna a uživatel má možnost okno umístit kamkoliv na obrazovku. Tento způsob umístění okna se nazývá **ruční umístění** (manual placement). Jestliže se před vámi objeví rám okna, jednoduše jej pomocí kurzoru myši nastavte na požadovanou pozici a stiskněte levé tlačítko.

Je také možné, že správce oken umístí nové okno na pracovní ploše systému X Window dle „svého uvážení“. Takové umístění okna se nazývá **náhodné umístění** (random placement).

Třetí varianta spočívá v tom, že se okno náležící jisté aplikaci pokaždé otevře na stejné pozici. Správce oken může být nakonfigurován tak, aby pro vybrané aplikace otevíral okna s předem definovanou velikostí na předem definované pozici.

Jako příklad může sloužit program `xclock`, kdy asi budete chtít zobrazovat hodiny v horním levém rohu pracovní plochy systému X Window.

5.4.2 Aktivní okno

Správce oken řídí spoustu důležitých nastavení. Vás bude určitě zajímat, jak nastavit dané okno (například terminálové okno) tak, abyste v něm mohli zadávat příkazy. Aktivní okno je nejčastěji určeno pozicí kurzoru myši. Jestliže je kurzor myši umístěn na jednom z terminálových oken,[2] pak v tomto okně můžete zadávat příkazy z klávesnice. V tom spočívá rozdíl od

[2] V daném okamžiku můžete spustit několik programů `xterm` současně.

jiných operačních systémů, jako je OS/2, Microsoft Windows nebo Macintosh, kde musíte na dané okno (chcete-li, aby bylo aktivní) klepnout myší. V systému X Window obvykle platí, že když kurzor myši opustí rámec okna, pak v tomto okně již nemůžete zadávat příkazy.

Poznamenejme, že oba správce oken (tedy program twm a fvwm) lze konfigurovat tak, že se aktivace oken bude realizovat stejně jako například v operačním systému OS/2. Popis konfigurace správce oken najdete v příslušné dokumentaci, nebo můžete požadované konfigurace dosáhnout metodou pokusů a omylů.

5.4.3 Přemisťování oken

V systému X Window lze rovněž konfigurovat způsob, kterým mají být okna přemisťována. Moje osobní konfigurace programu twm obsahuje tři způsoby, jak posouvat okna po pracovní ploše. Nejznámější způsob spočívá v tom, že se kurzor myši umístí na titulní (hlavní) lištu okna, stiskne kterékoliv tlačítko (levé, pravé nebo střední[3]), a pak se okno přesune na novou pozici. Je ovšem velmi pravděpodobné, že je váš systém X Window konfigurován tak, aby se okna přemisťovala pomocí levého tlačítka (tak, jak je tomu u ostatních operačních systémů).

Jiný způsob přemisťování oken může spočívat v tom, že se využívá některé klávesy na klávesnici. Například moje vlastní konfigurace umožňuje stisknout klávesu Alt, nastavit kurzor myši kamkoliv do rámce okna a pak po stisknutí levého tlačítka okno přesunout na novou pozici.

I zde platí, že si konfiguraci správce oken můžete měnit metodou pokusů a omylů, nebo si prostudujte příslušnou dokumentaci. Budete-li se pokoušet interpretovat konfigurační soubor správce oken, přečtěte si oddíl 9.2.1 (pro program twm) nebo oddíl 9.2.2 (pro program fvwm).

5.4.4 Překrývání oken

Protože v systému X Window může být otevřeno několik oken najednou, má smysl hovořit o překrývání oken. Přestože okna a samotná pracovní plocha systému X Window jsou dvourozměrné, mohou se okna navzájem překrývat, jako by představovaly třírozměrné útvary. To znamená, že vybrané okno může být zobrazeno v popředí a částečně nebo úplně překrývat okna na pozadí. V souvislosti s překrýváním oken existuje několik operací:

- **Vyzvednutí** (raising) okna do popředí. Toho lze zpravidla dosáhnout klepnutím jistým tlačítkem myši na titulní lištu vybraného okna. Podle konfigurace správce oken to může být kterékoliv tlačítko; také je možné dosáhnout vyzvednutí okna do popředí pomocí více než jednoho tlačítka.

[3] Některé osobní počítače mají dvoutlačítkovou myš. Pak můžete prostřední tlačítko simulovat současným stisknutím pravého a levého tlačítka.

- **Zasunutí** (lowering) okna do pozadí. Toho lze obvykle dosáhnout klepnutím jiného tlačítka myši na titulní lištu okna. Dokonce lze nakonfigurovat správce oken tak, že k vyzvednutí i zasunutí okna se použije totéž tlačítko myši - je-li okno v popředí, pak se zasune do pozadí a je-li v pozadí, vyzvedne se do popředí.

- **Cyklické** procházení všech oken. Správce oken může být konfigurován tak, že po stisknutí jisté klávesy se postupně objevuje ve stanoveném pořadí každé otevřené okno v popředí.

5.4.5 Transformace okna do ikony

Nyní se podívejme na další operace, které umožňují transformovat (nebo také skrýt) okno do ikony. V závislosti na konfiguraci správce oken lze tohoto efektu dosáhnout několika různými způsoby. Při používání správce oken twm si většina lidí nakonfiguruje tzv. **správce ikon** (icon manager). Jedná se o speciální okno, které obsahuje seznam jmen všech ostatních oken otevřených na pracovní ploše. Pokud klepnete jistým tlačítkem myši na konkrétní jméno, dané okno se transformuje na ikonu. Okno je stále aktivní, ale nevidíte jej. Pomocí jiného tlačítka můžete provést inverzní operaci - ikona se zpět transformuje na okno.

Uvedené vlastnosti mohou být velmi užitečné. Předpokládejme například, že máte otevřenu spoustu terminálových oken pro příležitostnou komunikaci se vzdálenými počítači. Kdyby zůstala všechna okna otevřena, měli byste na pracovní ploše nepřehledno. Protože však s těmito terminálovými okny pracujete pouze příležitostně, můžete je transformovat do ikon a získat tak větší přehled. Jediné malé nebezpečí spočívá v tom, že zapomenete na některé okno transformované do ikony a zbytečně si otevřete pro stejný účel nové okno.

Pokud jde o umisťování ikon, lze správce oken nakonfigurovat tak, aby se ikony automaticky řadily na spodní okraj pracovní plochy.

5.4.6 Jak měnit velikost oken

V systému X Window existuje několik způsobů, jak měnit velikost oken. Tyto způsoby opět závisejí na konfiguraci správce oken. První a asi nejpoužívanější metoda spočívá v tom, že klepnete na rámeček ohraničující okno levým tlačítkem myši, držíte je stisknuté a rozměr okna nastavíte na požadovanou velikost (tento způsob je běžný i u ostatních operačních systémů, jako je Microsoft Windows nebo OS/2).

Další metoda je založena na speciálním tlačítku umístěném v titulní liště okna. Na obrázku 5.1 si můžete všimnout malého tlačítka na pravé straně titulní lišty každého okna. Stačí klepnout levým tlačítkem myši na toto tlačítko a nastavit rozměr okna na požadovanou velikost. Jakmile má okno příslušné rozměry, levé tlačítko myši uvolněte.

5.4.7 Nastavení maximální velikosti okna

Většina správců oken podporuje tzv. maximalizaci oken, tedy nastavení rozměrů okna tak, aby pokrylo celou obrazovku. Při použití správce oken `twm` lze dokonce nastavit takovou konfiguraci, aby bylo možné maximalizovat pouze vertikální rozměr, horizontální rozměr nebo oba rozměry najednou. Tento proces je v programu `twm` označován jako „zooming“ (zvětšování), ale častěji se dává přednost termínu „maximization“. Různé aplikace reagují na změnu rozměrů okna různě. Například `xterm` vám poskytne větší pracovní prostor a nezvětšuje přitom velikost fontů.

Obecně lze říci, že proces maximalizace oken není bohužel v systému X Window standardizován.

5.4.8 Nabídky

Dalším úkolem správce oken je řídit funkci nabídek a umožnit tak uživateli rychle realizovat úkony, které se stále dokola opakují. Například lze vytvořit nabídku, která umožní automaticky a rychle spustit editor emacs nebo terminál `xterm`. Obecně platí, že programy běžící v systému X Window můžete spouštět buď z terminálového okna, což je pomalejší a nepohodlné, nebo z vhodně nakonfigurovaných nabídek, což je rychlé a pohodlné.

Různé nabídky mohou být zpřístupněny klepnutím myši v základním okně (často se pro toto okno používá označení pracovní plocha), tedy okně, které nelze přemisťovat a které obsahuje všechna ostatní okna. Pozadí základního okna je implicitně šedé[4]. Chcete-li vyvolat nabídku, stačí umístit kurzor myši na pozadí základního okna a stisknout tlačítko. V nabídce si opět pomocí kurzoru myši vyberte požadovanou položku (aniž uvolníte tlačítko myši) a uvolněním tlačítka ji spusťte.

5.5 Atributy systému X Window

Existuje spousta programů, jež využívají vlastností systému X Window. Některé programy, jako je editor emacs, mohou být spuštěny jako textově orientované aplikace a zároveň mohou být spuštěny v prostředí systému X Window. Existuje však řada programů, které mohou běžet pouze v prostředí X Window.

[4] Existuje program xfishtank, který na pozadí základního okna simuluje akvárium a také spousta dalších programů pro změnu pozadí v systému X Window.

5.5.1 Geometrie

Existuje několik aspektů, které mají programy běžící pod systémem X Window společné. První z nich lze označit jako geometrii. Atributy geometrie popisují velikost a umístění okna a tvoří je čtyři komponenty.

- Horizontální rozměr, zpravidla měřený v grafických bodech. (Grafický bod je nejmenší jednotka na obrazovce, které lze přiřadit barvu. Systém X Window běžně pracuje s rozlišovací schopností 1024 bodů horizontálně a 768 bodů vertikálně.) Některé aplikace (`xterm` nebo `emacs`) měří velikost okna ve znacích. Například šířka okna je dána počtem znaků na řádku (nejčastěji osmdesát), který může aplikace zobrazit.

- Vertikální rozměr, opět obvykle měřený v grafických bodech. Rovněž je možné stanovit vertikální rozměr ve znacích.

- Horizontální vzdálenost od okrajů obrazovky. Například číslo `+35` znamená, že levý okraj okna bude vzdálen 35 grafických bodů od levého okraje obrazovky. Na druhé straně číslo `-50` bude znamenat, že pravý okraj okna bude vzdálen padesát grafických bodů od pravého okraje obrazovky. Obecně platí, že není možné inicializovat okno mimo rámec obrazovky, i když je pak možné okno mimo rámec obrazovky přesunout.

- Vertikální vzdálenost od horního nebo dolního okraje obrazovky. Kladná vertikální vzdálenost je měřena od horního okraje obrazovky a záporná je měřena od spodního okraje obrazovky. Všechny čtyři komponenty definující geometrii okna se uvádějí prostřednictvím jediného řetězce. Například `503x73-78+0` znamená, že okno bude mít šířku 503 grafických bodů, výšku 73 grafických bodů a že bude umístěno vpravo bezprostředně pod horním okrajem obrazovky. Obecný tvar řetězce pro specifikaci geometrie okna je tedy: `hsizexvsize+hplace+vplace`.

5.5.2 Display

Každé aplikaci v systému X Window je přiřazen tzv. display, což je jméno obrazovky, kterou řídí server systému X Window. Displej se skládá ze tří komponent:

- Jméno počítače, na kterém server běží. Při samostatné instalaci operačního systému Linux běží server vždy na stejném systému jako klient. Ve většině případů může být jméno počítače vynecháno.

- Číslo serveru, který na daném počítači běží. Na každém jednotlivém počítači může běžet několik serverů systému X Window současně (v případě operačního systému Unix to není příliš častý případ). Pak každý z nich musí mít unikátní číslo.

- Číslo obrazovky. Systém X Window podporuje servery řídící v daném okamžiku více nežjednu obrazovku. Některé aplikace je účelné provozovat na více než jednom monitoru a pak je nutné, aby server řídil více monitorů. Není totiž efektivní, aby byl pro každý monitor spuštěn zvláštní server.

Uvedené tři atributy lze specifikovat prostřednictvím jediného řetězce v následujícím formátu: *machine:server-number.screen-number.*

Například moje aplikace mají nastavenu hodnotu pro display jako `:0.0`. To znamená, že systém pracuje s první obrazovkou, řídí jej první server a běží na lokálním počítači. Kdybych však používal vzdálený počítač, pak bude display nastaven jako `mousehouse:0.0`.

Implicitně se řetězec definující hodnotu display čte ze systémové proměnné `DISPLAY` (viz oddíl 9.1.4) a může být předefinován parametry příkazového řádku pro spuštění systému X Window (viz. tabulka 5.2). Chcete-li se přesvědčit, zda je systémová proměnná nastavena, zadejte příkaz `echo $DISPLAY`.

5.6 Společné vlastnosti

Systém X Window představuje uživatelské grafické rozhraní, které lze prakticky ve všech aspektech konfigurovat. Je téměř nemožné říci, jak jeho jednotlivé komponenty fungují, protože to především závisí na konfiguraci.

jméno	následováno čím	příklad
`-geometry`	geometrií okna	`xterm -geometry 60x24+0+90`
`-display`	display, ve kterém se má program objevit	`xterm -display lionsden:0.0`
`-fg`	primární barva popředí	`xterm -fg yellow`
`-bg`	primární barva pozadí	`xterm -bg blue`

Tabulka 5.2
Standardní volby pro programy běžící pod systémem X Window

Každá vlastnost může být překonfigurována, změněna nebo úplně nahrazena jinou. Proto je obtížné jednoznačně popsat chování jednotlivých prvků tvořících toto rozhraní. Zatím jsme popsali to, co popsat lze: různé správce oken a možnosti jejich konfigurace.

Situaci ještě více komplikuje skutečnost, že různé aplikace jsou založeny na různých grafických knihovnách. Pro tyto knihovny se vžil název „knihovny přípravků“ - widget sets. Spolu se systémem X Window se distribuuje „Athena“, grafický systém vyvinutý v Massachusetts

Institute of Technology. Systém Athena se běžně používá ve volně šířitelném programovém vybavení. Má tu nevýhodu, že nevytváří příliš dobře vyhlížející grafické objekty a jeho používání je poměrně těžkopádné.

Další populární knihovnou je „Motif“. Motif patří ke komerčním produktům a má podobné vlastnosti jako uživatelské grafické rozhraní používané v operačním systému Microsoft Windows. Tuto knihovnu využívá spousta komerčních a některé volně šířitelné aplikace. Populární program pro prohlížení stránek WWW Netscape také využívá knihovnu Motif.

Pokusme se nyní popsat některé společné vlastnosti systému X Window.

5.6.1 Tlačítka

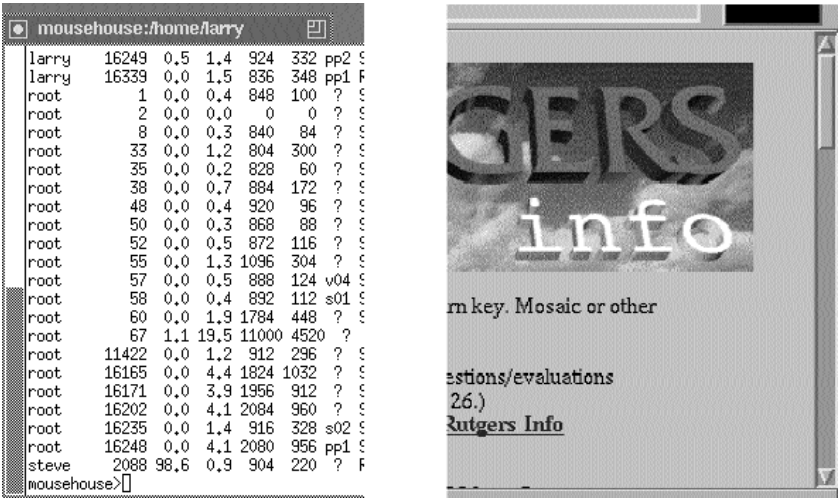
Tlačítka jsou prvky grafického rozhraní, jež se nejsnadněji používají. Stačí na tlačítko přesunout kurzor myši a klepnout levým tlačítkem. Tlačítka systémů Athena a Motif fungují stejně, rozdíly mezi nimi jsou pouze kosmetické.

5.6.2 Nabídkové lišty

Nabídková lišta je soubor příkazů zpřístupněných pomocí myši. Na obrázku 5.3 je například zobrazena nabídka editoru `emacs`. Každé slovo v nabídce představuje záhlaví jisté množiny dalších příkazů. Například položka File obsahuje příkazy pro zavádění a ukládání souborů. Podle zažité konvence zde najdete i příkaz k ukončení editoru.

Chcete-li zpřístupnit některý příkaz z nabídky, přesuňte kurzor myši na příslušnou nabídku (například File) a stiskněte levé tlačítko myši (držte je stále stisknuté). Pak se vám zobrazí další nabídka příkazů. Chcete-li si jeden z nich vybrat, přesuňte kurzor myši na požadovaný příkaz a uvolněte tlačítko myši. Některé nabídkové lišty vyžadují, abyste na požadované položce klikli myší. Pak program vyčkává, až klepnete na nějaký příkaz. Také můžete klepnout mimo nabídku a tím dáte programu najevo, že žádný příkaz realizovat nemá.

Obrázek 5.3
Editor `Emacs` mění nabídkovou lištu podle toho, s jakým typem souboru zrovna pracujete. Zde je ukázka lišty obsahující nabídky.



Obrázek 5.4

Na levé straně tohoto terminálového okna je ukázka posuvné lišty vytvořené v systému Athena. Na druhé straně je ukázka posuvné lišty (v okně programu Netscape) vytvořené systémem Motif.

5.6.3 Posuvné lišty

Posuvná lišta je nástroj umožňující zobrazit pouze část dokumentu, zatímco zbytek dokumentu je mimo okno. Například na obrázku 5.4 zobrazuje terminálové okno dolní třetinu textu. Díky posuvným lištám se také snadno určí, jaká část dokumentu je právě zobrazena. Tmavá část posuvné lišty ukazuje relativní pozici právě zobrazené stránky textu a zároveň její relativní velikost. Jestliže se celý text vleze na obrazovku, je celá posuvná lišta zobrazena tmavě. Jestliže je na obrazovce zobrazena polovina textu, pak bude zobrazena tmavě polovina posuvné lišty.

Vertikální posuvné lišty mohou být zobrazeny jak v pravém okraji okna, tak i v levém. Horizontální posuvné lišty bývají zpravidla zobrazeny u dolního okraje okna.

Posuvné lišty v systému Athena

Posuvné lišty v systému Athena fungují jinak, než u ostatních systémů. Každé tlačítko myši má jinou funkci. Má-li se text rolovat nahoru (tj. má-li se zobrazit text nad momentálně zobrazeným textem), pak stačí klepnout pravým tlačítkem myši, přičemž je kurzor myši umístěn kdekoliv v posuvné liště. Pokud chcete text rolovat dolů, pak klepněte levým tlačítkem myši.

Dále si můžete pomocí prostředního tlačítka myši zobrazit text v konkrétní pozici. Stačí klepnout prostředním tlačítkem myši na odpovídající pozici v posuvné liště.

Posuvné lišty v systému Motif

Posuvné lišty v systému Motif fungují téměř stejně jako v operačním systému Windows nebo Macintosh. Příklad posuvné lišty je na obrázku 5.4. Všimněte si, že na začátku a konci posuvné lišty jsou šipky. Ty jsou určeny pro „jemné“ rolování textu. Pokud na jedné z těchto šipek klepnete levým nebo prostředním tlačítkem myši, posune se text jen o několik málo řádků. Pravé tlačítko myši v tomto případě neplní žádnou funkci.

Pokud jde o používání myši uvnitř posuvné lišty, existují významné rozdíly mezi systémy Athena a Motif. Pravé tlačítko myši nemá žádný význam. Klepnutí levým tlačítkem myši nad momentální pozicí tmavé části posuvné lišty způsobí posun textu nahoru. Podobně platí, že klepnutí levým tlačítkem myši pod momentální pozicí tmavé části lišty způsobí posun textu dolů. Dále lze zobrazovaným textem přímo manipulovat tak, že se klepne levým tlačítkem myši na tmavou část posuvné lišty, drží se tlačítko stisknuté a nastaví se pozice tmavé části na požadované místo. Jakmile se tlačítko uvolní, požadovaná pozice se zobrazí.

Prostřední tlačítko má podobnou funkci jako u systému Athena. Zobrazí se ta část textu, která svou pozicí relativně odpovídá pozici kurzoru myši v posuvné liště.

Kapitola

6

Práce s operačním systémem Unix

Operační systém Unix je velmi výkonný nástroj pro ty, kdo jej umí používat. V této kapitole si popíšeme pokročilejší techniky práce s příkazovým procesorem `bash`.

6.1 Pseudoznaky

V předcházející kapitole jsme se naučili pracovat s příkazy pro manipulaci se soubory. Příležitostně se může stát, že budete chtít pracovat s více soubory najednou. Například budete chtít zkopírovat všechny soubory začínající posloupností „data“ do adresáře `~/backup`. Můžete to udělat tak, že použijete několikrát příkaz `cp`, nebo v jednom příkazu `cp` uvedete seznam všech souborů, které se mají kopírovat. Oba způsoby jsou těžkopádné a jistě vám seberou hodně času. Navíc máte velkou šanci udělat chybu.

Na následujícím příkladě si uvedeme daleko elegantnější postup:

```
/home/larry/report# ls -F
1993-1 1994-1 data1 data5
1993-2 data-new data2
/home/larry/report# mkdir ~/backup
/home/larry/report# cp data* ~/backup
/home/larry/report# ls -F ~/backup
data-new data1 data2 data5
/home/larry/report#
```

Z uvedeného příkladu vidíte, že po zadání hvězdičky zkopíroval příkaz `cp` všechny soubory začínající řetězcem data do adresáře `~/backup`. Zkuste uhodnout, jak bude fungovat příkaz `cp d*w ~/backup`

6.1.1 Co se ve skutečnosti stalo?

Rozhodně dobrá otázka. Příkazový interpret `bash` je schopen interpretovat jisté znaky, kterým se říká pseudoznaky (wildcards). Znak hvězdička (`*`) je interpretován tak, že za něj dosadí jakoukoliv posloupnost znaků. Proto příkaz z našeho příkladu `cp data* ~/backup` byl interpretován jako příkaz `cp data-new data1 data2 data5 ~/backup`.

Abychom ještě lépe ilustrovali interpretaci znaku hvězdička, uvedeme si nový jednoduchý příkaz `echo`, který zkopíruje své parametry jako text na obrazovce.

```
/home/larry# echo Hello!
hello!
/home/larry# echo How are you?
How are you?
/home/larry# cd report
/home/larry/report# ls -F
1993-1 1994-1 data1 data5
1993-2 data-new data2
/home/larry/report# echo 199*
1993-1 1993-2 1994-1
/home/larry/report# echo *4*
1994-1
/home/larry/report# echo *2*
1993-2 data2
/home/larry/report#
```

Jak sami vidíte, příkazový procesor rozšířil hvězdičku, vybral všechny soubory vyhovující specifikaci s hvězdičkou a předal je programu, který byl zadán ke spuštění. Naskýtá se přirozená otázka. Co se stane, když specifikaci s hvězdičkou nevyhovuje žádný soubor? Vyzkoušejte si například příkaz `echo /rc/fr*og` a uvidíte - příkazový procesor předá specifikaci přesně tak, jak byla zadána (žádnou interpretaci hvězdičky neprovede).

Ostatní příkazové procesory, například tcsh, se chovají jinak - nepředávají přesnou specifikaci, ale vypíší zprávu `No match`. Uveďme si příklad s příkazovým procesorem `tcsh`:

```
mousehouse>echo /rc/fr*og
echo: No match.
mousehouse>
```

Také vás asi napadá otázka, zda je možné pomocí příkazu `echo` zobrazit například `data*` (a nikoliv seznam souborů vyhovujících specifikaci `data*`). Řešení je jednoduché, stačí požadovaný text uzavřít do uvozovek. Například:

```
/home/larry/report# echo "data*"
data*
/home/larry/report#
```

```
mousehouse>echo "data*"
nebo    data*
mousehouse>
```

6.1.2 Znak otazník

Kromě znaku hvězdičky je příkazový procesor schopen interpretovat jako speciální znak také znak otazník. Uvede-li se ve specifikaci otazník, pak to znamená, že má být nahrazen právě jedním znakem. Například příkaz `ls /etc/??` zobrazí všechny soubory v adresáři `/etc`, jejichž jména se skládají právě ze dvou znaků.

6.2 Jak ušetřit čas pomocí příkazu `bash`

6.2.1 Editování příkazového řádku

Někdy se stane, že napíšete v příkazovém řádku příkazového interpretu `bash` dlouhý příkaz a pak si všimnete, že jste udělali chybu. Samozřejmě můžete postupně smazat všechny znaky, až se dostanete k chybně zadanému znaku, ale pak budete muset celý zbytek příkazu znovu psát. Příkazový interpret `bash` umožňuje používat kurzorové klávesy (šipka vlevo a šipka vpravo), pomocí kterých si můžete nastavit kurzor na chybně zadaný znak a opravit jej.

Dále existuje řada speciálních klávesových zkratek, pomocí nichž můžete příkazový řádek editovat. Tyto klávesové zkratky jsou většinou podobné příkazům používaným v editoru Emacs. Například kombinace Ctrl+T provede výměnu dvou sousedních znaků.[1] Popis většiny důležitých klávesových zkratek najdete v kapitole 8, která je věnována editoru Emacs.

6.2.2 Doplňování příkazů a jmen souborů

Další vynikající vlastnost příkazového procesoru bash spočívá v tzv. doplňování příkazových řádků. Podívejme se na následující příklad.

[1] Zkratka Ctrl+T znamená, že máte stisknout klávesu Ctrl, držet ji stisknutou, a pak stisknout klávesu T.

```
/home/larry# ls -F
this-is-a-long-file
/home/larry# cp this-is-a-long-file shorter
/home/larry# ls -F
shorter       this-is-a-long-file
/home/larry#
```

Samozřejmě je velmi nepohodlné a namáhavé vypisovat každý znak jména souboru `this-is-a-long-file` pokaždé, když chcete tento soubor zpřístupnit. Vytvořte si takový soubor, například příkazem `cp /etc/passwd this-is-a-long-file`. Nyní si ukážeme, jak zadat předcházející příkaz rychleji a s menším rizikem, že uděláme chybu.

Místo vypisování celého jména souboru zadejte `cp th` a pak stiskněte klávesu Tab. Jako mávnutím kouzelného proutku se vám celé jméno souboru objeví v příkazovém řádku a vám již stačí napsat `shorter`. Příkazový procesor `bash` neumí bohužel číst vaše myšlenky, proto jméno souboru `shorter` musíte napsat sami.

Když stisknete klávesu Tab, příkazový interpret `bash` se pokusí najít soubor, který začíná znaky, jež jste doposud napsali. Když například napíši `/usr/bin/ema` a pak stisknu Tab, příkazový procesor bash najde soubor `/usr/bin/emacs`. Když však napíši `/usr/bin/ld` a pak stisknu klávesu Tab, příkazový interpret pípne, aby mě upozornil, že našel více souborů. Skutečně, v adresáři `/usr/bin` jsou soubory `ld`, `ldd` a `ld86`.

Jestliže se pokoušíte doplnit jméno souboru a příkazový procesor pípne, pak můžete znovu stisknout klávesu Tab a objeví se vám seznam všech nalezených souborů. Proto nemusíte znát přesně jména svých souborů, příkazový procesor vám je vždy tímto způsobem připomene.

6.3 Standardní vstup a standardní výstup

Zkuste si provést příkaz, který vypíše seznam všech souborů v adresáři `/usr/bin`: `ls /usr/bin`. Adresář `/usr/bin` obsahuje velké množství souborů, proto po zobrazení jejich jmen zůstane na obrazovce jen několik a ostatní „utečou“ nahoru. Jak tento problém řešit?

6.3.1 Standardní vstup a výstup

Operační systém Unix nabízí programátorům velmi jednoduchý způsob zápisu na terminál. Pokud nějaký program něco vypisuje na vaši obrazovku, pak používá tzv. **standardní výstup** (standard output). Standardní výstup se zkracuje zkratkou stdout a slouží k předávání informací uživateli. Na druhé straně existuje **standardní vstup** (standard input), který slouží k pře-

dávání informací od uživatele. Samozřejmě je možné, aby program komunikoval s uživatelem bez použití standardního vstupu a výstupu, ale většina programů, kterými se zabýváme v této knize, standardní vstup a standardní výstup používá.

Například příkaz `ls` vypisuje seznam adresářů a souborů na standardní výstup, jenž je normálně spojen s terminálem. Příkazový interpret `bash` čte vámi zadané příkazy ze standardního vstupu.

Programy také mohou zapisovat do tzv. **standardního chybového výstupu** (standard error). Standardní chybový výstup je téměř výlučně spjat s terminálem, proto mohou být uživatelé informováni o případných chybách.

V následujících odstavcích si ukážeme, jak využívat standardní vstup a standardní výstup v konstrukcích, kterým se říká přesměrování vstupu, přesměrování výstupu a roura vstupu/výstupu.

6.3.2 Přesměrování výstupu

Velmi důležitou vlastností operačního systému Unix je schopnost přesměrovat výstup. Tato vlastnost vám umožní uložit výsledky příkazu do souboru nebo vytisknout na tiskárně. Jestliže chcete například přesměrovat výstup z příkazu `ls /usr/bin` do souboru, přidejte na konec příkazu znak `>` a za něj uveďte jméno souboru (nejlépe neexistujícího). Například:

```
/home/larry# ls
/home/larry# ls -F /usr/bin > listing
/home/larry# ls
listing
/home/larry#
```

Jak vidíte, seznam souborů se nezobrazil na terminálu, ale místo toho se vytvořil ve vašem domovském adresáři nový soubor `listing`. Zkusme se na tento soubor podívat pomocí příkazu `cat`. Jestli si vzpomínáte, příkaz `cat` se jevil jako zbytečný příkaz, který zkopíroval na obrazovku (standardní výstup) to, co jste napsali (standardní vstup). Příkaz `cat` také umí vypsat obsah souboru na standardní výstup, pokud tento soubor uvedete jako parametr:

```
/home/larry# cat listing
 ...
/home/larry#
```

Soubor `listing` nyní obsahuje přesný výstup příkazu `ls /usr/bin`, tedy seznam všech souborů v adresáři `/usr/bin`. To je v pořádku, ale stále to neřeší náš původní problém.[2]

Nyní je příkaz `cat` o něco zajímavější - lze jej použít spolu s přesměrováním výstupu. Co asi udělá příkaz `cat listing > newfile`? Přesměrování `> newfile` pro příkazový interpret znamená toto: „vezmi celý výstup z příkazu a ulož jej do nového souboru“. Výstup z příkazu `cat listing` je ovšem soubor `listing`. Jinými slovy, pravě jsme popsali jeden ze způsobů kopírování souborů, i když ne příliš efektivní.

Co asi udělá příkaz `cat > fox`? Příkaz `cat` čte data ze standardního vstupu (v tomto případě terminálu) a kopíruje je na standardní výstup, dokud nenarazí na znak konce souboru Ctrl+D. V tomto případě bude standardní vstup přesměrován do souboru `fox`. Nyní nám příkaz `cat` slouží jako základní editor:

```
/home/larry# cat > fox
The quick brown fox jumps over the lazy dog.
Stiskni Ctrl-d
```

Právě jste vytvořili soubor se jménem `fox` obsahující větu „`The quick brown fox jumps over the lazy dog.`“ Další důležitou funkcí příkazu `cat` je spojování souborů dohromady. Příkaz `cat` bude vypisovat každý soubor, který mu byl předán jako parametr. Proto například příkaz `cat listing fox` nejprve vypíše seznam souborů adresáře `/usr/bin` a nakonec vypíše naši hloupou větu. Příkaz `cat listing fox > listandfox` vytvoří nový soubor tvořený obsahem souborů `listing` a `fox`.

6.3.3 Přesměrování vstupu

Tak, jako je možné přesměrovat standardní výstup, je možné přesměrovat i standardní vstup. Místo toho, aby program četl z klávesnice, bude číst ze souboru. Protože se přesměrování vstupu vztahuje k přesměrování výstupu, je přirozené pro něj vyhradit znak `<`. Tento znak se také uvádí za příkazem, který chcete realizovat.

Přesměrování vstupu je užitečné zejména v případě, kdy máte soubor obsahující data a program, který data čte se standardního vstupu. Na druhé straně platí, že většina programů vyžaduje specifikaci souboru se vstupními daty, proto se přesměrování vstupu nepoužívá tak často, jako přesměrování výstupu.

[2] Pro netrpělivé čtenáře poznamenejme, že zde mohou použít program `more`. Než se však k němu dostaneme, musíme ještě několik věcí vysvětlit.

6.3.4 Roura

Řada příkazů v operačním systému Unix produkuje velké množství informací. Jak jsme si již ukázali, příkaz `cp /usr/bin` vyprodukuje příliš mnoho informací, než aby mohly být zobrazeny na terminálu. Abyste si mohli prohlédnout všechny informace z příkazů, jako je `ls /usr/bin`, budete muset použít další důležitý příkaz operačního systému Unix, který má název `more`.[3] Příkaz `more` způsobí pozastavení výpisu na obrazovku, jakmile se celá obrazovka zaplní. Například příkaz `more < /etc/rc` zobrazí stejným způsobem soubor `/etc/rc` jako příkaz `cat`, ale s tím rozdílem, že výpis pozastaví po každém naplnění obrazovky.

Příkaz `more` také můžete použít ve tvaru `more /etc/rc`, což představuje normální způsob jeho použití.

Co je však platné, že příkaz `more` umožňuje zobrazit více informací, než se vleze na obrazovku? Příkaz `more < ls /usr/bin` nebude fungovat, protože přesměrování vstupu funguje pouze se soubory a nikoliv s příkazy. Budete tedy muset postupovat takto:

```
/home/larry# ls /usr/bin > temp-ls
/home/larry# more temp-ls
...
/home/larry# rm temp-ls
```

Naštěstí nabízí operační systém Unix mnohem elegantnější způsob. Stačí, když zadáte příkaz `ls /usr/bin | more`. Znak | indikuje tzv. **rouru**. Roura v operačním systému Unix řídí tok dat.

Užitečnost roury ještě více vzrůstá ve spojení s dalšími nástroji, kterým se říká **filtry**. Filtr je program, který čte standardní vstup, nějakým způsobem jej modifikuje a odešle jej na standardní výstup. Příkladem filtru je právě příkaz `more` - čte data ze standardního vstupu, zobrazuje je na standardní výstup (obrazovku) a umožňuje vám prohlédnout celý soubor. `more` však není úplně dokonalý filtr, protože jeho výstup není vhodný pro to, aby mohl být předán jako vstup jinému programu.

Mezi další filtry patří příkazy `cat`, `sort`, `head` a `tail`. Chcete-li například přečíst pouze prvních deset řádků výstupu z příkazu `ls`, můžete použít příkaz `ls /usr/bin | head`.

[3] Název `more` pochází z nápovědy tohoto programu, kterou byl původně řetězec --more--. V mnoha verzích operačního systému Linux se vyskytuje identický příkaz, který postupně programátoři zdokonalovali.

6.4 Současný běh úloh

6.4.1 Jak řídit úlohy

Řízení úloh (job control) představuje možnost zajistit, aby daný proces (proces není v podstatě nic jiného, než běžící program) běžel na pozadí nebo naopak, aby běžel na popředí. Jinými slovy, zřejmě potřebujete nástroje k tomu, abyste mohli dlouhodobě běžící program přesunout do pozadí a tak mohli pracovat na jiných věcech, a přitom měli možnost do programu běžícího na pozadí kdykoliv zasahovat. Tímto nástrojem je v operačním systému Unix příkazový procesor. Ukážeme si, jak jej využívat k řízení úloh.

V souvislosti s řízením úloh jsou v operačním systému Unix vyhrazena dvě důležitá klíčová slova. První z nich je `fg` pro běh programů v popředí a druhé je `bg` pro běh programů na pozadí. Abychom vyzkoušeli, jak fungují, použijeme příkaz `yes`:

```
/home/larry# yes
```

Po spuštění příkazu `yes` se na levé straně obrazovky vypisuje velkou rychlostí písmeno „y“.[4] Pokud byste chtěli příkaz `yes` zastavit, stiskli byste klávesu Ctrl+C. Místo toho však stiskněte klávesu Ctrl+Z. Nyní se vám bude zdát, že se provádění příkazu `yes` zastavilo. Na obrazovce se však objeví následující zpráva:

```
[1]+ Stopped        yes
```

To znamená, že proces `yes` byl pozastaven. Pozastavený proces můžete obnovit pomocí příkazu `fg`, který jej aktivuje v popředí a program poběží dále. Zatímco je program pozastaven, můžete realizovat další příkazy. Než zadáte příkaz `fg`, zkuste si zadat několikrát jiný příkaz, například `ls`.

Jakmile se příkaz `yes` „vrátí“ do popředí, bude pokračovat ve výpisu písmen „`y`“ stejnou rychlostí jako na počátku po jeho spuštění. Nemusíte mít obavy, že se výstup z programu, který byl pozastaven, někam ztratí. Je-li program pozastaven uvedeným způsobem, pak až do okamžiku, kdy jej obnovíte, žádný výstup neprodukuje. Nyní stiskněte klávesu Ctrl+C a program `yes` zrušte.

Nyní se vraťme k předcházející zprávě:

```
[1]+ Stopped        yes
```

[4] Možná se vám zdá příkaz `yes` divný. Řada příkazů však potřebuje potvrdit nějakou volbu, proto v operačním systému Unix takový příkaz existuje.

Číslo v hranatých závorkách je **pořadové číslo úlohy**, na které se můžete odvolávat, kdykoliv bude chtít na běhu této úlohy něco změnit. Číslo úlohy se zavádí z toho důvodu, aby existovalo nějaké rozlišení mezi více současně běžícími úlohami. Znak „+“ za hranatými závorkami indikuje, že uvedená úloha je aktuální úlohou, tedy úlohou, jež byla jako poslední převedena z popředí na pozadí. Jestliže zadáte příkaz `fg`, pak do popředí převedete právě tu úlohu, která je označena znakem „+“. Slovo `Stopped` znamená, že úloha byla pozastavena a nachází se ve zvláštním stavu - není zrušena, ale také není aktivní. Operační systém Linux ji uvedl do speciálního stavu, kdy může pokračovat v realizaci, jakmile bude zadán příslušný příkaz. Jako poslední je uvedeno jméno procesu, tedy v našem případě `yes`.

Než pokročíme dále, zrušme tuto úlohu a nastartujme ji, tentokrát jiným způsobem. Příkaz ke zrušení úlohy se jmenuje `kill` (doslova zabít, zničit) a používá se následujícím způsobem:

```
/home/larry# kill %1
[1]+ Stopped yes
/home/larry#
```

Zpráva, která se objevila na obrazovce (říká, že proces byl pozastaven) je zavádějící. Abychom zjistili, v jakém stavu se proces nachází (t.j. zda je aktivní, zmrazen nebo pozastaven), zadáme příkaz `jobs`:

```
/home/larry# jobs
[1]+ Terminated yes
/home/larry#
```

Až zde máte napsáno, v jakém stavu se úloha `yes` skutečně nachází - byla ukončena a nikoliv pozastavena. Je také možné, že zadáte příkaz `jobs` a vůbec žádná zpráva se nevypíše. To znamená, že na pozadí opravdu neběží žádná úloha. Jestliže jste právě zrušili nějakou úlohu a příkaz `jobs` nic nevypíše, pak jste ji zrušili úspěšně. Obvykle se ale objeví zpráva, že úloha byla ukončena (`Terminated`).

Nyní spusťme příkaz `yes` znovu, tentokrát následujícím způsobem:

```
/home/larry# yes > /dev/null
```

Pokud jste četli oddíl o přesměrování vstupu a výstupu, pak asi tušíte, že příkaz `yes` bude odesílat svůj výstup do souboru `/dev/null`. Soubor `/dev/null` má v operačním systému Unix speciální význam. Představuje „černou díru“, do které se ztratí jakékoliv množství informací, aniž by například hrozilo, že se zaplní váš pevný disk. Nemáte-li dostatek fantazie, pak si představte, že je ve vašem počítači vyvrtána díra, kterou proudí informace ven a ztrácejí se v hlubinách vesmíru.

Po zadání uvedeného příkazu se vám nevrátí nápověda příkazového řádku, ani se nebudou na obrazovce vypisovat písmena „y“. I když je výstup z příkazu `yes` přesměrován do souboru `/dev/null`, úloha stále běží na popředí. Jako obvykle ji můžete pozastavit pomocí klávesy Ctrl+Z. Pak se vám samozřejmě znovu objeví výzva příkazového řádku:

```
/home/larry# yes > /dev/null
[Příkaz "yes" běží, nyní stiskněme Ctrl+Z]
[1]+ Stopped        yes > /dev/null
/home/larry#
```

Je nějaká možnost přesunout úlohu do pozadí tak, aby dále běžela a aby se nám objevila výzva příkazového řádku? K tomuto účelu slouží příkaz `bg`:

```
/home/larry# bg
[1]+ yes > /dev/null &
/home/larry#
```

Nyní byste měli věřit následujícímu tvrzení: po zadání příkazu `bg` běží proces `yes > /dev/null` dále, a to na pozadí. Poznáte to podle toho, že když v příkazovém řádku zadáte nějaký příkaz, například `ls` nebo `stuff`, bude jeho realizace poněkud zpomalena. Jiný efekt úlohy běžící na pozadí nemají. Nadále můžete v příkazovém řádku zadávat jakékoliv příkazy a proces `yes > /dev/null` poběží na pozadí a přitom bude výstup odesílat do „černé díry“.

Existují dvě možnosti, jak proces běžící na pozadí zrušit (doslova „zabít”). Buď použijete příkaz `kill`, nebo přesunete proces do popředí a ukončíte jej pomocí klávesy Ctrl+C. Vyzkoušíme si druhý způsob, abychom lépe pochopili vztah mezi příkazy `fg` a `bg`:

```
/home/larry# fg
yes > /dev/null
[Nyní běží proces opět v popředí. Stiskněte klávesu Ctrl+C
a ukončete jej.]
/home/larry#
```

Jako další si vyzkoušíme spustit současně více procesů, například:

```
/home/larry# yes > /dev/null &
[1] 1024
/home/larry# yes | sort > /dev/null &
[2] 1026
/home/larry# yes | uniq > /dev/null
```

```
[nyní stiskněte klávesu Ctrl+Z]
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Určitě jste si všimli, že jsme na konec prvních dvou příkazů zapsali znak `&`. Pokud na konec příkazu zadáte znak `&`, příkazový procesor spustí úlohu hned od počátku na pozadí. Je to mnohem jednodušší, než spustit úlohu normálním způsobem, stisknout klávesu Ctrl+Z a pak zadávat příkaz `bg`. První dvě úlohy jsme tedy přímo spustili na pozadí. Třetí úloha je v tomto okamžiku pozastavena a tedy neaktivní. Jistě jste zpozorovali, že počítač nyní reaguje na další příkazy pomaleji, protože dvě běžící úlohy spotřebovávají jistý čas procesoru.

Zrušme nyní druhou úlohu, protože ta patrně výrazně snižuje výkon vašeho počítače. Mohli bychom použít příkaz `kill %2`, ale to by bylo příliš jednoduché. Místo toho zadejme následující příkazy:

```
/home/larry# fg %2
yes | sort > /dev/null
[Stiskněte klávesu Ctrl+C]
/home/larry#
```

Uvedený příklad demonstruje, že příkaz `fg` akceptuje parametry začínající znakem `%`. Ve skutečnosti jsme mohli použít následující posloupnost příkazů:

```
/home/larry# %2
yes | sort > /dev/null
[Stiskněte klávesu Ctrl+C]
/home/larry#
```

První příkaz bude opět fungovat, protože příkazový procesor interpretuje číslo úlohy jako požadavek, že má být úloha přenesena do popředí. Čísla úloh odlišuje od ostatních čísel právě pomocí znaku `%`. Nyní zadejte příkaz `jobs`, abyste věděli, které úlohy momentálně běží:

```
/home/larry# jobs
[1]- Running yes > /dev/null &
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Znak „-“ znamená, že úloha číslo 1 byla do pozadí přesunuta jako první v pořadí a bude jako druhá v pořadí přesunuta do popředí, pokud zadáte příkaz `fg` bez parametrů. Znak „+“ znamená, že úloha číslo 3 bude do popředí přesunuta jako první v pořadí, pokud zadáte příkaz `fg` bez parametrů. Uvedené implicitní pořadí můžete změnit takto:

```
/home/larry# fg %1
yes > /dev/null
[nyní stiskněte klávesu Ctrl+Z]
[1]+ Stopped yes > /dev/null
/home/larry#
```

Pomocí uvedených příkazů jste pozastavili úlohu číslo 1 a změnili jste také pořadí priorit všech úloh. K jakým změnám došlo? Na to vám odpoví příkaz `jobs`:

```
/home/larry# jobs
[1]+ Stopped yes > /dev/null
[3]- Stopped yes | uniq > /dev/null
/home/larry#
```

Nyní jsou obě úlohy pozastaveny (v obou případech jsme použili klávesu Ctrl+Z). Úloha číslo 1 je nyní na pozici první, pokud jde o pořadí, v jakém budou úlohy implicitně přenášeny do popředí. Je tomu tak z toho důvodu, že jsme úlohu nejprve přesunuli do popředí a pak ji pozastavili. Znak „+“ vždy označuje tu úlohu, která byla jako poslední pozastavena, když předtím běžela na popředí. Úlohu číslo 1 můžeme opět aktivovat tímto způsobem:

```
/home/larry# bg
[1]+ yes > /dev/null &
/home/larry# jobs
[1]- Running yes > /dev/null &
[3]+ Stopped yes | uniq > /dev/null
/home/larry#
```

Všimněte si, že úloha 1 nyní běží a u druhé úlohy se objevil znak „+“. Nakonec obě úlohy zrušíme, protože se už nemůžeme dále dívat na to, jak snižují výkon počítače:

```
/home/larry# kill %1 %2
[3] Terminated yes | uniq > /dev/null
/home/larry# jobs
[1]+ Terminated yes > /dev/null
/home/larry#
```

Jistě jste si všimli zpráv upozorňujících na ukončení úloh - jak se zdá, nic neumírá tiše. Tabulka 6.1 obsahuje stručný přehled o příkazech, které se vztahují k řízení úloh.

6.4.2 Teorie řízení úloh

Především je nutné pochopit, že úlohy řídí příkazový procesor. Ve skutečnosti v systému neexistují programy jako fg, bg, jobs, & nebo kill. Tyto programy jsou vnitřními příkazy příkazového procesoru a jsou jeho součástí. (Výjimku někdy tvoří příkaz `kill`, který je v některých operačních systémech Unix implementován jako externí program. Pokud však jde o Linux, příkaz `kill` je integrován v příkazovém procesoru bash). Tento mechanismus řízení úloh je logický. Každý uživatel chce mít k realizaci úloh svůj prostor. Protože každý uživatel pracuje se svým příkazovým procesorem, je logické, aby příkazový procesor úlohy řídil. Odtud dále vyplývá, že čísla úloh se vztahují právě k jednomu uživateli. Moje úloha číslo 1 bude zřejmě zcela odlišná od vaší úlohy číslo 1, a to i v případě, kdy budeme oba přihlášeni k témuž systému. Ve skutečnosti platí, že přihlásíte-li se do systému více než jednou, bude každý spuštěný příkazový procesor vlastnit unikátní datové struktury pro řízení úloh. Jako jediný uživatel tedy můžete mít několik zcela různých úloh s pořadovým číslem 1 běžících v různých příkazových interpretech.

`fg %job`	Tento příkaz je příkazem příkazového procesoru a přesouvá úlohu do popředí. Chcete-li zjistit, která úloha se přesune implicitně jako první, zadejte příkaz `jobs` a podívejte se, která úloha je označena znakem „+“. Parametry: volitelným parametrem je číslo úlohy. Implicitní je úloha označená znakem „+“.
`&`	Pokud se znak „&“ uvede na konec příkazového řádku, bude úloha spuštěna automaticky přímo na pozadí. Pro takto spuštěný proces platí všechny metody řízení úloh, které jsme zde uvedli.
`bg %job`	Tento příkaz je příkazem příkazového procesoru a přesouvá úlohu do pozadí. Chcete-li zjistit, která úloha se přesune implicitně jako první, zadejte příkaz `jobs` a podívejte se, která úloha je označena znakem „+“. Parametry: volitelným parametrem je číslo úlohy. Implicitní je úloha označená znakem „+“.
`kill %job PID`	Tento příkaz je příkazem příkazového procesoru a ukončuje úlohu, ať běží na pozadí nebo byla pozastavena. Pokaždé byste měli použít jako parametr číslo úlohy, kterému předchází znak `%`. Parametry: buď číslo úlohy, kterému předchází znak `%`, nebo PID (identifikační číslo procesu), pak znak `%` není nutné uvádět. Na jednom příkazovém řádku může být uvedeno více úloh, které se mají ukončit.
`jobs`	Tento příkaz je příkazem příkazového procesoru. Vypisuje informace o běžících úlohách a o úlohách, které byly pozastaveny.

(pokračování)

Ctrl+C	Klávesová zkratka vyhrazená k bezpodmínečnému ukončení úlohy. Běžně se používá k ukončení úlohy běžící v popředí. Je však nutné upozornit na skutečnost, že ne všechny programy na tuto klávesovou zkratku reagují.
Ctrl+Z	Klávesová zkratka vyhrazená k pozastavení úlohy. Opět platí, že některé programy jej ignorují. Jakmile je jednou úloha pozastavena, může být znovu přesunuta do popředí nebo ukončena.

Tabulka 6.1
Přehled příkazů a kláves používaných k řízení úloh

Jediný bezpečný způsob, jak identifikovat proces, představuje identifikační číslo procesu (process identification number, zkratka PID). Každá úloha běžící v systému má svoje unikátní identifikační číslo procesu. Dva různí uživatelé mohou k odkazu na proces použít totéž identifikační číslo a pak mají jistotu, že se jedná o tentýž proces (samozřejmě za předpokladu, že jsou přihlášeni k témuž systému).

Podívejme se na další příkaz, který vám pomůže lépe pochopit význam identifikačních čísel procesů. Příkaz `ps` vypíše seznam všech běžících nebo pozastavených procesů, včetně vašeho příkazového procesoru. Tento příkaz má také několik voleb, z nichž nejdůležitější jsou `a`, `u` a `x`. Zadáte-li volbu `a`, pak se vám zobrazí seznam všech procesů, t.j. procesů, které spustili i ostatní uživatelé. Pomocí volby `x` se zobrazí seznam těch procesů, které nejsou nijak svázány s terminálem.[5] Poslední volba `u` zajistí, že se vám zobrazí další užitečné informace o běžících nebo pozastavených procesech.

Jestliže chcete získat komplexní představu, co se ve vašem systému odehrává, zadejte příkaz `ps -aux`. (V novějších verzích je možno znaménko „–“ vynechat a použít pouze příkaz `ps aux`.) Pak se můžete podívat, kolik paměti každý proces potřebuje (sloupec `%MEM`) a jak zatěžuje procesor (sloupec `%CPU`). Ve sloupci `TIME` je uveden celkový čas procesoru, který spuštěný proces spotřeboval.

Ještě jedna poznámka o identifikačním číslu procesu. Kromě parametru `%číslooúlohy#` můžete v příkazu `kill` použít identifikační číslo procesu. Spusťte příkaz `yes > /dev/null` na pozadí, pak spusťte příkaz `ps` a podívejte se na identifikační číslo náležící procesu `yes`. Toto identifikační číslo můžete také použít ke zrušení procesu `yes`.[6]

5 To má smysl jen v případě jistých systémových programů, které nekomunikují s uživatelem prostřednictvím klávesnice.

6 Obecně je mnohem jednodušší zrušit úlohu pomocí jejího čísla.

Jestliže začnete programovat v jazyku C, pak se brzy dozvíte, že příkazy pro řízení úloh jsou interaktivní verze systémových volání `fork` a `execl`. Jedná se o příliš složité mechanismy, než abychom je zde mohli popsat. Pokud budete vytvářet programy, které jsou schopny spouštět více procesů, pak se s těmito systémovými funkcemi budete muset seznámit v příslušné dokumentaci.

6.5 Virtuální konzoly

Operační systém Linux podporuje tzv. virtuální konzoly. Jedná se o metodu, která budí dojem, že váš počítač není jedním počítačem ale několika počítači najednou. Linux je schopen spustit několik terminálů současně a přitom jsou všechny spojeny s jedním jádrem. Naštěstí je používání virtuálních konzol jednou z nejjednodušších záležitostí v operačním systému Linux. Chcete-li si je vyzkoušet, stiskněte klávesu Alt a pak klávesu F2.[7]

Najednou se vám objeví nová výzva k přihlášení se do systému. Nepodléhejte panice. Nyní pracujete s virtuální konzolou číslo 2. Přihlaste se a proveďte několik příkazů, abyste se přesvědčili, že nově nastartovaný příkazový procesor skutečně funguje. Budete-li se chtít vrátit do virtuální konzoly číslo 1, stiskněte opět klávesu Alt a pak F1. Nebo můžete vytvořit třetí konzolu pomocí kláves Alt a F3.

Operační systém Linux má implicitně povoleno šest virtuálních konzol. Pokud budete chtít vědět jak, prostudujte si manuál „*Příručka správce operačního systému Linux*“. Budete muset udělat nějaké úpravy v souborech v adresáři `/etc`. Šest virtuálních konzol však většině uživatelů stačí.

Začnete-li používat virtuální konzoly, brzy zjistíte, že jsou ideálním nástrojem k realizaci mnoha úkonů současně. Na první konzole můžete například provozovat editor Emacs, na druhé programy pro komunikaci s Internetem a na třetí můžete mít spuštěn příkazový procesor, kdybyste chtěli spustit ještě nějaký další program.

[7] Ujistěte se, že pracujete s textovou konzolou. V případě systému X Window by uvedená kombinace kláves nemusela fungovat.

Kapitola

7

Malé a výkonné programy

7.1 V čem spočívá síla operačního systému Unix

Síla operačního systému Unix spočívá v používání malých a jednoduchých příkazů, které se sami o sobě zdají neužitečné. Pokud se je naučíte spojovat dohromady, vytvoříte systém, jenž je mnohem pružnější a výkonnější než všechny ostatní operační systémy. V této kapitole se budeme zabývat příkazy `sort`, `grep`, `more`, `cat`, `wc`, `spell`, `diff` a `tail`. Z názvů těchto programů nelze bohužel intuitivně předpokládat, jakou funkci plní.

Nejdříve se budeme věnovat každému příkazu odděleně a nakonec uvedeme několik příkladů, jak je spojovat a používat dohromady.[1]

7.2 Práce se soubory

Kromě příkazů `cd`, `mv` a `rm`, o kterých jsme se zmínili v kapitole 4, existují další příkazy určené k manipulaci se soubory (a nikoliv s daty, jež soubory obsahují). Patří mezi ně `touch`, `chmod`, `du` a `df`. Žádný z těchto příkazů se nestará o obsah souborů - pouze mění některé jejich atributy, se kterými operační systém Unix pracuje.

Uveďme si seznam atributů, se kterými uvedené příkazy manipulují:

[1] Poznamenejme, že výklad uvedených příkazů nebude zcela vyčerpávající. Podrobnosti naleznete v manuálových stránkách.

- Časové údaje. S každým souborem jsou spjaty tři časové údaje.[2] První časový údaj obsahuje informaci o době vytvoření souboru, druhý o době jeho poslední modifikace a třetí o době, kdy byl naposledy zpřístupněn.
- Vlastník souboru. Každý soubor v operačním systému Unix je vlastněn některým uživatelem.
- Skupina. Každý soubor je také spojen se skupinou uživatelů. Nejčastěji se tato skupina nazývá `users`, což znamená, že soubor je sdílen každým uživatelem přihlášeným do systému.
- Přístupová práva. Každý soubor má přístupová práva (pro která se někdy používá označení privilegia). Jedná se o mechanismus, jenž určuje, kdo může k danému souboru přistupovat, kdo může spouštět dané programy a podobně. Každé z přístupových práv může být odděleně přiřazeno vlastníku souboru, skupině nebo všem uživatelům operačního systému.

`touch` *soubor1 soubor2 ... souborN*

Příkaz `touch` provede aktualizaci časových údajů každého souboru uvedeného jako parametr - nastaví čas vytvoření na aktuální čas. Pokud soubor uvedený jako parametr neexistuje, program touch jej vytvoří. Je také možné specifikovat čas, který má být souborům přiřazen. Podrobnosti si nastudujte v manuálových stránkách.

`chmod [-Rfv]` *mód soubor1 soubor2 ... souborN*

Příkaz umožňující změnit přístupová práva se nazývá `chmod` (change mode). Než se pustíme do jeho popisu, musíme si probrat, jaká přístupová práva operační systém Unix bere v úvahu. Každý soubor je spojen se skupinou přístupových práv. Podle těchto přístupových práv operační systém Unix určuje, zda může být soubor čten, zda do něj může být zapisováno, nebo, jedná-li se o spustitelný program, může být spuštěn. V následujících odstavcích budeme hovořit o přístupových právech uživatelů. Každý program, jenž uživatel spustí, má shodná přístupová práva. Pokud přesně nevíte, co program dělá, mohou nastat problémy s bezpečností systému.

Operační systém Unix rozlišuje tři různé typy uživatelů. Prvním z nich je vlastník souboru. Tím je osoba, která má právo v souvislosti s tímto souborem používat příkaz `chmod`. Druhý typ představuje skupina. Většina souborů ve vašem systému by měla mít přiřazen atribut „`users`“, a tak by měla být přístupna každému normálnímu uživateli. Chcete-li znát hodnotu atributu skupina, zadejte příkaz `ls -l` *file*.

[2] Starší souborové systémy v operačním systému Linux uchovávaly pouze jeden časový údaj, protože byly odvozeny od operačního systému Minix. Pokud máte takový souborový systém, pak pro vás budou některé informace neaktuální.

Nakonec operační systém identifikuje ty osoby, které nejsou ani vlastníky žádného souboru, ani členové žádné skupiny. Pro ty je vyhrazen atribut „other“ (ostatní).

Typické nastavení přístupových práv je: pro vlastníka právo číst soubor a zapisovat do souboru, pro skupinu právo číst soubor a pro ostatní jsou všechna práva potlačena. Může se však stát, že skupina má právo soubor číst i do něj zapisovat a přitom vlastník nemá žádná práva.

Nyní si vyzkoušejme, jak pomocí příkazu `chmod` změnit některá přístupová práva. Nejdříve si vytvořte nový soubor, řekněme pomocí příkazu `cat` nebo pomocí editoru Emacs. Implicitně je vám přiděleno právo soubor číst i právo do něj zapisovat. Práva ostatních uživatelů jsou určena podle toho, jak je nastaven váš systém a jak je nastaven proces přihlašování. Ujistěte se, že nově vytvořený soubor jste schopni číst pomocí příkazu `cat`. Nyní si odeberte právo číst tento soubor pomocí příkazu `chmod u-r` *filename*. Parametr `u-r` se interpretuje jako „user minus read“. Když se nyní pokusíte soubor přečíst, obdržíte chybové hlášení `Permission denied!`. Chcete-li získat zpět právo soubor číst, zadejte příkaz `chmod u+r` *filename*.

Přístupová práva k adresářům používají stejné atributy (pro čtení, zapisování a spouštění), ale poněkud odlišným způsobem je interpretují. Právo číst znamená, že uživatel, skupina nebo ostatní mohou vypisovat seznam souborů v adresáři. Právo zapisovat znamená, že uživatel, skupina nebo ostatní mohou přidávat soubory do adresáře nebo rušit soubory. Právo spouštět znamená, že uživatel může zpřístupňovat soubory v adresáři i v jeho podadresářích. Pokud nemá uživatel žádná práva, nemůže ani použít příkaz `cd`.

Při používání příkazu `chmod` se specifikuje, kdo má k danému souboru přístupová práva práva (uživatel, skupina, ostatní nebo všichni). Dále se specifikují atributy, které definují, co se s daným souborem může dělat. Znaménko plus indikuje udělení daného práva, znaménko mínus popření. Pomocí znaménka rovná se (`=`) se specifikují přesná přístupová práva. Přípustná přístupová práva jsou read (čtení), write (zápis) a execute (spouštění).

Pokud se použije v příkazu chmod volba `R`, pak se změna přístupových práv týká všech souborů v adresáři a všech podadresářů (`R` je označením pro rekurzivní). Jestliže se použije volba `f`, pak se příkaz `chmod` pokusí změnit přístupová práva práva i v případě, že uživatel není vlastníkem daného souboru. Zadá-li se volba `v`, pak bude příkaz `chmod` podrobně vypisovat informace o všech krocích, které realizuje.

7.3 Systémová statistika

Příkazy uvedené v tomto oddílu jsou určeny k výpisu informací o systému nebo o jeho částech.

du [-abs] [*cesta1 cesta2 ... cestaN*]

Příkaz du je zkratkou pro „disk usage“ (využívání disku). Zobrazuje informaci o velikosti diskového prostoru, který je přidělen danému adresáři a všem jeho podadresářům. Samotný příkaz du zobrazuje seznam, jehož položky uvádějí u každého adresáře velikost diskového prostoru (který adresář obsazuje), kolik prostoru obsazuje aktuální adresář a jako poslední udává velikost diskového prostoru spotřebovanou aktuálním adresářem a všemi jeho podadresáři. Pokud příkazu du předáte nějaké parametry (jméno souboru nebo adresáře), pak vám vypíše uvedené informace o tomto souboru či adresáři.

Použije-li se volba a, pak se vypíší uvedené informace jak o souborech, tak i o adresářích. Po zadání volby b se informace o velikosti diskového prostoru nebudou uvádět v kilobajtech, ale přímo v bajtech. Jeden bajt je ekvivalentní jednomu znaku v textovém souboru.

Pokud se použije volba s, pak příkaz du vypíše zmíněné informace o adresářích uvedených jako parametry a nikoliv o jejich podadresářích.

df

Příkaz df je zkratkou pro „disk filling“. Sumarizuje množství použitého diskového prostoru. Pro každý souborový systém (připomeňme, že souborový systém zaujímá buď samostatný disk, nebo samostatný diskový oddíl) vypíše informaci o celkovém množství diskového prostoru, informaci o velikosti použité části, o volné kapacitě a nakonec o celkové kapacitě souborového systému.

Paradoxně se může stát, že se vám zobrazí kapacita větší než 100 %. Je to způsobeno tím, že operační systém Unix rezervuje pro každý souborový systém jistý prostor pro superuživatele. Proto je zajištěno, že i když uživatel zaplní celý disk, zůstává na disku něco málo volného místa, aby se mohly realizovat některé operace.

Pro většinu uživatelů nemá příkaz df žádné užitečné volby.

uptime

Příkaz uptime dělá přesně to, co byste podle jeho názvu očekávali. Vypisuje celkový čas, který uplynul od posledního zavedení operačního systému.

Dále příkaz `uptime` uvádí informaci o aktuálním čase a o tzv. „load average“. Termínem „load average“ se míní průměrný počet úloh čekajících na spuštění v průběhu daného časového intervalu. Příkaz `uptime` uvádí tyto hodnoty pro poslední minutu, posledních pět minut a posledních deset minut. Jestliže se hodnota „load average“ blíží k nule, pak to znamená, že je systém téměř nevytížen. Hodnota blízká k jedničce znamená, že je systém plně vytížen, ale není zahlcen. Vysoké hodnoty jsou výsledkem skutečnosti, že v systému běží několik programů současně.

Příkaz `uptime` patří k několika málo programům operačního systému Unix, které nemají parametry a žádné volby. (V nových verzích programu `uptime`, který je součástí balíku `procps`, je akceptována volba V, která je určena pro zobrazení verze balíku `procps`).

`who`

Příkaz `who` slouží k zobrazení seznamu momentálně přihlášených uživatelů systému. Pokud se k příkazu přidají parametry `am i`, pak se zobrazí informace o aktuálním uživateli.

`w [-f] [`*uživatelské jméno*`]`

Příkaz `w` zobrazuje informace o momentálních uživatelích operačního systému a informace o tom, co dělají. V podstatě tento příkaz kombinuje příkazy `uptime` a `who`. Záhlaví výstupu z příkazu `w` je přesně stejné jako v případě příkazu `uptime` a každý řádek obsahuje informace o uživateli, například kdy se přihlásil. Kolonka `JCPU` ukazuje celkový čas procesoru, který uživatel spotřeboval, zatímco kolonka `PCPU` ukazuje celkový čas procesoru spotřebovaný jejich momentálně běžící úlohou.

Jestliže u příkazu `w` uvedete volbu `f`, pak se nezobrazí vzdálený systém, ze kterého je daný uživatel přihlášen (ve výpisu bude chybět kolonka `FROM`).

7.4 Co obsahují soubory

V operačním systému Unix existují dva hlavní příkazy pro výpis obsahu souborů - `cat` a `more`. Již jsme o nich hovořili v kapitole 6.

`cat [-nA] [`*soubor1 soubor2 ... souborN*`]`

Příkaz `cat` nepatří mezi uživatelsky přátelské příkazy. Nečeká, až si přečtete obsah souboru a spíše se používá v souvislosti s rourami. Má však několik užitečných voleb. Například volba `n` zajistí, že všechny řádky vypisovaného souboru budou číslovány. Při zadání volby `A` se budou řídící znaky zobrazovat jako normální znaky a ne jako podivné sekvence paznaků.

Opět připomínáme, že kompletní seznam voleb akceptovatelných příkazem `cat` naleznete v manuálových stránkách. Pokud se v příkazovém řádku neuvede ani jeden parametr, použije příkaz `cat` standardní vstup.

`more [-l] [+`*linenumber*`] [`*file1 file2 ... fileN*`]`

Příkaz `more` je mnohem užitečnější a budete jej často používat zejména k zobrazování souborů ve formátu ASCII. Jedinou zajímavou volbou je `l`, po jejímž uvedení příkazu `more` sdělíte, že nechcete interpretovat znak Ctrl-L jako znak „nová stránka“. `more` zahájí zobrazení od specifikovaného čísla řádku (*linenumber*).

Protože je more interaktivním příkazem, uvádíme v následujícím seznam příkazů, kterými lze jeho činnost řídit.

Mezerník	Zobrazí se následující stránka.
`d`	Text se posune o 11 řádků, což je přibližně polovina stránky.
`/`	Vyhledání regulárního výrazu. Zatímco regulární výraz může být opravdu komplikovaný, můžete zadat textový řetězec, který se má vyhledat. Zadáte-li například `/toad` Enter, vyhledá se v celém textu řetězec „`toad`“. Jestliže uvedete jen / Enter, pak se vyhledá další výskyt posledně zadaného řetězce.
`n`	Také tento příkaz je určen k vyhledání posledně zadaného regulárního výrazu.
`: n`	Pokud jste zadali prostřednictvím parametrů více než jeden soubor, zahájí se prohlížení následujícího souboru.
`: p`	Prohlížení se nastaví na předcházející soubor.
`q`	Program `more` se ukončí.

`head [-`*lines*`] [`*file1 file2 ... fileN*`]`

Příkaz `head` zobrazí prvních deset řádků každého z uvedených souborů nebo prvních deset řádků ze standardního vstupu, pokud žádný soubor není jako parametr v příkazovém řádku uveden. Jakákoliv numerická hodnota uvedená jako volba změní implicitní nastavení deset. Například `head -15 frog` zobrazí prvních patnáct řádků souboru `frog`.

`tail [-lines] [file1 file2 ... fileN]`

Podobně jako příkaz `head` zobrazuje příkaz `tail` jen část souboru. Normálně zobrazuje posledních deset řádků souboru nebo posledních deset řádků ze standardního vstupu. Také akceptuje volbu pro nastavení počtu zobrazených řádků.

`file [file1 file2 ... fileN]`

Příkaz `file` se pokouší identifikovat formát souborů uvedených v seznamu v příkazovém řádku. Protože ne všechny soubory mají příponu charakterizující formát souboru nebo neobsahují posloupnosti znaků, podle kterých by se dal formát identifikovat, pokouší se příkaz `file` provádět některé základní testy a tak odhadnout, co soubor obsahuje.

Buďte opatrní, často se může stát, že je formát souboru identifikován chybně.

7.5 Informační příkazy

Následující oddíl je věnovaný příkazům, které mění soubor, provádějí jisté operace se souborem nebo zobrazují některé statistické informace o souboru.

`grep [-nvwx] [-number] expression [file1 file2 ... fileN]`

Jeden z nejužitečnějších příkazů operačního systému Unix je příkaz `grep` (generalized regular expression parser). Jeho jméno se zdá být příliš nadsazené na to, že jde o program, jenž umí pouze vyhledávat regulární výrazy v textu. Následující příklad demonstruje nejjednodušší způsob používání příkazu grep:

```
/home/larry# cat animals
Animals are very interesting creatures. One of my favorite animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# grep iger animals
the tiger, a fearsome beast with large teeth.
/home/larry#
```

Z uvedeného příkladu vyplývá jedna nevýhoda. I když příkaz vypsal všechny řádky obsahující hledané slovo, neuvedl, kde se dané slovo v souboru vyskytuje (neuvedl číslo řádku).

Někdy to může stačit, v závislosti na tom, co potřebujete udělat. Když například hledáte chyby ve výstupu z programu, stačí zadat příkaz `a.out | grep error`, kde `a.out` je jméno vašeho programu.

Jestliže však chcete přesně vědět, kde se hledaný regulární výraz v souboru nachází, zadejte volbu `n`. Pak bude příkaz `grep` zobrazovat i pořadová čísla řádků. Volbu `v` použijte tehdy, když budete chtít zobrazit seznam všech řádků neobsahujících daný regulární výraz.

K dalším vlastnostem příkazu `grep` patří to, že implicitně vyhledává neúplná slova. V předcházejícím příkladu jsme zadali neúplné slovo `iger` a `grep` našel slovo `tiger`. Jestliže má program `grep` pracovat pouze s celými slovy, zadejte volbu `w`. Po zadání volby `x` bude `grep` pracovat s celými řádky.

Pokud neuvedete v příkazu `grep` žádné parametry, bude prohledávat standardní vstup.

`wc [-clw] [`*soubor1 soubor2 ... souborN*`]`

`wc` je zkratka pro „word count“ (počet slov). Tento příkaz spočítá slova, znaky a řádky v souborech uvedených v seznamu. Pokud se v příkazovém řádku neuvedou jako parametry žádné soubory, pracuje příkaz `wc` se standardním vstupem.

Pro příkaz `wc` existují tři volby: `c` (character - znak), `l` (line - řádek) a `w` (word - slovo). Pomocí volby se specifikuje, co má program `wc` počítat. Pokud například zadáte příkaz `wc -cw`, pak spočítá znaky a slova, ale nikoliv řádky. Příkaz zadaný bez voleb spočítá vše - znaky, slova i řádky.

Jedna z aplikací příkazu `wc` spočívá v tom, že lze s jeho pomocí určit počet souborů v adresáři: `ls | wc -w`. Pokud chcete například vědět, kolik souborů končí s příponou `.c`, pak zadejte příkaz `ls *.c | wc -w`.

`spell [`*soubor1 soubor2 ... souborN*`]`

`spell` je velmi jednoduchý program operačního systému Unix pro kontrolu pravopisu textových souborů. Zpravidla kontroluje pravopis americké angličtiny.[3] Program `spell` je tedy filtr, který prochází soubor ve formátu ASCII a na výstupu vypisuje slova, která považuje za pravopisně nesprávná. `spell` pracuje se soubory uvedenými v příkazovém řádku jako parametry, jinak zpracovává standardní vstup.

[3] I když existuje několik verzí tohoto programu pro evropské jazyky, distribuce operačního systému Linux většinou obsahuje kontrolu pravopisu americké angličtiny.

Pravděpodobně máte k dispozici i dokonalejší program pro kontrolu pravopisu `ispell`. Program `ispell` navíc nabízí možnosti opravy chybně napsaného slova. Pokud se v příkazovém řádku specifikuje soubor, pak se program `ispell` ovládá pomocí nabídky, jinak pracuje jako klasický filtr.

Chcete-li se dozvědět o programu ispell více, podívejte se do manuálových stránek.

`cmp` *soubor1* [*soubor2*]

Příkaz `cmp` porovnává obsahy dvou souborů. První musí být uveden jako parametr v příkazovém řádku, druhý je buď specifikován jako parametr, nebo se čte ze standardního vstupu. Příkaz `cmp` je velmi jednoduchý a jeho úkolem je ukázat, kde se dva soubory liší.

`diff` *soubor1 soubor2*

Jedním z nejkomplikovanějších standardních programů operačního systému Unix je příkaz `diff`. GNU verze programu `diff` má více než dvacet voleb! Jedná se o zdokonalenou verzi programu `cmp`, která ukazuje nejen kde se soubory liší, ale také v čem se liší.

Nebudeme probírat kompletní možnosti programu `diff`, protože to překračuje rámec této knihy. Místo toho se zaměříme na základní operace. Krátce řečeno, program `diff` akceptuje jako parametry dva soubory a zobrazí rozdíly mezi nimi na principu „řádek po řádku“. Uveďme si příklad:

```
/home/larry# cat frog
Animals are very interesting creatures. One of my favorite
  animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# cp frog toad
/home/larry# diff frog toad
/home/larry# cat dog
Animals are very nteresting creatures. One of my favorite
  animals is
the tiger, a fearsome beast with large teeth.
I also like the lion---it's really neat!
/home/larry# diff frog dog
1c1,2
< Animals are very interesting creatures. One of my favorite
  animals is
---
```

```
> Animals are very nteresting creatures. One of my favorite
  animals is
>
3c4
< I also like the lion---it's really neat!
---
> I also like the lion---it's really neat!
/home/larry#
```

Jak vidíte v našem příkladu, program `diff` neprodukuje žádný výstup, pokud jsou soubory identické. Pokud se porovnávaly dva různé soubory, pak řádek `1c1,2` říká, že byl porovnán první řádek levého souboru (`frog`) s prvním a druhým řádkem pravého souboru (`dog`) a že zde byly nalezeny rozdíly. Pak byly porovnány řádky 3 (v souboru `frog`) a 4 (v souboru `dog`) a i zde byl nalezen rozdíl. Může se zdát podivné, že se nejdříve porovnávají řádky s různým pořadovým číslem. Je to však z toho důvodu, aby program pracoval efektivněji.

`gzip [-v#] [`*soubor1 soubor2 ... souborN*`]`
`gunzip [-v] [`*soubor1 souborX (x∈{1,2,...,N})*`]`
`zcat [`*soubor1 soubor2 ... souborN*`]`

Tyto tři programy se používají ke kompresi a dekompresi dat.

Příkaz `gzip` (GNU zip) je program, který čte původní soubory a produkuje soubory menší. Přitom soubory specifikované v příkazovém řádku ruší a nahrazuje je komprimovanými soubory. Komprimované soubory mají stejná jména, ale navíc mají příponu „`gz`“.

`tr` *řetězec1 řetězec2*

Příkaz `tr` (translate characters) pracuje pouze se standardním vstupem - neakceptuje žádné soubory jako parametry. Jeho dvěma parametry jsou dva řetězce. Příkaz nahradí všechny výskyty znaků řetězce *řetězec1* na vstupu odpovídajícím znakem z řetězce *řetězec2*. Kromě relativně jednoduchého tvaru tohoto příkazu, například `tr frog toad`, může příkaz `tr` akceptovat poměrně složité příkazy. Uvedeme si příklad, kdy se pomocí příkazu `tr` převádějí malá písmena na velká:

```
/home/larry# tr [:lower:] [:upper:]
this is WEIRD sentence.
THIS IS WEIRD SENTENCE.
/home/larry#
```

Program `tr` je dosti složitý a často se využívá v malých programech pro příkazové procesory.

Kapitola

8

Editování souborů pomocí editoru Emacs

8.1 Co je to Emacs?

Abyste mohli dělat něco rozumného s počítačem, potřebujete mít možnost ukládat texty do souborů a měnit texty, které jsou v souborech uloženy. Takové úkony vám umožní realizovat textový editor. Emacs je jeden z nejpopulárnějších editorů na světě - zejména proto, že i začátečníkům umožňuje bez problémů pracovat s textovými soubory. Klasický editor dodávaný s operačním systémem Unix, `vi`, probereme v dodatku A.

Než se začnete učit pracovat s editorem Emacs, musíte si najít nějaký soubor obsahující obyčejný text a zkopírovat si jej do domovského adresáře.[1] Na začátek by bylo riskantní editovat originální soubor, mohl by obsahovat důležité informace. Editor Emacs se spouští jednoduše:

```
/home/larry# emacs README
```

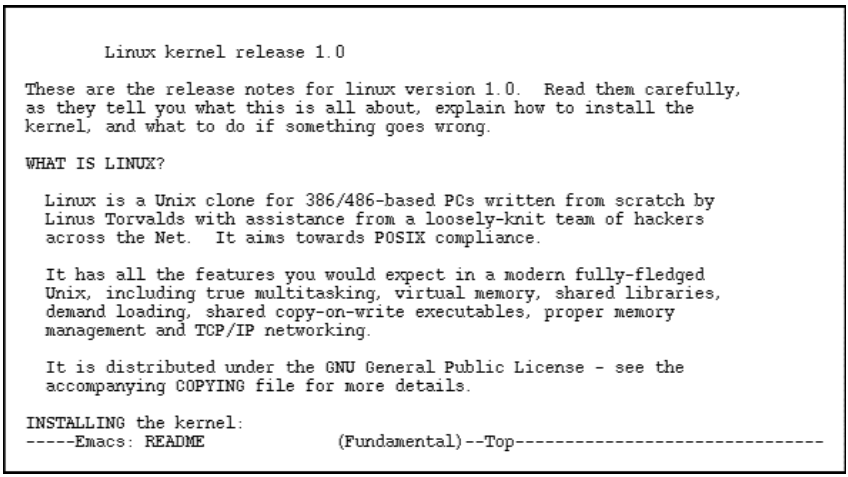
Pokud jste se rozhodli zkopírovat soubor `/etc/rc`, `/etc/inittab` nebo jiný, pak samozřejmě jako parametr uvedete jméno tohoto souboru - například `emacs rc`.

Spuštění editoru Emacs může mít různý efekt. Záleží to na tom, v jakém prostředí jej spouštíte. V textovém terminálu zobrazujícím pouze textové znaky vytvoří nastartovaný editor Emacs okno přes celou obrazovku. Pokud jej spustíte v systému X Window, vytvoří si editor své vlastní okno. Budeme předpokládat, že jste spustili editor Emacs v textovém terminálu. Vše, co zde budeme uvádět, bude platit i pro editor Emacs spuštěný v okně systému X Window. V systému X Window nezapomínejte přesunout kurzor myši na okno obsahující editor, jinak nebudete moci nic psát.

---

[1] Například `cp /usr/src/linux/README ./README`

Vaše obrazovka (nebo okno v systému X Window) by měla vypadat přibližně tak, jak je zobrazena na obrázku 8.1. Většina obrazovky obsahuje text vašeho souboru. Obzvláště důležité jsou poslední dva řádky, zejména chcete-li se naučit pracovat s tímto editorem. Řádek obsahující dlouhé posloupnosti pomlček je řádek specifikující mód editoru (modeline).



Obrázek 8.1
Editor Emacs byl právě nastartován příkazem `emacs README`

Na uvedeném obrázku vidíte v předposledním řádku slovo „Top“. Může zde být také uvedeno slovo „All“ a mohou se zde také vyskytovat malé rozdíly, což záleží na verzi editoru. Někteří uživatelé mají v tomto řádku zobrazen aktuální čas. Pod uvedeným řádkem se nachází informační a **příkazový řádek**, někdy označovaný jako „minibuffer“, jindy jako „echo area“. Informační řádek slouží editoru ke komunikaci s uživatelem, k vypisování zpráv a někdy zde jeho prostřednictvím budete zadávat příkazy. Budou se zde objevovat takové pokyny, jako: „`For information about GNU project and its goals, type C-h C-p`.“ Zatím takové pokyny ignorujte, budeme se jimi zabývat později.

Než opravdu změníte text v nějakém souboru, musíte se naučit „pohybovat“ textem a pohybovat kurzorem. Kurzor by měl být umístěn na začátku souboru v levém horním rohu obrazovky. Chcete-li kurzor posunout o jeden znak dopředu, stiskněte kombinaci kláves C-F (stiskněte klávesu Ctrl, držte ji stisknutou a stiskněte klávesu F). Kurzor se posune o jeden znak dopředu. Pokud budete obě klávesy stále držet stisknuty, bude se pohyb kurzoru opakovat. Všimněte si, že když kurzor dospěje na konec řádku, automaticky se přesune na začátek následujícího řádku. Opačného směru pohybu kurzoru dosáhnete pomocí kláves C-B. Pokud chcete posunout kurzor o jeden řádek dolů, použijte klávesy C-N, pokud jej chcete posunout o jeden řádek nahoru, stiskněte klávesy C-P[2].

2 Zajisté jste si již všimli, že příkazy editoru Emacs jsou většinou realizovány pomocí kombinace dvou kláves.

Používání klávesy Ctrl zajišťuje nejrychlejší způsob, jak při editování textu pohybovat kurzorem. Editor Emacs používá takové kombinace kláves, které při psaní udržují vaše ruce nad klávesnicí. Jestliže jste však zvyklí používat kurzorové klávesy, budou také fungovat.

Pokud pracujete v systému X Window, můžete k nastavení pozice textového kurzoru použít kurzor myši. Stačí kurzor myši nastavit na požadovanou pozici v textu a klepnout levým tlačítkem. Tento způsob je však velmi pomalý (jednou rukou musíte opustit klávesnici, uchopit myš, nastavit kurzor a pak se na klávesnici zase vrátit) a nedoporučujeme si na něj zvykat. Většina uživatelů, kteří pracují s editorem Emacs dlouhodobě, používá k pohybu kurzoru výhradně klávesnici.

Pomocí kláves C-P a C-B nastavte kurzor opět na začátek souboru, tedy horní levý roh obrazovky. Nyní podržte klávesy C-B stisknuty poněkud déle. Uslyšíte pípnutí a v informačním řádku uvidíte zprávu „`Beginning of buffer`“.

V tomto okamžiku se asi podivíte. Co je to buffer?

Když editor Emacs zpracovává soubor, nepracuje ve skutečnosti se souborem přímo. Zkopíruje si jeho obsah do speciální pracovní oblasti, která se nazývá **buffer** - zde můžete obsah souboru modifikovat. Až ukončíte editaci souboru, dáte příkaz, aby editor buffer uložil. Do té doby zůstane obsah původního souboru nezměněn a změny se provádějí pouze uvnitř pracovní oblasti editoru.

Nyní jsme tedy připraveni obsah bufferu modifikovat. Vše, co jsme dělali doposud, bylo „nedestruktivní“, což znamená, že jsme obsah bufferu neměnili. Předpokládejme, že chcete do editovaného souboru zapsat znak „X“. Jakmile stisknete klávesu X, změní se předposlední řádek obrazovky. Editor Emacs registruje změny v editovaném textu a jakmile nějaké nastanou, informuje o tom uživatele pomocí dvou hvězdiček před slovem Emacs. Pak bude uvedený řádek vypadat asi takto:

```
--**-Emacs: some_file.txt      (Fundamental)--Top--------------------
```

Hvězdičky zůstanou zobrazeny do té doby, než obsah bufferu uložíte. Obsah bufferu můžete v průběhu editace ukládat průběžně vícekrát - stačí stisknout klávesy C-X C-S (stiskněte klávesu Ctrl, držte ji stisknutou a pak stiskněte klávesu „X“ a „S“). Průběžné ukládání obsahu bufferu má význam především při vytváření nebo modifikaci velkých souborů.

Nyní si uvedeme seznam několika málo příkazů používaných v editoru Emacs spolu s těmi, které jsme již popsali. Je na vás, abyste si jejich používání procvičili. Než pokročíme dále, měli byste s těmito příkazy být důkladně seznámeni.

C-F	Posunutí kurzoru o jeden znak doprava.
C-B	Posunutí kurzoru o jeden znak doleva.
C-N	Posunutí kurzoru o jeden řádek dolů.
C-P	Posunutí kurzoru o jeden řádek nahoru.
C-A	Umístění kurzoru na začátek řádku.
C-E	Umístění kurzoru na konec řádku.
C-V	Zobrazení následující stránky.
C-L	Zobrazení stránky s aktuálním řádkem uprostřed.
C-D	Zrušení znaku, na kterém je umístěn kurzor.
C-K	Zrušení textu od pozice kurzoru do konce řádku.
C-X C-S	Uložení obsahu bufferu do odpovídajícího souboru.
Backspace	Zrušení znaku vlevo od kurzoru.

8.2 Používání editoru pod systémem X Window

Pokud chcete rychle editovat soubory pod systémem X Window, máte poněkud ulehčenou situaci. Editovaný text je zobrazen pod nabídkou funkcí editoru, proto je práce s editorem intuitivnější.

V textovém módu uvedená nabídka není dostupná.

Když poprvé spustíte editor Emacs pod systémem X Window, obsahuje nabídka čtyři hlavní položky: Buffers, File, Edit a Help. Potřebujete-li si některou z položek hlavní nabídky zpřístupnit, nastavte kurzor myši na tuto položku, stiskněte levé tlačítko myši (držte je stále stisknuté). Pak přesuňte kurzor na požadovanou funkci a tlačítko myši uvolněte. Jestliže žádnou funkci vybrat nechcete, přesuňte kurzor myši mimo nabídku a opět uvolněte tlačítko myši.

Nabídka Buffers obsahuje seznam různých souborů, které v této relaci s editorem Emacs editujete. Nabídka File obsahuje příkazy pro zavádění a ukládání souborů. Funkce této nabídky podrobněji popíšeme později. Nabídka Edit obsahuje některé nejdůležitější příkazy pro editování textového souboru a prostřednictvím nabídky Help si můžete interaktivně zobrazovat vybrané části dokumentace k editoru.

Jistě jste si všimli, že u každé funkce v nabídce je uvedena ekvivalentní kombinace kláves pro realizaci této funkce. Na jedné straně můžete funkce vyvolávat pomocí myši a menu, na druhé straně můžete k tomuto účelu používat ekvivalentní kombinace kláves, což je samozřejmě rychlejší.

8.3 Editování více souborů současně

V daném okamžiku je editor Emacs schopen pracovat s více soubory. Ve skutečnosti platí, že počet souborů, které mohou být současně uloženy v bufferech editoru, je omezen pouze velikostí paměti počítače. Po zadání příkazu Ⓒ-Ⓧ Ⓒ-Ⓕ se do bufferu editoru Emacs zavádí nový soubor. Když tento příkaz zadáte, v příkazovém řádku se objeví výzva k zadání jména souboru:

```
Find file: ~/
```

Syntaxe používaná k zadání jména souboru je stejná, jako v příkazovém řádku příkazového procesoru; lomítka se používají k oddělení adresářů a podadresářů, znak ~ je interpretován jako váš domovský adresář. I zde funguje možnost automatického **doplňování jména souboru** - zadáte-li dostatek znaků k tomu, aby mohlo být jméno souboru jednoznačně identifikováno, pak stačí stisknout klávesu Tab a zbytek jména souboru se doplní automaticky (nebo se zobrazí seznam jmen všech souborů, která zadané specifikaci vyhovují). Podobnou funkci jako klávesa Tab má klávesa mezerník. Necháme na vás, abyste zjistili, v čem se funkce vyvolané těmito klávesami liší. Máte-li zadáno jméno souboru, který chcete editovat, stiskněte klávesu Enter; editor zavede obsah souboru do své pracovní oblasti a zobrazí jej na obrazovce. V terminologii editoru Emacs se tento proces nazývá vyhledání souboru. Najděte si nějaký další soubor a popsaným způsobem jej zaveďte do editoru. Nyní máte v editoru nový buffer. Budeme předpokládat, že původní má název `some_file.txt` a nový má název `another_file.txt`. Zdá se, že se váš první buffer ztratil. Pravděpodobně se divíte, kam se poděl.

Neobávejte se, stále zůstává uvnitř editoru a k jeho opětovnému zobrazení stačí stisknout klávesu Ⓒ-Ⓧ Ⓑ. Pak se vás editor v příkazovém řádku zeptá, do kterého bufferu se má přepnout. Nabídne implicitní buffer, t.j. buffer, jehož obsah se zobrazí po stisknutí klávesy Enter (nemusíte jméno bufferu uvádět). Implicitní buffer je ten, který jste „opustili“ jako poslední.

Jestliže tedy pracujete se dvěma soubory a často mezi nimi přepínáte, používejte klávesu C-X B a nemusíte při každém přepnutí zadávat jméno souboru. Samozřejmě můžete jméno uvádět, ale bude vás to zdržovat.

I při přepínání mezi buffery lze použít funkci k doplňování jména - stejně jako v případě vyhledávání souboru. Po stisknutí klávesy Tab doplní editor automaticky jméno bufferu, je-li schopen jej z doposud zadaných znaků jednoznačně identifikovat. Kdykoliv jste v příkazovém řádku vyzváni k zadání jména, vyzkoušejte si, zda je editor schopen jméno automaticky doplnit. Automatické doplňování vám ušetří spoustu času a editor je schopen je realizovat pokaždé, když má vybrat nějakou položku z nějakého předdefinovaného seznamu.

Vše, co jste se naučili o příkazech pro editování v prvním bufferu platí i v druhém. Pokročme dále - v novém bufferu změňte nějaký text, ale neukládejte jej (pomocí kláves C-X C-S). Nyní předpokládejme, že nechcete provedené změny uložit a že chcete buffer zrušit. K tomu slouží příkaz C-X K. Nejdříve se vás editor zeptá, který buffer chcete zrušit. Stisknete-li klávesu Enter, zruší se implicitní buffer (což je asi nejčastější případ). Editor se vás znovu zeptá, zda chcete buffer opravdu zrušit, a když zadáte „yes“ a stisknete Enter, pak jej konečně zruší.

Nyní byste se měli důkladně procvičit v zavádění souborů, jejich modifikaci, ukládání, přepínání mezi buffery a rušení bufferů. Ujistěte se, že needitujete nějaké důležité soubory, které by mohly poškodit funkci vašeho systému.[3] Vytvořte si alespoň pět bufferů a vyzkoušejte si přepínání mezi nimi.

8.4 Ukončení práce s editorem

Když ukončíte práci s editorem Emacs, ujistěte se, že všechny buffery, které mají být uloženy jsou skutečně uloženy. Pak můžete editor ukončit pomocí kláves C-X C-C. Někdy se vás editor po zadání kláves C-X C-C v příkazovém řádku na něco zeptá. Nebuďte znepokojeni a odpovězte normálním způsobem. Jestliže se domníváte, že se budete do editoru chtít později vrátit, nepoužívejte klávesy C-X C-C, ale klávesu C-Z. Pak bude aktivita editoru pouze pozastavena. K opětovné aktivaci editoru můžete použít známý příkaz `fg`. Pozastavení editoru je mnohem efektivnější, než jej stále dokola ukončovat a znovu startovat, zejména když opakovaně editujete tytéž soubory.

3 Pokud nejste přihlášeni jako uživatel root, neměli byste být schopni vašemu systému ublížit, ale stejně buďte opatrní.

V systému X Window má funkce C-Z jiný efekt - provede transformaci okna s editorem do ikony. Podrobně jsme tento mechanismus popsali v kapitole 5. Existují tedy dvě možnosti, jak transformovat okno obsahující editor do ikony - buď normálním způsobem pomocí správce oken, nebo pomocí klávesy C-Z. V systému X Window však k opětovné aktivaci editoru nelze použít příkaz fg - musíte použít správce oken.

8.5 Klíče Meta

Doposud jsme při práci s editorem Emacs používali kombinace kláves s klávesou Ctrl. Existují však další možné kombinace, a to s klávesou Meta. Těmto kombinacím se v terminologii editoru Emacs říká meta-klíče. Bohužel ne všechny klávesnice mají klávesu Meta umístěnu ve stejné poloze a některé ji nemají vůbec. V případě klávesnic u počítačů IBM PC je klávesa Meta totožná s klávesou Alt.

Klávesu Meta si můžete otestovat. Stiskněte tu klávesu, o které si myslíte, že by mohla být klávesou Meta, a zároveň stiskněte klávesu X. Pokud se v příkazovém řádku objeví malá nápověda (zpravidla M-X), pak jste kýženou klávesu nalezli. Stiskněte klávesu C-G a znovu se vrátíte do bufferu editoru Emacs.

Jestliže se vám v příkazovém řádku nic neobjeví, stále existuje řešení. Místo klávesy Meta můžete použít klávesu Escape. Avšak v tomto případě ji nedržte stisknutou - stiskněte ji, uvolněte a zadejte další klávesu reprezentující danou funkci. Vyzkoušejte si naši oblíbenou kombinaci, tedy Escape a pak „X“. Nyní se vám v příkazovém řádku nápověda určitě objeví. Opět stiskněte klávesu C-G. Klávesa C-G je v editoru Emacs určena ke zrušení čehokoliv, co nehodláte realizovat. Editor zpravidla vyšle zvukový signál, aby vás upozornil, že danou funkci rušíte.[4]

Označení M-X je analogické označení C-X (kde x je klávesa reprezentující nějakou funkci). Pokud jste nalezli skutečný klíč Meta, pak jej používejte. Jinak budete muset používat popsanou sekvenci s klávesou Escape. Kdekoliv v následujícím textu uvidíte M-X, pak to znamená, že máte použít meta-klíč.

8.6 Práce s bloky textu

Editor Emacs, tak jako prakticky každý editor, umožňuje pracovat s bloky textu. Abyste tyto funkce mohli využívat, musíte mít prostředek pro definování **začátku bloku** a **konce bloku**. V editoru Emacs se definice bloku realizuje pomocí nastavení dvou pozic v bufferu, které se

[4] Příležitostně se může stát, že jedno stisknutí klávesy C-G nepřesvědčí editor Emacs, že chcete opravdu přerušit činnost, kterou právě děláte. Stačí však trvat na svém a pak se editor vrátí do předcházejícího módu.

označují jako mark (značka) a point. Chcete-li v bufferu nastavit začátek bloku, nastavte kurzor na požadovanou pozici a stiskněte klávesu C-SPC (SPC je zkratka pro mezerník). V příkazovém řádku se objeví zpráva „Mark set“.[5] Nyní je začátek bloku nastaven. Editor nepoužívá žádné zvýrazňovací prostředky k tomu, aby byl začátek bloku viditelný.

A co konec bloku? Konec bloku je definován aktuální pozicí kurzoru. Proto se pro něj používá v terminologii editoru Emacs termín point. Point zrovna tak například znamená místo, od kterého se má vkládat text při kopírování nebo vkládání bloků. Nastavením začátku bloku a přesunutím kurzoru na kteroukoliv jinou pozici v textu je definován začátek a konec bloku. Tento blok se nazývá **region**. Region je tedy vždy část textu mezi značkou a aktuální pozicí kurzoru.

Pouhá definice regionu nezpřístupňuje blok ke kopírování. Nejdříve musíte blok „**zkopírovat**“ (copy), abyste jej pak mohli „**nalepit**“ (paste) někam jinam. Má-li se blok zkopírovat, stiskněte klávesu M-W. Nyní je blok uložen ve speciálním bufferu editoru Emacs. Jestliže nyní chcete blok nalepit někam jinam, nastavte si kurzor na požadovanou pozici a stiskněte C-Y.

Pokud chcete blok textu přesunout (a ne zkopírovat), pak místo příkazu M-W použijte příkaz C-W. Tímto způsobem se blok po přenesení do speciálního bufferu vymaže. Když zjistíte, že jste blok nechtěli vymazat, stiskněte klávesu C-Y a blok se obnoví. Místo, do kterého editor Emacs ukládá části textu, se nazývá **kill-ring**. U jiných editorů se tato oblast nazývá „clipboard“ nebo „paste buffer“.

Existuje ještě jeden způsob, jak vystřihovat a nalepovat text: kdykoliv stisknete klávesu C-K, dojde ke zrušení textu od pozice kurzoru do konce řádku a přitom se zrušený text také ukládá do oblasti kill-ring. Pokud takto zrušíte více než jeden řádek, uloží se všechny řádky do oblasti kill-ring a vy máte možnost je později nalepit najednou.

Někdy je tento způsob přesouvání a kopírování textu rychlejší, než používat systém s označováním začátku a konce bloku. Záleží jen na vás, kterému způsobu dáte přednost.

8.7 Vyhledávání a náhrada řetězců

V editoru Emacs máte několik možností, jak vyhledávat text. Některé způsoby jsou komplikované a nemá smysl je zde popisovat. Nejjednodušší a nejčastěji používaná metoda je tzv. postupné vyhledávání označované jako „isearch“ (incremental search). Předpokládejme, že potřebujete vyhledat řetězec „gadfly“ v následujícím textu:

5 Na některých počítačích příkaz C-SPC nebude fungovat. Místo toho použijte C-@.

```
I was growing afraid that we would run out of gasoline, when my
passenger exclaimed
''Gadzooks! There's gadfly in here!''.
```

Nastavte pozici kurzoru na začátek textu nebo na místo, o kterém víte, že předchází hledanému řetězci, a zadejte příkaz C-S. Tím nastavíte editor Emacs do módu vyhledávání. Nyní začněte zadávat řetězec, který chcete vyhledat. Jakmile však napíšete první znak, tedy „g“ „přeskočí“ kurzor na první výskyt písmene „g“ v textu. Jestliže máte v editoru text uvedený v našem příkladu, pak kurzor skočí na začátek slova „growing“. Nyní napište písmeno „a“ (druhé písmeno ve slově „gadfly”) a editor přesune kurzor na slovo „gasoline“, protože vyhledal první výskyt dvojice znaků „ga“. Když zadáte další písmeno „d“, skočí kurzor na slovo „gadzooks“ a nakonec po zadání písmene „f“ skočí na hledané slovo „gadfly“. Přitom jste nemuseli zadat kompletní hledaný řetězec.

Při postupném vyhledávání funguje editor Emacs tak, že po každém zadání dalšího znaku hledaného řetězce vyhledá první výskyt toho slova, jehož začátek je shodný s doposud zapsanými znaky. Jakmile zadáte tolik znaků, aby vyhledávání bylo jednoznačné, můžete funkci ukončit stisknutím klávesy Enter. Jestliže se domníváte, že hledaný řetězec je nad aktuální pozicí kurzoru, pak zadejte příkaz C-R, čímž inicializujete zpětné vyhledávání.

Pokud naleznete první výskyt zadaného řetězce, ale zajímá vás jeho další výskyt, pak znovu zadejte příkaz C-S. Editor Emacs vyhledá následující výskyt zadaného řetězce a tak můžete pokračovat dále. Když editor výskyt hledaného řetězce nenalezne, vypíše zprávu, že řetězec nenašel a začne znovu vyhledávat od začátku bufferu. Podobná pravidla platí pro příkaz C-R.

Nyní si popsané funkce vyzkoušejte. Najděte si soubor s anglickým textem a vyhledejte v něm výskyt slova „the“. Pak pomocí opakované funkce C-S najděte další výskyty. Všimněte si, že editor přitom vyhledá i jiné řetězce, například „them“, protože vyhovují zadané specifikaci. Jestliže chcete vyhledat pouze řetězec „the“, pak na konec hledaného řetězce budete muset přidat mezeru. Při editování hledaného řetězce můžete používat standardní editační klávesy Backspace a Delete. Chcete-li vyhledávání ukončit, vždy použijte klávesu Enter.

Editor Emacs také umožňuje nahradit výskyt jednoho řetězce jiným. Tento proces se v terminologii editoru Emacs označuje jako **query-replace**. Proces zahájíte tak, že stisknete M-X a napíšete `query-replace` a stisknete Enter.

I v případě zadávání příkazů funguje v editoru Emacs doplňování, a proto stačí, když například napíšete „query-re“ a stisknete klávesu Tab. Předpokládejme, že chcete řetězec „gadfly“ nahradit řetězcem „housefly“. V příkazovém řádku „`Query replace:` “ zadejte „gadfly“ a stiskněte Enter. Pak budete opět vyzváni k zadání řetězce, kterým se má původní řetězec nahradit - zadejte „housefly“. Editor Emacs bude nyní procházet text, zastavovat kurzor na každém výskytu řetězce „gadfly“ a bude se vás ptát, zda má nalezený řetězec nahradit. Pokud

ano, stiskněte klávesu Y, pokud ne, stiskněte klávesu N. Jestliže je pro vás předcházející výklad příliš komplikovaný, popsané funkce si vyzkoušejte. Bude to mít pro vás větší význam, než kdybyste předcházející odstavce četli desetkrát.

8.8 Vnitřní funkce editoru Emacs

Všechny funkce editoru Emacs vyvolané stisknutím nějaké kombinace kláves mají nějaký název, kterému editor „rozumí“. Například klávesa C-P znamená pro editor Emacs provedení vnitřní funkce `previous-line`. Všechny vnitřní funkce mohou být volány prostřednictvím svého jména, a to po stisknutí klávesu Alt+X. Když například zapomenete, jaký klíč máte použít k nastavení kurzoru na předcházející řádek, stačí zadat: M-X `previous-line` a Enter. Vyzkoušejte si to a přesvědčte se, že C-P a M-X `previous-line` realizují tutéž funkci.

Autor editoru Emacs postupoval tak, že nejdříve definoval celou množinu vnitřních funkcí editoru a pak teprve navrhl klávesové zkratky pro realizaci nejčastěji používaných funkcí. Někdy je jednodušší použít explicitní volání funkce prostřednictvím M-X, než si pamatovat klávesovou zkratku svázanou s touto funkcí. Například funkce `query-replace` je u některých verzí editoru Emacs svázána s klávesou M-%. Kdo si ale má pamatovat takovou divnou kombinaci? Pokud budete náhradu řetězců provádět extrémně často, pak má samozřejmě smysl si klávesovou zkratku pamatovat.

Většina kláves, které stisknete, jsou písmena, číslice, případně další znaky, které se mají vkládat do textového bufferu. Každá z těchto kláves je **svázána** s funkcí, jež má jméno `self-insert-command`. Tato funkce nedělá nic jiného, než že dané písmeno nebo číslici vloží do bufferu. Kombinace kláves, například s klávesou Ctrl, jsou obecně svázány s jinými funkcemi - pohyb kurzoru a podobně. Například kombinace C-V je svázána s funkcí `scroll-up`, což znamená, že se editovaný text posune o jednu obrazovku dolů.

Jak ale postupovat, když do textu chcete vložit řídící znak? Konec konců, řídící znaky jsou také znaky ASCII, i když zřídkakdy používané. Může se stát, že budete chtít do textu takový znak vložit. Proto v editoru Emacs existuje prostředek, který zabrání editoru interpretovat kombinaci klávesy s klávesou Ctrl jako příkaz. Klávesa C-Q[6] je svázána se speciální funkcí, která má název `quoted-insert`. Jediné, co funkce `quoted-insert` dělá, je, že přečte následující klávesu a vloží ji do textového bufferu bez interpretace. Pokud chcete do textu vložit samotný znak C-Q, pak zadejte C-Q dvakrát.

6 Pro kombinaci kláves C-Q se používá označení „klávesa“, protože představuje jediný znak z tabulky ASCII.

V editoru Emacs existují některé funkce, jež nejsou svázány s žádnou kombinací kláves. Když například píšete dlouhý text, pak asi nebudete chtít ukončovat každý řádek klávesou Enter. Po editoru Emacs můžete chtít, aby to dělal za vás (po editoru Emacs můžete chtít cokoliv). Příkaz, který takovou funkci realizuje, má název auto-fill-mode. Není však implicitně svázán s žádnou kombinací kláves. Jestliže chcete tento příkaz inicializovat, zadejte `M-X auto-fill-mode`. Jak jsme si již řekli, „M-X“ je klíč umožňující vyvolat vnitřní funkci editoru jménem. Takto byste mohli vyvolat i funkci `next-line` (následující řádek) nebo `previous-line` (předcházející řádek), ale to by bylo velmi neefektivní, protože uvedené funkce jsou svázány s klávesami C-N a C-P.

Mimochodem, když se po vyvolání funkce `auto-fill-mode` podíváte na předposlední řádek, uvidíte zde na pravé straně slovo Fill. Dokud se zde toto slovo vyskytuje, bude editor Emacs ukončovat řádky za vás. Když funkci „`M-X auto-fill-mode`“ vyvoláte znovu, automatické ukončování řádků přestane fungovat - funkce je vytvořena jako přepínač.

Může se vám zdát, že zadávání funkcí prostřednictvím jejich dlouhých jmen není příliš pohodlné. Naštěstí i v tomto případě v editoru Emacs funguje doplňování jmen (stejně jako doplňování jmen souborů). Proto platí, že zřídkakdy budete muset vypisovat celé jméno funkce, písmeno po písmenu. Pokud si nejste jisti, zda bude editor schopen doplnit jméno, stiskněte klávesu Tab. V horším případě se vám objeví znak Tab, v lepším případě editor provede doplnění.

8.9 Nápověda v editoru Emacs

Editor Emacs má velmi rozsáhlou nápovědu. Tak rozsáhlou, že se o ní zmíníme jen velmi stručně. Nejvyšší úroveň nápovědy se vyvolá stisknutím klíče C-H a nějakého písmene. Například C-H K vyvolá nápovědu vztahující se ke klíčům (budete vyzváni k zadání klíče a pak se objeví text s vysvětlením, jakou funkci klíč realizuje). C-H T vyvolá výukový program editoru Emacs. K důležitým klíčům patří C-H C-H C-H, kdy se vám objeví „nápověda o nápovědě“. Zde se dozvíte vše o systému nápovědy. Když znáte jméno funkce editoru Emacs (například `save-buffer`), ale nemůžete si vzpomenout na klíč k jejímu vyvolání, použijte C-H W („`where is`“) a zadejte jméno funkce. Zrovna tak máte možnost zjistit podrobné informace o dané funkci - zadejte C-H F a pak jméno funkce.

Mějte na paměti, že editor Emacs je sice schopen doplňovat jména funkcí, avšak nemůžete na tuto vlastnost příliš spoléhat, pokud k nějaké funkci (jejíž název přesně neznáte) potřebujete nápovědu. Jestliže si myslíte, že můžete odhadnout slovo, kterým funkce začíná, zkuste je napsat a stiskněte Tab. Uvidíte, zda editor byl schopen funkci identifikovat. Pokud ne, zkuste něco jiného. Totéž platí pro jména souborů. I když si nemůžete vzpomenout, jak se jmenu-

je soubor, který jste editovali před mnoha měsíci, můžete název odhadnou a zkusit, co na to editor odpoví. Doplňování jmen je tedy nejen užitečné v tom, že šetří čas, ale pomáhá vám nalézt to, co jste již zapomněli, nebo to, co si přesně nepamatujete.

Existuje několik dalších znaků, které můžete zadat po příkazu C-H a tak získat nápovědu různým způsobem. Nejčastěji však budete používat C-H K, C-H W a C-H F. Až budete blíže seznámeni s editorem Emacs, vyzkoušejte si například C-H A. Pak se vás editor zeptá na řetězec a mezi všemi jmény funkcí nalezne to, které daný řetězec obsahuje. (Písmeno „a“ je zkratkou pro „apropos“ nebo „about“.)

Jiný zdroj informací o editoru Emacs nabízí systém pro čtení dokumentace v hypertextovém formátu **Info**. Ten můžete inicializovat přímo z editoru Emacs tak, že stisknete klíč C-H I. Pak se vám objeví základní stránka systému Info, ve které najdete další pokyny, jak postupovat při vyhledávání informací.

8.10 Pracovní módy editoru Emacs

Každý buffer editoru Emacs je spjat s tzv. módem.[7] Módy byly zavedeny z toho důvodu, že například při psaní zprávy pro elektronickou poštu má uživatel jiné požadavky, než při psaní programu v jazyku C. Kdyby měl editor splňovat všechny požadavky najednou, bylo by jeho používání velmi komplikované.

Proto se autor editoru Emacs[8] rozhodl řešit tuto situaci pomocí módů. Chování editoru se mění podle toho, s jakým módem je konkrétní buffer spjat. Jednotlivé módy se od sebe liší vazbou mezi klíči a funkcemi, ale jsou zde i jiné rozdíly.

Nejzákladnějším módem je mód `fundamental`, který nemá žádné speciální funkce. Uvedeme zde, co o základním módu říká samotný editor Emacs:

```
Fundamental mode:
Major mode not specialized for anything in particular.
Other major modes are defined by comparison with this one.
```

Právě uvedenou informaci jsem získal takto: zadal jsem příkaz C+X B (což znamená vyvolání funkce `switch-to-buffer`) a zadal jsem „foo“ jako jméno bufferu, do kterého se chci přepnout. Protože buffer s takovým jménem nebyl doposud vytvořen, editor jej vytvořil a přepnul se do něj. Implicitně v něm nastavil základní mód (`fundamental-mode`). Všech-

7 Aby nebyla situace tak jednoduchá, existují zde hlavní módy (Major Modes) a vedlejší módy (Minor Modes). Zatím však pro nás nejsou podstatné.

8 Autor editoru Emacs je Richard Stallman.

ny názvy módů mají tvar `<modename>-mode` a pomocí funkce „M-X“ lze pro každý buffer specifikovat mód explicitně právě pomocí jeho názvu. Abych získal více informací o základním módu, zadal jsem příkaz C-H M. Nakonec se zobrazila výše uvedená informace o základním módu.

Od základního módu je odvozen mírně užitečnější mód `text-mode` (textový mód), který má dva speciální příkazy: M-S pro funkci `center-paragraph` a M-S pro funkci center-line. Příkaz M-S znamená, že máte stisknout klávesu Meta, držet ji stisknutou, a pak stisknout klávesu Shift a „S“.

Nyní si můžete vyzkoušet vytvořit nový buffer a definovat v něm textový mód. Pak stiskněte klíč C-H M. Objeví se vám informace o textovém módu. Možná nebudete všemu rozumět, ale jistě zde najdete užitečné informace.

V dalších oddílech si probereme některé z nejpoužívanějších módů. Budete-li s nimi pracovat, nezapomeňte na klíč C-H M, kdykoliv budete chtít znát o aktuálním módu nějaké podrobnosti.

8.11 Programovací módy

8.11.1 Mód pro jazyk C

Jestliže použijete editor Emacs pro psaní programů v jazyku C, můžete po něm chtít, aby prováděl automatické odsazování. Soubory s příponou „.c“ nebo „.h“ zavádí editor Emacs automaticky v módu „c-mode“. To znamená, že jsou k dispozici některé speciální funkce vhodné pro psaní programů v jazyku C. Klávesa Tab je v módu `c-mode` svázána s funkcí `c-indent-command`. To znamená, že se po stisknutí klávesy Tab nevloží do textu znak Tab, ale dojde k automatickému odsazení řádku podle kontextu ve vytvářeném programu. Z toho vyplývá, že editor Emacs má jisté znalosti o syntaxi programů napsaných v jazyku C. V žádném případě však nepočítejte s tím, že vás bude editor upozorňovat na chyby v programu!

Navíc editor předpokládá, že jsou předcházející řádky odsazeny správně. Pokud v předcházejícím řádku chybí závorka, středník, složená závorka či cokoliv jiného, pak editor provede odsazení chybně. To je pro vás signál, že jste na něco zapomněli a můžete předcházející řádek opravit.

Uvedenou vlastnost editoru Emacs můžete využít ke kontrole oddělovačů ve zdrojovém programu. Nemusíte celý program číst od začátku a pracně hledat chybu, ale stačí zahájit odsazování řádků od začátku souboru pomocí klávesy Tab. Když dojde k nesprávnému odsazení, zkontrolujte předcházející řádek. Jinými slovy, v tomto směru za vás editor Emacs může udělat spoustu práce.

8.11.2 Mód pro jazyk Scheme

Tento mód pro vás bude užitečný jen tehdy, když používáte programovací jazyk Scheme. Tento jazyk není tak běžně používaným jazykem jako jazyk C nebo Pascal, ale v poslední době jeho popularita vzrůstá, a proto se mu budeme také věnovat. Většina toho, co platí pro jazyk Scheme, platí i pro jazyk Lisp.

Aby to nebylo tak jednoduché, v editoru Emacs jsou definovány dva módy pro programovací jazyk Scheme a každý uživatel se může rozhodnout, který mu bude lépe vyhovovat. Zaměříme se na popis módu s názvem `cmuscheme` a později, v oddíle o konfiguraci editoru Emacs, si vysvětlíme rozdíly mezi oběma módy. Nebuďte při čtení následujících řádků zneklidněni, když se váš editor Emacs bude chovat trochu jinak, než zde bude popisováno. V editoru lze konfigurovat prakticky vše a různé distribuce operačního systému Linux obsahují různě konfigurované editory Emacs.

V editoru Emacs můžete inicializovat interaktivní proces Scheme pomocí příkazu `M-X run-scheme`. Takto vytvoříte buffer nazvaný „`*scheme*`“, ve kterém se nachází obvyklý příkazový řádek jazyka Scheme. V tomto řádku můžete zadávat výrazy v jazyku Scheme ukončené klávesou Enter, jazyk Scheme je bude vyhodnocovat a bude zobrazovat příslušné odpovědi. Tímto způsobem můžete, máte-li interaktivně komunikovat s procesem Scheme, zadat definice všech svých funkcí a aplikací v příkazové řádku. Jiná situace nastane v případě, že máte zdrojový kód zapsaný v nějakém samostatném souboru. Pak by mohlo být jednodušší editovat tento soubor a definice odeslat prostřednictvím bufferu Scheme.

Jestliže do editoru Emacs zavedete soubor s příponou „`.ss`“ nebo „`.scm`“, pak se automaticky nastartuje mód **Scheme mode**. Pokud se z nějakých důvodů tento mód nenastartuje, můžete jej aktivovat prostřednictvím příkazu `M-X scheme-mode`. Tento mód ovšem není totožný s bufferem, ve kterém běží proces Scheme. V módu scheme-mode však máte k dispozici některé příkazy pro komunikaci s uvedeným bufferem.

Jestliže ve zdrojovém kódu jazyka Scheme máte vytvořenu definici nějaké funkce, pak ji stisknutím kláves C-C C-E můžete odeslat do bufferu, ve kterém běží proces Scheme. Pokud stisknete klávesy C-C M-E, pak se po odeslání definice funkce dostanete přímo do uvedeného bufferu a zde můžete zadávat interaktivní příkazy. Klávesy C-C C-L jsou určeny k zavedení souboru s kódem v jazyku Scheme (fungují pro buffer s procesem Scheme i pro buffer se zdrojovým kódem). Stejně jako u ostatních programovacích jazyků je klávesa Tab určena k odsazování řádků.

Pokud pracujete v bufferu, ve kterém běží proces Scheme, můžete používat klávesy M-P a M-N k zobrazení předcházejících nebo následujících příkazů (tzv. **input history**).

Předpokládejme, že ladíte nějakou funkci, řekněme '`rotate`', pro kterou jste použili argumenty, například:

```
> (rotate '(a b c d e))
```

pak můžete tento příkaz znovu zavést do příkazového řádku pomocí M-P. Nemusíte znovu v příkazovém řádku Scheme zadávat dlouhou sekvenci znaků a ušetříte tak spoustu času.

Editor Emacs má speciální módy jen pro několik málo programovacích jazyků. Patří mezi ně jazyk C, C++, Lisp a Scheme. (V současné době je dostupný mód snad pro každý programovací jazyk – např. Java, Python nebo JavaScript.)

8.11.3 Mód pro elektronickou poštu

Prostřednictvím editoru Emacs můžete také vytvářet a odesílat zprávy pro elektronickou poštu. K tomu je určen speciální buffer, tzv. „mail buffer“, který se inicializuje prostřednictvím klávesy C-X M. Nejdříve vyplníte položky „`To:`“ a „`Subject:`“ a pak se pomocí klíče C-N přenesete přes oddělovací čáru do pole, ve kterém můžete napsat zprávu. Oddělovací čáru nikdy needitujte ani nerušte, protože pak by editor Emacs nebyl schopen elektronickou zprávu odeslat. Oddělovací čáru používá k odlišení záhlaví od textu zprávy.

V poli pro text zprávy (pod oddělovací čarou) můžete uvést cokoliv. Po dokončení zprávy ji odešlete pomocí kláves C-C C-C. Editor Emacs obsah bufferu odešle.

8.12 Jak zvýšit efektivitu práce s editorem Emacs

Zkušení uživatelé editoru Emacs jsou až fanatičtí, pokud jde o zvyšování efektivity práce s editorem. Ve skutečnosti někdy stráví více času zvyšováním efektivity, než kolik pak ušetří. I když nechci, abyste se stali takovými fanatiky, existuje několik opatření, pomocí kterých se vám bude s editorem Emacs pracovat snadněji. Někteří zkušení uživatelé pohlížejí na začátečníky jako na hlupáky, protože neznají všechny „triky“ s editorem. Já osobně tento druh elitářství odsuzuji, protože jsem také kdysi byl začátečníkem. Nyní se však opět věnujme editoru.

Pro pohyb kurzoru existují některé další klávesy. Víme, že pro pohyb kurzoru o jeden znak doprava je určen klíč C-F. Chcete-li „poskočit“ s kurzorem o celé slovo, zadejte M-F. Vyzkoušejte si, co udělá příkaz M-B. To ale není vše. Pokud zapisujete věty tak, aby za poslední tečkou byly vždy dvě mezery, můžete pomocí klávesy M-E přeskakovat celé věty. Dvě mezery jsou podmínkou, protože jinak by editor nebyl schopen konec věty identifikovat. Opět si vyzkoušejte, jak funguje klíč M-A.

Možná si to ani neuvědomujete a používáte opakovaně klíč C-F k posunu kurzoru na konec řádku. Připomínáme, že k tomuto účelu je určen klíč C-E a k nastavení kurzoru na začátek řádku je určen klíč C-A. Možná, že často používáte klíč pro posun kurzoru na následující řádek, a přitom byste měli použít klíč C-V, kdy se vám zobrazí následující stránka. Zrovna tak využívejte klíč M-V.

Pokud jste někde u konce řádku a všimnete si, že jste někde na začátku udělali chybu, pak nepoužívejte klávesy Backspace nebo Delete, abyste se dostali k chybnému místu. Museli byste jinak dobře napsaný řádek psát znovu. Místo toho používejte klíčů M-B, C-B, případně C-F, opravte chybu a pak se pomocí C-E vraťte na konec řádku.

Pokud zadáváte jméno souboru, nezadávejte je nikdy celé. Zadejte jen nezbytný počet znaků, které soubor jednoznačně identifikují. Pak vám editor Emacs zbývající znaky po stisknutí klávesy Tab nebo mezerníku automaticky doplní. Šetřte sebe a ne procesor!

Když píšete nějaký dlouhý text, používejte funkci pro automatické ukončování řádků. Klíč M-Q vyvolá funkci `fill-paragraph` a lze jej použít ve všech textových módech. Také jej můžete použít k „zarovnání“ již napsaného odstavce. Stačí nastavit kurzor na nějakou pozici uvnitř odstavce a stisknout M-Q.

Někdy je užitečné použít klíč C-X U, kdy se editor pokusí vrátit zpět provedené změny. Editor Emacs v tomto případě odhadne, kolik změn má vrátit, a jeho odhad je obvykle velmi inteligentní. Klíč C-X U můžete použít opakovaně až do okamžiku, kdy editor vrátí poslední změnu, kterou si „pamatuje“.

8.13 Konfigurace editoru Emacs

Editor Emacs je tak velký a tak komplexní program, že má i svůj vlastní programovací jazyk. Vlastnosti editoru můžete modifikovat pomocí vlastních programů. Programovací jazyk integrovaný v editoru Emacs se jmenuje Emacs Lisp a je dialektem jazyka Lisp. Pokud máte s jazykem Lisp nějaké zkušenosti, pak se vám bude zdát opravdu přátelským. Pokud nemáte, ničeho se neobávejte. V tomto oddíle se nebudeme programováním v editoru Emacs zabývat příliš do hloubky a pokud se s jazykem Emacs Lisp budete chtít seznámit podrobněji, pak si prostudujte stránky dostupné v systému Info.

Většina funkcí editoru Emacs je definována pomocí kódu napsaného v jazyku Emacs Lisp.[9] Většina z těchto souborů je distribuována spolu s editorem a kolektivně se nazývají „Emacs Lisp library“. Umístění této knihovny závisí na tom, jakým způsobem je editor Emacs instalován ve vašem systému. Nejčastěji jej můžete najít v adresáři: `/usr/lib/emacs/lisp`, `/usr/lib/emacs/19.19/lisp` a podobně. Číslo `19.19` značí verzi editoru Emacs a může se lišit od verze ve vašem systému.

Informace o uložení knihovny je uložena v interní proměnné **load-path** editoru Emacs, proto ji nemusíte pracně hledat v souborovém systému. Jestliže chcete zjistit hodnotu této proměnné, musíte ji **vyhodnotit**. To znamená, že musíte aktivovat interpretr Emacs Lisp. V editoru

9 Někdy se neoficiálně nazývá „Elisp“.

Emacs existuje speciální mód pro vyhodnocování výrazů jazyka Lisp zvaný **lisp-interaction-mode**. S tímto módem je obvykle spjat buffer „`*scratch*`“. Pokud se vám jej nepodaří nalézt, vytvořte nový buffer s jakýmkoliv jménem a zadejte příkaz M-X lisp-interaction-mode.

Nyní se nacházíte v pracovním prostoru pro interaktivní komunikaci s interpretrem Emacs Lisp. Zadejte příkaz:

```
load-path
```

a pak stiskněte C-J. V módu interaktivní komunikace je klíč C-J svázán s funkcí `eval-print-last-sexp`. Slovo „sexp“ znamená „s-expression“ (**s-výraz**), což znamená vyváženou skupinu závorek - to je opravdu řečeno velmi zjednodušeně, ale brzy budete tušit, k čemu jsou takové výrazy užitečné při práci z jazykem Emacs Lisp. V každém případě po vyhodnocení proměnné `load-path` obdržíte zprávu, která bude vypadat přibližně takto:

```
load-path Ctrl+j
("/usr/lib/emacs/site-lisp/vm.5.35" "/home/kfogel/elithp"
"/usr/lib/emacs/site-lisp" "/usr/lib/emacs/19.19/lisp")
```

Toto hlášení nebude vypadat v každém systému stejně, protože závisí na způsobu instalace editoru Emacs. Uvedený příklad pochází z mého počítače s procesorem 386, na němž běží operační systém Linux. Jak vyplývá z předcházejícího výpisu, proměnná `load-path` je seznam řetězců. Každý řetězec uvádí adresář, který by mohl obsahovat soubory náležící do systému Emacs. Když potřebuje editor Emacs zavést soubory obsahující kód Lisp, hledá tyto soubory v uvedených adresářích v uvedeném pořadí. Pokud je v proměnné `load-path` uveden adresář, který v souborovém systému neexistuje, editor jej ignoruje.

Ve fázi startování se editor Emacs pokouší nalézt soubor .emacs ve vašem domovském adresáři. Proto platí, že pokud chcete mít svoji vlastní konfiguraci editoru Emacs, měli byste používat soubor `.emacs`. Nejvýznamnější konfigurační nastavení se týkají vazeb mezi klíči a funkcemi, proto se jim nyní budeme věnovat.

```
(global-set-key "\Ctrl+cl" 'goto-line)
```

Funkce `global-set-key` má dva argumenty: prvním z nich je klíč a druhým je funkce, která se má po stisknutí tohoto klíče realizovat. Slovo „`global`“ znamená, že uvedená vazba mezi klíčem a funkcí bude platit ve všech hlavních módech. Existuje jiná funkce, `local-set-key`, jež nastavuje vazbu mezi klíčem a funkcí pro jediný buffer. V uvedeném příkladu jsme nastavili vazbu mezi klíčem C-C J a funkcí `goto-line`. Při definici klíče se musí použít řetězec, což znamená, že se klíč musí uzavřít do uvozovek. Speciální syntaxe „`\Ctrl+<char>`“ znamená, že se má stisknout klávesa Ctrl, držet stisknutá, a pak se má stisknout klávesa `<char>`. Podobně platí, že „`\Alt+<char>`“ indikuje kombinaci s klávesou Meta.

Konfigurace vazby mezi klíčem a funkcí vypadá jednoduše, ale kde získat informace o funkci „`goto-line`“? Nebo naopak, předpokládejme, že chci klíč Ⓒ-Ⓒ Ⓛ svázat s funkcí, která umožní přeskočit na řádek specifikovaného pořadového čísla, ale jak mám zjistit jméno takové funkce?

V tomto okamžiku využijete vynikajících vlastností systému nápovědy, který je integrován v editoru Emacs. Jakmile se jednou rozhodnete, jaký druh funkce chcete použít, můžete použít editor Emacs k „vystopování“ jejího jména. Jedna sice rychlá, ale ne příliš přímočará metoda spočívá v tom, že se využije schopnosti editoru Emacs doplňovat jména funkcí. Měli byste si pamatovat, že klíč Ⓒ-Ⓗ Ⓕ vyvolá nápovědu popisující funkci (t.j. vyvolá funkci `describe-function`). Pak stiskněte Tab, aniž cokoliv zadáte. Tak „donutíte“ editor Emacs doplnit prázdný řetězec. Jinými slovy, editor vypíše jména všech funkcí, které jsou v něm interně definovány. Uvedená operace bude chvíli trvat, protože takových funkcí je v editoru definováno opravdu mnoho.

Nyní stiskněte klíč Ⓒ-Ⓖ, čímž funkci `describe-function` přerušíte. Nyní v editoru existuje buffer nazvaný „`*Completions*`“, který obsahuje požadovaný seznam všech interních funkcí editoru Emacs. Přepněte se do tohoto bufferu a pomocí postupného vyhledávání najděte funkci, kterou chcete použít. Můžete například využít předpokladu, že funkce pro přechod na řádek specifikovaného čísla bude ve svém názvu obsahovat slovo „`line`“. Proto zadejte vyhledávání slova „`line`“ a najdete tak všechny eventuality.

Vhodnější metoda spočívá v tom, že použijete klíče Ⓒ-Ⓗ Ⓐ (t.j. vyvoláte funkci `command-apropos`), kdy se vám zobrazí všechny funkce obsahující specifikovaný řetězec. Výstup z funkce `command-apropos` se poněkud hůře třídí, než výstup z funkce `describe-function`. Vyzkoušejte si obě metody a vyberte tu, která vám bude lépe vyhovovat.

Samozřejmě se může stát, že budete hledat funkci, která v editoru Emacs neexistuje. V takovém případě si ji budete muset napsat sami. Nebudeme zde popisovat, jak se v jazyku Emacs Lisp programuje. Doporučujeme prostudovat si příklady v knihovně Emacs Lisp a přečíst si stránky systému Info popisující jazyk Emacs Lisp. Pokud znáte někoho, kdo programování v jazyku Emacs Lisp ovládá, jistě vám pomůže.

Definice vlastních funkcí editoru Emacs není nic těžkého. Abych vás trochu navnadil, za poslední rok jsem jich bez problémů vytvořil 131. Vyžaduje to trochu zkušeností, ale programování v jazyku Emacs Lisp zvládnete poměrně rychle.

Další konfigurační možnosti spočívají v tom, že se v souboru `.emacs` nastaví hodnoty jistých proměnných. Přidejte například do vašeho souboru .emacs následující řádek a pak znovu nastartujte editor Emacs:

```
(setq inhibit-startup-message t)
```

Editor Emacs kontroluje ve fázi startování proměnnou inhibit-startup-message a podle její hodnoty se rozhodne, zda zobrazovat jisté informace (o verzi editoru, zárukách a podobně) nebo ne. Uvedený výraz jazyka Lisp používá příkaz `setq` k nastavení proměnné `inhibit-startup-message` na hodnotu „t“ což je speciální hodnota v jazyce Lisp pro „true“. Opakem je hodnota „nil“ což je speciální hodnota pro „false“. V mém konfiguračním souboru `.emacs` se nacházejí některá nastavení, jež možná shledáte užitečnými:

```
(setq case-fold-search nil) ; gives case insensitivity in searching
;; make C programs ident the way I like them to:
(setq c-indent-level 2)
```

První výraz nastavuje způsob vyhledávání na tzv. case-insensitive, což znamená, že se při vyhledávání nerozlišují malá a velká písmena (a to dokonce i tehdy, když hledaný řetězec obsahuje buď pouze malá, nebo pouze velká písmena). V druhém výrazu se nastavuje odsazování řádků při psaní programů v jazyku C na hodnotu 2. Je to poněkud méně, než je implicitní nastavení, ale to záleží na individuálním vkusu a na tom, jaká hodnota odsazení učiní váš program napsaný v jazyku C čitelnějším.

V jazyku Lisp se komentářový řádek označuje znakem „`;`“. Editor Emacs ignoruje vše, co se nachází za tímto znakem. Výjimku tvoří případ, kdy se znak „`;`“ nachází uvnitř řetězce. Například:

```
;; Tyto dva řádky jsou v jazyku Lisp ignorovány, ale
;; následující s-výraz bude plně vyhodnocen:
(setq some-literal-string "An awkward pause; for no purpose.")
```

Doporučuje se, abyste změny ve zdrojových souborech pro jazyk Lisp komentovali, protože jinak například po šesti měsících určitě zapomenete, jakou změnu jste dělali. Komentář na celý řádek uvádějte dvojicí znaků `;;`. Pak bude editor Emacs správně provádět odsazování řádků.

To, co jsme si řekli o vyhledávání interních funkcí editoru Emacs, platí i pro vyhledávání proměnných. Chcete-li vytvořit seznam všech proměnných, zadejte příkaz C-H v (describe-variable), nebo použijte C-H C-A (apropos). Použijete-li druhou možnost, pak musíte počítat s tím, že vám příkaz vyhledá funkce a proměnné dohromady.

Soubory se zdrojovým kódem v jazyku Emacs Lisp mají implicitní příponu „`.el`“, například „`c-mode.el`“. Aby však mohl kód vytvořený v jazyku Emacs Lisp běžet rychleji, umožňuje editor provést jakousi **předkompilaci** (soubory označované jako „**byte-compiled**“) a pak mají tyto soubory implicitní příponu „`.elc`“. Výjimku, pokud jde o implicitní příponu, tvoří soubor `.emacs`, jenž příponu „`.el`“ nepotřebuje, neboť jej editor hledá automaticky ve fázi startování.

Chcete-li zavést interaktivně soubor s kódem v jazyku Lisp, použijte příkaz `M+X load-file`. Editor vás vyzve k zadání jména souboru a pak specifikovaný soubor zavede do své pracovní oblasti. Jestliže chcete zavést soubor „zevnitř“ jiného souboru napsaného v jazyku Lisp, postupujte takto:

```
(load "c-mode") ; tento příkaz zavede buď soubor c-mode.el, nebo
  c.mode.elc
```

Editor Emacs nejdříve k uvedenému jménu přidá příponu `.elc` a pokusí se jej nalézt v některém adresáři specifikovaném v proměnné `load-path`. Pokud jej nenajde, přidá příponu `.el` a hledání zopakuje. Jestliže není úspěšný ani v tomto případě, pak použije přímo řetězec předaný funkci `load`. Překlad můžete realizovat prostřednictvím příkazu `Alt+X byte-compile-file`. Pokud však zdrojový soubor často aktualizujete, pak to zpravidla nemá smysl. Soubor .emacs nikdy nepřekládejte, ani mu nepřidávejte příponu `.el`.

Bezprostředně po zavedení souboru `.emacs` vyhledá editor Emacs soubor `default.el` a zavede jej. Tento soubor je obvykle umístěn v adresáři uvedeném v proměnné `load-path` s názvem `site-lisp` nebo `local-elisp` či v nějakém podobném adresáři (podívejte se na proměnnou `load-path`). Uživatelé, kteří provádějí údržbu editoru Emacs používají soubor `default.el` k nastavení globálních konfigurací, které pak platí pro každého uživatele v systému. Soubor `default.el` by také neměl být kompilován, protože se v něm často provádějí změny.

Jestliže váš osobní soubor `.emacs` obsahuje nějakou chybu, pak se editor Emacs nebude pokoušet načíst soubor `default.el`, ale zastaví svou činnost a vypíše hlášení: „`Error in init file`“. Uvidíte-li takové hlášení, pak jste pravděpodobně udělali nějakou chybu v souboru `.emacs`.

V souboru `.emacs` se ještě vyskytuje jeden druh výrazů. Knihovna Emacs Lisp někdy nabízí více sad programů, které realizují tytéž funkce různým způsobem. To znamená, že musíte specifikovat tu sadu, která se má používat (implicitní sada nemusí být pro vás vždy tou nejlepší). Tyto sady se například uplatňují v oblasti interaktivního rozhraní pro jazyk Scheme. S editorem Emacs se standardně distribuují dvě sady funkcí interaktivního rozhraní pro jazyk Scheme: `xscheme` a `cmuscheme`.

```
prompt> /usr/lib/emacs/19.19/lisp/*scheme*
/usr/lib/emacs/19.19/lisp/cmuscheme.el
/usr/lib/emacs/19.19/lisp/cmuscheme.elc
/usr/lib/emacs/19.19/lisp/scheme.el
/usr/lib/emacs/19.19/lisp/scheme.elc
/usr/lib/emacs/19.19/lisp/xscheme.el
/usr/lib/emacs/19.19/lisp/xscheme.elc
```

Já osobně dávám přednost sadě cmuscheme před sadou `xscheme`, avšak editor Emacs implicitně používá sadu `xscheme`. Jak „donutit“ editor Emacs, aby byl nakonfigurován v souladu s mým přáním? Do souboru `.emacs` jsem vložil následující řádky:

```
;; notice how the expression can be broken across two lines. Lisp
;; ignores whitespaces, generally:
(autoload 'run-scheme "cmuscheme"
"Run an inferior Scheme, the way I like it". t)
```

Funkce `autoload` akceptuje jako argument jméno funkce (začínající apostrofem ') a „sdělí“ editoru Emacs, že je tato funkce definována v jistém souboru. Jméno tohoto souboru se předává jako druhý argument, tedy řetězec (buď s příponou „`.el`“, nebo „`.elc`“). Soubor pak bude vyhledán v adresářích definovaných v proměnné `load-path`.

Zbývající argumenty jsou volitelné, ale jsou nezbytné v tomto případě: Třetí argument je dokumentačním řetězcem pro specifikovanou funkci. Pokud použijete funkci `describe-function`, pak se jako popis k vyhledané funkci objeví právě tento řetězec.

Čtvrtý argument „sdělí“ editoru Emacs, že specifikovaná funkce může být volána interaktivně, t.j. pomocí klíče Alt+X. V tomto případě je to velmi důležité, protože jedině tak lze spustit proces Scheme v prostředí editoru Emacs pomocí příkazu `Alt+x run-scheme`.

Nyní, když je funkce `run-scheme` definována jako automaticky zaveditelná, co se stane, když se zadá příkaz `Alt+x run-scheme`? Editor Emacs vyhledá funkci `run-scheme`, zjistí, zda je automaticky zaveditelná a zavede specifikovaný soubor (v našem případě „`cmuscheme`“). Protože kompilovaný soubor `cmuscheme.elc` existuje, editor jej zavede. Tento soubor musí definovat funkci `run-scheme`, jinak dojde při jeho zavádění k chybě. Naštěstí tuto funkci skutečně definuje, proto jde vše hladce a já mám k dispozici své oblíbené rozhraní pro jazyk Scheme.[10]

Automatické zavedení funkce zajišťuje, že bude editor Emacs schopen funkci najít, až ji budete potřebovat. Navíc můžete nad automaticky zaváděnými funkcemi získat jistou kontrolu. Dále platí, že automaticky zaveditelné funkce šetří paměť spotřebovanou editorem Emacs, protože se tyto funkce zavádějí až v okamžiku, kdy jsou potřebné. Řada příkazů není ve fázi startování editoru ve skutečnosti definována. Místo toho jsou nastaveny jako automaticky zaveditelné funkce. Pokud takový příkaz nezadáte, nikdy se odpovídající funkce nezavede. Uvedená vlastnost automatického zavádění funkcí je pro editor Emacs (a hlavně pro uživatele) „životně“ důležitou. Kdyby editor ve fázi startování zaváděl všechny funkce, trvala by tato fáze asi dvacet minut a celá dostupná paměť vašeho počítače by byla obsazena. O automatické zavádění implicitních funkcí se nemusíte starat, editor Emacs je realizuje sám.

10 Jistě jste si všimli, že o rozhraní cmuscheme jsme hovořili již dříve. Proto byste si měli zavedení sady cmuscheme vyzkoušet.

8.14 Kde získat další informace

V této kapitole jsme zdaleka neuvedli vše, co byste měli znát o editoru Emacs. Je zde zhruba jedno procento informací. Budete-li editor intenzivně využívat, pak budete potřebovat znát spoustu dalších „triků“ šetřících čas. Nejjednodušší bude, když vyčkáte, až budete muset řešit nějaký konkrétní problém. Pak vyhledáte funkci, která váš problém vyřeší.

Snad nejdůležitější je, abyste uměli plně využívat systém nápovědy integrovaný v editoru Emacs. Řekněme například, že budete chtít vložit do editovaného textu obsah jiného souboru. Snadno uhodnete, že asi existuje funkce `insert-file` (vložení souboru). Máte-li svou domněnku potvrdit, použijte příkaz C-H F. V příkazovém řádku zadejte jméno funkce, kterou hledáte a o které si chcete přečíst nějaké informace. Protože víte, že editor Emacs je schopen doplňovat jména funkcí, můžete jako „počáteční“ odhad uvést řetězec „insert“. Pak stačí stisknout klávesu Tab. Objeví se vám seznam všech funkcí obsahujících řetězec „insert“ a funkce „`insert-file`“ je jedna z nich.

Nyní si můžete přečíst spoustu informací o funkci `insert-file` a také ji můžete použít - stačí zadat příkaz `Alt+x insert-file`. Chcete-li také zjistit, zda je tato funkce svázána s nějakým klíčem, zadejte příkaz `Ctrl+h w insert-file` a stiskněte Enter. Čím lépe budete znát vlastnosti systému nápovědy v editoru Emacs, tím snáze naleznete potřebné informace. Máte-li navíc dostatek erudice zkoumat nové věci a ochotu učit se, ušetříte mnoho času.

Jestliže si chcete objednat kopii manuálu k editoru Emacs a/nebo manuál „*Emacs Lisp Programming*“, pošlete objednávku na adresu:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

Oba tyto manuály jsou distribuovány v elektronické podobě spolu s editorem Emacs a jsou čitelné prostřednictvím systému Info (použijte klávesu C-H I, pomocí které inicializujete rozhraní mezi editorem Emacs a systémem Info). Na druhé straně, cena za uvedené manuály je opravdu mírná a navíc se získané peníze budou věnovat na vývoj kvalitního volně šiřitelného programového vybavení. Také chceme poznamenat, že pomocí klíčů C-H C-C si můžete zobrazit informace o licenčních podmínkách platných pro editor Emacs. Jsou určitě zajímavější, než si teď myslíte, a pomohou vám vyjasnit si pojem „volné programové vybavení“. Pokud si myslíte, že pojem „volné programové vybavení“ znamená, že daný program nic nestojí, pak si licenční podmínky rychle přečtěte.

Kapitola

9

Konfigurace operačního systému Unix

9.1 Konfigurace příkazového interpretu bash

Filosofie operačního systému Unix se od filosofie jiných operačních systémů podstatně liší v jedné věci. Autoři Unixu se nepokoušeli předvídat všechny potřeby všech uživatelů. Místo toho se pokusili navrhnout operační systém tak, aby si každý uživatel pracovní prostředí svého operačního systému snadno upravil sám podle svých potřeb. Konfigurace jednotlivých programů operačního systému Unix se definuje prostřednictvím tzv. **konfiguračních souborů**. Někdy jsou označovány jako „ini-files“ nebo „rc files“ nebo dokonce jako „dot files“ (jedná se zpravidla o soubory, jejichž jména začínají tečkou). Pokud si vzpomínáte, tak jména souborů začínající znakem „.“ nejsou normálně příkazem `ls` zobrazována.

Nejdůležitější konfigurační soubory jsou ty, které používá příkazový interpret. Implicitním příkazovým procesorem v operačním systému Linux je `bash` a právě jemu bude věnována tato kapitola. Než začneme popisovat, jak příkazový interpret `bash` konfigurovat, podívejme se na soubory, které `bash` vyhledává.

9.1.1 Inicializace příkazového interpretu bash

Existuje několik různých způsobů, jak příkazový procesor `bash` spustit. Tzv. **login-shell** se automaticky spouští poté, co se přihlásíte do systému.

Další způsob spočívá ve spuštění **interaktivního příkazového procesoru** (interactive shell). To je jakýkoliv příkazový procesor, který se prezentuje příkazovým řádkem. Příkazový interpret, jenž se spustí bezprostředně po přihlášení se do systému, je také interaktivní. Jiným příkladem interaktivního příkazového interpretu je program xterm spuštěný z prostředí X Window.

Existují také **neinteraktivní příkazové interprety.** Tyto procesory se používají k vykonávání příkazů uvedených v souboru, jako jsou dávkové soubory s příponou .BAT v operačním systému MS-DOS. Analogií v operačním systému Unix jsou tzv. **skripty**. Skript příkazového procesoru je něco jako miniprogram. Jsou sice výrazně pomalejší než kompilovaný program, ale snadno se vytvářejí a modifikují.

Příkazové procesory v operačním systému Unix používají v závislosti na typu následující inicializační soubory:

Typ příkazového procesoru	inicializační soubor
Interaktivní příkazový interpreter spuštěný při přihlášení se do systému	`.bash_profile`
Interaktivní příkazový procesor	`.bashrc`
Neinteraktivní příkazový procesor	skript příkazového procesoru

9.1.2 Inicializační soubory

Protože většina uživatelů chce mít stejné uživatelské prostředí bez ohledu na to, jaký typ příkazového procesoru se spustí, začneme konfiguraci tím, že do konfiguračního souboru `.bash_profile` vložíme jednoduchý příkaz „source ~/.bashrc“. Příkaz source „sdělí“ příkazovému procesoru, že má jeho argument interpretovat jako skript. To znamená, že se při každém spuštění skriptu `.bash_profile` také spustí skript `.bashrc`.

Nyní budeme přidávat příkazy do našeho souboru `.bashrc`. Do souboru `.bash_profile` se zadávají pouze příkazy, které se mají spouštět při přihlášení se do systému.

9.1.3 Vytváření druhých jmen

Jakým způsobem lze měnit konfigurační nastavení? Hned uvedeme příklad, který si do svého konfiguračního souboru `.bashrc` zadává devadesát procent uživatelů operačního systému Unix:

```
alias ll="ls -l"
```

Příkaz definuje tzv. **alias** (druhé jméno) příkazu. V našem případě se jméno příkazu `ll` rozšiřuje na příkaz příkazového procesoru „`ls -l`“. Za předpokladu, že příkazový procesor bash přečetl uvedenou definici ve vašem konfiguračním souboru `.bashrc`, pak má příkaz `ll` stejný efekt jako příkaz `ls -l`. Přitom ušetříte polovinu stisknutých kláves. Když zadá-

te v příkazovém řádku `ll` a stisknete Enter, příkazový procesor zadaný příkaz rozvine podle definice a pak realizuje příkaz `ls -l`. Ve skutečnosti v systému příkaz `ll` neexistuje, ale příkazový procesor `bash` jej automaticky transformuje na platný program.

Některé příklady druhých jmen jsou uvedeny na této straně dole. Můžete si je vložit do vašeho souboru `.bashrc`. Zvláště zajímavý je první z nich. Je-li definován alias `ls="ls -F"`, pak po každém zadání příkazu `ls` bude automaticky aplikovat volbu `-F`. Platí, že se alias nikdy nepokusí rekurzivně rozšířit sám sebe. Uvedený příklad demonstruje nejčastěji používaný způsob automatického přidávání voleb do příkazů.

Všimněte si znaku „#“. Ve skriptech příkazového procesoru se tento znak používá k označení komentáře a příkazový procesor zbytek řádku za znakem # ignoruje.

Dále jste si mohli všimnout několika dalších věcí. Především se v některých definicích nevyskytují uvozovky, například v definici `pu`. Platí totiž, že pokud se na pravé straně znaku rovná se (=) vyskytuje jediné slovo, pak se uvozovky nemusejí uvádět.

Nic by se nestalo, kdyby zde uvozovky byly uvedeny. Určitě však uvozovky používejte v případě, když budete definovat nové jméno pro příkaz s volbami a/nebo argumenty. Například:

```
alias rf="refrobnicate -verbose -prolix -wordy -o foo.out"
alias ls="ls -F"          # give characters at end of listing
alias ll="ls -l"          # special ls
alias la="ls -a"
alias ro="rm  *~; rm.*~"  # removes backup files created by Emacs
alias rd="rmdir"          # saves typing!
alias md="mkdir"
alias pu=pushd            # pushd, popd, and dirs weren't covered
alias po=popd             # manual---you might want to look them up
alias ds=dirs             # in the bash manpage
# these all are just keyboard shortcuts
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wombat.gnu.ai.mit.edu"
alias tko="tpalk kold@cs.oberlin.edu"
alias tjo="talk jimb@cs.oberlin.edu"
alias mroe="more"         # spelling correction!
alias moer="more"
alias email="emacs -f rmail"     # my mail reader
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
                          # one way of invoking emacs
```

Asi se vám zdá, že poslední alias má divně umístěné uvozovky:

```
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
```

Jak asi tušíte, chtěl jsem umístit uvozovky do samotných voleb. Proto jsem musel před každou uvozovku vložit obrácené lomítko. Příkazový procesor pak takovou uvozovku nebude interpretovat obvyklým způsobem.

Všimněte si také dvou definic pro opravu chybně zadaného příkazu `more`. Pokud omylem zadám „`mroe`“ nebo „`moer`“, bude příkazový procesor „vědět“, že jsem chtěl zadat „`more`“. Aliasy neinterferují s argumenty předávanými programům. To znamená, že například příkaz

```
/home/larry# mroe hurd.txt
```

bude fungovat dobře.

Určitě platí, že správné používání definice druhých jmen představuje polovinu konfiguračních možností, které jsou u příkazových procesorů k dispozici. Při práci v operačním systému Unix si všímejte, které příkazy často zadáváte, a pak si pro ně pořiďte druhá jména, tedy zkratky. Zjistíte, že se vám pak bude pracovat mnohem radostněji.

9.1.4 Systémové proměnné

V souboru `.bashrc` se definují další důležitá konfigurační nastavení prostřednictvím systémových proměnných (environment variables). Co jsou to systémové proměnné? Pojďme na to z druhé strany: předpokládejme, že čtete dokumentaci k programu `fruggle` a že narazíte na následující věty:

> *Fruggle normally looks for its configuration file, .frugglerc, in the user's home directory. However, if the environment variable FRUGGLEPATH is set to a different filename, it will look there instead.*

Každý program běží v nějakém **prostředí** a to je definováno příkazovým interpretem, jenž program volá.[1] Lze si představit, že prostředí existuje uvnitř příkazového interpretu. Programátoři mají k dispozici speciální funkci pro získání hodnoty systémové proměnné a program `fruggle` tuto funkci využívá. To znamená, že ověří hodnotu v systémové proměnné `FRUGGLEPATH`. Pokud není tato systémová proměnná definována, pak program použije soubor `.frugglerc` ve vašem domovském adresáři. Pokud však je definována, program `fruggle` použije její hodnotu místo implicitního souboru `.frugglerc`.

[1] Nyní vidíte, proč jsou příkazové procesory tak důležité. Představte si, že byste museli definovat celé prostředí, kdykoliv budete spouštět nějaký program!

Nyní si uveďme ukázku, jak změnit prostředí v příkazovém procesoru `bash`:

```
/home/larry# export PGPPATH=/home/larry/secrets/pgp
```

Pod příkazem `export` si můžete představit následující větu: „Exportuj tuto proměnnou do prostředí, ze kterého budu spouštět program, tak, aby proměnná byla z tohoto programu viditelná.“ Později uvidíte, že jsou i jiné důvody pro používání příkazu `export`.

Uvedenou systémovou proměnnou používá program `pgp` pro šifrování, jehož autorem je Phil Zimmerman. Program `pgp` implicitně používá váš domovský adresář jako adresář, ve kterém hledá jisté soubory (šifrovací klíče). Také tento adresář využívá k vytvoření dočasných pracovních souborů. Nastavením systémové proměnné `PGPPATH` jsem změnil pracovní adresář na `/home/larry/secrets/pgp`. Abych zjistil přesné jméno této systémové proměnné, musel jsem si přečíst manuál k programu `pgp`. Systémové proměnné se zpravidla uvádějí velkými písmeny a končí na „`PATH`“.

Je užitečné vědět, jak hodnotu systémové proměnné zjistit:

```
/home/larry# echo $PGPPATH
/home/larry# .pgp
/home/larry#
```

Všimněte si, že jsme před jméno systémové proměnné uvedli znak `$`. Jedině tak lze zjistit hodnotu systémové proměnné. Pokud byste znak dolaru neuvedli, dostali byste následující výpis:

```
/home/larry# echo PGPPATH
PGPPATH
/home/larry#
```

Znak dolaru se používá k vyhodnocení systémové proměnné, avšak pouze v kontextu s příkazovým procesorem - přesněji s příkazovým procesorem, jenž realizuje interpretaci příkazu. V jakých případech realizuje příkazový procesor interpretaci?

Proměnná	Obsahuje	Příklad
`HOME`	Váš domovský adresář	`/home/larry`
`TERM`	Typ vašeho terminálu	`xterm, vt100, console`
`SHELL`	Cesta k vašemu příkazovému procesoru	`/bin/bash`

(pokračování)

Proměnná	Obsahuje	Příklad
USER	Jméno vašeho účtu	larry
PATH	Seznam adresářů, ve kterých se automaticky vyhledávají programy ke spuštění	/bin:/usr/local/bin:/usr/bin/X11

Tabulka 9.1
Některé důležité systémové proměnné

Je to v těch případech, kdy zadáváte příkaz z příkazového řádku nebo kdy příkazový procesor čte příkazy z nějakého souboru, například ze souboru .bashrc.

Existuje další důležitý příkaz, pomocí kterého lze získat informace o prostředí. Tímto příkazem je env. Po zadání příkazu env se vám zobrazí seznam všech systémových proměnných. Jste-li uživateli systému X Window, pak bude tento seznam velmi dlouhý, proto používejte příkaz env | more.

Některé systémové proměnné jsou opravdu důležité, proto jsou uvedeny v tabulce 9.1. Tyto systémové proměnné se automaticky definují po přihlášení se do systému. Není proto nutné je nastavovat v souboru .bashrc nebo .bash_login.

Nyní se blíže podívejme na systémovou proměnnou TERM. Abychom jí mohli porozumět, podívejme se zpět do historie operačního systému Unix. Operační systém musí znát jisté údaje o vaší konzole, aby mohl realizovat takové funkce, jako je zápis znaků na obrazovku, pohyb kurzoru v textovém řádku a podobně. V počátcích rozvoje výpočetní techniky výrobci terminálů průběžně rozšiřovali jejich vlastnosti: nejdříve inverzní video, později znaky pro evropské jazyky, dokonce i první funkce pro kreslení (připomínáme, že jde o dobu, kdy se ještě nikomu ani nesnilo o grafických oknech a myších). Každá nová vlastnost však přinášela programátorům problémy: jak měli vědět, co terminál podporuje a co ne? Jak měli podporovat nové vlastnosti a přitom „neodepsat“ staré terminály?

V operačním systému Unix najdete odpověď na tyto otázky v souboru /etc/termcap. Soubor /etc/termcap obsahuje seznam všech terminálů, o kterých váš operační systém „ví,“ a dále informace, jakým způsobem se má řídit kurzor. Pokud systémový správce dostane nový terminál, pak pouze do souboru /etc/termcap přidá záznam vztahující se k novému terminálu a nemusí pracně konfigurovat celý operační systém Unix. Někdy je situace ještě jednodušší. Kdysi se stal terminál vt100 od firmy Digital Equipment Corporation jakýmsi pseudostandardem, který převážná většina moderních terminálů respektuje a umí emulovat.

V operačním systému Linux je někdy hodnota systémové proměnné TERM nastavena jako console, což znamená emulaci terminálu vt100 s některými speciálními funkcemi.

Další proměnná, PATH, je rovněž kritickou systémovou proměnnou z hlediska funkčnosti příkazového procesoru. Zde uvádím nastavení na mém počítači:

```
/home/larry# env | grep ^PATH
PATH=/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/
  TeX/bin
/home/larry#
```

Systémová proměnná PATH obsahuje seznam adresářů oddělených dvojtečkou, ve kterých systém automaticky vyhledává spustitelné programy. Když například zadám příkaz ls a stisknu klávesu Enter, bude příkazový procesor bash hledat program ls nejdříve v adresáři /home/larry/bin, který jsem vytvořil pro ukládání mých vlastních programů.

Program ls jsem však nenapsal já (ten byl pravděpodobně napsán ještě před tím, než jsem se narodil), proto zde příkazový procesor tento příkaz nenašel. Jako další prohledává příkazový procesor adresář /bin. A zde program ls našel. Protože soubor ls je spustitelný program, přestane příkazový procesor dále hledat a spustí jej. Může se stát, že v jiném adresáři bude také uložen spustitelný program ls (například v adresáři /usr/bin), ale příkazový procesor jej nespustí, pokud mu to výslovně nepřikážete:

```
/home/larry# /usr/bin/ls
```

Systémová proměnná PATH existuje hlavně proto, abychom nemuseli při každém spouštění nějakého programu zadávat celou cestu k tomuto programu. Zadáte-li tedy jakýkoliv příkaz, prohledá příkazový procesor všechny adresáře uvedené v systémové proměnné PATH. Když jej najde, spustí jej, a pokud ne, vypíše následující zprávu:

```
/home/larry# clubly
clubly: command not found
```

Všimněte si, že moje systémová proměnná PATH neobsahuje aktuální adresář, tedy „.“. Pokud by obsahovala, pak by vypadala takto:

```
/home/larry# echo $PATH
.:/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr
  /TeX/bin
/home/larry#
```

Zda zadávat nebo nezadávat aktuální adresář do systémové proměnné PATH je předmětem diskuse v kuloárech okolo operačního systému Unix. Problém tkví v tom, že aktuální adresář v systémové proměnné PATH by mohl představovat „díru“ v bezpečnosti systému.

Předpokládejme, že se přepnete do adresáře, ve kterém někdo nechal virový program „Trójský kůň“ a nazval jej `ls`. Vy nic zlého netušíte a zadáte příkaz `ls`, což je přirozené, chcete-li se seznámit s novým adresářem. Protože je aktuální adresář uveden v systémové proměnné `PATH` jako první, spustí příkazový procesor právě tuto verzi příkazu `ls`. I když jste nechtěli udělat nic špatného, rozpoutali jste ve vašem systému virovou nákazu. Proti takovým haváriím neexistuje dost účinná ochrana. Virový program může spustit osoba, která ani nemá privilegia uživatele root. Stačí, aby měla právo zapisovat do adresáře, ve kterém se virový program nachází. Dokonce to může být její domovský adresář.

Pokud jde o váš systém, je pravděpodobné, že jeden uživatel neklade druhému různé nástrahy ve formě „Trojských koňů“ a že kolektiv uživatelů je založen na přátelských a kolegiálních vztazích. Avšak ve velkých systémech s mnoha uživateli (jako jsou například universitní počítače), mohou být desítky programátorů, které byste nejraději ani nepotkali. Z toho vyplývá, že zařazení aktuálního adresáře do systémové proměnné PATH závisí na konkrétní situaci.[2]

Skutečný způsob, kterým je nastavena systémová proměnná `PATH` v mém počítači, je dostatečně ilustrativní. Zde je uvedeno nastavení v souboru `.bashrc`:

```
export PATH=${PATH}:.:${HOME}/bin:/bin:/usr/bin:/usr/local/bin:/usr
  /bin/X1:/usr/TeX/bin
```

Zde jsem využil skutečnosti, že proměnná `HOME` je nastavena před tím, než příkazový procesor `bash` přečte můj soubor `.bashrc`. Systémová proměnná `PATH` je tedy nastavena prostřednictvím systémové proměnné `HOME`. Složené závorky („`{...}`") představují další úroveň uvozovek. Oddělují to, co se vyhodnotí po znaku `$`, proto bude příkazový procesor moci jednoznačně identifikovat následující část příkazu (tedy „`/bin`“ v tomto případě). Zde uvádíme další příklad efektu, který vyvolají složené závorky:

```
/home/larry# echo ${HOME}foo
/home/larryfoo
/home/larry#
```

Bez uvedení složených závorek se nevypíše nic, protože neexistuje proměnná prostředí `HOMEfoo`:

```
/home/larry# echo $HOMEfoo
/home/larry#
```

2 Pamatujte si, že program z aktuálního adresáře můžete vždy spustit explicitně, tedy například „`./foo`“.

Nyní se podívejme na další zvláštní konstrukci v uvedeném příkazu. Jaký je význam řetězce „$PATH"? Tento řetězec zařadí do proměnné prostředí PATH hodnotu proměnné prostředí PATH dříve definovanou. Kde byla nastavena původní hodnota? Soubor /etc/profile slouží jako jistý typ globálního souboru .bash_profile, kde se nacházejí nastavení společná pro všechny uživatele systému. Jeden soubor s globálním nastavením má jistou výhodu. Má-li například systémový správce přidat do proměnné prostředí PATH důležitou cestu, stačí, když to udělá v souboru s globální platností a nemusí opravovat inicializační soubory všech uživatelů. Proto je proměnná prostředí PATH již nastavena a vy ji můžete použít.

Prostřednictvím jisté proměnné prostředí můžete také specifikovat, jak má vypadat váš příkazový řádek. Tato proměnná prostředí má označení jako PS1. Předpokládejme, že dáváte přednost tomu, aby se stále zobrazovala cesta do aktuálního adresáře. Pak nastavte proměnnou prostředí PS1 takto:

```
export PS1='$PWD#'
```

Jak asi tušíte, jsou zde ve skutečnosti použity dvě proměnné prostředí. První, která se nastavuje, je PS1 a druhou je PWD. Proměnná prostředí PWD může znamenat buď „Print Working Directory", nebo „PATH to Working Directory". Vyhodnocení proměnné prostředí však probíhá uvnitř jednoduchých uvozovek. Jednoduché uvozovky slouží k vyhodnocení vnitřního výrazu. Výsledkem tohoto vyhodnocení je proměnná prostředí PWD. Kdybyste použili výraz export PS1=$PWD, pak by se vám stále zobrazoval ten adresář, který byl aktuální v době vyhodnocení tohoto výrazu. Trochu jsme pohled na vyhodnocování proměnných prostředí zkomplikovali, ale to není tak důležité. Pouze si pamatujte, že budete-li chtít zobrazovat v příkazovém řádku aktuální adresář, budete muset použít jednoduché uvozovky.

Možná, že budete chtít podobný příkazový řádek jako v operačním systému MS-DOS. Pak zadejte export PS1='$PWD>'. Chcete-li mít v příkazovém řádku jméno vašeho systému, zadejte PS1='hostname''>'.

V posledním příkladu jsme použili nový typ uvozovek, tzv. zpětné uvozovky. Tyto uvozovky ve skutečnosti nic nechrání. Výraz uzavřený ve zpětných uvozovkách se vyhodnotí jako příkaz a výstup se objeví na místě zpětných uvozovek.

Vyzkoušejte si příkazy echo `ls` nebo wc `ls`. Čím více budete seznámeni s příkazovým procesorem, tím užitečnější vám budou uvedené techniky.

V souboru .bashrc je mnoho dalších možností, jak konfigurovat váš příkazový procesor, ale zde není dostatek prostoru všechny prodiskutovat. Další podrobnosti si nastudujte v manuálových stránkách k příkazovému procesoru bash nebo se zeptejte zkušených uživatelů. Dále uvádíme kompletní soubor .bashrc, abyste si jej mohli nastudovat. Představuje typický standard, i když je proměnná prostředí PATH poněkud dlouhá.

```
# some random stuff:
ulimit -c unlimited
export history_control=ignoredups
export PS1='$PWD>'
umask 022
# application-specific PATHs:
export MANPATH=/usr/local/man:/usr/man
export INFOPATH=/usr/local/info
export PGPPATH=${HOME}/.pgp
# make the main PATH:
homepath=${HOME}:~/bin
stdpath=/bin:/usr/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc:/urs
  /games
pubpath=/usr/public/bin:/usr/gnusoft/bin:/usr/local/contribs/bin
softpath=/usr/bin/X11:/usr/local/bin/X11:/usr/TeX/bin
export PATH=.:${homepath}:${stdpath}:${pubpath}:${softpath}
# Technically, the curly braces were not necessary, because the
  colons
# were valid delimiters; nevertheless, the curly braces are a good
# habit to get into, and they can't hurt.
# aliases
alias ls="ls -CF"
alias fg1="fg %1"
alias fg2="fg %2"
alias tba="talk sussman@tern.mcs.anl.gov"
alias tko="take kold@cs.oberlin.edu"
alias tji="talk jimb@totoro.bio.indiana.edu"
alias mroe="more"
alias moer="more"
alias ll="ls -l"
alias la="ls -a"
alias ro="rm *~; rm.*~"
alias rd="rmdir"
alias pu=pushd
alias po=popd
alias ds=dirs
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
```

```
alias tg="telnet wobat.gnu.ai.mit.edu"
alias email="emacs -f rmail"
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
function gco
{
gcc -o $1 $1.c -g
}
```

9.2 Inicializační soubory systému X Window

Většina lidí dává při své práci přednost grafickému uživatelskému prostředí, kterým je v opeačním systému Unix systém X Window. Jestliže jste seznámeni s operačním systémem Macintosh nebo Microsoft Windows, pak vám systém X Window bude připadat velmi známý. Méně známé vám však budou připadat konfigurační možnosti, které systém X Window nabízí.

V případě operačního systému Macintosh nebo Microsoft Windows se konfigurace realizuje přímo v grafickém uživatelském prostředí. Když chcete například změnit barvu pozadí, klepnete myší na novou barvu nějakého speciálního inicializačního grafického programu. V systému X Window se implicitní hodnoty řídí textovými soubory, které můžete přímo editovat - jinými slovy, chcete-li například zadat novou barvu pozadí, musíte její jméno uvést v jistém inicializačním souboru.

Nikdo nepopírá, že inicializační metody v systému X Window jsou poněkud těžkopádnější než u komerčních programů. Domnívám se, že tendence zachovávat textově orientované inicializační metody dokonce i v grafickém uživatelském prostředí tkví v tom, že například systém X Window vytvořila poměrně nesourodá skupina programátorů, kteří až příliš lpí na tradicích dodržovaných v operačních systémech typu Unix. Tyto tendence se mohou v příštích verzích systému X Window změnit (alespoň doufám, že se změní), ale nyní se budeme zabývat inicializací prostřednictvím textových souborů. Tak alespoň máte k dispozici velmi flexibilní a přesnou kontrolu nad konfigurací.

Nejdůležitější konfigurační soubory pro systém X Window jsou tyto:

`.xinitrc`	Skript, který systém X Window spouští ve fázi startování.
`.twmrc`	Soubor, který čte správce oken `twm`.
`.fvwmrc`	Soubor, který čte správce oken `fvwm`.

Všechny uvedené soubory by měly být uloženy ve vašem domovském adresáři.

Soubor `.xinitrc` je jednoduchý skript příkazového procesoru, jenž se automaticky spouští ve fázi startování systému X Window. Může dělat vše, co mohou dělat ostatní skripty, avšak nejvíce ze všeho má smysl jej použít k nastartování různých programů systému X Window a k nastavení parametrů. Posledním příkazem v souboru .xinitrc je obvykle příkaz pro spuštění správce oken, například `/usr/bin/X11/twm`.

Jaký druh konfiguračních nastavení má smysl v souboru `.xinitrc` uvádět? Snad nějaká volání programu `xsetroot`, čímž si můžete nastavit pozadí okna a vlastnosti kurzoru. Dále volání programu `xmodmap`, jenž předá serveru[3] informace o tom, jak má interpretovat signály z vaší klávesnice. Všechny ostatní programy, které se mají spustit pokaždé spolu se systémem X Window (například `xclock`).

Zde uvádím některé řádky z mého souboru `.xinitrc`. Váš konfigurační soubor bude jistě vypadat jinak, proto je považujte za pouhý příklad.

```
#!/bin/sh
# The first line tells the operating system which shell to use in
# interpreting this script. The script itself ought to be marked as
# executable; you can make it so with "chmod +x ~/.xinitrc".
# xmodmap is a program for telling the X server how to interpret your
# keyboard's signals. It is *definitely* worth learning out. You
# can do "man xmodmap", "xmodmap -help", "xmodmap -grammar", and more.
# I don't guarantee that the expressions below will mean anything on
# your system (I don't even guarantee that they mean anything on
# mine):
xmodmap -e 'clear Lock'
xmodmap -e 'keycode 176 = Control_R'
xmodmap -e 'add control = Control_R'
xmodmap -e 'clear Mod2'
xmodmap -e 'add Mod1 = Alt_L Alt_R'
# xset is a program for setting other parameters of the X server:
xset m 3 2 & # mouse parameters
xset s 600 5 & # screen saver prefs
xset s noblank # ditto
xset fp+ /home/larry/x/fonts # for cxterm
# To find out more, do "xset -help"
```

[3] Server představuje hlavní proces systému X Window, tedy program, se kterým musejí všechny ostatní programy běžící pod systémem X Window komunikovat, pokud chtějí využívat grafické prostředí. Tyto ostatní programy se nazývají klienti a systém jako celek bývá označován termínem systém „klient-server“.

```
# Tell the X server to superimpose fish.cursor over fish.mask, and use
# the resulting pattern as my mouse cursor:
xsetroot -cursor /home/lab/larry/x/fish.cursor
/home/lab/larry/x/fish.mask &
# a pleasing background pattern and color:
xsetroot -bitmap /home/lab/larry/x/pyramid.xbm -bg tan
# todo: xrdb here? What about .Xdefaults file?
# You should do "man xsetroot", or "xsetroot -help" for
# more information on the program above.
# A client program, the imposing circular color-clock by Jim Blandy:
/usr/local/bin/circles
# Maybe you'd like to have a clock on your screen at all time:
/usr/bin/X11/xclock -digital &
# Allow client program running at occs.cs.oberlin.edu to display
# themselves here, do the same thing for juju.mcs.anl.gov:
xhost occs.cs.oberlin.edu
xhost juju.mcs.anl.gov
# You can simply tell the X server to allow clients running on any
# other host (a host being a remote machine) to display here, but this
# is a security hole -- those clients can be run by someone else,
# and watch your keystrokes as you type your password or something!
# However, if you wanted to do it anyway, you could use a "+" to stand
# for all possible hostnames, instead of a specific hostname, like
# this:
# xhost +
# And finally, run the window namager:
/usr/bin/X11/twm
# Some people prefer other window manager. I use twm, but fvwm is
# often distributed with Linux too:
# /usr/bin/X11/fvwm
```

Všimněte si, že některé programy běží na pozadí. Jsou to ty programy, jejichž zadání je ukončeno znakem `&`. Jiné programy se na pozadí nespouštějí. Rozdíl spočívá v tom, že se startují současně se systémem X Window a běží na pozadí, dokud systém X Window neukončíte. Jiné se vykonají bezprostředně a ihned skončí - jedním z nich je `xsetroot`, který pouze nastaví základní okno a chování kurzoru a pak skončí.

Jakmile se nastartuje správce oken, načte svůj vlastní inicializační soubor. Tento inicializační soubor řídí takové věci, jako je nastavení nabídek, pozice oken, řízení ikon a další. Pokud používáte jako správce oken program `twm`, pak tímto inicializačním souborem je soubor `.twmrc`, který se nachází ve vašem domovském adresáři. Jestliže však používáte `fvwm`, pak je inicializačním souborem soubor `.fvwmrc`. V následujících oddílech se budeme zabývat pouze těmito dvěma, protože se běžně distribuují s operačním systémem Linux.

9.2.1 Konfigurace programu twm

Soubor `.twmrc` není skript příkazového procesoru – je napsán v jazyku speciálně vyvinutém pro program `twm`.[4] Při konfiguraci prostřednictvím souboru `.twmrc` si uživatelé nejraději hrají s nastavením oken (barvy a podobně) a nabídek. Zde uvádíme příklad konfiguračního souboru `.twmrc`:

```
# Set colors for various parts of windows. This has a great
# impact no the "feel" of your environment.
Color
{
BorderColor "OrangeRed"
BorderTitleForeground "Black"
BorderTitleBackground "Black"
TitleForeground "black"
TitleBackground "gold"
MenuForeground "black"
MenuBackground "LightGrey"
MenuTitleForeground "LightGrey"
MenuTitleBackground "LightSlateGrey"
MenuShadowColor "black"
IconForeground "DimGray"
IconBackground "Gold"
IconBorderColor "OrangeRed"
IconManagerForeground "black"
IconManagerBacground "honeydew"
}
```

[4] Toto je jedna z odpuzujících vlastností inicializačních souborů: některé z nich mají svůj vlastní příkazový jazyk. To znamená, že uživatel musí být velmi zdatný a rychle se naučit další jazyk. Předpokládám, že takové vymyšlenosti mohly být zajímavé v ranných dobách operačního systému Unix, kdy programátoři stále toužili po něčem novém. Dnes je ale vyčerpávající učit se stále dokola nové a nové syntaxe jazyků, které konec konců slouží jedinému programu.

```
# I hope you don't have a monochrome system, but if you do...
Monochrome
{
BorderColor "black"
BorderTitleForeground "black"
BorderTitleBackground "white"
TitleForeground "black"
TitleBackground "white"
}
# I created beifang.bmp with the program "bitmap". Here I tell twm to
# use it as the default highlight pattern on windows' title bars:
Pixmaps
{
TitleHighlight "/home/larry/x/beifang.bmp"
}
# Don't worry about this stuff, it's only for power users :-)
BorderWidth 2
TitleFont "-adobe-new century schoolbook-bold-r-normal--14-140-75-75
   -p-87-iso8859-1"
MenuFont "6x13"
IconFont "lucidasans-italic-14"
ResizeFont "fixed"
Zoom 50
RandomPlacement
# These programs will not get a window titlebar by default:
NoTitle
{
"stamp"
"xload"
"xclock"
"xlogo"
"xbiff"
"xeyes"
"oclock"
"xoid"
}
# "AutoRaise" means that a window is brought to the front whenever the
# mouse pointer enters it. I find this annoying, so I have turned
```

```
# off. As you can see, I inherited my .twmrc from people who also
# did not like autoraise.
AutoRaise
{
"nothing" # I don't like auto-raise # Me either # nor I
}
# Here is where the mouse button functions are defined. Notice the
# pattern: a mouse button pressed on the root window, with no modifier
# key being pressed, always brings up a menu. Other locations usually
# result in window manipulation of some kind, and modifier keys are
# used in conjuction with mouse buttons to get at the more
# sophisticated window manipulations.
#
# You don't have to follow this pattern in your own .twmrc -- it's
# entirely up to you how you arrange your environment.
# Button = KEYS : CONTEXT : FUNCTION
# ----------------------------------
Button1 = : root : f.menu "main"
Button1 = : title : f.raise
Button1 = : frame : f.raise
Button1 = : icon : f.iconify
Button1 = m : window : f.iconify
Button2 = : root : f.menu "stuff"
Button2 = : icon : f.move
Button2 = m : window : f.move
Button2 = : title : f.move
Button2 = : frame : f.move
Button2 = s : frame : f.zoom
Button2 = s : window : f.zoom
Button3 = : root : f.menu "z"
Button3 = : title : f.lower
Button3 = : frame : f.lower
Button3 = : icon : f.raiselower
# You can write your own functions; this one gets used in the menu
# "windowops" near the end of this file:
Function "raise-n-focus"
{
f.raise
```

```
f.focus
}
# Okay, below are the actual menus reffered to in the mouse button
# section. Note that many of these menu entries themselves call
# sub-menus. You can have as many levels of menus as you want, but be
# aware that recursive menus don't work. I tried it.
menu "main"
{
"Vanilla" f.title
"Emacs" f.menu "emacs"
"Logins" f.menu "logins"
"Xlock" f.menu "xlock"
"Misc" f.menu "misc"
}
# This allows me to invoke emacs on several different machines. See
# the section on .rhosts files for more information about how this
# works:
{
"Emacs" f.title
"here" !"/usr/bin/emacs &"
"" f.nop
"phylo" !"rsh phylo \"emacs -d floss:0\" &"
"geta" !"rsh geta \"emacs -d floss:0\" &"
"darwin" !"rsh darwin \"emacs -d floss:0\" &"
"ninja" !"rsh ninja \"emacs -d floss:0\" &"
"indy" !"rsh indy \"emacs -d floss:0\" &"
"oberlin" !"rsh cs.oberlin.edu \"emacs -d floss.life.uiuc.edu:0\" &"
"gnu" !"rsh gate-1.gnu.ai.mit.edu \"emacs -d floss.life.uiuc.edu:0\" &"
}
# This allows me to invoke xterms on several different machines. See
# the section on .rhosts files for more information about how this
# work:
menu "logins"
{
"Logins" f.title
"here" !"/usr/bin/X11/xterm -ls -T 'hostname' -n 'hostname' &"
"phylo" !"rsh phylo \"xterm -ls -display floss:0 -T phylo\" &"
"geta" !"rsh geta \"xterm -ls -display floss:0 -T geta\" &"
```

```
"darwin" !"rsh darwin \"xterm -ls -display floss:0 -T darwin\" &"
"ninja" !"rsh ninja \"xterm -ls -display floss:0 -T ninja\" &"
"indy" !"rsh indy \"xterm -ls -display floss:0 -T indy\" &"
}
# The xlock screensaver, called with various options (each of which
# gives a different pretty picture):
menu "xlock"
{
"Help" !"xlock -mode hop &"
"Qix" !"xlock -mode qix &"
"Flame" !"xlock -mode flame &"
"Worm" !"xlock -mode worm &"
"Swarm" !"xlock -mode swarm &"
"Hop NL" !"xlock -mode hop -nolock &"
"Qix NL" !"xlock -mode qix -nolock &"
"Flame NL" !"xlock -mode flame -nolock &"
"Worm NL" !"xlock -mode worm -nolock &"
"Swarm NL" !"xlock -mode swarm -nolock &"
}
# Miscellaneous programs I run occassionally:
menu "misc"
{
"Xload" !"/usr/bin/X11/xload &"
"XV" !"/usr/bin/X11/xv &"
"Bitmap" !"/usr/bin/X11/bitmap &"
"Tetris" !"/usr/bin/X11/xtetris &"
"Hextris" !"/usr/bin/X11/xhextris &"
"XRoach" !"/usr/bin/X11/xroach &"
"Analog Clock" !"/usr/bin/X11/xclock -analog &"
"Digital Clock" !"/usr/bin/X11/xclock -digital &"
}
# This is the one I bound to the middle mouse button:
menu "stuff"
{
"Chores" f.title
"Sync" !"/bin/sync"
"Who" !"who | xmessage -file - -columns 80 -lines 24 &"
"Xhost +" !"/usr/bin/X11/xhost + &"
```

```
"Rootclear" !"/home/larry/bin/rootclear &"
}
# X functions that are sometimes convenient
menu "x"
{
"X Stuff" f.title
"Xhost +" !"xhost + &"
"Refresh" f.refresh
"Source .twmrc" f.twmrc
"(De)Iconify" f.iconify
"Move Window" f.move
"Resize Window" f.resize
"Destroy Window" f.destroy
"Window Ops" f.menu "windowops"
"" f.nop
"Kill twm" f.quit
}
# This is submenu from above:
menu "windowops"
{
"Window Ops" f.title
"Show Icon Mgr" f.showiconmgr
"Hide Icon Mgr" f.hideiconmgr
"Refresh" f.refresh
"Refresh Window" f.winrefresh
"twm version" f.version
"Focus on Root" f.unfocus
"Source .twmrc" f.twmrc
"Cut File" f.cutfile
"(De)Iconify" f.iconify
"DeIconify" f.deiconify
"Move Window" f.move
"ForceMove Window" f.forcemove
"Resize Window" f.resize
"Raise Window" f.raise
"Lower Window" f.lower
"Raise or Lower" f.raiselower
"Focus on Window" f.focus
```

```
"Raise-n-Focus" f.function "raise-n-focus"
"Destroy Window" f.destroy
"Kill twm" f.quit
}
```

Věřte mi, že to není zdaleka nejobsáhlejší soubor `.twmrc`, jaký jsem kdy viděl. Je vysoce pravděpodobné, že nějaký příklad souboru .twmrc bude obsažen ve vaší distribuci systému X Window. Prohledejte adresář `/usr/lib/X11/twm/` nebo `/usr/X11/lib/X11/twm`. Zde byste měli uvedený soubor nalézt.

Často dělají uživatelé chybu a zapomínají uvádět znak `&` na konec příkazů. Pokud systém X Window „zatuhne“ právě když spustíte nějaký příkaz, pak pravděpodobně tkví příčina v tom, že jste zapomněli uvést znak `&`. V takovém případě ukončete systém X Window kombinací kláves Ctrl-Alt-Backspace opravte soubor `.twmrc` a spusťte systém X Window znovu.

9.2.1 Konfigurace programu fvwm

Pokud používáte jako správce oken program `fvwm`, prohledejte tento adresář:

`/usr/lib/X11/fvwm/ nebo /usr/X11/lib/X11/fvwm.`

Zde najdete nějaké příklady konfiguračních souborů.

Poznámka: O programu `fvwm` nic nevím. Asi bych byl schopen něco vyčíst z příkladů konfiguračních souborů, ale zůstal bych pouze čtenářem a těžko bych dokázal něco vysvětlovat. Zájemcům o program fvwm a jeho konfiguraci doporučuji přečíst si příslušné manuálové stránky a prostudovat příklady konfiguračních souborů nacházejících se ve výše zmíněných adresářích.

9.3 Ostatní inicializační soubory

Za zmínku stojí následující inicializační soubory:

`.emacs`	Inicializační soubor editoru Emacs. Editor jej čte ve fázi startování.
`.netrc`	Soubor obsahující implicitní jména a hesla pro `ftp`.
`.rhosts`	Zpřístupňuje váš účet vzdáleným systémům.
`.forward`	Inicializační soubor pro automatické přesměrování elektronické pošty.

9.3.1 Inicializační soubor pro editor Emacs

Pokud používáte editor Emacs jako primární editor, pak má pro vás soubor `.emacs` mimořádný význam. Podrobně jsme jej popsali v kapitole 8.

9.3.2 Implicitní nastavení pro FTP

V souboru `.netrc` můžete mít uložena některá implicitní nastavení pro `ftp`. Následující řádky obsahují příklad takového souboru:

```
machine floss.life.uiuc.edu login larry password fishSticks
machine darwin.life.uiuc.edu login larry password fishSticks
machine geta.life.uiuc.edu login larry password fishSticks
machine phylo.life.uiuc.edu login larry password fishSticks
machine ninja.life.uiuc.edu login larry password fishSticks
machine indy.life.uiuc.edu login larry password fishSticks

machine clone.mcs.anl.gov login fogel password doorm@
machine osprey.mcs.anl.gov login fogel password doorm@
machine tern.mcs.anl.gov login fogel password doorm@
machine altair.mcs.anl.gov login fogel password doorm@
machine dalek.mcs.anl.gov login fogel password doorm@
machine juju.mcs.anl.gov login fogel password doorm@
machine sunsite.unc.edu login anonymous password
larry@cs.oberlin.edu
```

Každý řádek souboru `.netrc` specifikuje jméno počítače, přihlašovací jméno, které se má použít jako implicitní pro tento počítač, a heslo. Uvedená nastavení vám ušetří velké množství času, protože jinak byste při přihlašování se k jednotlivým serverům `ftp` museli pokaždé zadávat dlouhé identifikační řetězce. Pokud se budete přihlašovat k serveru ftp, jehož jméno je uvedeno v souboru `.netrc`, pokusí se program `ftp` aplikovat uživatelské jméno a heslo z tohoto souboru.

Při spouštění programu `ftp` můžete pomocí parametru -n specifikovat, že nechcete použít implicitní nastavení ze souboru .netrc. Příkaz pak bude mít tvar „`ftp -n`“.

Musíte se ujistit, že soubor `.netrc` jste schopni číst pouze vy. K nastavení příslušných přístupových práv použijte program chmod. Kdyby soubor .netrc mohli číst jiní uživatelé, pak by mohli odhalit vaše hesla platná pro servery ftp uvedené v tomto souboru.

Takový případ by představoval značnou „bezpečnostní díru“. Naštěstí program `ftp` a ostatní programy, které čtou soubor `.netrc`, odmítnou pokračovat ve své činnosti, pokud zjistí, že přístupová práva k tomuto souboru nejsou v pořádku.

Další informace týkající se souboru .netrc si najděte v manuálových stránkách prostřednictvím příkazu „`man .netrc`“ nebo „`man ftp`“.

9.3.3 Povolení snadného vzdáleného přístupu k vašemu účtu*

Jestliže se ve vašem domovském adresáři nachází soubor `.rhosts`, pak lze ze vzdálených počítačů spouštět aplikace na vašem počítači. Uveďme si příklad. Předpokládejme, že jste přihlášeni na počítači cs.oberlin.edu. Na počítači `floss.life.uiuc.edu` je správně konfigurován soubor `.rhosts`.

Pak ze svého počítače můžete na počítači `floss.life.uiuc.edu` spustit aplikaci, jejíž výstup bude přesměrován na vaši obrazovku, a dokonce se před tím nebudete muset přihlašovat a zadávat heslo.

Soubor `.rhosts` může vypadat takto:

```
frobnozz.cs.knowledge.edu jsmith
aphrodite.classics.havhaahd.edu wphilps
frobbo.hoola.com trixie
```

Formát souboru je velmi jednoduchý: jméno počítače následované jménem uživatele. Předpokládejme nyní, že uvedený příklad je mým skutečným souborem .rhosts na vzdáleném počítači `floss.life.uiuc.edu`. To by znamenalo, že bych mohl spustit program na počítači floss a výstup by mohl být přesměrován na kterýkoliv počítač uvedený v tomto souboru, pokud bych byl přihlášen jako odpovídající uživatel.

Přesný mechanismus, prostřednictvím kterého uživatel uskutečňuje spouštění vzdáleného programu, je realizován programem rsh, jehož název je zkratkou pro „remote shell“, tedy „vzdálený příkazový procesor“. Ten nastartuje příkazový procesor na vzdáleném počítači a spustí specifikovaný program. Uveďme si příklad:

```
frobbo$ whoami
trixie
frobbo$ rsh floss.life.uiuc.edu "ls ~"
foo.txt mbox url.ps snax.txt
```

* Poznámka korektora: Používání programu `rsh` a `rlogin` je v současné době nahrazeno podobným programem ssh, který je bezpečnější, ale nemůže být standardní součástí distribucí Linuxu.

```
frobbo$ rsh floss.life.uiuc.edu "more ~/snax.txt"
[nyní se zobrazí stránky souboru snax.txt]
```

Uživatel trixie na počítači floss.life.uiuc.edu, který má soubor `.rhosts` uvedený v předcházejícím příkladě, umožňuje uživateli trixie na počítači `frobbo.hoola.com` spouštět programy z počítače floss.

Soubor `.rhosts` bude pracovat správně, i když nemáte na všech počítačích stejné uživatelské jméno. Chcete-li „sdělit“ vzdálenému počítači, jaké jméno uživatele chcete použít k přihlášení se, použijte při zadání příkazu `rsh` volbu „`-l`“. Pokud takový uživatel na vzdáleném počítači existuje a pokud zde existuje soubor `.rhosts` obsahující jméno vašeho (t.j. lokálního) počítače a jméno uživatele, pak se příkaz `rsh` provede úspěšně.

```
frobbo$ whoami
trixie
frobbo$ rsh -l larry floss.life.uiuc.edu "ls ~"
[zde se objeví seznam souborů mého adresáře na počítači floss]
```

Uvedený příklad bude fungovat, pokud má uživatel `larry` na počítači `floss.life.uiuc.edu` takový soubor `.rhosts`, jenž umožňuje uživateli `trixie` z počítače `frobbo.hoola.com` spouštět programy. Není relevantní, zda se jedná nebo nejedná o tutéž osobu. Důležitá jsou pouze jména uživatelů, jména počítačů a záznamy v souboru `.rhosts`.

V souboru .rhosts mohou být uvedeny i jiné kombinace - například jméno uživatele následující za jménem vzdáleného počítače se může vynechat a tak umožnit kterémukoliv uživateli na vzdáleném počítači spouštět programy na vašem počítači. To je však spojeno s jistými riziky. Nepovolaná osoba by mohla například zrušit vaše soubory. Pokud se rozhodnete pro takto organizovaný soubor `.rhosts`, pak byste se měli ujistit, že právo číst soubor .rhosts máte pouze vy.

9.3.4 Přesměrování elektronické pošty

Kromě jiných souborů můžete také mít soubor `.forward`, který nelze přímo nazvat „inicializačním“ souborem. Pokud tento soubor obsahuje adresu elektronické pošty, pak veškeré zprávy elektronické pošty budou odesílány na tuto adresu. Soubor je užitečný v případě, kdy máte účet na několika různých systémech, ale elektronickou poštu chcete číst pouze na jednom.

Na vašem počítači se mohou nacházet i jiné inicializační soubory. Jejich počet se liší podle operačního systému, jenž používáte, a také podle programů, které máte nainstalovány. Jeden ze způsobů, jak získat více informací o inicializačních souborech, spočívá v tom, že si prostudujete soubory ve vašem domovském adresáři začínající tečkou. Není sice garantováno, že všechny takové soubory jsou inicializační, ale převážná většina z nich určitě bude.

9.4 Kde si můžete prohlédnou některé příklady

Jestliže na svém počítači máte instalován operační systém Linux a jestliže máte přístup do sítě Internet, pak se můžete přihlásit prostřednictvím služby telnet do systému `floss.life.uiuc.edu`. Přihlašte se jako „guest“ a heslo uveďte jako „explorer“. Připravili jsme pro vás spoustu příkladů, z nichž většina je uložena v adresáři `/home/kfogel`. Tyto příklady si můžete zkopírovat a prostudovat. Buďte ale opatrní. Počítač floss nepředstavuje zcela zabezpečený systém a když se budete dostatečně snažit, podaří se vám získat přístupová práva uživatele root. Dali jsme přednost důvěře před neustálou ostražitostí a doufáme, že toho nikdo nezneužije.*

* Poznámka korektora: Autor knihy to zřejmě myslel jako vtip.

Kapitola

10

Komunikace s ostatními systémy

Moderní operační systémy typu Unix jsou dobře přizpůsobeny ke komunikaci s ostatními počítači, což znamená, že jsou vybaveny prostředky pro práci v síti. Dva různé počítače s operačním systémem Unix si mohou vyměňovat informace mnoha způsoby. Tato kapitola je věnována metodám, pomocí kterých budete moci komunikovat prostřednictvím sítě s ostatními počítači.

Budeme se zabývat elektronickou poštou, zájmovými skupinami a některými základními programy pro komunikaci.

10.1 Elektronická pošta

Mezi nejpopulárnější vlastnosti operačního systému Unix patří možnost odesílat a přijímat zprávy elektronické pošty. Máte-li k dispozici elektronickou poštu, nepotřebujete papír, inkoust a pero, obálky, známky a nesrovnatelně pomalejší poštovní službu.

10.1.1 Odesílání elektronické pošty

Potřebujete-li odeslat zprávu elektronickou poštou, stačí zadat příkaz `mail username` a pak zapsat vaši zprávu.

Předpokládejme například, že chcete odeslat zprávu elektronickou poštou uživateli jménem sam:

```
/home/larry# mail sam
Subject: The user documentation
```

```
Just testin out the mail system.
EOT
/home/larry#
```

Program `mail` je velmi jednoduchý. Podobně jako program cat čte text ze standardního vstupu řádek po řádku, dokud nenarazí na znak „konec textu“ jenž je reprezentován klávesou Ⓒ-Ⓓ. To znamená, že po dokončení textu zprávy stisknete Enter a pak Ⓒ-Ⓓ.

Příkaz `mail` představuje nejrychlejší způsob, jak odeslat zprávu elektronické pošty, a je užitečný zejména ve spojení s prostředky pro přesměrování vstupu/výstupu či rourami. Jestliže chcete například odeslat soubor `report1` uživateli jménem „Sam“, můžete použít příkaz `mail sam < report1` nebo dokonce „`sort report1 | mail sam`“.

Používání příkazu `mail` je však spojeno s velkou nevýhodou. Jestliže uděláte v nějakém řádku chybu, pak ji již nemůžete opravit. Proto vám doporučuji (pokud nepoužijete možnost s přesměrováním vstupu/výstupu) k odesílání elektronické pošty používat editor Emacs. Postup jsme popsali v oddílu 8.10.

10.1.2 Čtení zpráv elektronické pošty

`mail [`*user*`]`

Program `mail` poskytuje poněkud těžkopádnou možnost číst zprávy elektronické pošty. Zadáte-li příkaz `mail` bez jakýchkoliv parametrů, objeví se vám následující hlášení:

```
/home/larry# mail
No mail for larry
/home/larry#
```

Předpokládejme, že chcete odeslat zprávu elektronickou poštou sami sobě, abyste si mohli vyzkoušet způsoby jejího čtení:

```
/home/larry# mail larry
Subject: Frogs!
and toads!
EOT
/home/larry# echo "snakes" | mail larry
/home/larry# mail
Mail version 5.5. 6/1/90 Type ? for help.
"/usr/spool/mail/larry" 2 messages new
```

```
>N 1 larry Tue Aug 30 18:11 10/211 "Frogs!"
N 2 larry Tue Aug 30 18:12 10/211
&
```

Příkazový řádek uvnitř programu pro čtení elektronické pošty je uvozen znakem ampersand („&"). Umožňuje specifikovat spoustu jednoduchých příkazů a po zadání znaku ? a enter zobrazuje stručnou nápovědu.

Základními příkazy pro práci s programem mail jsou:

`t` *message-list*	Na obrazovce se zobrazí zprávy uvedené v seznamu *message-list*.
`d` *message-list*	Zprávy uvedené v seznamu *message-list* se zruší.
`s` *message-list file*	Zprávy uvedené v seznamu *message-list* se uloží do souboru *file*.
`r` *message-list*	Odpověď na zprávy - program `mail` zahájí proces sestavování nových zpráv, které budou odeslány každému, kdo vám odeslal zprávu uvedenou v seznamu *message-list*.
`q`	Program `mail` se ukončí a uloží každou zprávu, která nebyla zrušena příkazem `d` do souboru mbox ve vašem domovském adresáři.

Jak vypadá seznam *message-list*? Skládá se ze seznamu čísel oddělených mezerami, nebo může být vyjádřen ve formě intervalu, tedy například 2-4 (což znamená „2 3 4"). Také můžete uvést uživatelské jméno odesílatele. Například `t sam` zobrazí všechny zprávy odeslané uživatelem sam. Jestliže se seznam *message-list* neuvede, zobrazí se vždy poslední zpráva.

Se čtením zpráv elektronické pošty je spojeno několik problémů. Především platí, že pokud má zpráva více řádků, než se vejde na obrazovku, program mail se nezastaví! Takou zprávu budete muset uložit do souboru a pak si ji prohlédnout prostřednictvím příkazu `more`. Dále program mail nemá dobré prostředky pro čtení starých zpráv, tedy zpráv uložených do souborů.

Editor Emacs má prostředek pro čtení zpráv elektronické pošty, jenž se jmenuje `rmail`. V této knize se však programem rmail nebudeme zabývat. V operačním systému Linux jsou navíc k dispozici další programy pro čtení elektronické pošty, jako je `elm` nebo .*

* Poznámka korektora: Samozřejmě můžete používat i jiné poštovní klienty, např. i Netscape Communicator.

10.2 Jak vyhledat uživatele sítě

10.2.1 Příkaz finger

Příkaz `finger` vám umožní získat informace o ostatních uživatelích vašeho systému nebo o uživatelích sítě Internet. Jméno příkazu nepochybně vzniklo jako zkratka s reklamním podtextem ve firmě AT&T.

`finger [-slpm] [`*user*`][@`*machine*`]`

Volitelné parametry v příkazu `finger` mohou být mírně matoucí. Pomocí příkazu `finger` můžete získat informace o lokálním uživateli (například „sam“), o jiném počítači (například „@lionsden“), informace o uživateli vzdáleného počítače (například „sam@liondsen“) nebo informace o lokálním počítači (neuvede se žádný parametr).

Příkaz finger má další zajímavou vlastnost. Pokud se pokusíte získat informace o uživateli, jehož jméno přesně neznáte, pokusí se příkaz `finger` najít jméno sám (zkouší různé kombinace založené na původně zadaném jménu). To znamená, že když například zadám příkaz `finger Greenfield`, obdržím zprávu, že účet `sam` existuje pro jméno `Sam Greenfield`.

```
/home/larry# finger sam
Login: sam Name: Sam Greenfield
Directory: /home/sam Shell: /bin/tcsh
Last login Sun Dec 25 14:47 (EST) on tty2
No Plan.
/home/larry# finger greenfie@gauss.rutgers.edu
[gauss.rudgers.edu]
Login name: greenfie In real life: Greenfie
Directory: /gauss/u1/greefie Shell: /bin/tcsh
On since Dec 25 15:19:41 on ttyp0 from tiptop-slip-6439
13 minutes Idle Time
No unread mail
Project: You must be joking!
No Plan.
/home/larry# finger
Login Name Tty Idle Login Time Office Office Phone
larry Larry Greenfield 1 3:51 Dec 25 12:50
larry Larry Greenfield p0 Dec 25 12:51
/home/larry#
```

Použijete-li u příkazu `finger` volbu `-s`, pak se vám vždy zobrazí stručný výpis (stejný, který obdržíte, když program `finger` použijete k získání informací o počítači) a pokud použijete volbu `-l`, zobrazí se vám vždy kompletní výpis (i tehdy, když program `finger` použijete k získání informací o počítači). Po zadání volby `-p` se nezobrazí informace ze souborů `.forward`, `.plan` a `.project`. Zadáte-li volbu `-m` a žádáte-li pouze informace o uživateli, pak se vám zobrazí pouze přihlašovací jméno.

10.2.2 Soubory .plan a .project

Nyní bychom měli vysvětlit, jaký je význam souborů `.plan` a `.project`. Jedná se o soubory, které jsou uloženy v domovském adresáři uživatele a jejichž obsah se zobrazí pokaždé, když je na daného uživatele aplikován program `finger`. Soubory `.plan` a `.project` si můžete vytvořit sami - jediné omezení spočívá v tom, že ze souboru .project se zobrazuje pouze první řádek.

Dále platí, že každý, kdo chce aplikovat program `finger` musí mít možnost procházet vaším domovským adresářem (`chmod a+x ~/`) a každý musí být schopen číst soubory `.plan` a `.project` (`chmod a+r ~/.plan ~/.project`).

10.3 Používání systémů vzdálenými počítači

`telnet` *vzdálený systém*

Hlavní prostředek pro využívání vzdálených systémů založených na operačním systému Unix představuje program Telnet. (V současné době se spíše používá program `ssh`, který na rozdíl od příkazu `telnet` umožňuje šifrovanou komunikaci se vzdáleným systémem.) Používání programu telnet je velmi jednoduché:

```
/home/larry# telnet lionsden
Trying 128.2.36.41...
Connected to lionsden
Escape character is '^]'.
lionsden login:
```

Jak vidíte z následujícího příkladu, po zadání příkazu `telnet` budete vyzváni k přihlášení se ke vzdálenému systému. Uživatelské jméno lze zadat jakékoliv (ovšem heslo musíte znát správné) a pak je vám vzdálený systém k dispozici téměř stejně jako váš lokální systém.

Normální způsob ukončení programu `Telnet` spočívá v odhlášení se, ale je také možnost zadat znak Escape, kterým je zpravidla Ctrl-]. Tak obdržíte nový příkazový řádek uvozený řetězcem `telnet>`. Nyní stačí napsat `quit` a pak stisknout klávesu Enter - spojení se přeruší a program `Telnet` se ukončí. Pokud si to rozmyslíte a nechcete relaci ukončit, stiskněte pouze klávesu Enter.

Pokud jste uživateli systému X Window, pak si pro komunikaci se vzdáleným uživatelem otevřete nové terminálové okno - například prostřednictvím příkazu „`xterm -title "lionsden" -e telnet lionsden &`“. Uvedený příkaz otevře nové terminálové okno, ve kterém automaticky poběží program `Telnet`. Jestliže takový příkaz budete používat častěji, pak doporučujeme, abyste si pro něj vytvořili alias.

10.4 Přenášení souborů

`ftp` *vzdálený systém*

Normální způsob přenášení souborů zprostředkovává v operačním systému Unix program `ftp`, což je zkratka pro „**file transfer protocol**“. Po zadání příkazu `ftp` budete vyzváni, abyste se přihlásili ke vzdálenému systému téměř stejným způsobem, jako v případě programu `telnet`. Pak se vám zobrazí speciální příkazový řádek.

Nyní máte k dispozici několik příkazů běžných v operačním systému Unix, jež můžete aplikovat v příkazovém řádku programu `ftp`. Například příkaz `cd` i zde slouží k přepínání se mezi adresáři a příkaz `ls` slouží k zobrazení seznamu souborů uložených v aktuálním adresáři.

Navíc máte k dispozici dva důležité příkazy: `get` a `put`. Příkaz `get` je určen k přenosu souborů ze vzdáleného počítače do vašeho lokálního počítače a příkaz `put` slouží k opačnému úkolu. Oba příkazy jako implicitní používají váš domovský adresář a lokální adresář na vzdáleném počítači (který můžete měnit prostřednictvím příkazu `cd`).

S používáním programu `ftp` je spojen jeden problém. Spočívá v rozlišení mezi znakovými a binárními soubory. Protokol pro přenos dat programem `ftp` je velmi starý a implicitně předpokládá, že přenášené soubory jsou textové. Aplikuje-li se tento implicitní přenos na binární soubory, pak budou s největší pravděpodobností po přenosu poškozeny. Před přenosem binárního souboru proto používejte příkaz `binary`.

Program `ftp` ukončíte příkazem `bye`.

10.5 Putování po stránkách WWW

WWW, neboli World Wide Web (doslova „celosvětová pavučina“) zřejmě představuje nejpopulárnější způsob využívání Internetu. Skládá se ze stránek, z nichž každá je spojena s tzv. lokátorem URL (**uniform resource locator**). Lokátory URL jsou komické řetězce následujícího tvaru: `http://www.rutgers.edu/`. Stránky jsou vytvořeny v jazyku HTML (**hypertext markup language**).

Jazyk HTML umožňuje autorům stránek WWW začlenit do dokumentů odkazy na kterékoliv jiné stránky WWW (nebo obrázky) lokalizované kdekoliv jinde v celosvětovém systému WWW. Když uživatel čte nějaký dokument, pak se pouhým klepnutím na odkaz (prezentovaný klíčovým slovem nebo tlačítkem) přenese na jinou stránku WWW, která může být uložena v počítači na druhém konci světa.

`netscape [`*url*`]`

Nejpopulárnějším programem pro prohlížení stránek WWW je v operačním systému Linux program Netscape od firmy Netscape Corporation. Program Netscape může běžet pouze pod systémem X Window.

Program Netscape je velmi jednoduchý, pokud jde o ovládání. Je založen na knihovně Motif a připomíná program napsaný pro Microsoft Windows (samozřejmě, že pro Microsoft Windows také existuje verze programu Netscape). Odkazy na další stránky WWW jsou v programu Netscape zobrazeny modře. Stačí na odkaz klepnout levým tlačítkem myši a vzápětí se vám zobrazí nová stránka WWW.

Operační systém Linux podporuje i jiné programy pro prohlížení stránek WWW, například program lynx, což je textově orientovaný program, proto není schopen zobrazovat obrázky a jiné grafické prvky dokumentů WWW. Je ovšem schopen pracovat pod systémem X Window.

`lynx [`*url*`]`

Naučit se pracovat s programem `lynx` je poněkud obtížnější, než naučit se pracovat s programem `Netscape`. Při práci si budete muset zvyknout na používání kurzorových kláves. Klávesy „šipka nahoru“ a „šipka dolů“ jsou určeny k přepínání mezi odkazy na dané stránce WWW, klávesa „šipka doprava“ zobrazí novou stránku (na kterou odkazuje zvýrazněný odkaz) a klávesa „šipka doleva“ je určena k zobrazení předcházející stránky. K ukončení programu `lynx` použijte klávesu Q. Program `lynx` má mnohem více příkazů, jejichž kompletní popis najdete v manuálových stránkách.

Kapitola

11

Zábavné příkazy

Většina lidí, která má co do činění s operačním systémem Unix, asi nebude souhlasit s titulem této kapitoly. Někteří se dokonce rozčílí, protože si u každého příkazu musejí pamatovat až desítky parametrů a najednou se jim někdo snaží namluvit, že by používání takových příkazů mohlo být zábavné. V nadpisu této kapitoly se spíš odráží až sarkastický humor autorů operačního systému Unix. Dále uvedené příkazy totiž nemají ekvivalentní příkazy v operačním systému MS-DOS. Přitom jsou velmi výkonné, a když je zatvrzelí příznivci operačního systému MS-DOS chtějí používat, musejí si je koupit (speciální verze těchto příkazů pro MS-DOS byly dodatečně vytvořeny). Protože operační systém Unix odjakživa soupeří s operačními systémy od firmy Microsoft, jsou touto skutečností příznivci operačního systému Unix pobaveni.

V této kapitole se budeme zabývat příkazy `find` (příkaz pro vyhledávání skupin souborů v adresářové struktuře), `tar` (příkaz pro archivaci souborů nebo celých adresářů), `dd` (příkaz pro kopírování) a `sort` (příkaz pro třídění souborů). Předem je nutno upozornit na skutečnost, že tyto příkazy nejsou standardizovány. Zaměříme se na popis verzí náležejících do projektu GNU, protože právě tyto verze jsou implementovány v operačním systému Linux a právě tyto verze pravděpodobně budou (tak jako ostatní programy vytvořené v rámci projektu GNU) v blízké budoucnosti představovat standard. Používáte-li jiný operační systém typu Unix, pak nezapomeňte konfrontovat příslušné manuálové stránky.

11.1 Příkaz find

11.1.1 Všeobecné poznámky

Mezi doposud probranými příkazy jsou takové, které umožňují rekurzivní procházení stromovou strukturou adresářů. Příkladem jsou příkazy `ls -R` nebo `rm -R`. Příkaz `find` je také rekurzivní. Kdykoliv byste měli něco dělat s jistým typem souborů v adresáři a ve všech jeho podadresářích, vzpomeňte si na příkaz `find`. V jistém smyslu platí, že vyhledávání souborů příkazem `find` představuje spíše vedlejší efekt.

Základní struktura příkazu `find` je následující:

```
find cesta [...] výraz [...]
```

Uvedená syntaxe platí pouze pro verzi GNU. Ostatní verze neumožňují specifikovat více než jednu cestu (path), což z praktického hlediska není tak důležité. Hrubé vysvětlení funkce příkazu je následující: prostřednictvím prvního parametru zadáte, odkud má prohledávání začít (parametr path; u verze GNU nemusíte tento parametr vůbec uvádět a prohledávání pak začne od aktuálního adresáře), a druhý parametr (expression) určuje typ vyhledávání.

Standardní chování příkazu `find` je poněkud lstivé a stojí za to tomuto faktu věnovat trochu pozornosti. Předpokládejme, že vaším domovským adresářem je adresář garbage a že obsahuje soubor foobar. Řekněme, že náhodou napíšete příkaz `find .-name foobar`. Tento příkaz by měl podle všech předpokladů vyhledat soubory nazvané `foobar` - avšak nic se nestane. Potíž je v tom, že program `find` je tzv. tichý (silent) příkaz. Pouze vrátí 0 (ať už nějaký soubor najde nebo ne), nebo vrátí záporné číslo, pokud nastal nějaký problém. Uvedený případ nenastane, pokud používáte operační systém Linux, ale je dobré jej mít na paměti.

11.1.2 Výrazy

Výraz neboli parametr expression může být rozdělen do čtyř různých skupin klíčových slov: *volby*, *testy*, *akce* a *operátory*. Každé klíčové slovo může vracet hodnotu true nebo false a přitom může produkovat nějaký vedlejší efekt. Rozdíl mezi skupinami klíčových slov popisuje následující seznam:

volby celkově ovlivňují operaci vyhledávání a netýkají se zpracování souboru. Příkladem volby je `-follow`, která „sdělí“ příkazu `find`, aby procházel symbolické odkazy. Volby vždy vracejí hodnotu true.

testy jsou skutečnými testy. Například test `-empty` ověřuje, zda je soubor prázdný. Testy mohou vracet hodnotu true nebo false.

akce produkují jako vedlejší efekt jméno programu. Také mohou vracet hodnotu true nebo false.

operátory ve skutečnosti nevracejí žádnou hodnotu (dle konvence vracejí hodnotu true) a používají se ke skládání výrazů. Příkladem je operátor `-or`, jenž jako výsledek produkuje logickou operaci OR nad operandy, kterými jsou dva výrazy. Jsou-li vedle sebe uvedeny dva výrazy bez operátoru, pak se implicitně aplikuje operátor `-and`.

Poznamenejme, že program `find` spoléhá na syntaktickou analýzu příkazu, kterou realizuje příkazový procesor. To znamená, že všechna klíčová slova musejí být oddělena nezobrazitelnými znaky a zejména platí, že spousta znaků musí být oddělena znaky Escape - jinak by je příkazový procesor nemohl správně interpretovat. Jako znaky Escape mohou být použita obrácená lomítka, jednoduché nebo dvojité uvozovky. V později uvedených příkladech se jednoznaková klíčová slova uvozují obráceným lomítkem, protože je to nejjednodušší.

11.1.3 Volby

V následujícím seznamu uvádíme přehled všech voleb, které je schopen příkaz `find` akceptovat (připomínáme, že se jedná o verzi GNU). Nezapomeňte, že volby vždy vracejí hodnotu true.

- `-daystart`

  Volba `-daystart` měří dobu, která uplyne od poslední půlnoci. Počítačový nadšenec asi nemůže pochopit, proč má nějaký program takovou volbu, ale programátor, který pracuje od osmi do pěti, ji ocení.

- `-depth`

  Tato volba zajistí, že příkaz `find` bude zpracovávat obsah každého podadresáře před zpracováním adresáře samotného. Abych řekl pravdu, neumím si představit užitečné uplatnění této volby, kromě případu, kdy se emuluje příkaz `rm -F` (podadresáře nemůžete zrušit, dokud obsahují nějaké soubory).

- `-noleaf`

  Volba `-noleaf` vypíná optimalizaci, která indikuje, že adresář obsahuje o dva podadresáře méně, než by měl obsahovat. Kdyby byl svět dokonalý, pak by bylo možné odkazovat se na všechny adresáře z prostředí každého jejich podadresáře (pomocí volby `..`), pomocí odkazu `.` uvnitř samotného adresáře a prostřednictvím jeho skutečného jména z jeho nadřazeného adresáře.

To znamená, že na každý adresář musejí existovat alespoň dva odkazy (jeden z adresáře samotného a jeden z adresáře nadřazeného) a případně další odkazy ze všech podadresářů. V praxi se však může stát, že symbolické odkazy a distribuované souborové systémy mohou toto pravidlo porušit.[1]

- `-maxdepth` *levels*, `-mindepth` *levels*

  Parametry `levels` jsou nezáporná čísla, jež udávají maximální a minimální úroveň podadresářů, které má příkaz `find` prohledávat. Uveďme příklady: `-maxdepth 0` indikuje, že příkaz `find` má být realizován pouze pro argumenty uvedené v příkazovém řádku a že se žádné podadresáře prohledávat nemají. `-mindepth 1` zakazuje zpracování příkazu pro argumenty uvedené v příkazovém řádku, zatímco ostatní soubory v podadresářích budou zpracovány.

- `-version`

  Po zadání parametru `-version` se pouze vypíší informace o verzi programu find.

- `-xdev`

  Parametr `-xdev` má poněkud zavádějící označení. Pro program `find` předává informaci o tom, že se dané zařízení (tedy souborový systém) nemá prohledávat. Volba `-xdev` je velmi užitečná v případě, kdy se má vyhledávat něco v hlavním souborovém systému. Hlavní souborový systém většinou obsazuje malou diskovou oblast. Kdyby se použil příkaz `find /` bez parametru `-xdev`, pak by se prohledávala celá adresářová struktura!

11.1.4 Testy

První dva testy jsou velmi jednoduché: `-false` vždy vrací hodnotu false a `-true` vždy vrací hodnotu true. K dalším testům, které nevyžadují specifikaci hodnoty, patří test `-empty` (vrací hodnotu true, pokud je daný soubor prázdný) a dále dvojice `-nouser / -nogroup`, kdy test vrací hodnotu true, právě když se v souboru `/etc/passwd` nebo `/etc/group` nevyskytuje záznam identifikující vlastníka souboru jako uživatele nebo skupinu. Jedná se o postižení situace, kdy je v systému zrušen účet uživatele, ale soubory jím vlastněné jsou stále uloženy někde v souborovém systému. Podle Murphyho zákonů jsou takové soubory neobyčejně velké.

Samozřejmě je možné vyhledávat soubory vlastněné jistým uživatelem nebo jistou skupinou. Odpovídající testy jsou `-uid nn` a `-gid nn`. Naneštěstí nelze přímo zadat uživatelské jméno, ale musí se vždy uvést jeho identifikační číslo `nn`.

[1] Distribuované souborové systémy umožňují, aby se soubory lokalizované někde jinde jevily jako lokální.

Je povoleno použít zápis identifikačního čísla ve tvaru +nn, což znamená „ostře větší než“ nebo -nn, což znamená „ostře menší než“. Ve spojení s identifikačními čísly uživatelů se tato možnost jeví jako hloupá, ale v souvislosti s ostatními testy může být užitečná.

Další užitečnou volbou je volba -type c, která vrací hodnotu true, pokud je soubor typu c. Mnemotechnické zkratky jsou stejné, jako v případě příkazu ls: **b** pro binární soubor, **c** pro znakový soubor, **d** pro adresáře, **p** pro pojmenované roury, **l** pro symbolické odkazy a **s** pro sokety (sockets). Obyčejné soubory jsou specifikovány prostřednictvím zkratky **f**. K testu -type se vztahuje test -xtype, který je podobný, ale chová se jinak v případě symbolických odkazů - pokud není zadána volba -follow, ověřuje se místo vlastního symbolického odkazu soubor, na který odkaz odkazuje.

Testy -inum nn a -links nn ověřují, zda je číslo i-uzlu příslušné k danému souboru nn nebo zda má soubor nn symbolických odkazů. Test -size nn má hodnotu true, jestliže je pro soubor alokováno nn bloků po 512 bajtech. Dnes již však neplatí, že se velikost souboru vždy měří (například po zadání příkazu ls -s) v blocích po 512 bajtech - Linux například používá bloky po 1024 bajtech. Proto je možné číslo nn doplnit znakem b (pak se měří velikost v bajtech) nebo k (pak se měří velikost v kilobajtech).

Bity specifikující přístupová práva se testují pomocí testu -perm *mode*. Pokud argumentu *mode* nepředchází žádné znaménko, pak bity specifikující přístupová práva musejí přesně souhlasit. Pokud předchází znaménko -, pak musejí být nastavena příslušná přístupová práva, ale o ostatních se nic nepředpokládá. Pokud předchází znak +, pak je podmínka splněna, právě když je kterýkoliv z bitů nastaven. Důležité je, že hodnota mode musí být uvedena buď symbolicky, nebo jako oktalové číslo, tak jako v případě příkazu chmod.

Následující skupina testů se vztahuje k času, kdy byl soubor naposledy použit. Takové testy jsou zejména užitečné tehdy, když dojde k zaplnění diskového prostoru a uživatel potřebuje vyhledat staré soubory, které lze zrušit. Staré a zapomenuté soubory se těžko hledají a příkaz find vám dává naději, že se vám je podaří nalézt. Test -atime nn vrací hodnotu true, pokud byl daný soubor zpřístupněn před nn dny, -ctime nn vrací hodnotu true, když byly atributy přístupových práv daného souboru změněny před nn dny (například příkazem chmod). Test -mtime nn vrací hodnotu true, když byl soubor naposledy modifikován před nn dny. Užitečným bude zřejmě test -newer file, který vrací hodnotu true, když byl uvažovaný soubor modifikován později než soubor uvedený jako parametr file. GNU-verze příkazu find také akceptuje testy -anewer a -cnewer, které se chovají podobně jako test -newer, a dále testy -amin, -cmin a -mmin, které pracují s časem uvedeným v minutách.

V neposlední řadě se musíme zmínit o testu -name pattern. Tento test vrací hodnotu true, pokud jméno souboru vyhovuje šabloně uvedené prostřednictvím parametru pattern. Pro šablonu platí prakticky stejná pravidla jako pro používání pseudoznaků ve jménech souborů, například při používání příkazu ls. Jistě si pamatujete, že příkazový procesor je schopen

zvláštním způsobem interpretovat tzv. pseudoznaky (hvězdičku a otazník). Avšak test `-name foo*` nevrátí tu hodnotu, kterou byste očekávali. Místo toho budete muset použít test `-name foo` nebo `-name "foo*"`. Dobře si tuto situaci zapamatujte, většina uživatelů používá nesprávný test. Je zde ještě jeden rozdíl oproti příkazu `ls` - tečky na začátku nejsou interpretovány. Pokud se potřebujete vyrovnat s tímto problémem, použijte test `-path pattern`, který se o tečky a zpětná lomítka při porovnávání cest nestará.

11.1.5 Akce

Jak jsme si již řekli, argumenty charakterizující akce slouží k inicializaci nějaké operace. Přesto mezi tyto parametry patří například akce `-prune`, která nic neaktivuje, pouze realizuje krok dolů ve stromové struktuře adresářů. Většinou je tato akce spjata s parametrem `-fstype`, prostřednictvím kterého se realizuje výběr mezi různými souborovými systémy. Ostatní akce mohou být rozděleny do dvou širokých kategorií.

- Akce, které něco tisknou. Je zřejmé, že implicitní akcí bude akce `-print`. Ta v průběhu činnosti programu find tiskne jména souborů, které se právě zpracovávají (ovšem za předpokladu, že ostatní podmínky uvedené v příkazovém řádku vracejí hodnotu true). Akce `-print` má několik variant. První z nich, `-fprint file`, používá soubor uvedený jako parametr `file` místo standardního výstupu. Další, `-ls`, zobrazuje jméno aktuálního souboru jako příkaz `ls -dils`. Akce `-printf format` se chová podobně jako funkce `printf()` v jazyku C (v této variantě lze specifikovat formát výstupu). Totéž realizuje akce `-fprintf file`, ale do souboru uvedeném prostřednictvím parametru `file`. Tato akce rovněž vždy vrací hodnotu true.

- Akce, které aktivují nějaký příkaz. Jejich syntaxe je poněkud zvláštní, a proto se na ně podívejme podrobněji.

  ```
  -exec command \;
  ```

  Akce `-exec` spustí příkaz zadaný prostřednictvím parametru `command` a vrací hodnotu true, pokud má příkaz status ukončení 0. Za příkazem je uvedena dvojice znaků „`\;`“, protože jinak by příkaz `find` nebyl schopen rozeznat, kde příkaz `command` končí (trik s uvedením akce `-exec` na konec příkazu `find` není aplikovatelný). Proto se příkaz `command` ukončuje znakem „`;`“, což je zároveň znak k oddělování příkazů v jazyku příkazového procesoru. Bez znaku „`\`“ by středník byl příkazovým procesorem na základě syntaktické analýzy odstraněn a do příkazu `find` by se již nedostal. Proto se zde znak „`\`“ musí uvést. Další věc, kterou si musíte zapamatovat, spočívá ve způsobu specifikace jména aktuálního souboru uvnitř příkazu `command`. Jméno souboru musí být uvedeno pomocí dvojice složených závorek `{}`. Některé starší verze programu `find` vyžadují, aby bylo jméno odděleno netisknutelnými znaky (white spaces), například mezerami. To však není příliš šikovné, proto je ve verzi GNU zavedena konstrukce se složenými závorkami umožňující gene-

rovat řetězce. Asi vás napadá otázka, zda musejí být složené závorky uzavřeny do uvozovek. Podle mých zkušeností nemusejí - ani v případě, že použijete příkazový procesor `bash`, ani v případě, že použijete příkazový procesor `tcsh`. Proto zůstanou v příkazu `find` řetězce uvedené ve složených závorkách příkazovým procesorem nedotčeny.

```
-ok command \;
```

Akce `-ok` se chová podobně jako akce `-exec` s tím rozdílem, že pro každý vybraný soubor je uživatel vyzván k potvrzení příkazu (tj. zda se má provést nebo ne). Pokud odpověď začíná písmenem „y“ nebo „Y“, příkaz se provede a akce vrátí hodnotu true. Jinak se akce neprovede a vrácená hodnota je false.

11.1.6 Operátory

V příkazu `find` lze použít spoustu operátorů. Následující seznam je uvádí v pořadí s klesající prioritou.

```
\( expr \)
```

Kulaté závorky mění prioritu operandu. Závorkám musí předcházet znak „\“, protože v příkazovém procesoru mají speciální význam.

```
! expr
-not expr
```

Operátory `!` a `-not` mění pravdivostní hodnotu výrazu `expr`. Je-li hodnota `expr` true, změní se na false a naopak. Znak „`!`“ nemusí být oddělen obráceným lomítkem nebo uvozovkami, protože je následován netisknutelným znakem (mezerou).

```
expr1 expr2
expr1 -a expr2
expr1 -and expr2
```

Všechny tři varianty odpovídají logické operaci AND, přičemž se nejčastěji používá první varianta. Jestliže je první výraz `expr1` vyhodnocen jako false, pak se již druhý výraz nevyhodnocuje.

```
expr1 -o expr2
expr1 -or expr2
```

Obě varianty odpovídají logické operaci OR. Je-li první výraz `expr1` vyhodnocen jako true, pak se již druhý výraz `expr2` nevyhodnocuje.

```
expr1 , expr2
```

Uvedený operátor má poněkud speciální význam. Oba výrazy `expr1` i `expr2` se vyhodnotí (samozřejmě se všemi vedlejšími účinky) a hodnota konečného výrazu je nastavena na hodnotu výrazu `expr2`.

11.1.7 Příklady

Jak vyplývá z předcházejících seznamů, příkaz `find` má spoustu parametrů. Existuje však několik často používaných konstrukcí s příkazem `find`, které stojí za to si zapamatovat. Některé z nich si uvedeme jako příklady.

```
% find .-name foo\* -print
```

První příklad vyhledá všechny soubory, jejichž jména začínají řetězcem `foo`. Pokud se mají hledat soubory, jejichž jména mají řetězec `foo` někde uvnitř, pak je vhodné místo `foo` použít „`*foo*`“.

```
% find /usr/include/ -xtype f -exec grep foobar \
   /dev/null {}\;
```

V druhém příkladu se rekurzivně provádí příkaz `grep`, a to od adresáře `/usr/include`. V tomto případě se zajímáme o obyčejné soubory a symbolické odkazy, které na obyčejné soubory odkazují. Proto jsme použili test `-xtype`. V mnoha případech je jednodušší tento test nepoužít, zejména tehdy, kdy jsme si jisti, že žádný binární soubor neobsahuje požadovaný řetězec. A proč jsme použili příkaz `/dev/null`? Jedná se o trik, který příkaz `grep` „přinutí“ vypsat jméno souboru, pro který nebylo splněno výběrové kritérium. Příkaz `grep` je aplikován na každý soubor jinak, a proto „nepovažuje“ za nezbytné vypisovat jméno souboru. Ale nyní jsou zde dva soubory - aktuální soubor a soubor `/dev/null`. Jiná možnost spočívá v použití roury a příkazu `xargs`. Když jsem ji zkusil, zničil jsem si celý souborový systém.

```
%find / -atime +1 -fstype ext2 -name core \
   -exec rm {}\;
```

Tento příklad představuje klasickou úlohu pro crontab. Zruší ze souborového systému `ext2` všechny soubory s názvem `core`, které nebyly posledních 24 hodin zpřístupněny. Je možné, že někdo tyto soubory používá v souvislosti s programem `gdb` k ladění, ale je málo pravděpodobné, že je bude používat po 24 hodinách.

```
% find /home -xdev -size +500k -ls > piggies
```

Další příklad umožňuje zjistit, kdo vlastní soubory větší než 500 kilobajtů a kdo tedy zatěžuje souborový systém. Poznamenejme, že pokud se zajímáme pouze o jeden souborový systém, pak argument `-xdev` není nutný.

11.1.8 Slovo na závěr

Mějte na paměti, že příkaz `find` spotřebuje velmi mnoho času, pokud má provádět operace s každým souborem v celém souborovém systému. Proto je nutné počet operací vhodně optimalizovat, zejména když se na vašem systému pravidelně realizují úlohy pro údržbu prostřednictvím crontab. Jako poučný příklad vezměme následující: Předpokládejme, že chceme zrušit soubory končící na `.BAK`, přitom chceme změnit přístupová práva všech adresářů na 771 a přístupová práva souborů končících na `.sh` změnit na 755. Přitom chceme ještě připojit souborový systém NFS na telefonní linku, a v tomto systému nechceme žádné soubory kontrolovat. Proč bychom měli psát tři různé příkazy? Nejefektivněji uvedenou úlohu splníme takto:

```
% find .\( -fstype nfs -prune \) -o \
        \( -type d -a -exec chmod 771 {}\; \) -o \
        \( -name "*.BAK" -a -exec /bin/rm {}\; \) -o \
        \( -name "*.sh" -a -exec chmod 755 {}\; \)
```

Asi se vám bude takový příkaz zdát nesrozumitelný, ale když si jej pozorně prohlédnete, zjistíte, že je logický. Pamatujte si, že příkazy tohoto typu neprovádějí nic jiného, než že vyhodnocují výrazy, jejichž hodnota je buď false, nebo true. Samozřejmě mají vedlejší účinky - ty se realizují tehdy, když příkaz `find` musí vyhodnotit část obsahující výraz s akcí `-exec`, což nastane právě tehdy, když je levá strana výrazu vyhodnocena jako pravdivá (true). Když je například právě zpracovávaným souborem adresář, pak se bude realizovat první akce `-exec` a přístupová práva adresáře budou změněna na 771. Ostatní části příkazu budou vynechány. Budete-li takové příkazy aplikovat prakticky, přestanou se vám zdát nesrozumitelné a budete je používat zcela přirozeně.

11.2 Archivační program tar

11.2.1 Úvod

Tar je obecně použitelný program pro archivaci souborů, který je schopen sloučit (spakovat) velké množství souborů do jediného archivního souboru, přičemž zachovává veškeré informace o souborech, jako jsou uživatelská práva. Jméno `tar` je zkratkou „tape archive“, protože tento nástroj byl původně používán pro archivaci na magnetické pásky. Jak však uvidíme, používání příkazu `tar` není zdaleka omezeno pouze na záložní soubory pro magnetické pásky.

11.2.2 Hlavní funkce

Formát příkazu `tar` je následující:

`tar` *functionoptions files*`...`

---

kde *function* je jednoznaková indikace operace, která se má provést; *options* je seznam (jednoznakových) voleb pro tuto operaci a *files* je seznam souborů, jež se mají spakovat nebo rozpakovat. (Všimněte si, že argument function není oddělen od options mezerou.) Parametr `function` může být:

- `c` pro vytvoření nového archivního souboru
- `x` pro extrakci souborů z archivního souboru
- `t` pro výpis obsahu archivního souboru
- `r` pro přidání souborů na konec archivního souboru
- `u` pro aktualizaci souborů, které jsou novější než soubory v archivním souboru
- `d` pro porovnání souborů v archivním souboru se soubory v souborovém systému

Nejčastěji budete používat parametr `c`, `x` a `t`; ostatní budete používat jen zřídka.

11.2.3 Volby

Nejčastěji používané volby `options` jsou tyto:

- `v` pro tisk podrobné informace v průběhu pakování a rozpakování
- `k` pro zachování existujících souborů při rozpakování, tj. žádný existující soubor nebude přepsán souborem z archivního souboru

- `f` *filename* ...pro specifikaci jména archivního souboru

Další volby popíšeme v následujícím oddílu později.

11.2.4 Příklady

Ačkoliv se syntaxe příkazu `tar` zdá být na první pohled složitá, je jeho praktické použití velmi jednoduché. Předpokládejme, že máme adresář `mt` obsahující následující soubory:

```
rutabaga% ls -l mt
-rw-r--r-- 1 root root 24 Sep 21 1993 Makefile
-rw-r--r-- 1 root root 847 Sep 21 1993 README
-rw-r--r-- 1 root root 9220 Nov 16 19:03 mt
-rwxr-xr-x 1 root root 2775 Aug 7 1993 mt.1
-rw-r--r-- 1 root root 6421 Aug 7 1993 mt.c
-rw-r--r-- 1 root root 3948 Nov 16 19:02 mt.o
-rw-r--r-- 1 root root 11204 Sep 5 1993 st_info.txt
```

Nyní chceme spakovat pomocí příkazu `tar` obsah tohoto adresáře do jediného archivního souboru. Použijeme tedy příkaz:

```
tar cf mt.tar mt
```

Prvním argumentem v příkazu `tar` je operace (zde `c` pro vytvoření) následovaná volbou `options`, kde jsme použili `f mt.tar` a specifikovali tak jméno archivního souboru `mt.tar`. Jako poslední je uveden seznam souborů - pokud se místo seznamu uvede název adresáře, `tar` spakuje všechny soubory v tomto adresáři.

Poznamenejme, že prvním argumentem musí být písmeno, označující operaci, následované seznamem voleb „options“. Proto není důvod dávat před první argument pomlčku, jak to vyžaduje většina systémů Unix. Příkaz `tar` v systému Linux však tuto pomlčku připouští, proto by mohl mít posledně uvedený příklad tvar:

```
tar -cf mt.tar mt
```

V některých verzích musí být typ operace (jako je `c`, `t`, nebo `x`) na prvním místě, v jiných verzích na pořadí písmen nezáleží. Jistý přehled o průběhu pakování (nebo rozpakování) získáte prostřednictvím volby `v` - pak se bude každý soubor ukládaný do archivního souboru vypisovat na obrazovce. Například:

```
rutabaga% tar cvf mt.tar mt
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Použijete-li volbu `v` vícekrát, zobrazovaná informace bude podrobnější:

```
rutabaga% tar cvvf mt.tar mt
drwxr-xr-x root/root 0 Nov 16 19:03 1994 mt/
-rw-r--r-- root/root 11204 Sep 5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root 847 Sep 21 16:37 1993 mt/README
-rwxr-xr-x root/root 2775 Aug 7 05:50 1993 mt/mt.1
-rw-r--r-- root/root 24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root 6421 Aug 7 09:50 1993 mt/mt.c
-rw-r--r-- root/root 3948 Nov 16 19:02 1994 mt/mt.o
-rw-r--r-- root/root 9220 Nov 16 19:03 1994 mt/mt
```

Tyto podrobné informace jsou důležité zejména tehdy, když chcete zkontrolovat, zda `tar` realizuje pakování dle vašich představ. U některých verzí programu `tar` musí být volba `f` uvedena jako poslední. Je to z toho důvodu, že se za volbou `f` předpokládá jméno souboru. Pokud neuvedete volbu `f`, předpokládá `tar` (z historických důvodů), že má použít zařízení `/dev/rmt0`, což je první magnetopásková jednotka.

Nyní můžeme soubor `mt.tar` předat někomu jinému a ten si jej může rozpakovat na svém počítači. K rozpakování by měl použít následující příkaz:

```
tar xvf mt.tar
```

Tento příkaz vytvoří adresář `mt` a umístí do něj všechny soubory, které jsme dříve uvedli - s týmiž přístupovými právy jako v původním systému. Nové soubory budou vlastněny uživatelem, který provedl příkaz `tar xvf` - pokud ovšem nepoužijete tento příkaz jako root (pak je původní vlastník zachován). Parametr `x` zajistí rozpakování souborů a volba `v` opět zobrazí soubory, které se ukládají na disk:

```
courgette% tar xvf mmt.tar
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Všimněte si, že tar zachoval cestu každého souboru relativní k poloze v původní struktuře adresářů. To znamená, že když jsme vytvářeli archivní soubor pomocí příkazu `tar cf mt.tar mt`, jediný soubor, který jsme specifikovali místo seznamu souborů, byl adresář `mt` obsahující soubory. Proto tar uložil do archivního souboru samotný adresář a soubory, které se v něm nacházely. Při rozpakování se napřed vytvořil adresář `mt` a pak do něj byly uloženy soubory - tedy přesně opačný proces, než jaký probíhal při vytváření archivního souboru. tar implicitně rozpakuje všechny soubory relativně k pracovnímu adresáři, v němž jej spustíte. Pokud se například pokusíte spakovat obsah adresáře `/bin` příkazem:

```
tar cvf bin.tar /bin
```

dostanete varovné hlášení:

```
tar: Removing leading / from absolute path names in the archive.
```

To znamená, že soubory jsou ukládány v archivním souboru uvnitř adresáře /`bin`. Když tento soubor rozpakujete, adresář `/bin` bude vytvořen jako podadresář vašeho pracovního adresáře, v němž spouštíte `tar` - ne jako absolutní adresář `/bin`. Toto je velmi důležitý mechanismus, který byl vymyšlen za účelem ochrany před fatálními chybami při rozpakování pomocí příkazu `tar`. Jinak byste, podle předchozího příkladu, mohli přepsat soubory v adresáři `/bin`, a tak zničit systém.

Pokud byste však opravdu chtěli rozpakovat takový archivní soubor do adresáře `/bin`, museli byste mít jako pracovní adresář nastaven `/`. Výše uvedený mechanismus můžete potlačit pomocí volby `p`, ale nedoporučuje se to. Jiný způsob, jak vytvořit soubor `mt.tar`, spočívá v tom, že se přepnete do adresáře `mt` pomocí příkazu `cd` a zadáte příkaz:

```
tar cvf mt.tar *
```

Potom ovšem nebude podadresář `mt` ukládán do archivního souboru a při rozpakování budou soubory ukládány přímo do vašeho pracovního adresáře. Proto doporučujeme vždy pakovat soubory tak, aby archivní soubor obsahoval podadresář, jak jsme ukázali pomocí příkazu `tar cvf mt.tar mt`. Pak se vždy před rozpakováním vytvoří adresář pro uložení souborů a nemůže se stát, že přepíšete soubory ve svém pracovním adresáři. Navíc zbavíte osobu, která provádí rozpakování, starostí s vytvářením adresáře pro ukládání souborů a ušetříte jí dost času. Samozřejmě existuje mnoho situací, při nichž nebude vhodné doporučený postup dodržovat, ale všeobecně se takový postup považuje za jakousi etiketu při práci s programem `tar`.

Při vytváření archivních souborů můžete uvést seznam souborů nebo adresářů, které se mají do archivního souboru uložit. V prvním příkladu jsme použili příkaz `tar` pro jediný adresář a ukázali jsme použití hvězdičky, kterou příkazový procesor rozšíří na seznam všech souborů v pracovním adresáři. Před rozpakováním archivního souboru pomocí příkazu `tar` je vhodné se podívat, co je jeho obsahem. Tak se například dozvíte, zda budete muset vytvořit adresář pro uložení souborů vlastními silami, nebo bude vytvořen automaticky. K prohlížení obsahu archivního souboru použijte příkaz:

```
tar tvf tarfile
```

Ten zobrazí obsah archivního souboru `tarfile`. Poznamenejme, že pokud použijete funkci t, stačí uvést jednu volbu `v` a obdržíte podrobnou informaci o obsahu archivního souboru:

```
courgette% tar tvf mt.tar mt
drwxr-xr-x root/root 0 Nov 16 19:03 1994 mt/
-rw-r--r-- root/root 11204 Sep 5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root 847 Sep 21 16:37 1993 mt/README
-rwxr-xr-x root/root 2775 Aug 7 05:50 1993 mt/mt.1
-rw-r--r-- root/root 24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root 6421 Aug 7 09:50 1993 mt/mt.c
-rw-r--r-- root/root 3948 Nov 16 19:02 1994 mt/mt.o
-rw-r--r-- root/root 9220 Nov 16 19:03 1994 mt/mt
```

Žádné rozpakování v tomto případě neproběhne, ale pouze se zobrazí informace o obsahu archivního souboru. Zde vidíme, že jména souborů jsou doplněna jménem adresáře `mt`, a proto při rozpakování bude tento adresář nejdříve vytvořen a pak teprve do něj budou ukládány soubory. Příkaz `tar` můžete rovněž použít k extrakci jednotlivých souborů z archivního souboru. Potom použijte příkaz ve tvaru:

```
tar xvf tarfile files
```

kde `files` je seznam souborů, které se mají rozpakovat. Jak jsme si již ukázali, pokud neuvedeme žádný seznam, `tar` rozpakuje všechny soubory z archivního souboru.

Jestliže specifikujete soubory, které se mají rozpakovat, musíte uvést jejich plná jména včetně cesty tak, jak jsou uvedena v archivním souboru. Chceme-li například rozpakovat soubor `mt.c` z výše uvedeného archivního souboru `mt.tar`, použijeme následující příkaz:

```
tar xvf mt.tar mt/mt.c
```

který nejdříve vytvoří podadresář `mt` a do něj pak umístí soubor `mt.c`. Program `tar` má mnohem více možností, než které jsme zde uvedli. Ty, o nichž jsme se zde zmínili, budete zřejmě používat nejčastěji. Verze GNU implementovaná operačnímu systému Linux má řadu přípon a představuje ideální nástroj pro pořizování záložních kopií. Více informací naleznete v manuálové stránce k příkazu `tar`.

11.2.5 Jak používat program tar spolu s programem gzip

Program tar neprovádí při ukládání dat do archivního souboru žádnou komprimaci. Jestliže vytvoříte soubor `tar` ze tří souborů o velikosti 200 KB, pak výsledný soubor bude mít velikost 600 KB. Proto patří k běžné praxi komprimovat soubory `tar` pomocí programu `gzip` (nebo starším programem `compress`). Komprimovaný soubor tar můžete vytvořit pomocí následujících příkazů:

```
tar cvf tarfile files...
gzip -9 tarfile
```

Provedení takových příkazů je však těžkopádné a vyžaduje, abyste měli na disku dostatek místa pro nekomprimovaný soubor `tar`, než spustíte `gzip`.

Mnohem efektivnější způsob spočívá v tom, že se využije možnost programu tar zapisovat archivní soubor do standardního výstupu. Pokud použijete místo jména archivního souboru pomlčku (`-`), pak bude `tar` číst data ze standardního vstupu nebo zapisovat do standardního výstupu. K vytvoření komprimovaného souboru tar tedy můžeme použít příkaz:

```
tar cvf - files... | gzip -9 > tarfile.tar.gz
```

Zde vytváří `tar` archivní soubor ze souborů uvedených v seznamu `files`, zapisuje jej do standardního výstupu; `gzip` čte data ze standardního vstupu, komprimuje je a zapisuje do svého standardního výstupu. Nakonec jsme přesměrovali komprimovaný soubor `tar` do `tarfile.tar.gz`. Podobně bychom mohli k rozpakování takového souboru použít příkaz:

```
gunzip -9c tarfile.tar.gz | tar xvf -
```

Zde `gunzip` provede dekomprimaci uvedeného archivního souboru a zapisuje výsledek do standardního výstupu sloužícího jako standardní vstup pro program `tar`, jenž soubory rozpakuje a ukládá. Není práce v systému Unix zábavná? Samozřejmě, že jsou oba výše uvedené příkazy poněkud těžkopádné. Naštěstí GNU-verze programu tar má možnost uvést volbu `z`, která zajistí automatické vytváření nebo rozpakování komprimovaných archivních souborů `tar`. (Diskusi o použití volby `z` jsme úmyslně zařadili až na toto místo, abyste si uvědomili její výhody.) K vytvoření a rozpakování archivních souborů máte tedy možnost používat následující příkazy:

```
tar cvzf tarfile.tar.gz files...
```

a

```
tar xvzf tarfile.tar.gz
```

Poznamenejme, že byste měli soubory vytvořené tímto způsobem označovat pomocí přípony `.tar.gz`, aby byl zřejmý jejich formát. Volba funguje i ve spojení s dalšími parametry, jako je například `t`. Možnost používat volbu `z` podporuje pouze GNU-verze programu `tar`. Jestliže používáte `tar` na jiných systémech Unix, musíte pro realizaci stejného úkolu zadávat delší příkazy, jak jsme uvedli dříve. Téměř všechny verze operačního systému Linux používají verzi GNU.

Nyní využijeme svých znalostí a napíšeme krátké skripty pro příkazový procesor, které vám mohou sloužit jako návod pro vytváření a rozpakování souborů `tar`. V příkazovém procesoru bash doplňte následující řádky do souboru `.bashrc`:

```
tarc () {tar czvf $1.tar.gz $1 }
tarx () {tar xzvf $1 }
tart () {tar tzvf $1 }
```

Budete-li nyní chtít vytvořit komprimovaný archivní soubor obsahující soubory z jednoho adresáře, stačí zadat příkaz:

```
tarc directory
```

Výsledný archivní soubor bude mít název `directory.tar.gz`. (Přesvědčte se, že jste neuvedli před jménem adresáře lomítka (`/`) - jinak bude archivní soubor vytvořen jako `.tar.gz` uvnitř daného adresáře.) K zobrazení obsahu archivního souboru stačí zadat příkaz

```
tart file.tar.gz
```

a k jeho rozpakování příkaz

```
tarx file.tar.gz
```

11.2.6 Triky při používání programu tar

Protože tar ukládá do archivního souboru informace o vlastnictví souborů, přístupových právech, adresářové struktuře i symbolických odkazech, velmi dobře se hodí ke kopírování nebo přesouvání celých adresářových stromů z jednoho místa na druhé v rámci jednoho systému (a dokonce i mezi systémy, jak uvidíte). Použijete-li syntaxi s pomlčkou (`-`), budete zapisovat výsledný soubor do standardního výstupu, který může být čten jako standardní vstup a rozpakován kdekoliv.

Předpokládejme například, že máme adresář obsahující dva podadresáře: `from-stuff` a `to-stuff`. Adresář `from-stuff` obsahuje celý strom dalších podadresářů s mnoha soubory, symbolickými odkazy atd. a bylo by velmi obtížné zkopírovat jej pomocí příkazu `cp`. Ke zkopírování celého adresáře `from-stuff` do adresáře `to-stuff` však můžeme použít následující příkazy:

```
cd from-stuff
tar cf - .| (cd ../to-stuff; tar xvf -)
```

Velice jednoduché a elegantní! Začínáme v adresáři `from-stuff`, kde vytvoříme soubor tar obsahující celý adresář, a zapíšeme jej do standardního výstupu. Tento výstup pak čte podřízený příkazový procesor (příkazy uvedené v závorkách), který se nejdříve přepne do adresáře `../to-stuff` a pak spustí příkaz `tar xvf`, jenž čte ze standardního vstupu. Přitom se žádný archivní soubor `tar` neukládá na disk - data jsou přímo posílána z jednoho procesu tar do druhého. Druhý proces tar používá volbu `v` pro zobrazení informací o každém souboru, který se ukládá, a vy můžete kontrolovat, zda kopírování obsahu adresářů probíhá správně.

Pomocí tohoto „triku“ můžete ve skutečnosti přenášet celé adresáře mezi jednotlivými systémy - stačí vložit vhodný příkaz `rsh` mezi příkazy uzavřené v závorkách. Pak vzdálený příkazový procesor spustí `tar`, který bude číst archivní soubor jako standardní vstup. (Poznamenejme, že verze GNU příkazu `tar` má možnost automaticky číst nebo zapisovat soubory `tar` z/do jiných počítačů v síti; informace najdete v příslušné manuálové stránce.)

11.3 Program dd

Existuje jakási legenda, podle které v ranných dobách vývoje operačního systému Unix potřebovali programátoři program pro binární kopírování dat mezi jednotlivými zařízeními. Protože velmi spěchali, „vypůjčili“ si syntaxi příkazu používanou v operačním systému na počítačích IBAlt+360 a později vyvinuli rozhraní konzistentní s ostatními příkazy operačního systému Unix. Nevím, zda je tato legenda pravdivá, ale pěkně se vypráví.

11.3.1 Volby

Přes svůj legendou opředený původ není program `dd` úplně jiný než ostatní příkazy operačního systému Unix. Ve skutečnosti představuje filtr, který implicitně čte data ze standardního vstupu a zapisuje je na standardní výstup. Pokud zadáte na terminálu pouze příkaz `dd`, pak bude program `dd` tiše očekávat vstup z klávesnice.

Syntaxe příkazu `dd` je následující:

```
dd [if=file] [of=file] [ibs=bytes] [obs=bytes]
   [bs=bytes] [cbs=bytes] [skip=blocks] [seek=blocks]
   [count=blocks] [conv={ascii, ebcdic, ibm, block,
   unblock, lcase, ucase, swab, noerror, notrunc, sync}]
```

Všechny volby mají tvar *option=value*. Před a za znakem „=“ se nesmí objevit mezera, což je nepříjemné, protože v takové situaci neprovede příkazový procesor doplnění jména souboru, pokud se uvedou pseudoznaky. Verze příkazového procesoru bash v operačním systému Linux je však dostatečně inteligentní, proto nemusíte mít obavy. Všechny výše uvedené číselné hodnoty (`bytes` a `blocks`) mohou být následovány multiplikativní konstantou. Pro tyto konstanty lze použít následující zkratky: **b** pro bloky (multiplikativní konstanta 512), **k** pro kilobajty (multiplikativní konstanta 1024), **w** pro slova (multiplikativní konstanta 2) a prostřednictvím **xm** se číselná hodnota násobí konstantou **m**.

Význam voleb příkazu `dd` je popsán v následujícím seznamu.

- `if=`*filein* a of=*fileout* jsou volby specifikující jméno vstupního a výstupního souboru. Výstupní soubor je zkrácen na velikost danou volbou `seek`. Pokud není klíčové slovo `seek` uvedeno, pak je hodnota `seek` rovna nule (soubor je před provedením operace zrušen). Volba `notrunc` (viz dále) však může toto implicitní chování příkazu `dd` změnit.

- `ibs=`*nn* a `obs=`*nn* jsou volby specifikující počet bajtů, které se mají najednou přečíst a najednou zapsat. Domnívám se, že implicitní hodnoty jsou jeden blok (t.j. 512 bajtů), ale nejsem si tím tak docela jist. Uvedené parametry jsou velmi důležité v případě, kdy se jako vstup nebo výstup používají speciální zařízení - například při čtení ze sítě je hodnota

ibs nastavena na 10 kilobajtů, zatímco disketa 3,5 palce má přirozenou délku bloku 18 kilobajtů. Nevhodné nastavení těchto hodnot může nejen značně prodloužit realizaci programu, ale také může vést k neodstranitelným chybám. Proto buďte opatrní.

- bs=*nn* je volba, která předefinuje nastavení ibs a obs - obě hodnoty se nastaví na stejnou konstantu *nn*.

- Volba cbs=*nn* nastavuje vyrovnávací paměti pro konverzi na *nn* bajtů. Vyrovnávací paměť se používá při konverzi z formátu ASCII na EBCDIC nebo při konverzi mezi textovými a binárními soubory a podobně. Například soubory vytvořené pod operačním systémem VMS mají zpravidla velikost bloku 512 bajtů, proto byste měli například při čtení dat zapsaných na magnetickou pásku v operačním systému VMS nastavit hodnotu cbs na 1 bajt.

- skip=*nbl* a seek=*nbl* jsou volby, které „sdělí“ programu dd, kolik bloků má „přeskočit“ při čtení vstupu a při zápisu na výstup. Samozřejmě platí, že druhá volba má smysl jedině tehdy, když je specifikována konverze notrunc. Velikost bloků je dána volbami ibs a obs. Uvědomte si, že pokud nezadáte hodnotu ibs a zadáte hodnotu skip=1b, pak ve skutečnosti dojde k „přeskočení“ 512x512 bajtů, což je 256 kilobajtů.

- Volbou count=*nbl* se nastavuje, kolik bloků se má ze vstupu kopírovat. Velikost bloku je přitom dána hodnotou ibs. Uvedená volba spolu s předcházející volbou je užitečná v případě, kdy chcete obnovit co nejvíce bajtů z porušeného souboru - „přeskočíte“ nečitelnou část souboru a zbytek zkopírujete do nového souboru.

- Volba conv=*conversion*,[*conversion*,...] je určena ke konverzi podle specifikace. Možné konverze jsou následující: ascii - konverze formátu EBCDIC na formát ASCII; ebcdic nebo ibm - konverze z formátu ASCII na EBCDIC (jednoznačná konverze z formátu EBCDIC na formát ASCII neexistuje; první představuje standardní konverzi a druhá funguje lépe, pokud se má soubor tisknout na tiskárně IBM); block - vyplňuje mezerami záznamy ukončené znakem newline na délku uvedenou prostřednictvím volby cbs; unblock - opačná konverze ke konverzi block; lcase - konverze z velkých písmen na malá; ucase - konverze z malých písmen na velká; swab - konverze, při které se každý pár vstupních bajtů zamění (chcete-li například použít na počítači s procesorem Intel soubor obsahující celá čísla, který byl vytvořen na počítači s procesorem 680x0, budete takovou konverzi potřebovat); noerror - program dd bude pokračovat v činnosti i poté, co nastane nějaká chyba; sync - každý vstupní blok se doplní znaky NUL tak, aby měl délku definovanou prostřednictvím volby ibs.

11.3.2 Příklady

Dříve či později se setkáte s případem, kdy budete chtít pod operačním systémem Linux vytvořit svoji první disketu. Jak zapsat data na disketu bez souborového systému MS-DOS? Řešení je jednoduché:

```
% dd if=disk.img of=/dev/fd0 obs=18k count=80
```

V uvedeném příkladu jsem se rozhodl nepoužít volbu `ibs`, protože nevím jaká je optimální volba pro velikost bloku na pevném disku. Nemůže však uškodit, když se místo volby `obs` použije volba `bs` - pak je proces kopírování dokonce rychlejší. Všimněte si nastavení počtu sektorů pro zápis (18 kilobajtů je velikost sektoru, proto je hodnota `count` nastavena na `80`). Pro disketovou jednotku se v operačním systému Linux používá jméno zařízení `/dev/fd0`.

Další užitečné použití příkazu `dd` je v oblasti zálohování, zejména při zapojení počítače do sítě. Předpokládejme, že máte počítač alfa a že na počítači beta je magnetopásková jednotka `/dev/rst0` obsahující soubor, který chcete zkopírovat. Dále předpokládejme, že máte stejná přístupová práva na obou počítačích a že na pevném disku počítače beta není dostatek místa pro kopii uvažovaného souboru. Pak můžete použít příkaz:

```
% rsh beta 'dd if=/dev/rst0 ibs=8k obs=20k' | tar xvBf -
```

Celou operaci takto realizujete „na jeden průchod“. Zajisté jste si všimli, že se v příkazu využila schopnost příkazového procesoru číst data z magnetopáskové jednotky. Velikosti vstupních a výstupních bloků jsou nastaveny na implicitní hodnoty, tedy 8 kilobajtů pro čtení a 20 kilobajtů pro zápis do sítě ethernet.

Poznámka na závěr: zapomněl jsem říci, že označení příkazu `dd` je zkratkou pro výraz „data duplicator“.

Kapitola

12

Chyby, skryté závady a další nepříjemnosti

12.1 Jak předcházet chybám

Řada lidí na nejrůznějších fórech dává najevo zklamání z operačního systému Unix. Jejich zklamání zpravidla vyplývá z toho, že jej neumějí efektivně využívat - obvykle se jedná o uživatele dříve zvyklé na operační systémy s pohodlnou obsluhou, jako je Microsoft Windows, Macintosh Operating System a podobně. Naopak lidé, kteří zvládli filosofii operačního systému Unix a jsou schopni v něm efektivně pracovat, na něj nedají dopustit. Cení si zejména toho, že jen málokterý příkaz vyžaduje v průběhu své realizace nějakou komunikaci s uživatelem, a proto v operačním systému vše běží hladce, rychle a bez problémů.

Právě skutečnost, že například příkazy `rm` a `mv` nikdy nevyžadují potvrzení, zda mají skutečně zrušit specifikované soubory, vede často k problémům. Proto si nyní projděme malý seznam, v němž uvádíme zásady, jak se takovým a podobným problémům vyhnout.

- Pořizujte si záložní kopie. Tato výzva platí zejména pro uživatele, kteří používají systém sami. Každý systémový administrátor by měl pravidelně pořizovat záložní kopie – dostatečný interval je asi jeden týden. Podrobnosti najdete v *„Příručce správce operačního systému Linux“*.

- Každý uživatel by si měl pořizovat své vlastní záložní kopie. Pokud používáte více jak jeden systém, pak si uchovávejte aktualizované kopie všech svých souborů na každém systému. Jestliže máte přístup k disketové jednotce, pak si na ni ukládejte kopie důležitých souborů. Přinejhorším si kopie důležitých souborů uchovávejte alespoň v oddělených adresářích.

- Důkladně si promyslete, zda jste správně zadali „destruktivní“ příkazy, jako je `mv`, `rm` a `cp`. Zrovna tak si dejte pozor na přesměrování (`>`) souborů - i to může být nebezpečné a vyžaduje zvláštní pozornost. Nevinně vyhlížející opomenutí může způsobit katastrofu:

```
/home/larry/report# cp report-1992 report-1993 backups
```

Stačí vynechat poslední parametr a neštěstí je hotovo (místo zálohy máte zrušen jeden soubor):

```
/home/larry/report# cp report-1992 report-1993
```

- Autor rovněž doporučuje na základě vlastních zkušeností neprovádět zálohování ani další úlohy údržby pozdě v noci, kdy jste unaveni a snadno uděláte chybu. Jestli o půl druhé v noci zjistíte, že máte příliš plný disk, pak jej nechte plným a nezačínejte hned rušit soubory - takovou práci si nechte až na ráno, kdy budete vyspaní a odpočatí.

- Používejte takovou výzvu příkazového řádku, která vás bude informovat o aktuálním adresáři. Pokud takovou výzvu nepoužíváte, rovněž hrozí nebezpečí. Následující příklad nám zaslal člen diskusní skupiny `comp.unix.admin`[1], který si nevšiml, že není v adresáři `/tmp`:

```
mousehouse> pwd
/etc
mousehouse> ls /tmp
passwd
mousehouse> rm passwd
```

Série právě uvedených příkazů by vás mohla učinit velmi nešťastnými při pohledu na to, jak si rušíte soubor `passwd`, bez kterého se nemůže nikdo do operačního systému Unix přihlásit.

12.2 Chyba není ve vás

Naneštěstí pro programátory na celém světě platí, že ne všechny problémy pocházejí z chyb uživatelů. Operační systémy Unix a Linux jsou velmi komplikované a všechny známé verze mají chyby či skryté závady. Některé z těchto závad lze jen velmi těžko odstranit, protože se projevují jen za velmi speciálních okolností.

V souvislosti s chybami programového vybavení se používá termín „bug“ (doslova štěnice). Asi nejvýstižnější překlad bude „skrytá chyba“, protože se jedná o chybu, o které autoři programového vybavení neví. Chyba tohoto druhu se odhalí až při testování daného programu, někdy až po velmi dlouhé době.

[1] Jedná se o mezinárodní diskusní skupinu, která řeší otázky kolem správy operačního systému Unix.

12.2.1 Co dělat, když objevíte skrytou chybu

Když vám počítač dá chybnou odpověď (nejdříve důkladně prověřte, zda je odpověď opravdu chybná) nebo nějaký program zhavaruje, pak lze uvažovat o skryté chybě.

Skrytou chybu může obsahovat program, který stále běží, přesto že by měl skončit - i v tomto případě byste však měli nejdříve zkontrolovat, zda neprovádí příliš složité a zdlouhavé operace. Než použijete nový příkaz, důkladně si prostudujte příslušnou dokumentaci (nejlépe manuálové stránky).

Některá hlášení vám budou připadat jako skryté chyby, i když ve skutečnosti skrytými chybami nebudou. Prostudujte si oddíl 3.4 a příslušnou dokumentaci, než učiníte nějaké závěry o skrytých chybách.

Například taková hlášení, jako „`disk full`“ nebo „`lp0 in fire`“ neznamenají, že je program vadný, ale že je něco v nepořádku s vaším technickým vybavením - nedostatek diskového prostoru nebo špatně připojená tiskárna.

Pokud nemůžete najít něco v dokumentaci, pak je to chybou dokumentace a měli byste autora programového vybavení kontaktovat a o nedostatku informovat. Zrovna tak je chybou dokumentace, když popisuje jiné vlastnosti programového vybavení, než jaké ve skutečnosti jsou.[2] Je-li něco nekompletního nebo nejasného v dokumentaci, pak se jedná o chybu dokumentace.

Naopak, jestliže nejste schopni porazit šachový program `gnuchess`, pak je to tím, že algoritmus tohoto programu je opravdu dobrý a neznamená to nutně, že je ve vašem mozku skrytá chyba.

12.2.2 Jak ohlásit chybu

Jakmile jste si jisti, že jste odhalili skrytou chybu, je nutné zajistit, aby se hlášení o ní dostalo na správné místo. Pokuste se zjistit, co chybu způsobilo a pokuste se rekonstruovat všechny okolnosti, za jakých chyba nastala. Jestli se vám chybu nepodaří znovu „vyvolat“ pročtěte si zprávy z diskusní skupiny `comp.os.linux.help` nebo `comp.unix.misc`. Také si důkladně pročtěte manuálové stránky k danému programu.

Při odesílání zpráv o skryté chybě se dává přednost elektronické poště. Pokud nemáte možnost odesílat zprávy prostřednictvím elektronické pošty, kontaktujte osobu, od které máte operační systém Linux nebo zkuste kontaktovat někoho, kdo přístup k elektronické poště má. Ta-

2 To bude asi také platit o tomto manuálu.

ké se můžete obrátit na komerčního dodavatele operačního systému Linux. Ten má určitě zájem na tom, aby se skryté chyby rychle odstraňovaly. Mějte na paměti, že nikdo není povinen odstraňovat chyby, dokud s ním nemáte podepsanou příslušnou smlouvu.

Když odesíláte hlášení o chybě, pak uveďte všechny informace a okolnosti, za kterých se chyba projevila. Dodržujte dále uvedené zásady:

- Popište, co si myslíte, že má program dělat a co ve skutečnosti dělá. Například: „Program vrací hodnotu 5, když mu byl zadán výraz 2+2“. Nebo „Program ohlásil `segment violation -- core dumped`“. Je velmi důležité popsat, co se stalo, aby mohl autor chybu odstranit.
- Uveďte všechna nastavení proměnných prostředí.
- Uveďte verzi jádra operačního systému (najdete ji v souboru `/proc/version`) a verzi systémových knihoven (podívejte se do adresáře `/lib`, nebo pošlete výpis obsahu tohoto adresáře).
- Popište, jak jste program spustili, a popište, co jste dělali, když k chybě došlo.
- Uveďte všechny okolnosti, za kterých k chybě došlo. Řekněme například, že příkaz `w` některému uživateli nezobrazí informace o aktuálním procesu. Nestačí napsat: „Příkaz `w` nefunguje pro jistého uživatele“. Chyba může nastat proto, že jméno uživatele má osm znaků nebo proto, že se přihlásil prostřednictvím sítě. Správná informace bude tedy znít takto: „Příkaz `w` nezobrazuje informace o aktuálním procesu uživateli greenfie, když se přihlásí prostřednictvím sítě.“.
- Buďte zdvořilí. Většina lidí pracuje obětavě na volném programovém vybavení z vlastní iniciativy, vlastní dobré vůle a především proto, aby vám předložili program, který potřebujete. Nebuďte na lidi, kteří pracují od rána do noci, hrubí - právě hrubé osočování dokázalo odradit nejednoho nadějného programátora od práce na operačním systému Linux.

A

Technické informace

Editor `vi` (vyslovuje se [ví áj]) je jediným editorem, který se nachází v každé instalaci operačního systému Unix. Původně byl vytvořen na univerzitě California v Berkeley, jeho různé verze se nacházejí ve všech distribucích operačního systému Unix a je také součástí distribuce operačního systému Linux. Editor `vi` se poměrně těžko učí, ale má mnoho velmi výkonných funkcí. Obecně se doporučuje začátečníkům editor Emacs, protože se mnohem snáze používá. Avšak uživatelé, kteří používají více počítačových platforem, raději pracují s editorem `vi`.

Abychom pochopili, proč klávesa K znamená posun kurzoru o jeden řádek nahoru a proč editor pracuje ve třech odlišných módech, musíme se podívat do historie. Pokud máte pokušení naučit se pracovat s editorem `vi`, pak můžete považovat tento dodatek za učebnici, která vás provede veškerými základy. Také zde předkládáme přehled příkazů, který můžete považovat za referenční příručku.

Dokonce i v případě, že editor `vi` nebude editorem pro vaši běžnou práci, není čas věnovaný jeho základům úplně zbytečný. Je téměř jisté, že v operačním systému typu Unix, který používáte, je editor `vi` instalován. Někdy je nezbytné použít editor `vi` při instalaci jiných programových produktů, například editoru Emacs. Spousta nástrojů operačního systému Unix, aplikací a her používá jistou podmnožinu příkazů editoru `vi`.

A.1 Stručná historie editoru vi

První textové editory byly orientovány na zpracování textových souborů řádek po řádku a používaly se na neinteligentních terminálech. Typickým editorem, který takto funguje, je editor **Ed**. Editor Ed je velmi výkonný a využívá velmi málo zdrojů počítače. V porovnání s editorem Ed nabízí editor `vi` uživateli vizuální alternativu s mnohem širší množinou příkazů.

Editor vi se startuje stejně jako řádkový editor ex. Platí, že editor ex je vlastně editor vi spuštěný ve speciálním módu. Vizuální složka editoru ex může být inicializována z příkazového řádku pomocí příkazu editoru vi nebo přímo z editoru ex.

Editor ex/vi byl vyvinut na univerzitě California v Berkeley a jeho autorem je William Joy. Původně byl dodáván jako nepodporovaný obslužný program. Oficiálně byl zařazen až v operačním systému System 5 Unix od firmy AT&T. Postupně se stal velmi populárním a dodnes konkuruje moderním celoobrazovkovým editorům.

V důsledku své popularity se editor vi objevil v mnoha verzích a dnes existují verze pro většinu operačních systémů (i jiných, než Unix). Cílem této kapitoly není popsat všechny dostupné příkazy editoru vi a jejich modifikace. Právě v důsledku toho, že vzniklo mnoho verzí editoru vi, není množina jeho příkazů standardizována. Řada klonů editoru vi dokonce nepodporuje některé původní příkazy.

Jestliže máte nějaké zkušenosti s editorem ed, pak se editor vi naučíte mnohem snáze. I když editor vi nehodláte používat, budou se vám základní znalosti hodit zejména v nouzových situacích.

A.2 Stručný výklad příkazů editoru Ed

Cílem tohoto oddílu je naučit vás pracovat s editorem ed. Byl navržen tak, aby se jej každý snadno naučil. Než jej budete moci používat, budete muset chvíli trénovat. Při probírání jednotlivých příkazů si hned vyzkoušejte uvedené příklady - tak se naučíte s editorem pracovat velmi rychle.

A.2.1 Vytvoření souboru

Editor ed je schopen editovat v daném okamžiku pouze jeden soubor. Vyzkoušejte si následující příklad a vytvořte si svůj první soubor prostřednictvím editoru ed.

```
/home/larry# ed

a

This is my first text file using Ed.
This is really fun.
.
w firstone.txt
q
/home/larry#
```

Nyní si můžete obsah souboru ověřit pomocí příkazu `cat` nebo `more`:

```
/home/larry# cat firstone.txt
```

Předcházející příklad ilustruje spoustu důležitých aspektů. Po spuštění editoru se objeví prázdný řádek. Klíč `a` je určen k přidání textu do souboru. Pokud chcete ukončit zadávání textu, zapište do prvního sloupce na novém řádku tečku. Chcete-li text uložit, pak zadejte příkaz `w` a jméno souboru. Samotný klíč `q` činnost editoru ukončí.

Nejdůležitější je poznatek, že editor pracuje ve dvou módech - v **textovém módu** a v **příkazovém módu**. Svoji činnost editor zahajuje v příkazovém módu, kdy lze na prázdných řádcích zadávat příkazy. Ty jsou definovány prostřednictvím jisté množiny klíčů.

A.2.2 Jak editovat existující soubor

Pokud chcete do existujícího souboru přidat řádek, postupujte dle následujícího příkladu:

```
/home/larry# ed firstone.txt

a

This is a new line of text.
.

q
```

Když zkontrolujete soubor `firstone.txt` pomocí příkazu `cat`, zjistíte, že nový řádek byl vložen mezi původní první a druhý řádek. Jak editor `ed` pozná, kam má vložit nový textový řádek?

Po načtení souboru si editor `ed` uchovává informaci o aktuálním řádku. Příkaz a vkládá nový text za aktuální řádek. V editoru `ed` také můžete vkládat nový řádek před aktuální řádek a to prostřednictvím příkazu `i`.

Nyní je patrné, že editor `ed` pracuje s textem řádek po řádku. Všechny příkazy mohou být aplikovány na specifikovaný řádek.

Dále uveďme příklad, jak přidat textový řádek na konec souboru:

```
/home/larry# ed firstone.txt
$a
The last line of text.
```

```
.
w
q
```

Modifikátor příkazu `$` „sdělí“ editoru `ed`, že má přidat řádek za poslední řádek stávajícího textu. Pokud byste chtěli přidat textový řádek za první řádek, použijte modifikátor `1`. Nyní máte k dispozici příkazy, pomocí kterých lze vkládat nový text před nebo za řádek specifikovaného čísla.

Jak se dozvíme, který řádek je aktuální? Stačí zadat příkaz `p` a zobrazí se obsah aktuálního řádku. Pokud chcete změnit aktuální řádek na hodnotu 2, pak postupujte dle následujícího příkladu:

```
/home/larry# ed firstone.txt
2p
q
```

A.2.3 Podrobnosti o číslování řádků

Ukázali jsme si, jak prostřednictvím příkazu `p` zobrazit obsah aktuálního řádku. Také víme, že pro příkazy existují modifikátory ve formě pořadových čísel řádků. Chceme-li vypsat obsah druhého řádku, stačí zadat příkaz:

```
2p
```

Existují také jiné speciální modifikátory, prostřednictvím kterých lze měnit nastavení aktuálního řádku. Znak dolaru `$` se používá pro poslední řádek textu. Chcete-li vypsat obsah posledního řádku, zadejte:

```
$p
```

Pro aktuální číslo řádku se používá speciální modifikátor tečka. Obsah aktuálního řádku prostřednictvím tohoto modifikátoru lze vypsat takto:

```
.p
```

Možná se vám takový příkaz zdá zbytečný. Když však často měníte obsah aktuálního řádku, pak zjistíte, jak je příkaz užitečný.

Pokud chcete zobrazit text v rozsahu řádků od 1 do 2, pak musíte specifikovat rozsah:

```
1,2p
```

První číslo odkazuje na počáteční řádek, jenž se má zobrazit, a druhé číslo odkazuje na poslední řádek, jenž se má zobrazit. Aktuální řádek se po provedení tohoto příkazu nastaví na druhou hodnotu.

Další příklad ukazuje, jak zobrazit obsah souboru od prvního řádku po aktuální řádek:

```
1,.p
```

Také můžete zobrazit obsah souboru od aktuálního řádku po řádek poslední:

```
.,$p
```

Nyní budete jistě umět zadat příkaz, který zobrazí obsah celého souboru.

Dále si ukážeme, jak zrušit první dva řádky souboru:

```
1,2d
```

Příkaz `d` ruší text řádek po řádku. Chcete-li zrušit celý text, stačí zadat:

```
1,$d
```

Pokud provedete v editovaném textu velké množství změn a nechcete obsah souboru uložit, pak je nejlepší editor ukončit bez zápisu.

Většina uživatelů nepoužívá editor `ed` jako svůj hlavní editor. Moderní editory jsou celoobrazovkové a nabízejí mnohem pružnější množinu příkazů. Editor `ed` představuje dobrý úvod k editoru `vi` a pomůže vám pochopit, odkud příkazy editoru `vi` pocházejí.

A.3 Stručný výklad příkazů editoru vi

Cílem tohoto oddílu je naučit vás pracovat s editorem `vi`. Předpokládáme, že nemáte s editorem `vi` žádné zkušenosti a probereme si deset nejzákladnějších příkazů. Tyto příkazy vám budou stačit k realizaci nejnutnějších kroků při editování souboru a další příkazy se pak snadno naučíte podle svých potřeb. Při probírání jednotlivých příkazů si hned vyzkoušejte uvedené příklady - tak se naučíte s editorem pracovat velmi rychle.

A.3.1 Jak spustit editor vi

Chcete-li spustit editor `vi`, zadejte jméno programu a jako parametr uveďte jméno editovaného textového souboru. Objeví se vám obrazovka, na jejíž levé straně bude zobrazen sloupec znaků tilda (`~`). Editor `vi` se nyní nachází v příkazovém módu - cokoliv napíšete, bude považováno za příkaz a nikoliv za vstupní text. Jestliže chcete zadávat text, musíte nejdříve zadat příkazy. Pro vkládání textu existují tyto dva základní příkazy:

- `i` vložení textu vlevo od kurzoru
- `a` přidání textu vpravo od kurzoru.

Protože se v tomto okamžiku nacházíte na začátku prázdného souboru, nezáleží na tom, který z uvedených příkazů použijete. Zadejte tedy jeden z nich a zapište následující text. (Jedná se o báseň, jejímž autorem je Augustus DeMorgan. Je uvedena v manuálu „*The Unix Programming Environment*“, který napsali pan B.W. Kernighan a R. Pike.):

```
Great fleas have little fleas<Enter>
  upon their backs to bite 'em,<Enter>
And little fleas have lesser fleas<Enter>
  and so and infinitum.<Enter>
And great fleas themselves, in turn,<Enter>
  have greater fleas to go on;<Enter>
While these again have greater still,<Enter>
  and greater still, and so on.<Enter>
<Esc>
```

Všimněte si, že k ukončení vkládaného textu se používá klíč Esc. Pak editor opět přejde do příkazového módu.

A.3.2 Příkazy pro pohyb kurzoru

- `h` posunutí kurzoru o jeden znak doleva
- `j` posunutí kurzoru o jeden řádek dolů
- `k` posunutí kurzoru o jeden řádek nahoru
- `l` posunutí kurzoru o jeden znak doprava

Uvedené příkazy mohou být opakovány tak, že danou klávesu budete držet stisknutou. Nyní si vyzkoušejte pohyb kurzoru všemi směry. Pokud se pokusíte posunout kurzor na neexistující pozici (například když stisknete klávesu K a přitom je kurzor na prvním řádku souboru), pak obrazovka blikne nebo se ozve zvukový signál. Ničeho se neobávejte, váš soubor nebude v takovém případě nijak poškozen.

A.3.3 Jak rušit text

- `x` zrušení znaku na pozici kurzoru
- `dd` zrušení řádku

Posuňte kurzor na druhý řádek a nastavte jeho pozici pod apostrof ve slově `'em`. Stiskněte klávesu X a apostrof zmizí. Nyní stiskněte klávesu I (přepnete tak editor `vi` do módu, ve kterém se vkládá text) a zapište „`th`“. Nakonec stiskněte klávesu Esc.

A.3.4 Uložení souboru

- `:w` uložení souboru (na disk)
- `:q` ukončení editoru `vi`

Nejdříve se přesvědčte, že jste v příkazovém módu – stiskněte klávesu Esc. Nyní zadejte `:w`. V tomto okamžiku se vámi vytvořený text uloží do diskového souboru.

Příkaz pro ukončeni editoru `vi` je reprezentován klíčem `:q`. Jestliže chcete zkombinovat uložení souboru a ukončení editoru, pak použijte příkaz `:wq`. Pro příkaz `:wq` existuje ekvivalentní zkratka `ZZ`. Zkratka `ZZ` se bude hodit zejména programátorům. Podívejme se na typickou posloupnost akcí, které provádí programátor při ladění programu: spuštění programu; zjištění problému; editování souboru obsahujícího zdrojový kód programu; provedení změn v souboru a jeho uložení na disk; přeložení programu; spuštění programu; ...a tak stále dokola. Pak bude programátor příkaz `ZZ` zřejmě používat velmi často. Ve skutečnosti není příkaz `ZZ` přesným ekvivalentem příkazu `:wq`. Příkaz `ZZ` totiž neprovede uložení souboru, pokud v textu neprovedete žádné změny, zatímco příkaz `:wq` soubor před ukončením editoru soubor vždy uloží.

Jestliže jste udělali nějaké změny, ale pak nechcete soubor ukládat, použijte příkaz `:q!` (nezapomeňte předtím stisknout klávesu Esc). Pokud byste zapomněli znak „`!`“ editor `vi` vám nepovolí ukončit práci bez uložení souboru.

A.3.5 Co bude následovat

Deset příkazů, které jsme v předcházejících odstavcích probrali, stačí k tomu, abyste byli schopni pracovat s editorem vi. Tyto příkazy však představují pouze základ práce s editorem. Existují další příkazy, například pro kopírování textu z jednoho místa na druhé, pro přesouvání textu z jednoho souboru do druhého, pro konfiguraci editoru a podobně. V editoru vi lze aplikovat asi 150 příkazů.

A.4 Pokročilejší techniky práce s editorem vi

Velká výhoda editoru `vi` spočívá v tom, že jej můžete používat, i když znáte jen několik málo základních příkazů. Většina uživatelů je nadšena, že stačí znát jen pár příkazů, ale jakmile s editorem pracují déle, potřebují k realizaci některých úloh další příkazy.

V následujícím oddílu se předpokládá, že se uživatel seznámil se základními příkazy uvedenými v předcházejícím oddílu. Nyní si probereme další příkazy – od kopírování textů až po definici maker.

Také uvedeme oddíl o konfiguraci editoru `vi`, kde se dozvíte, jak nastavit některé vlastnosti editoru, aby byla vaše práce co nejefektivnější. Následující odstavce jsou zaměřeny spíše na popis příkazů a nejsou již tolik zaměřeny na jejich procvičování. Proto doporučujeme, abyste si probírané příkazy průběžně procvičovali sami.

Nebudeme uvádět zcela vyčerpávající seznamy příkazů editoru `vi`, ale zaměříme se na příkazy nejčastěji používané zejména z praktického hlediska. I když si nakonec zvolíte pro svou běžnou práci jiný editor, budou se vám nabyté znalosti vždy hodit.

A.4.1 Příkazy pro změnu polohy kurzoru

K nejzákladnějším funkcím editoru patří příkazy pro změnu polohy kurzoru. Zde uvádíme jejich přehled.

- h — posunutí kurzoru o jeden znak doleva
- j — posunutí kurzoru o jeden řádek dolů
- k — posunutí kurzoru o jeden řádek nahoru
- l — posunutí kurzoru o jeden znak doprava

Některé implementace editoru vi také umožňují používat kurzorové klávesy.

- w — posunutí kurzoru na začátek následujícího slova
- e — posunutí kurzoru na konec následujícího slova
- E — posunutí kurzoru na konec následujícího slova před mezeru
- b — posunutí kurzoru na začátek předcházejícího slova
- 0 — posunutí kurzoru na začátek řádku
- ^ — posunutí kurzoru na první slovo v aktuálním řádku
- $ — posunutí kurzoru na konec řádku
- <CR> — posunutí kurzoru na začátek následujícího řádku
- - — posunutí kurzoru na začátek předcházejícího řádku

G posunutí kurzoru na konec souboru

1G posunutí kurzoru na začátek souboru

nG posunutí kurzoru na řádek s pořadovým číslem n

<Ctrl>+<Shift>+G zobrazení čísla aktuálního řádku

% posunutí kurzoru na odpovídající hranatou závorku

H posunutí kurzoru na první řádek obrazovky

M posunutí kurzoru na prostřední řádek obrazovky

L posunutí kurzoru na spodní řádek obrazovky

nl posunutí kurzoru na sloupec n

Jestliže kurzor dospěje na spodní nebo horní řádek obrazovky, pak se zobrazený text vždy příslušným směrem posune. Nyní uvedeme alternativní příkazy, které umožňují posouvání zobrazovaného textu na obrazovce (tzv. rolování).

<Ctrl>+f posunutí textu o stránku nahoru

<Ctrl>+b posunutí textu o stránku dolů

<Ctrl>+d posunutí textu o půl stránky dolů

<Ctrl>+u posunutí textu o půl stránky nahoru

Právě uvedené příkazy jsou určeny k pohybu kurzoru. Některé z nich mohou být modifikovány pomocí čísla, které se uvede před příkaz. Předřazené číslo n znamená, že se příkaz bude opakovat n-krát.

Uveďme si tvar příkazu pro posunutí kurzoru o n pozic doleva:

nl posunutí kurzoru o n pozic (znaků) doprava

Jestliže chcete například zařadit před text větší počet mezer, můžete použít podobnou modifikaci příkazu i pro vkládání textu. Zadejte počet mezer, pak příkaz i následovaný mezerou a nakonec stiskněte klávesu ESC.

ni opakované vložení textu n-krát

Příkazy odkazující na řádky mohou jako modifikátor použít číslo řádku. Ukázkovým příkladem je příkaz G:

1G posunutí kurzoru na první řádek textového souboru

Editor `vi` má velké množství příkazů, které lze použít ke změně pozice kurzoru - od jednoduchých příkazů, kde se kurzor posune o jednu pozici, až po složitější příkazy, kdy se kurzor nastavuje na předem specifikovanou pozici. Také lze zadat nastavení kurzoru na specifikovaný řádek při spouštění editoru, například:

```
vi +10 myfile.tex
```

Uvedený příkaz otevře soubor myfile.tex a umístí kurzor na desátý řádek od začátku souboru.

Vyzkoušejte si příkazy uvedené v tomto oddíle a procvičte se v jejich používání. Většina uživatelů používá pouze jistou podmnožinu uvedených příkazů. V dalším oddílu si probereme příkazy umožňující text měnit.

A.4.2 Modifikace textu

Nyní je naším cílem měnit obsah textového souboru a editor vi za tímto účelem nabízí spoustu příkazů.

V tomto oddíle budeme probírat příkazy určené k editování textu, rušení textu a podobně. Až oddíl dočtete, budete mít dostatek znalostí potřebných k vytvoření jakéhokoliv textového souboru. Následující oddíly jsou pak zaměřeny na další užitečné příkazy.

Při vkládání textu lze vložit více textových řádků prostřednictvím klávesy Enter. Předpokládejme, že jste udělali chybu a že jste stále na řádku, ve kterém se chyba vyskytuje.

Pak můžete použít klávesu Backspace a vrátit se k místu, kde se nachází chyba. Pokud jde o klávesu Backspace, různé implementace editoru vi se chovají různě. Některé z nich pouze posunují kurzor zpět a zapsaný text nechávají nedotčený. Jiné text postupně zprava ruší. Některé implementace dokonce umožňují používat v módu zadávání textu kurzorové klávesy. To vše však nepatří k normálnímu chování editoru `vi`. Pokud je text viditelný a použijete klávesu Esc (přitom jste na řádku, ve kterém jste použili klávesu Backspace), pak budou znaky za kurzorem zrušeny. Funkci klávesy Backspace si budete muset vyzkoušet, abyste zjistili, jak se tato klávesa ve vaší konkrétní implementaci editoru vi chová.

- a — Přidání textu od aktuální pozice kurzoru
- A — Přidání textu na konec řádku
- i — Vložení textu vlevo od pozice kurzoru
- I — Vložení textu vlevo od prvního výskytu zobrazitelného znaku (non-white character) v aktuálním řádku
- o — Otevření nového řádku a vložení textu za aktuální řádek
- O — Otevření nového řádku a vložení textu před aktuální řádek

Nyní máme několik příkazů pro vkládání textu a dále si uvedeme příkazy pro zrušení textu. Editor `vi` má několik příkazů pro zrušení textu, které mohou být spojeny s modifikátorem.

- x — zrušení znaku, na jehož pozici je umístěn kurzor
- dw — zrušení textu od aktuální pozice kurzoru do konce slova
- dd — zrušení aktuálního řádku
- D — zrušení textu od aktuální pozice kurzoru do konce řádku

Prostřednictvím modifikátorů se síla příkazů zvýší. Následující příklady jsou pouze podmnožinou všech možností:

- nx — n-krát opakované zrušení znaku, na jehož pozici je umístěn kurzor
- ndd — zrušení n řádků
- dnw — zrušení n slov (stejnou funkci má příkaz ndw)
- dG — zrušení textu od aktuální pozice kurzoru do konce souboru
- d1G — zrušení textu od aktuální pozice kurzoru do začátku souboru
- d$ — zrušení textu od aktuální pozice kurzoru do konce řádku
- dn$ — zrušení textu od aktuální pozice kurzoru do konce n-tého řádku

Uvedené příklady ukazují, že operace zrušení textu mohou být velmi efektivní. Je to zejména evidentní tehdy, když tyto operace použijete ve spojení s příkazy pro změnu polohy kurzoru. Poznamenejme, že příkaz `D` tvoří výjimku, protože ignoruje jakékoli modifikátory.

Příležitostně nastanou situace, kdy budete chtít provedené změny vrátit zpět. Následující příkazy jsou určeny pro obnovení textu:

- u — zrušení posledně provedeného příkazu
- U — zrušení všech změn provedených v tomto řádku
- :e! — obnova stavu souboru od okamžiku, kdy byl naposledy uložen

Editor `vi` umožňuje nejen vracet zpět provedené změny, ale také vracet zpět vrácené změny. Například pomocí příkazu `5dd` zrušíte pět řádků a pak je obnovíte pomocí příkazu `u`. Když použijete příkaz `u` znovu, budou řádky opět zrušeny.

Nové verze editoru `vi` nabízejí příkazy, které umožňují editovat text v tzv. přepisovacím módu. Znamená to, že chcete-li měnit nějaký text, pak jej nemusíte nejdříve rušit.

r c	přepsání znaku na pozici kurzoru znakem
R	přepsání textu novým textem
c w	změna textu v aktuálním slově
c $	změna textu od aktuální pozice kurzoru do konce řádku
c n w	změna následujících n slov (totéž jako ncw)
c n $	změna do konce n-tého řádku
C	změna do konce řádku (totéž jako c$)
c c	změna aktuálního řádku
S	v textu, který právě píšete, nahradí aktuální znak
n s	v textu, který právě píšete, nahradí n aktuálních znaků

Série příkazů realizujících změny vám umožní zadat řetězec znaků, který musí být ukončen klávesou Esc.

Příkaz c w platí od aktuální pozice kurzoru ve slově do konce slova. Když použijete příkaz realizující změnu a přitom zadáte „vzdálenost“, editor `vi` zobrazí znak `$` na pozici, do které bude změna provedena. Nový text může být delší nebo kratší než text původní.

```
c w
```

A.4.3 Kopírování a přesouvání částí textu

Pro přesouvání textu existuje řada příkazů, jejichž kombinací lze dosáhnout kýženého efektu. V tomto oddíle popíšeme pojmenované a nepojmenované buffery spolu s příkazy umožňujícími „vystřihnout“ a „nalepit“ text.

Základní kopírování textu probíhá ve třech krocích:

1. Kopírování textu do bufferu („**vystřižení**“ textu).
2. **Nastavení kurzoru** na cílovou pozici.
3. Kopírování textu z bufferu na novou pozici („**nalepení**“ textu).

K vystřižení textu do nepojmenovaného bufferu použijte některý z následujících příkazů:

y y	Přesunutí kopie aktuálního řádku do nepojmenovaného bufferu.
Y	Přesunutí kopie aktuálního řádku do nepojmenovaného bufferu.
c y y	Přesunutí kopie následujících n řádků do nepojmenovaného bufferu.
c Y	Přesunutí kopie následujících n řádků do nepojmenovaného bufferu.

y w	Přesunutí slova do nepojmenovaného bufferu.
y n w	Přesunutí n slov do nepojmenovaného bufferu.
n y w	Přesunutí n slov do nepojmenovaného bufferu.
y $	Přesunutí textu od aktuální pozice do konce řádku.

Nepojmenovaný buffer je dočasná oblast paměti, jejíž obsah může být zrušen prostřednictvím jiných často používaných příkazů. Někdy se stane, že jistou část textu budete potřebovat po dlouhou dobu. V takovém případě byste měli použít pojmenovaný buffer. Editor `vi` nabízí 26 pojmenovaných bufferů. Jako identifikační jméno bufferu se používá jednoznakové písmeno. Pro rozlišení pojmenovaného bufferu editor `vi` používá znak ". Při odkazu na pojmenovaný buffer se používají malá písmena (pokud má být obsah bufferu zcela nahrazen) nebo velká písmena (pokud má být obsah bufferu doplněn). Uveďme si příklady:

" a y y	Přesunutí aktuálního řádku do pojmenovaného bufferu a.
" a Y	Přesunutí aktuálního řádku do pojmenovaného bufferu a.
" b y w	Přesunutí aktuálního slova do pojmenovaného bufferu b.
" B y w	Přidání aktuálního slova k obsahu pojmenovaného bufferu b.
" b y 3 w	Přesunutí následujících tří slov do pojmenovaného bufferu b.

Ke zkopírování („nalepení") obsahu bufferu použijte příkaz `p`:

p	Zkopírování obsahu nepojmenovaného bufferu VPRAVO od pozice kurzoru.
P	Zkopírování obsahu nepojmenovaného bufferu VLEVO od pozice kurzoru.
n P	Zkopírování obsahu nepojmenovaného bufferu n-krát VLEVO od pozice kurzoru.
" a p	Zkopírování obsahu pojmenovaného bufferu a VPRAVO od pozice kurzoru.
" b 3 P	Zkopírování obsahu pojmenovaného bufferu b třikrát VLEVO od pozice kurzoru.

Jestliže používáte editor `vi` v terminálovém okně systému X Window, pak máte k dispozici ještě jeden prostředek pro kopírování textů. Oblast textu, kterou chcete kopírovat, vyznačte kurzorem myši (levé tlačítko myši přitom držte stisknuté). Vyznačená oblast textu bude zobrazena inverzně a přitom se automaticky přenese do speciálního bufferu vyhrazeného pro systém X Window. Pokud chcete text „nalepit", stačí stisknout prostřední tlačítko myši. Nezapomeňte, že předtím musíte editor vi přepnout do vkládacího módu, protože jinak by mohl být vstup interpretován jako příkaz a výsledek by byl nepředvídatelný. Prostřednictvím stejné techniky lze kopírovat jednotlivá slova. Slovo se přenese do bufferu dvojnásobným klepnutím levým tlačítkem myši. Obsah bufferu se změní pouze tehdy, když se vyznačí nová oblast textu.

Přesouvání textu se rovněž odehrává ve třech krocích:

1. **Zrušení textu** a jeho současné zkopírování do pojmenovaného nebo nepojmenovaného bufferu.
2. **Nastavení kurzoru** na cílovou pozici.
3. **Kopírování textu** z pojmenovaného nebo nepojmenovaného bufferu na novou pozici („nalepení“ textu).

Proces je stejný jako kopírování textu, avšak v prvním kroku se text zruší. Když se provede příkaz `dd`, přesune se zrušený řádek do nepojmenovaného bufferu. Pak můžete obsah bufferu „nalepit“ stejným způsobem, jako při kopírování textu.

"ad zrušení řádku a přesunutí do pojmenovaného bufferu a

"aAdd zrušení čtyř řádků a přesunutí do pojmenovaného bufferu a

dw zrušení slova a přesunutí do nepojmenovaného bufferuDalší příklady na zrušení textu jsme uvedli v oddíle o modifikaci textu

V případě, že systém zhavaruje, obsahy pojmenovaného i nepojmenovaných bufferů se ztratí. Obsah editovaného bufferu však může být obnoven (viz oddíl „Užitečné příkazy“).

A.4.4 Vyhledávání a nahrazování textů

Editor `vi` má spoustu příkazů pro vyhledávání řetězců. Můžete vyhledávat jednotlivé znaky, ale také můžete při vyhledávání použít regulární výrazy.

Hlavní vyhledávací příkazy jsou `f` a `t`:

fc vyhledání následujícího znaku c vpravo od kurzoru

Fc vyhledání následujícího znaku c vlevo od kurzoru

tc přesunutí kurzoru na pozici vlevo od následujícího znaku c

Tc Přesunutí kurzoru na pozici vpravo od předcházejícího znaku c
(V některých verzích editoru vi je tento příkaz shodný s příkazem Fc.)

; opakování posledního příkazu f, F, t nebo T

, stejný příkaz jako ;, ale mění směr vyhledávání původního příkazu

Jestliže hledaný znak neexistuje, upozorní vás editor `vi` zvukovým nebo jiným signálem.

Editor `vi` rovněž umožňuje v editovaném textu hledat řetězec:

/str	vyhledání specifikovaného řetězce od aktuální pozice kurzoru
?str	vyhledání specifikovaného řetězce před aktuální pozicí kurzoru
n	opakování předcházejícího příkazu / nebo ?
N	opakování předcházejícího příkazu / nebo ?, přičemž se změní směr vyhledávání

Když použijete příkaz / nebo ?, „vyčistí“ se spodní řádek obrazovky. Pak v tomto řádku zadáte hledaný řetězec a stisknete klávesu Enter.

Řetězec uvedený za příkazy / a ? může být regulárním výrazem. V regulárním výrazu se prostřednictvím pseudoznaků popisují jisté množiny znaků. Pseudoznaky používané v editoru `vi` jsou následující: `.`, `*`, `[`, `]`, `^` a `$`. Následující seznam specifikuje význam pseudoznaků:

.	vyhovuje jakýkoliv znak kromě znaku "newline"
\	uvozuje jakýkoliv speciální znak
*	vyhovuje výskyt předcházejících znaků
[]	vyhovuje právě jeden znak uvedený v hranatých závorkách
^	vyhovuje právě jeden znak, jen pokud se nachází na začátku řádku
$	vyhovuje znak předcházející konci řádku
[^]	vyhovují znaky, které nejsou uvedeny v hranatých závorkách
[-]	Vyhovují znaky z uvedeného rozsahu

Nejjednodušší způsob, jak se naučíte používat regulární výrazy, spočívá v tom, že je budete používat. Vyzkoušejte si následující příklady:

c.pe	vyhovují slova jako cope, cape, caper a podobně
c\.pe	vyhovují slova jako c.pe, c.per a podobně
sto*p	vyhovují slova jako stp, stop, stoop a podobně
cat.*n	vyhovují slova jako carton, cartoon a podobně

`xyz.*`	vyhovují xyz do konce řádku
`^The`	vyhovují všechny řádky začínající na The
`atime$`	vyhovují všechny řádky končící na atime
`^Only$`	vyhovují všechny řádky, ve kterých se vyskytuje pouze slovo Only
`b[aou]rn`	vyhovují slova barn, born nebo burn
`Ver[D-F]`	vyhovují slova VerD, VerE nebo VerF
`Ver[^1-9]`	vyhovují slova začínající na Ver, po kterých nenásleduje žádná číslice
`the[ir][re]`	vyhovují slova their, therr, there a theie
`[A-Za-z][A-Za-z]*`	vyhovuje jakékoliv slovo

Editor vi používá příkazový mód `ex` k realizaci operací vyhledávání a náhrady řetězců. Všechny příkazy začínající dvojtečkou představují požadavek v módu `ex`.

Příkazy pro vyhledání a náhradu řetězce umožňují, aby byly realizovány v jistém řádkovém rozsahu. Uživatel může vyžadovat, aby byl před náhradou řetězce vyzván k potvrzení, zda se má konkrétní výskyt řetězce nahrazovat nebo ne. Uveďme si obecný tvar příkazu pro náhradu řetězce a několik příkladů:

...syntaxe obecného příkazu `:<start>,<finisf>s/<find>/<replace>/g`

`:1,$s/the/The/g`	vyhledá všechny výskyty slova the a nahradí je slovem The
`:%s/the/The/g`	znak % znamená "celý soubor". Tento příkaz provede stejnou funkci jako předcházející příkaz
`:.,5s/^.*//g`	zruší obsah editovaného souboru od aktuálního řádku po pátý řádek
`:%s/the/The/gc`	nahradí všechny výskyty slova the slovem The a před každou náhradou si vyžádá potvrzení
`:%s/^..../g`	zruší první čtyři znaky každého řádku

Jak je z předcházejících příkladů patrné, jsou příkazy pro vyhledávání a náhradu řetězců velmi výkonné, zejména když se kombinují s regulárními výrazy. Pokud se direktiva `g` v příkazu neuvede, provede se náhrada pouze u prvního výskytu hledaného řetězce.

Někdy se může stát, že původní vyhledávaný řetězec se má použít v nahrazujícím řetězci. Za tímto účelem nabízí editor `vi` některé speciální znaky:

`:1,5s/help/&ing/g`	v prvních pěti řádcích nahradí slovo help slovem helping
`:%s/ */&&/g`	v celém souboru zdvojnásobí počet mezer mezi slovy

Používání kompletního řetězce má v editoru `vi` jisté omezení, protože k stanovení rozsahu náhrady editor používá kulaté závorky `(` a `)`. V takovém případě je nutno použít znak `\`:

`:s/^\(.*\):.*/\1/g`	zruší vše, co následuje po dvojtečce včetně dvojtečky samé
`:s/\(.*\):\(.*\)/\2:\1/g`	vymění slova na obou stranách dvojtečky

Posledně uvedené příklady asi budete muset číst velmi pozorně. Editor `vi` nabízí velmi výkonné příkazy, které moderní editory zpravidla nemají. Za to se ovšem platí jistá cena - naučit se všechny příkazy editoru vi není vůbec snadné. Pokud jsme vás neodradili, pokuste se znovu si projít uvedené příklady a uvidíte, že se používání editoru `vi` pro vás stane zcela přirozenou záležitostí.

B

General Public License

Tento dodatek se nachází na CD ke knize „Linux - dokumentační projekt“.

C

Library General Public License

Tento dodatek se nachází na CD ke knize „Linux - dokumentační projekt“.

LINUX

Příručka správce operačního systému Linux

Kapitola

1

Úvod

Na počátku byl soubor nesličný a pustý, a prázdno se vznášelo nad povrchem bitů. A prsty Autorovy dosedly na povrch klávesnice i řekl Autor: Buďte slova! A stalo se tak.

Manuál „Průvodce správce operačního systému Linux“ popisuje ty aspekty používání operačního systému, jež se vztahují k jeho správě neboli administraci. Je určený lidem, kteří o správě operačního systému neví zhola nic (ptají se teď, co to je), avšak zvládají přinejmenším základy jeho běžného užívání. Nenaleznete zde návod, jak Linux instalovat. Instalace systému je podrobně popsaná v dokumentu „Průvodce instalací a začátky“. Bližší informace o sadě manuálů systému Linux jsou uvedeny níže.

Administrací (správou systému), rozumíme všechny činnosti, které je nutno pravidelně vykonávat, aby počítačový systém zůstal v provozuschopném stavu. Zahrnuje například zálohování souborů (v případě potřeby jejich obnovování), instalaci nových programů, vytváření uživatelských účtů (jejich mazání v případě, že jsou nepotřebné), kontroly a opravy případných poškození systému souborů a další. Když si počítač představíte jako dům, pak by správou systému byla jeho údržba. Ta by zahrnovala například úklid, zasklívání rozbitých oken a další podobné věci. Místo prostého pojmu „údržba systému“ se používá termín „administrace“, aby tato oblast nevypadala na první pohled zas až tak jednoduše.[1]

Příručka je strukturovaná tak, že většinu kapitol můžete číst nezávisle na sobě. Když například hledáte nějaké informace o zálohování, stačí, když si přečtete příslušnou kapitolu.[2] Doufáme, že díky tomu bude možné používat tuto knihu i jako referenční příručku a v případě

1 Přesto se najdou lidé, kteří toto označení *používají*, ale jenom proto, že ještě nikdy nečetli tento manuál. Nešťastníci...

2 Tedy, jestli se k vám – zcela náhodou, dostane verze, která kapitolu o zálohování obsahuje.

potřeby místo „louskání“ celého textu číst pouze jeho menší části. Přesto manuál zůstává především a převážně učebnicí a jakýmsi průvodcem. To, že poslouží i jako příručka referencí je pouze šťastným řízením osudu.

Nelze předpokládat, že by tato knížka pokryla celou problematiku administrace systému. Správce systému bude potřebovat řadu další dokumentace operačního systému Linux. Koneckonců, administrátor je v podstatě jenom uživatel, který má zvláštní práva a povinnosti. Velmi významným pramenem jsou i manuálové stránky, po kterých by správce měl sáhnout pokaždé, když si není funkcí některého příkazu zcela jist.

Manuál je zaměřen především na operační systém Linux, ale v obecných principech může být užitečný i pro správce jiných unixových systémů. Bohužel je mezi různými verzemi Unixu tolik rozdílů (o správě systému to platí dvojnásob), že není prakticky možné postihnout všechny známé variety. Je totiž obtížné – vezmeme-li v potaz způsob, jakým se Linux vyvíjí – pokrýt i všechny možnosti tohoto operačního systému samotného.

Kromě toho neexistuje jediná oficiální distribuce Linuxu od jediného výrobce. Různí lidé používají různá nastavení a konfigurace. Navíc si řada uživatelů vytváří své vlastní. Proto tato kniha není zaměřená na některou z konkrétních distribucí (i když autor osobně dává téměř výlučně přednost systému Debian GNU/Linux). V rámci možností se v příručce snažíme upozornit na takovéto odlišnosti a objasnit i jiné možné alternativy.

Než abychom podali strohý seznam „pěti jednoduchých kroků“ pro řešení každého úkolu, dáváme přednost popisu základních principů, tedy objasnění toho, jak věci doopravdy fungují. V manuálu proto najdete hodně informací, které nejsou nezbytné pro každého správce. Takovéto části jsou v textu označeny a v případě, že používáte systém s předem nastavenou konfigurací, můžete je klidně přeskočit. Pochopitelně, přečtete-li si knihu celou, proniknete do systému hlouběji, no a pak by pro vás mohly být o něco příjemnější i jeho používání a jeho správa.

Tak jako vše ostatní spojené s vývojem Linuxu, byla i tato práce založená na principu dobrovolnosti. Pustili jsme se do ní, protože jsme si mysleli, že by to mohla být zábava. Dalším důvodem byl pocit, že je potřeba tuto práci udělat. Přesto – jako konečně u každé dobrovolné práce – jsou určité hranice nasazení a úsilí, které můžete vynaložit. Navíc vás omezuje také to, kolik vědomostí a zkušeností máte. Přirozeně, manuál není tak dobrý, jak by mohl být v případě, že by přišel někdo s kouzelnou hůlkou a dobře zaplatil za jeho napsání. Pak by bylo možné strávit i několik dalších let jeho zdokonalováním. Samozřejmě si myslíme, že je celkem povedený, takže to berte jako varování pro každý případ.

Je jeden konkrétní bod, ve kterém jsme manuál dost „ořezali“ – není v něm vyčerpávajícím způsobem popsaná řada věcí, které již jsou podrobně zdokumentované v jiných, volně dostupných příručkách. Vztahuje se to zvlášť na dokumentaci k jednotlivým programům. Neuvádíme například všechny podrobnosti použití programu `mkfs`. Popisujeme je-

nom funkci programu a pouze tolik z jeho dalších možností, kolik je potřeba pro dosažení účelu této knihy. Laskavého čtenáře, jenž hledá podrobnější informace, odkazujeme na onu další dokumentaci. Převážná většina dokumentů, na které se odvoláváme v odkazech, je součástí úplné sady dokumentace k operačnímu systému Linux.

Pokoušeli jsme se napsat tuto příručku co nejlépe, ale budeme vděčni za všechny vaše nápady jak ji vylepšit. Gramatické a věcné chyby, nápady týkajíce se nových oblastí, o které by bylo možno knihu rozšířit, opakující se části, informace o rozdílech mezi různými verzemi Unixu – to všechno jsou připomínky, které se zájmem očekáváme. Kontaktní informace najdete prostřednictvím služby World Wide Web na adrese *http://www.iki.fi/liw/mail-to-lasu.html*. Na této stránce najdete i pokyny potřebné pro doručení elektronické pošty přes filtry nevyžádaných zpráv.

Při práci na této knize nám přímo či nepřímo pomáhalo mnoho lidí. Rádi bychom zvlášť poděkovali Mattu Welshovi za inspiraci a vedení projektu LDP; Andy Oramovi za to, že nás znovu a znovu zaměstnával řadou velmi podnětných připomínek; Olafu Kirschovi za to, že nám dokázal, že vše lze zvládnout; Adamu Richterovi z Yggdrasil a dalším za to, že nám ukázali, že tato práce může být zajímavá i pro jiné lidi.

Stephen Tweedie, H. Peter Anvin, Rémy Card, Theodore Ts'o a Stephen Tweedie odvedli kus práce, kterou jsme si formou odkazů a referencí „zapůjčili" (tím pádem je naše kniha na pohled tenčí a o to víc působivá). Za toto jsme vděční vůbec nejvíc a zároveň se velmi omlouváme za předchozí verze manuálu, které občas v některých oblastech postrádaly odpovídající úroveň.

Kromě toho patří náš dík Marku Komarinskemu za jeho materiály z roku 1993 i mnoho dalších sloupků v Linux Journalu, jež se týkaly problematiky správy systému. Jsou velmi informativní a inspirující.

Dostali jsme množství užitečných připomínek od velkého počtu dalších lidí. Díky malé „černé díře" v našem archivu nelze dohledat všechna jména, takže alespoň některá z nich (v abecedním pořadí): Paul Caprioli, Ales Cepek, Marie-France Declerfayt, Dave Dobson, Olaf Flebbe, Helmut Geyer, Larry Greenfield a jeho otec, Stephen Harris, Jyrki Havia, Jim Haynes, York Lam, Timothy Andrew Lister, Jim Lynch, Michael J. Micek, Jacob Navia, Dan Poirier, Daniel Quinlan, Jouni K Seppänen, Philippe Steindl, G. B. Stotte. Omlouváme se všem, na které jsme zapomněli.

Linux – dokumentační projekt

Projekt tvorby dokumentace pro operační systém Linux (angl. The Linux Documentation Project, zkráceně LDP) je volné sdružení autorů, korektorů a editorů, kteří spolupracují na úplné dokumentaci systému. Hlavním koordinátorem projektu je Greg Hankins.

Tento manuál je jedním ze sady dokumentů, které jsou v rámci LDP šířeny. Tato sada obsahuje Průvodce uživatele systému, Průvodce správce systému, Průvodce správce sítě, Průvodce jádrem systému. Všechny příručky jsou k dispozici ve formátu zdrojového kódu pro sázecí systém LaTex, ve formátu .dvi a v jazyce Postscript, a to na anonymním serveru FTP s adresou `ftp://sunsite.unc.edu`, v adresáři `/pub/Linux/docs/LDP`.

Rádi bychom v závěru dodali odvahy všem čtenářům se sklonem k tvůrčímu psaní či redigování, aby se připojili k naší snaze o zdokonalení dokumentace operačního systému Linux. Máte-li zájem a přístup k účtu pro elektronickou poštu, kontaktujte prostřednictvím Internetu Grega Hankinse na adrese `gregh@sunsite.unc.edu`.

Óda na LDP[3]

Jak úžasná věc
a krásná
napsat knihu.

Zpíval bych rád
i o potu, krvi a slzách.
i z těch kniha vzniká.

Začalo to kdysi
v dvaadevadesátém,
uživatelé kňučeli
„Nemůžem nic dělat!“

Chtěli jenom vědět,
v čem je jejich problém
a jak jej vyřešit
(nejlépe do zítřka).

Dali jsme jim odpovědi
v Č-K-D[4] pro Linux,
doufajíce, že je amen –
hlavně žádné další psaní.

„Příliš dlouhé,
nepřehledné,
k nepřečtení –
ať děláme, co děláme!“
Pak několik z nás
spojilo síly
(samozřejmě virtuálně)
a odstartoval LDP.

Začali jsme psát,
plánovat,
přinejmenším několik knih
pro každou z oblastí.

Začátek byl zábavný,
dlouhé hovory,
hrubý obrys
a pak zával.

Padlo ticho,
začala práce,
někdo psal míň,
někdo víc.

Prázdná obrazovka,
ach, ta hrůza,
sedí a směje se
vám do očí.

Stále čekáme
na poslední den,
kdy řekneme
hotovo.

Než přijde,
je vše, co máme,
pouhým návrhem,
čekajícím na vaše
připomínky.

3 Autor si přeje zůstat v anonymitě (dílko zaslal do diskusní skupiny LDP Matt Welsh).

4 Často kladené dotazy, angl. Frequently-Asked Question, zkráceně F-A-Q (poznámka překladatele).

Kapitola

2

Operační systém Linux – přehled

A viděl Bůh, že vše, což učinil, bylo velmi dobré.
Genesis 1:31

Tato kapitola podává zevrubný přehled o operačním systému Linux. V první části jsou popsány nejdůležitější ze služeb, jež systém nabízí. Další části se bez přílišných podrobností zabývají programy, které popsané služby realizují. Cílem kapitoly je podat výklad principů systému jako celku s tím, že každá část bude podrobněji probraná později, na jiném místě knihy.

2.1 Různé části operačního systému

Operační systém typu Unix se skládá z **jádra systému** a **systémových programů**. Uživatel systému pracuje s **aplikačními programy**. Jádro je srdcem operačního systému[1]. Udržuje záznamy souborů na disku, spouští programy, řídí jejich současný běh, přiděluje paměť a další technické prostředky různým procesům, přijímá a odesílá pakety z a do počítačové sítě a tak dál. Jádro systému samotné toho dělá velmi málo, ale poskytuje základní služby různým nástrojům, pomocí kterých mohou být realizovány všechny ostatní služby. Jádro rovněž hlídá, aby nikdo nemohl přistupovat k zařízením přímo. Když chtějí uživatelé a procesy používat technické prostředky, musí používat nástroje, které nabízí jádro systému. Tímto způsobem je zabezpečená i vzájemná ochrana uživatelů. Nástroje jádra systému, o nichž byla řeč, lze využívat prostřednictvím **volání systému** (angl. **system calls**). Podrobnější informace o systémových voláních uvádí sekce 2 manuálových stránek.

[1] Skutečně často se jádro systému chybně ztotožňuje se samotným operačním systémem. Ale operační systém poskytuje ve srovnání s prostým jádrem o hodně více služeb.

Systémové programy realizují služby, které se vyžadují od operačního systému. Využívají při tom nástroje, které nabízí jádro systému. Systémové i všechny ostatní programy běží jakoby „na povrchu“ jádra. Říká se tomu **uživatelský režim** (angl. **user mode**). Rozdíl mezi systémovými a aplikačními programy je v jejich určení. Pomocí aplikačních programů mohou uživatelé dělat některé užitečné věci (popřípadě se bavit – je-li aplikace, kterou si zrovna spustili, počítačová hra). Systémové programy jsou potřebné k tomu, aby systém vůbec fungoval. Textový editor je aplikace, `telnet` je systémový program. Hranice mezi aplikačními a systémovými programy je často dost neostrá. Lze prohlásit, že takovéto rozdělení je samoúčelné – důležité čistě pro vymezení samotných klasifikačních kritérií.

Součástí operačního systému mohou být i překladače programovacích jazyků a jejich knihovny (v případě Linuxu knihovny překladačů GCC a C). Součástí operačního systému ale nejsou všechny programovací jazyky. Naopak, za jeho součást se často považuje dokumentace, někdy dokonce i některé hry. Tradičně se za operační systém pokládá obsah jeho instalační pásky, či instalačních disků. Pokud jde o systém Linux, není uvedená definice zcela jasná, protože na mnoha serverech FTP po celém světě existuje množství různých instalací systému.

2.2 Důležité části jádra systému

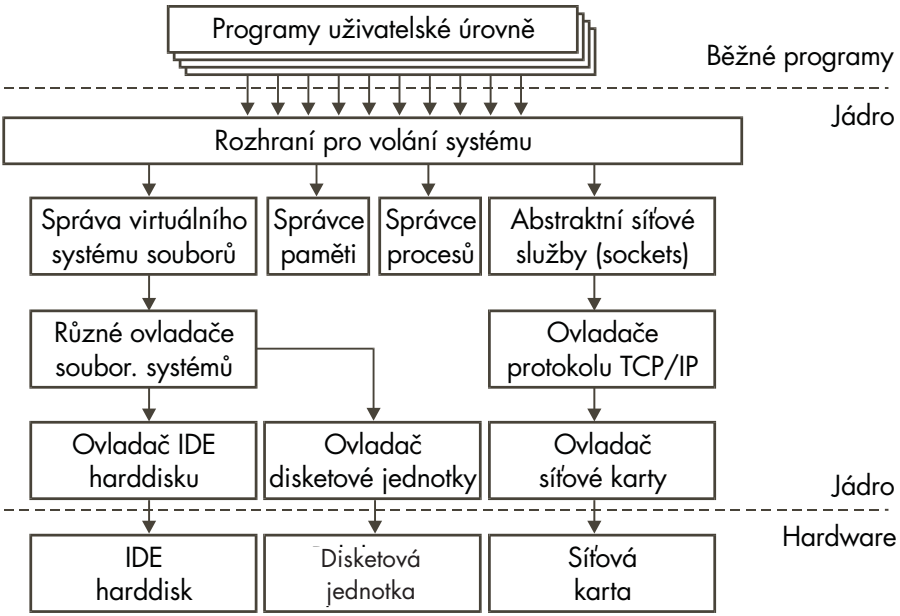
Jádro Linuxu sestává z několika důležitých subsystémů. Jsou to části řízení procesů, správy paměti, ovladačů technických prostředků, ovladačů souborových systémů, správy sítě a různé další kusy a kousky. Některé z nich jsou zobrazeny na obrázku 2.1.

Snad nejdůležitějšími subsystémy (bez nichž nic jiného nefunguje) jsou správa paměti a správa procesů. Subsystém správy paměti zajišťuje přidělování paměťových oblastí a odkládacího prostoru (angl. swap space) jednotlivým procesům, částem jádra a vyrovnávací paměti (angl. buffer cache). Subsystém správy procesů vytváří procesy a přepínáním mezi aktivními procesy, které využívají procesor, zabezpečuje multitasking.

Jádro systému na nejnižší úrovni obsahuje ovladače pro všechny druhy technických zařízení, které operační systém podporuje. Vzhledem k tomu, že na světě existuje celá řada různých typů hardwaru, je počet ovladačů zařízení velký. Je ale mnoho jinak podobných zařízení, které se často liší pouze v tom, jak spolupracují s programy. Takovéto podobnosti umožňují definovat obecné třídy ovladačů, jež podporují podobné operace. Každý člen takovéto třídy má stejné rozhraní k ostatním částem jádra. Liší se v tom, jak tyto operace implementuje. Například všechny ovladače disků vypadají pro zbytek jádra podobně. To znamená, že všechny znají operace jako „inicializuj diskovou jednotku“, „čti sektor N“ a „zapiš sektor N“.

Některé softwarové služby, jež poskytuje jádro samotné, mají rovněž podobné vlastnosti. Proto mohou být také rozdělené do tříd. Ku příkladu různé síťové protokoly byly vyčleněné do jednoho programového rozhraní – knihovny „BSD socket library“. Dalším příkladem je

vrstva **virtuálního souborového systému** (angl. **virtual filesystem**, zkráceně VFS). Ta odděluje operace souborového systému od jejich implementace. Každý typ souborového systému obstarává implementaci určité množiny operací, společné všem systémům souborů. Když se některý z prvků systému pokouší využít určitý souborový systém, žádost jde přes VFS. Ten ji směřuje k požadovanému ovladači konkrétního systému souborů.



Obrázek 2.1
Některé z důležitějších částí jádra systému Linux

2.3 Nejdůležitější služby v unixovém systému

Tato podkapitola popisuje některé významnější služby systému Unix, avšak opět bez větších podrobností. Všechny služby budou později podrobně vysvětlené v dalších kapitolách.

2.3.1 Proces init

Nejdůležitější služby v systému Unix poskytuje proces `init`. Spuštění procesu `init` (jako prvního z procesů) je v každém unixovém systému posledním krokem, který provede jádro systému při zavádění. Po spuštění proces `init` pokračuje v proceduře zavádění systému. Vykonává různé úkoly, které se při spouštění systému obvykle provádí (kontroluje a připojuje souborové systémy, spouští démony atd.).

Přesný seznam úloh, které `init` při zavádění dělá, závisí na verzi tohoto programu i operačního systému. Je na výběr několik možností. Proces `init` často obstarává tzv. **jednouživatelský režim** (angl. **single user mode**). V jednouživatelském režimu se do systému nemůže nikdo přihlásit. Příkazový interpret může z konzoly používat pouze superuživatel (správce). Běžným režimem práce je **víceuživatelský režim** (angl. **multiuser mode**). Tyto režimy práce některé systémy Unix zobecňují do tzv. **úrovní běhu systému** (angl. **run levels**). Jednouživatelský a víceuživatelský režim tak představují dvě různé úrovně, na kterých může systém běžet. Kromě nich mohou existovat i další. Například úroveň, při které se na konzole spustí grafické rozhraní X Window a podobně.

V běžné situaci program `init` kontroluje, zda fungují procesy `getty` (umožňující uživatelům připojit se do systému) a adoptuje procesy – sirotky. Sirotci jsou procesy, jejichž rodičovské procesy byly z různých důvodů ukončeny – říká se, že „umřeli“. V systému Unix *musí* být *všechny* procesy součástí jediné hierarchické stromové struktury. Proto musí proces `init` sirotky adoptovat.

Když se systém vypíná, proces `init` zodpovídá za ukončení všech ostatních procesů, odpojení všech souborových systémů, zastavení procesoru a za vše ostatní, co má podle dané konfigurace udělat.

2.3.2 Přihlášení z terminálů

Přihlášení uživatelů prostřednictvím terminálů (pomocí sériových linek) a konzoly (v případě, že neběží X Window) do systému obstarává program `getty`. Proces `init` spouští zvláštní instanci `getty` pro každý terminál, ze kterého se bude možno do systému přihlásit. Program `getty` dále čte zadávané uživatelské jméno a spouští další program `login`, jenž čte přístupové heslo. Jestli jsou uživatelské jméno a heslo správné, spustí program `login` příkazový interpret neboli shell. Když je příkazový interpret ukončen – jakmile se uživatel odhlásí ze systému, nebo když je program `login` ukončen proto, že nesouhlasí uživatelské jméno a heslo – proces `init` to zjistí a spustí pro daný terminál novou instanci programu `getty`. Samotné jádro systému nemá vůbec pojem o přihlašování uživatelů do systému. Všechno kolem toho obstarávají systémové programy.

2.3.3 Syslog

Jádro systému i mnoho systémových programů hlásí různé chyby, vypisuje varování a jiná hlášení. Velmi často je důležité, aby bylo možno tyto zprávy prohlížet později, dokonce i s velkým časovým odstupem. Je tedy vhodné je zapisovat do nějakých souborů. Program, který to má na starost, se jmenuje `syslog`. Lze jej nastavit tak, aby třídil zprávy a hlášení do různých souborů, a to podle původce, případně stupně významnosti. Hlášení jádra systému

jsou obvykle směřována do jiného souboru, než hlášení jiných procesů a programů. Jsou většinou významnější a je potřeba číst je pravidelně, aby bylo možné rozeznat případné problémy v zárodku.

2.3.4 Periodické vykonávání příkazů: cron a at

Uživatelé i správci systému často potřebují spouštět některé programy pravidelně. Například administrátor systému, který musí sledovat zaplněnost disku, by mohl chtít pravidelně spouštět příkaz, jenž by „vyčistil" adresáře dočasných souborů (`/tmp` a `/var/tmp`). Program by odstranil starší dočasné soubory, které po sobě programy z různých důvodů korektně nesmazaly.

Takovéto služby nabízí program `cron`. Každý uživatel má vlastní tabulku `crontab`, jež obsahuje seznam příkazů, které chce vlastník spustit, a časy, kdy se mají tyto příkazy provést. Démon `cron` má na starost spouštění těchto příkazů v požadovaném čase.

Služba `at` je podobná službě `cron`. Provede se ale jenom jednou. Příkaz je vykonán v určeném čase, ale jeho spouštění se neopakuje.

2.3.5 Grafické uživatelské rozhraní

Unix a Linux nezačleňují uživatelská rozhraní do jádra systému. Místo toho je implementují pomocí programů uživatelské úrovně. To se týká jak textového módu, tak grafického uživatelského prostředí.

Díky takovémuto řešení je samotný systém flexibilnější. Má to ale nevýhodu v tom, že je na druhou stranu velmi jednoduché implementovat pro každý program různá uživatelská rozhraní. Důsledkem je, že se takovýto systém uživatelé pomaleji učí.

Grafické prostředí, které Linux používá primárně, se nazývá „X Window System" (zkráceně X). Ale ani X přímo neimplementují uživatelské rozhraní. X Window pouze zavádí systém oken, tedy sadu nástrojů, pomocí kterých může být grafické uživatelské rozhraní implementované.

Tři z nejpopulárnějších stylů uživatelských rozhraní postavených na X jsou Athena, Motif a Open Look.*

* Poznámka korektora: V současné době patří mezi nejpopulárnější uživatelská rozhraní KDE (http://www.kde.org) nebo GNOME (http://www.gnome.org).

2.3.6 Komunikace prostřednictvím počítačové sítě

Komunikace pomocí počítačové sítě neboli síťování (angl. networking) je propojení dvou nebo více počítačů tak, že mohou komunikovat navzájem každý s každým. V současnosti používané metody propojování a komunikace jsou o něco komplikovanější, ale výsledný efekt stojí za to.

Operační systémy Unix mají řadu síťových funkcí. Většinu základních služeb – služby souborových systémů, tisky, zálohování atd., lze využívat i prostřednictvím sítě. To ulehčuje správu systému a umožňuje centralizovanou administraci. Zachovávají se výhody mikropočítačové technologie i přínos distribuovaných systémů (nižší náklady a lepší odolnost vůči poruchám).

Tato kniha se komunikací prostřednictvím počítačové sítě zabývá jenom zběžně. Podrobnosti o této problematice, včetně základního popisu principů počítačových sítí, přináší „Průvodce správce sítě“.

2.3.7 Přihlášení do systému ze sítě

Přihlášení do systému ze sítě funguje trochu odlišně, než běžné přihlášení přes terminál. Pro každý terminál, prostřednictvím kterého je možné se přihlásit, je vyhrazená samostatná fyzická sériová linka. Pro každého uživatele, který se přihlašuje prostřednictvím sítě, existuje jedno samostatné virtuální síťové spojení. Tímto spojením se může realizovat libovolný počet běžných přihlášení[2]. Proto není možné, aby běžely samostatné procesy `getty` pro všechna možná virtuální spojení. Kromě toho existuje několik různých způsobů přihlášení prostřednictvím sítě. Dva nejdůležitější způsoby v sítích TCP/IP jsou `telnet` a `rlogin`.

Síťová přihlášení mají místo řady procesů `getty` jednoho démona pro každý ze způsobů připojení (`telnet` a `rlogin` mají každý vlastního démona). Tento démon vyřizuje všechny přicházející žádosti o přihlášení. Dostane-li takovouto žádost, spustí svou novou instanci. Nová instance pak obsluhuje tuto jedinou žádost a původní instance nadále sleduje další příchozí žádosti o přihlášení. Nová instance pracuje podobně, jako program `getty`.[*]

2.3.8 Síťové souborové systémy

Jednou z nejužitečnějších věcí, kterou lze využít díky síťovým službám, je sdílení souborů pomocí **síťového souborového systému** (angl. **network file system**). Jeden z nejběžněji používaných typů se nazývá Network File System, zkráceně NFS, a byl vyvinut společností Sun.

---

2 No, může jich být přinejmenším mnoho. Ještě stále je totiž šířka přenosového pásma sítí problémem, takže existuje jakási praktická horní hranice počtu současných přihlášení do systému prostřednictvím jediného síťového spojení.

* Poznámka korektora: Od služeb `telnet` a `rlogin` se upouští a nahrazuje je program `ssh`.

V síťovém souborovém systému jsou všechny operace se soubory, které dělá program na jednom počítači, odesílány prostřednictvím počítačové sítě na jiný počítač. Pro program, který běží na lokálním počítači vzniká iluze, že soubory, které se nachází na vzdáleném počítači, jsou ve skutečnosti umístěny na počítači, na němž tento program běží. Takovýmto způsobem je velmi jednoduché sdílet informace, navíc lze používat již existující programy bez toho, že by je bylo potřeba měnit.

2.3.9 Pošta

Elektronická pošta je obvykle tím nejdůležitějším způsobem počítačové komunikace. Elektronický dopis je uložený v souboru se zvláštním formátem. K jeho odeslání nebo přečtení se používají speciální programy.

Každý uživatel systému má vlastní **schránku na příchozí poštu** (angl. **incoming mailbox**). Je to soubor určitého formátu, ve kterém jsou uložené všechny nově příchozí zprávy. Když někdo odesílá poštu, program zjistí adresu poštovní schránky příjemce a připojí dopis k jeho souboru s příchozí poštou. Jestli je schránka příjemce na jiném počítači, je dopis odeslán na tento stroj a ten se bude snažit doručit jej do schránky příjemce.

Systém elektronické pošty se skládá z několika typů programů. Doručení pošty do místních nebo vzdálených poštovních schránek má na starost první z nich - **agent pro přenos pošty** (angl. **Mail Transfer Agent** - zkráceně **MTA**), např. `sendmail` nebo `smail`. Uživatelé používají ke čtení pošty množství různých programů, tzv. **uživatelských poštovních agentů** (angl. **Mail User Agent** - zkráceně **MUA**), např. `pine` nebo `elm`. Poštovní schránky uživatelů jsou obvykle uloženy v adresáři `/var/spool/mail`.

2.3.10 Tisk

Tiskárnu může současně využívat pouze jeden uživatel. Nesdílet tiskárny mezi uživateli je ale dost neekonomické. Tiskárnu proto řídí program, jenž realizuje tzv. **tiskovou frontu**. Všechny tiskové úlohy všech uživatelů systému jsou zařazeny do fronty. Hned, jak tiskárna ukončí jednu úlohu, automaticky se jí odesílá další v pořadí. Uživatelé si nemusí zabezpečovat frontu požadavků na tisk organizačně a odpadá i nutnosti soupeřit a handrkovat se o přístup k tiskárně.[3]

Program pro obsluhu tiskové fronty navíc **ukládá metodou „spool“** všechny tiskové výstupy na disk, takže pokud je tisková úloha ve frontě, je text uložen v nějakém souboru. Tento mechanismus aplikačním programům umožňuje rychle odeslat tiskové úlohy programu, jenž tis-

[3] Nikdo se od programu, jenž obsluhuje tiskovou frontu, přesně nedoví, kdy bude jeho tiskový výstup skutečně ukončený. Uživatelé tedy místo toho - čekajíce na své tiskové výstupy - vytvoří novou frontu *u* tiskárny. To je ohromný příspěvek v oblasti podpory sociálních vztahů na pracovišti.

kovou frontu obsluhuje. Aplikace sama tak může pokračovat ve své práci. Nemusí čekat, než se úloha, která se právě tiskne, ukončí. To je v mnoha případech skutečně výhodné. Umožní vám to například zahájit tisk jedné verze dokumentu, přičemž nemusíte čekat, než se tisk ukončí, a můžete mezitím pracovat na nové, zcela pozměněné verzi.

2.4 Základní rysy systému souborů

Souborový systém je rozdělený na několik částí. Obvykle jsou hierarchicky uspořádané a nejvýše stojí kořenový souborový systém „root“ (angl. root filesystem). Říká se mu rovněž kořenový svazek, označuje se „/“. Souborový systém „root“ obsahuje adresáře `/bin`, `/lib`, `/etc`, `/dev` a několik dalších. Dalším je systém souborů `/usr`. Obsahuje programy a data, která se nemění. Následuje souborový systém `/var`. Ukládají se v něm data, jež se naopak často mění (například tzv. log-soubory). Posledním je souborový systém `/home`.

Ukládají si v něm svá data a soubory uživatelé systému. Rozdělení souborového systému na jednotlivé svazky může být jiné. Závisí zejména na hardwarové konfiguraci systému a rozhodnutích jeho správce. Všechna data a soubory mohou být nakonec uložené i v jediném systému souborů.

Některé další detaily týkající se uspořádání systému souborů popisuje kapitola 3. Ještě více podrobností o tomto tématu přináší „Standard systému souborů operačního systému Linux“*.

* Poznámka korektora: Dokument „Standard systému souborů operačního systému Linux“ se nyní jmenuje FHS (Filesystem Hierarchy Standard).

Kapitola

3

Přehled struktury adresářů

Dva dny nato seděl Pooh na své větvi, pohupoval nohama
a tam, vedle něj, stály čtyři hrníčky medu. . .
(A. A. Milne)

Kapitola popisuje důležité části standardní struktury adresářů operačního systému Linux, která je založená na systému souborů standardu FSSTND. Načrtneme v ní také běžný způsob rozdělení struktury adresářů do samostatných souborových systémů (svazků) s odlišnými účely a tento způsob rozdělení zdůvodníme. Naznačíme i některé alternativní způsoby rozdělení.

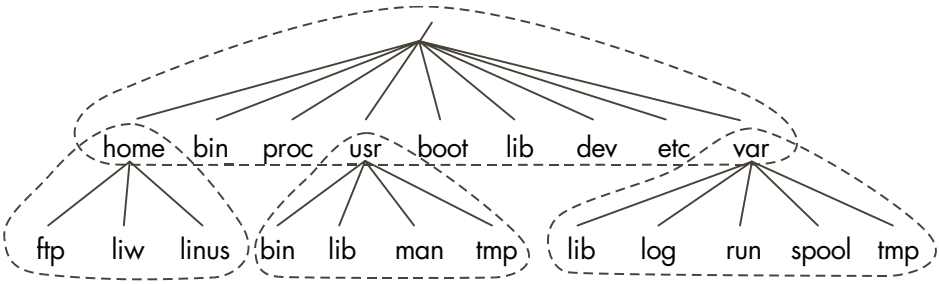
3.1 Základy

Tato kapitola volně vychází z normy „Standard systému souborů operačního systému Linux FSSTND, verze 1.2“ (viz bibliografie [Qui95]), který je pokusem zavést jisté konvence do organizace adresářového stromu operačního systému Linux. Výhodou přijetí takovéto normy je, že když bude vše na svém obvyklém místě, bude jednodušší psát programy a přenášet na Linux software z jiných platforem. Zároveň to ulehčí správu počítačů, na kterých běží operační systém Linux. I když neexistuje autorita, která by vývojáře, programátory a distributory donutila přizpůsobit se této normě, je její podpora v současnosti součástí většiny (ne-li všech) distribucí Linuxu. Není vhodné se neřídit standardem FSSTND, nejsou-li pro to velmi závažné důvody. Norma FSSTND se snaží sledovat tradice Unixu i současné trendy jeho vývoje. Motivací je snaha o usnadnění přechodu na Linux pro uživatele, kteří mají zkušenosti s jinými systémy Unix a naopak.

Tato kapitola není tak detailní jako FSSTND. Správce systému – chce-li plně proniknout do problematiky systémů souborů – by si měl přečíst i normu FSSTND.

Kapitola rovněž nepopisuje podrobnosti týkající se všech typů souborových systémů. Nebylo také cílem popsat všechny adresáře a konfigurační soubory, ale nabídnout čtenáři přehled o celém systému z perspektivy systému souborů. Podrobnější informace o popisovaných souborech jsou k dispozici na jiných místech této knihy, případně na manuálových stránkách.

Celou stromovou strukturu adresářů je možné rozdělit na menší části, tzv. svazky. Každá z těchto částí může být umístněna na vlastním disku nebo samostatné diskové oblasti – logické sekci. Tak se lze jednoduše přizpůsobit omezením velikosti disků a zároveň usnadnit zálohování i ostatní úkoly spojené se správou systému. Nejdůležitější z těchto částí jsou souborový systém „root“ (kořenový svazek), dále souborové systémy `/usr`, `/var` a `/home` (viz obrázek 3.1). Každý systém souborů má jiné určení. Adresářová stromová struktura byla navržená tak, aby fungovala i v síti počítačů s operačním systémem Linux. Uživatelé a programy tak mohou pomocí sítě sdílet některé části systémů souborů, a to buď prostřednictvím zařízení určených pouze pro čtení (např. CD-ROM), nebo pomocí sítě se systémem NFS.



Obrázek 3.1
Části adresářové struktury Unixu. Přerušované čáry označují hranice diskových oblastí.

Určení takto vymezených částí adresářové struktury je popsané v dalším textu.

- Souborový systém „root“ (kořenový svazek) je specifický pro každý počítač. Obecně je uložen na lokálním disku (avšak může to být i virtuální disk v paměti RAM – tzv. „ramdisk“, nebo síťová disková jednotka). Kořenový svazek obsahuje soubory nutné pro zavedení systému a jeho uvedení do stavu, ve kterém mohou být připojené ostatní souborové systémy. Obsah kořenového souborového systému postačuje pro práci v jednouživatelském režimu. Na tomto svazku jsou rovněž uloženy nástroje pro opravy poškozeného souborového systému a pro obnovení ztracených souborů ze záloh.

- Souborový systém `/usr` obsahuje všechny příkazy, knihovny, manuálové stránky a jiné soubory, jejichž obsah se nemění a které uživatel potřebuje při běžném provozu. Žádný ze souborů svazku `/usr` by neměl být specifický pro některý počítač. Rovněž by se neměl při normálním provozu měnit. Tyto podmínky zaručují, že soubory uložené v souborovém systému `/usr` bude možné efektivně sdílet v síti. Sdílení tohoto svazku je výhod-

né jak z hlediska nákladů – šetří se tím místo na disku (v souborovém systému `/usr` mohou být uloženy stovky megabajtů dat), tak z hlediska usnadnění správy systému (např. při instalaci novější verze aplikace se pak mění pouze systém `/usr` na hostitelském počítači a ne na každé stanici zvlášť). Je-li souborový systém `/usr` na lokálním disku, může být připojen pouze pro čtení. To snižuje pravděpodobnost poškození systému souborů při havárii systému.

- Souborový systém `/var` obsahuje soubory, které se v čase mění. Tedy především sdílené adresáře pro elektronickou poštu, systém „news“, tiskárny, tzv. log-soubory, formátované manuálové stránky a dočasné soubory. Historicky bývaly všechny soubory, které jsou nyní uloženy v systému `/var`, v souborovém systému `/usr`. To znemožňovalo připojit svazek `/usr` pouze pro čtení.

- Souborový systém `/home` obsahuje domovské adresáře uživatelů. Vyčlenění uživatelských domovských adresářů do vlastní adresářové stromové struktury nebo samostatného souborového systému ulehčuje zálohování. Ostatní části adresářového stromu totiž buď nevyžadují zálohování vůbec, nebo – vzhledem k tomu, že se nemění tak často – se zálohují jenom zřídka. Velký souborový systém `/home` je dobré rozdělit na několik menších částí hierarchicky nižší úrovně a rozlišit je jménem, např. `/home/students` a `/home/staff`.

Různé části, na které je hierarchická adresářová struktura rozčleněna, byly v našem přehledu označeny jako souborové systémy. Není ale žádný zvláštní důvod k tomu, aby ve skutečnosti ležely na samostatných oddělených svazcích. Všechny by mohly být nakonec i v jediném souborovém systému. Takové řešení má význam hlavně pro malé jednouživatelské systémy, kdy je prioritou jednoduchost. Celá stromová adresářová struktura může být rozdělena na souborové systémy i jinak. Způsob jejího rozčlenění závisí na tom, jak velký je disk a jak velký diskový prostor bude vyhrazený pro různé účely. Jedinou věcí, na kterou je potřeba dbát, jsou standardní unixová *jména*. Ty je potřeba zachovat. I když bude adresář `/var` a `/usr` ve stejné diskové oblasti, musí být zachována standardní jména, jako např. `/usr/lib/libc.a` nebo `/var/adm/messages`. Totéž platí i v případě, že se například přesune adresář `/var` do adresáře `/usr/var` a na původním místě přesunutého adresáře bude symbolický link z adresáře `/usr/var`.

Struktura souborového systému Unixu sdružuje soubory podle jejich účelu. Jenom tak lze zaručit, že budou například všechny příkazy na jednom místě, data na jiném, dokumentace na dalším a tak dál. Alternativou by bylo sdružovat soubory podle toho, ke kterému programu patří. Pak by mohly být například všechny soubory pro program Emacs v jednom adresáři, všechny soubory pro TEX v jiném a podobně. Problém druhého přístupu je v tom, že je velmi obtížné sdílet soubory (adresář určitého programu často obsahuje jak statické soubory, které lze sdílet, tak soubory, jejichž obsah se mění, a ty sdílet nelze). Rovněž by bylo velmi slo-

žité dohledávat v rámci celého systému soubory určitého typu, například manuálové stránky aplikací uložené na mnoha různých místech. Programátory by jistě strašila noční můra – jak v takovémto případě vytvořit programy, které by byly schopné v případě potřeby manuálové stránky všech aplikací nalézt.

3.2 Souborový systém „root“

Kořenový svazek „root“ by obecně měl být malý, protože obsahuje velmi kritické soubory. U malého souborového systému, který se mění jenom zřídka, je menší pravděpodobnost poškození. Poškození souborového systému „root“ většinou znamená, že operační systém na tomto svazku nebude možné zavést. Tento problém lze řešit pouze pomocí speciálních opatření (např. zavedením systému z diskety), a to by chtěl riskovat asi málokterý správce. Obecně by kořenový adresář systémového svazku neměl obsahovat žádné soubory, snad kromě standardního obrazu systému. Ten se obvykle jmenuje `/vmlinuz`. Všechny ostatní soubory by měly být uloženy v podadresářích kořenového adresáře, obvykle tímto způsobem:

`/bin`	Příkazy potřebné pro zavedení systému a pro práci běžných uživatelů po jeho zavedení.
`/sbin`	Stejné jako u adresáře `/bin`. Příkazy v tomto podadresáři ale nejsou určeny běžným uživatelům, i když je též mohou použít (je-li to nutné nebo možné).
`/etc`	Konfigurační soubory specifické pro daný počítač.
`/root`	Domovský adresář superuživatele.
`/lib`	Sdílené knihovny pro programy v kořenovém souborovém systému.
`/lib/modules`	Zaváděcí moduly jádra systému – zvláště ty, které jsou potřeba pro zavedení systému při zotavení po neočekávaných událostech (např. síťové ovladače a ovladače pro souborový systém).
`/dev`	Speciální soubory.
`/tmp`	Dočasné soubory. Programy, které se spouští až po zavedení systému, by správně měly používat místo adresáře `/tmp` adresář `/var/tmp`, protože je velmi pravděpodobné, že leží na větším disku.
`/boot`	Soubory, jež používá zavaděč operačního systému (angl. bootstrap loader), např. LILO. Je dobré mít v tomto podadresáři uložené obrazy jádra (místo toho, aby se ukládaly přímo v kořenovém adresáři). V přípa-

dě, že jich máte víc, může obsah adresáře `/boot` značně narůst. V tom případě bude lepší mít jej v samostatném souborovém systému. Tím se také zajistí, že obrazy jádra budou uloženy na prvních 1024 cylindrech disku IDE.

`/mnt` Přípojné místo pro dočasná připojení dalších systémů souborů správcem systému. Nepředpokládá se, že by tento adresář využívaly pro automatická připojení souborových systémů programy. Adresář `/mnt` může být rozdělen na podadresáře (např. `/mnt/dosa` pro disketovou mechaniku používanou v souborovém systému MS-DOS, `/mnt/exta` pro tutéž mechaniku využívanou v souborovém systému ext2 a podobně).

`/proc`, `/usr`, `/var`, `/home` Přípojná místa pro další souborové systémy.

3.2.1 Adresář /etc

Adresář `/etc` obsahuje mnoho souborů. Některé z nich jsou popsány v dalším textu. Pokud jde o ostatní, měli byste nejdřív zjistit, ke kterému programu patří, a pak si přečíst manuálové stránky k tomuto programu. V adresáři `/etc` je uloženo také hodně síťových konfiguračních souborů, které jsou popsané v „Průvodci správce sítě operačního systému Linux“.

`/etc/rc`, `/etc/rc.d` a `/etc/rc?.d`
Skripty a adresáře skriptů, které se spouští při startu, nebo v případě, že se mění úroveň běhu systému. Podrobnější informace najdete v kapitole o procesu `init`.

`/etc/passwd` Databáze uživatelů systému s položkami, v nichž je uloženo uživatelské jméno i skutečné jméno uživatele, domovský adresář, šifrované heslo a některé další informace. Formát je popsán v manuálové stránce pro program *passwd*.

`/etc/fdprm` Tabulka parametrů disketové jednotky. Popisuje jak vypadají různé formáty disket. Používá ji program `setfdprm`. Víc informací uvádí manuálová stránka programu *setfdprm*.

`/etc/fstab` Seznamy souborových systémů připojovaných automaticky při startu příkazem `mount -a` (skriptem `/etc/rc`, nebo odpovídajícím souborem, jenž se spouští při startu systému). V systému Linux obsahuje rovněž informace o odkládacích oblastech „swap“, které automaticky používá příkaz `swapon -a`. Podrobnější informace viz podkapitola 4.8.5 a manuálové stránky k příkazu *mount*.

`/etc/group` Soubor podobný souboru `/etc/passwd`, ale místo uživatelů popisuje pracovní skupiny. Podrobnější informace viz manuálová stránka pro soubor `group`.

`/etc/inittab` Konfigurační soubor procesu `init`.

`/etc/issue` Soubor obsahuje výstup programu `getty`, který se zobrazí před výzvou pro přihlášení uživatele. Obvykle obsahuje stručný popis systému nebo uvítací hlášku. Obsah určuje správce systému.

`/etc/magic` Konfigurační soubor programu `file`. Obsahuje popisy různých formátů souborů, podle kterých pak program `file` tyto typy rozpoznává. Více informací najdete v manuálových stránkách pro `magic` a `file`.

`/etc/motd` Tzv. „**zpráva pro tento den“** (angl. **message of the day**) – automatický výstup na terminál uživatele po úspěšném přihlášení do systému. Obsah volí správce systému. Často se využívá pro předávání informací (např. upozornění na plánovaná zastavení systému apod.) všem uživatelům systému.

`/etc/mtab` Seznam aktuálně připojených souborových systémů. Jeho obsah po zavedení systému a připojení určených souborových systémů prvotně nastavují inicializační skripty, v běžném provozu pak automaticky příkaz `mount`. Používá se v případech, kdy je potřeba zjistit, které souborové systémy jsou připojené, např. při zadání příkazu `df`.

`/etc/shadow` Soubor tzv. „stínových“ přístupových hesel (shadow password) uživatelů v systémech, které mají nainstalovanou podporu systému stínových hesel. Kódovaná stínová hesla jsou z bezpečnostních důvodů přeneseny ze souboru `/etc/passwd` do souboru `/etc/shadow`, který může číst pouze superuživatel. Snižuje se tak pravděpodobnost odhalení některého z přístupových hesel při průniku do systému.

`/etc/login.defs`
Konfigurační soubor příkazu `login`.

`/etc/printcap` Podobně jako u souboru `/etc/termcap`, až na to, že soubor je určený pro tiskárny. Odlišná je i jeho syntaxe.

`/etc/profile`, `/etc/csh.login`, `/etc/csh.cshrc`
: Soubory spouštěné při přihlášení uživatele nebo při startu systému interprety příkazů Bourne shell nebo C shell. Umožňují správci systému stanovit globální nastavení stejné pro všechny uživatele. Viz manuálové stránky k příslušným interpretům příkazů.

`/etc/securetty` Soubor identifikuje zabezpečené terminály, tedy terminály, ze kterých se může přihlašovat superuživatel. Typicky je v seznamu uvedena pouze virtuální konzola, takže je nemožné (nebo přinejmenším těžší) získat oprávnění superuživatele přihlášením se po modemu nebo ze sítě.

`/etc/shells` Soubor, jenž uvádí seznam důvěryhodných interpretů příkazů. Příkaz `chsh` umožňuje uživatelům změnit shell spuštěný při přihlášení, a to pouze na některý z interpretů uvedený v tomto souboru. Proces `ftpd`, jenž běží na hostitelském počítači a poskytuje služby FTP pro klientské počítače, rovněž kontroluje, zda je uživatelův příkazový interpret uveden v tomto souboru a nedovolí připojit se klientům, jejichž shell v tomto seznamu uveden není.

`/etc/termcap` Databáze vlastností terminálu. Popisuje, kterými „escape“ sekvencemi se řídí různé typy terminálů. Každý program je napsaný tak, že místo přímého výstupu „escape“ sekvence, jež by fungovala pouze s konkrétním typem terminálu, hledá v tabulce `/etc/termcap` sekvenci, která odpovídá tomu, co chce program na terminálu zobrazit. Pak může většina programů správně obsluhovat většinu typů terminálů. Více informací uvádí manuálové stránky pro `termcap`, `curs-termcap` a `terminfo`.

3.2.2 Adresář /dev

Adresář `/dev` obsahuje speciální soubory pro všechna zařízení. Speciální soubory jsou pojmenované podle určitých konvencí. Ty jsou podrobně popsán v „Seznamu zařízení operačního systému Linux“ (viz [Anv]). Speciální soubory se vytváří v průběhu instalace operačního systému, v běžném provozu pak skriptem `/dev/MAKEDEV`. Podobný je skript `/dev/MAKEDEV.local`. Ten upravuje a používá správce systému, když vytváří čistě lokální speciální soubory a linky. Lokální speciální soubory jsou ty, které nejsou vytvořené standardním postupem pomocí skriptu `MAKEDEV`, typicky například speciální soubory pro některé nestandardní ovladače zařízení.

3.3 Souborový systém /usr

Souborový systém /usr je často dost velký, protože jsou v něm instalované všechny programy. Všechny soubory v systému /usr jsou obvykle instalované přímo z distribuce systému Linux. Všechny další lokálně instalované programy se ukládají do adresáře /usr/local. To umožňuje správci systému instalovat vyšší verze Linuxu z nové verze distribuce nebo i úplně jiné distribuce operačního systému bez toho, že by bylo potřeba současně instalovat všechny programy znovu. Některé z podadresářů adresáře /usr jsou popsané níže, ty méně významné neuvádíme. Více informací přináší popis standardu FSSTND.

/usr/X11R6 — Všechny soubory systému X Window. Soubory pro X nejsou integrální součástí operačního systému z důvodů zjednodušení vývoje a instalace X. Adresářová struktura /usr/X11R6 je podobná stromu, který je vytvořený pod adresářem /usr samotným.

/usr/X386 — Podobně jako u předchozího adresáře /usr/X11R6, ale pro systém X11 Release 5.

/usr/bin — Zde se nachází téměř všechny uživatelské příkazy. Některé další příkazy jsou uložené v adresáři /bin nebo /usr/local/bin.

/usr/sbin — Obsahuje ty příkazy pro správu systému, které nejsou potřeba přímo v souborovém systému „root“ (zde je například uložená převážná většina serverových programů).

/usr/man, /usr/info, /usr/doc — Manuálové stránky, informační dokumenty o projektu GNU, případně různé jiné soubory s dokumentací.

/usr/include — Hlavičkové soubory pro programovací jazyk C. Z důvodů zachování konzistence by měly být spíš v adresáři /usr/lib, ale z historických důvodů jsou umístěné ve zvláštním adresáři.

/usr/lib — Datové soubory pro programy a subsystémy, které se nemění. Jsou zde rovněž uloženy některé globální konfigurační soubory. Jméno lib je odvozeno od anglického slova „library“ (knihovna). Původně totiž byly v adresáři /usr/lib uloženy knihovny podprogramů.

/usr/local — Místo pro lokálně instalovaný software a další soubory.

3.4 Souborový systém /var

Systém /var obsahuje data, která se při běžném provozu systému mění. Soubory jsou specifické pro každý systém, a proto se data mezi jinými počítači v síti nesdílí.

/var/catman — Vyrovnávací paměť pro manuálové stránky, které jsou formátované na požádání. Zdroje pro tyto manuálové stránky jsou obvykle uložené v adresáři /usr/man/man*. Manuálové stránky v předem formátované verzi jsou uložené v adresáři /usr/man/cat*. Manuálové stránky je běžně třeba při prvním prohlížení formátovat. Formátované verze jsou pak uložené právě v adresáři /var/catman. Další uživatel, který si chce stejné stránky prohlížet, tak nemusí čekat na jejich opakované formátování. (Soubory v uvedeném adresáři /var/catman je obvykle potřeba mazat, stejně jako dočasné soubory v adresářích /tmp a /var/tmp.)

/var/lib — Soubory, které se při normálním provozu systému mění.

/var/local — Měnící se data pro programy instalované v adresáři /usr/local (tj. programy instalované správcem systému). Upozorňujeme, že lokálně instalované programy by měly používat i ostatní podadresáře nadřazeného adresáře /var, např. /var/lock.

/var/lock — Soubory tzv. zámků. Většina programů dodržuje určitou konvenci a vytváří v adresáři /var/lock zámky. Tím dávají ostatním programům najevo, že dočasně využívají některé zařízení nebo soubor. Jiné programy, které by chtěly stejné zařízení či soubor ve stejném okamžiku používat, se o to nebudou pokoušet.

/var/log — Adresář obsahuje tzv. log-soubory různých programů. Například program login zaznamenává (do souboru /var/log/wtmp) všechna přihlášení a odhlášení uživatelů systému, program syslog ukládá (do souboru /var/log/messages) všechny hlášky jádra systému a systémových programů. Velikost souborů v adresáři /var/log dost často nekontrolovaně roste. Proto se musí v pravidelných intervalech mazat.

/var/run — Adresář, do něhož se ukládají soubory obsahující informace o systému, jež platí až do jeho dalšího zavedení. Tak například soubor /var/run/utmp obsahuje informace o současně přihlášených uživatelích systému.

`/var/spool` Adresáře pro elektronickou poštu, systém „news“, tiskové fronty a další subsystémy, které využívají metodu „spool“ a princip řazení úloh do fronty. Každý z těchto subsystémů má v tomto adresáři svůj vlastní podadresář, např. poštovní schránky uživatelů jsou uložené v podadresáři `/var/spool/mail`.

`/var/tmp` Do adresáře `/var/tmp` se ukládají velké dočasné soubory a dočasné soubory, které budou existovat déle než ty, které se ukládají do adresáře `/tmp`. (Avšak správce systému by měl dbát na to, aby stejně jako v adresáři `/tmp`, ani v adresáři `/var/tmp` nebyly uloženy velmi staré dočasné soubory.)

3.5 Souborový systém /proc

Systém souborů `/proc` je vlastně imaginárním souborovým systémem. Ve skutečnosti na disku neexistuje. Místo toho jej v paměti vytváří jádro systému. Ze systému souborů `/proc` lze získávat různé aktuální informace o systému (původně o procesech – z toho je odvozeno jeho jméno). Některé z významnějších souborů a adresářů popisujeme níže. Samotný souborový systém `/proc` je podrobněji popsán na manuálové stránce *proc*.

`/proc/1` Adresář s informací o procesu číslo 1. Každý z procesů má v adresáři `/proc` vlastní podadresář, kterého jméno je stejné, jako identifikační číslo procesu.

`/proc/cpuinfo` Různé informace o procesoru. Například typ, výrobce model, atd.

`/proc/devices` Seznam ovladačů zařízení konfigurovaných pro aktuálně běžící jádro systému.

`/proc/dma` Informuje o tom, které kanály DMA jsou právě využívány.

`/proc/filesystems`
Souborové systémy konfigurované v jádru systému.

`/proc/interrupts`
Informuje o tom, která přerušení jsou využívána, i o historii žádostí o využití každého z nich.

`/proc/ioports` Informuje o tom, který z vstupně-výstupních portů se momentálně využívá.

`/proc/kcore`	Obraz fyzické paměti systému. Má velikost odpovídající velikosti fyzické paměti systému. Ve skutečnosti ale samozřejmě nezabírá takovéto množství paměti, protože jde o soubor generovaný „na požádání“, tedy pokaždé jenom v okamžiku, kdy k němu různé programy přistupují. Uvědomte si, že soubory souborového systému `/proc` nezabírají ve skutečnosti (než je zkopírujete na nějaké jiné místo na disku) vůbec žádný diskový prostor.
`/proc/kmsg`	Výstupní hlášení jádra systému. Zde uložená hlášení využívá i program `syslog`.
`/proc/ksyms`	Tabulka symbolů jádra systému.
`/proc/loadavg`	Statistika zatížení systému – tři celkem nic neříkající indikátory toho, kolik práce systém momentálně má.
`/proc/meminfo`	Informace o využití paměti, jak fyzické, tak virtuální (swap).
`/proc/modules`	Informuje o tom, které moduly jádra jsou právě zavedeny v paměti.
`/proc/net`	Informace o stavu síťových protokolů.
`/proc/self`	Symbolický link do adresáře procesů toho programu, který zrovna přistupuje k souborovému systému `/proc`. Když k systému souborů `/proc` současně přistupují dva různé procesy, budou mít přidělené dva různé linky. Tímto způsobem se mohou programy pohodlně a jednoduše dostat k vlastnímu adresáři.
`/proc/stat`	Různé statistiky týkající se systému. Např. počet chyb stránkování během zavádění systému a podobné.
`/proc/uptime`	Informuje o tom, jak dlouho systém běží.
`/proc/version`	Verze jádra systému.

Většina výše popsaných souborů má textovou formu. Jsou tedy celkem dobře čitelné pomocí standardních nástrojů. Avšak velmi často jsou formátované takovým způsobem, že informace v nich obsažené jsou pro běžného uživatele na pohled dost těžce „stravitelné“. Proto existuje mnoho příkazů, které – kromě toho, že čtou informace obsažené v souborech systému `/proc`, upravují jejich formát do srozumitelnější podoby. Tak například program `free` čte data ze souboru `/proc/meminfo`, převádí velikost v bajtech na kilobajty a přidá něco málo dalších informací o využití paměti.

Kapitola

4

Používání disků a jiných záznamových médií

Na čistém disku lze hledat navěky.

Z pohledu diskového subsystému vás jako správce čeká skutečně nejvíc práce při instalaci operačního systému Linux, případně instalace jeho vyšší verze. Je potřeba vytvořit souborové systémy, do kterých se budou ukládat soubory, a vyhradit na discích prostor pro různé části systému.

Tato kapitola popisuje všechny tyto úvodní činnosti. Když tuto práci jednou podstoupíte a systém nastavíte, obvykle to už nebudete muset dělat znovu. Výjimkou je používání disket. K této kapitole se také budete vracet pokaždé, když budete přidávat nový pevný disk nebo pokud budete chtít optimálně vyladit diskový subsystém.

Mezi základní úkoly při správě disků patří:

- Formátování pevného disku. Formátování disku je posloupností několika různých dílčích činností (jako je například kontrola výskytu vadných sektorů), které tento disk připravují na další použití. (v současnosti je většina nových disků formátovaná výrobcem. Formátování po připojení do systému tedy není nutné.)

- Rozdělení pevného disku na oblasti. Když chcete disk využívat pro několik činností, o kterých se nepředpokládá, že by se vzájemně ovlivňovaly, můžete jej rozdělit na samostatné diskové oblasti, segmenty (angl. partitions). Jedním z důvodů pro rozdělení disku na oblasti je instalace a provozování různých operačních systémů na jednom disku. Jinou výhodou rozdělení pevného disku na oblasti je oddělení uživatelských souborů od souborů systémových. Tím se zjednoduší zálohování a sníží se pravděpodobnost poškození systémových souborů.

- Vytvoření souborového systému (vhodného typu). Na každém disku nebo diskové oblasti lze vytvořit samostatný souborový systém. Z pohledu operačního systému nestačí disk pouze zapojit, případně rozdělit na samostatné diskové oblasti. Linux může disk používat až poté, co na něm vytvoříte nějaký souborový systém. Pak lze na disk ukládat soubory a přistupovat k nim.

- Vytvoření jednoduché stromové struktury připojením různých souborových systémů. Systémy souborů se připojují buď automaticky, nebo manuálně – podle potřeby. (Ručně připojené souborové systémy se obvykle musí také ručně odpojit.)

Kapitola 5 obsahuje i informace o virtuální paměti a diskové vyrovnávací paměti. Pracujete-li s disky, měli byste být s jejich principy obeznámeni.

V této kapitole najdete vše, co jako správce systému potřebujete vědět o pevných discích, disketách, jednotkách CD-ROM a páskových jednotkách.

4.1 Dva druhy zařízení

Unix a tedy i Linux zná dva různé typy zařízení. Jednak bloková zařízení s náhodným přístupem (například disky) a jednak znaková zařízení (např. pásky a sériové linky). Znaková zařízení mohou být buď sériová, nebo s náhodným přístupem. Každému z podporovaných zařízení odpovídá v systému souborů jeden nebo několik speciálních souborů. Když se čtou či zapisují data z anebo do speciálního souboru, přenáší se ve skutečnosti na zařízení, které tento soubor reprezentuje. Takže pro přístup k zařízením nejsou nutné žádné zvláštní programy a žádné zvláštní programovací techniky (jako např. obsluha přerušení nebo nastavování parametrů sériového portu). Když například chcete vytisknout nějaký soubor na tiskárně, stačí zadat

```
$ cat jméno_souboru > /dev/lp1
$
```

Obsah tohoto souboru se vytiskne. Soubor musí mít přirozeně nějakou formu, kterou tiskárna zná. Avšak vzhledem k tomu, že není příliš rozumné, aby několik uživatelů současně kopírovalo soubory na jedinou tiskárnu, se pro tiskové úlohy běžně používá zvláštní program (nejčastěji `lpr`). Tento program zajistí, že se v určitém okamžiku bude tisknout pouze jeden soubor. Ihned po ukončení tiskové úlohy automaticky pošle na tiskárnu další ze souborů. Podobný mechanismus přístupu vyžaduje většina zařízení. Takže ve skutečnosti se běžný uživatel o speciální soubory skoro vůbec nemusí zajímat.

Protože se zařízení chovají jako soubory souborového systému (uložené v adresáři `/dev`), lze velmi lehce zjistit, které speciální soubory existují. Lze použít například příkaz `ls` nebo jiný vhodný program. v prvním sloupci výstupu příkazu `ls -l` je uvedený typ takového souboru a jeho přístupová práva. Chcete-li si prohlédnout sériová zařízení systému, zadáte

```
$ ls –l /dev/cua0
crw-rw-rw- 1 root uucp 5, 64 Nov 30 1993 /dev/cua0
$
```

Podle prvního písmene prvního sloupce, tedy písmene „c“ v řetězci „`crw-rw-rw-`“, informovaný uživatel pozná o jaký typ souboru jde. V tomto případě se jedná o znakové zařízení. U běžných souborů je prvním písmenem „`-`“, u adresářů je to „`d`“ a pro bloková zařízení se používá písmeno „`b`“. Podrobnější informace uvádí manuálová stránka k příkazu `ls`.

Všimněte si, že všechny speciální soubory obvykle existují, i když zařízení samotné není nainstalované. Takže například pouhý fakt, že v systému souborů je soubor `/dev/sda`, neznamená, že skutečně máte pevný disk SCSI. Díky tomu, že všechny speciální soubory po instalaci operačního systému existují, lze zjednodušit instalační programy. Méně složité je tím pádem i přidávání nových hardwarových komponent (pro nově přidávané součásti již není třeba hledat správné parametry a vytvářet speciální soubory).

4.2 Pevné disky

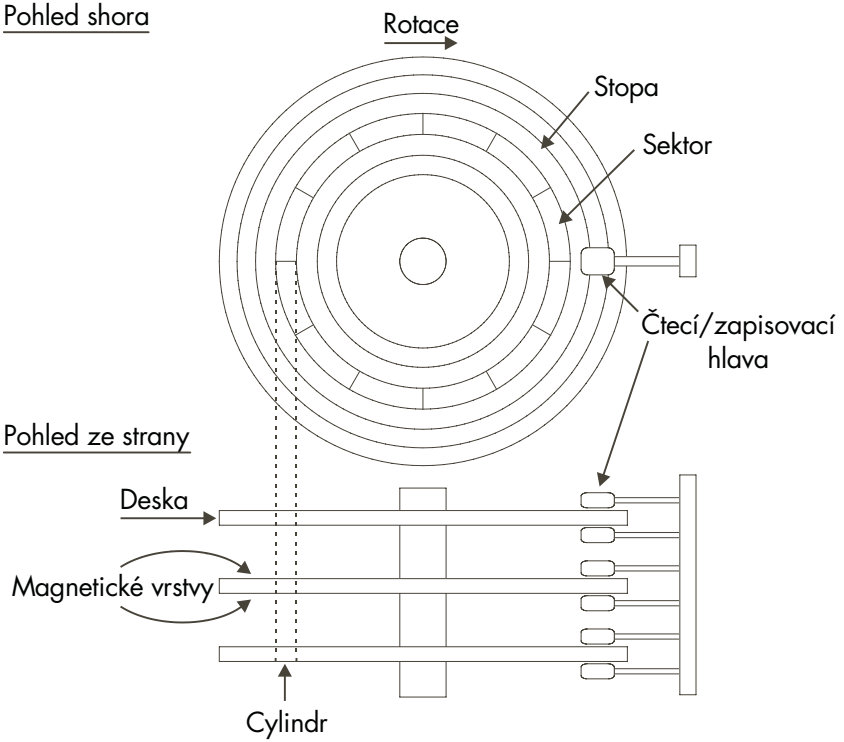
Tento odstavec zavádí terminologii, která se v oblasti pevných disků používá. Znáte-li tyto pojmy a principy, můžete tuto část přeskočit. Na obrázku 4.1 jsou schematicky znázorněny důležité části pevného disku, který se skládá z jedné nebo více kruhových **desek**.[1] Povrch, případně obě **strany** těchto desek, jsou pokryty magnetickou vrstvou, na kterou se zaznamenávají data. Každé takovéto vrstvě odpovídá jedna **čtecí a zapisovací hlava**, která čte nebo zaznamenává údaje. Disky rotují kolem běžné hřídele. Typická rychlost otáčení je 3600 otáček za minutu. Pevné disky s vysokým výkonem používají i vyšší rychlosti. Hlavy se pohybují po obvodu disků. Díky tomuto pohybu spojenému s rotací kruhových desek může hlava přistupovat ke všem částem magnetických povrchů pevných desek disku.

Procesor (CPU) a vybraný disk spolu komunikují prostřednictvím **řadiče disku**. Díky řadiči se zbytek systému nemusí zajímat o to, jak pracovat s diskem, protože lze použít pro různé typy disků řadiče, jež mají pro ostatní součásti počítače stejné rozhraní. Takže počítač může disku (místo zadávání sérií dlouhých a složitých elektrických signálů, podle nichž se hlava nejdřív přesune na odpovídající místo disku, pak čeká, až se pod ni dostane žádaná pozice

[1] Tyto desky jsou vyrobeny z pevného materiálu, např. hliníku. Proto se pevným diskům říká „pevné“.

a dělá další nepříjemností, které je pro danou operaci potřeba) jednoduše vzkázat „hele milý disku, dej mně to, co potřebuji“. (Ve skutečnosti je sice rozhraní řadiče stejně dost složité, ale rozhodně ne tak složité, jako by bylo v případě, kdyby ostatní prvky systému přistupovaly k disku přímo.) Řadič navíc dělá některé další operace, obsluhuje například vyrovnávací paměť, automaticky nahrazuje vadné sektory atd.

Výše uvedené obvykle každému postačí k pochopení toho, jak hardware pevného disku funguje. Existuje pochopitelně ještě řada dalších detailů, např. řízení motoru, který otáčí disky a přemísťuje hlavy, elektronika, jež řídí operace dalších mechanických částí atd., které ale většinou nejsou pro pochopení principu práce pevného disku tak důležité.



Obrázek 4.1
Schematické zobrazení pevného disku

Magnetické vrstvy disku jsou obvykle rozděleny do soustředných kružnic, kterým se říká **stopy** (angl. **tracks**). Stopy jsou po obvodu rozděleny na **sektory**. Toto rozdělení se používá pro určování místa na disku a přidělování diskového prostoru souborům. Když například chcete na disku najít určité místo, zadáte „souřadnice“: „vrstva 3, stopa 5, sektor 7“. Počet sektorů je většinou pro všechny stopy stejný, ale některé pevné disky mají na vnějších stopách víc sek-

torů (všechny sektory mají stejnou fyzickou velikost, takže většina z nich je umístěna na delších, vnějších stopách). Typicky bude v jednom sektoru uloženo 512 bajtů dat. Disk samotný pak neumí pracovat s menším množstvím dat, než je jeden sektor.

Každá vrstva je rozdělena na stopy a sektory stejným způsobem. Takže když je hlava jedné vrstvy nad určitou stopou, hlavy ostatních povrchů se také nachází nad odpovídajícími stopami. Všem těmto stopám dohromady se říká **cylindr**. Přemístění hlavy z jedné stopy (cylindru) na jinou trvá jistou dobu. Takže ukládáním dat, ke kterým se často přistupuje najednou (například data jednoho souboru) na stejný cylindr, se zamezí zbytečnému přesouvání hlav při jejich pozdějším čtení. Snižuje se přístupová doba a zvyšuje výkon. Není ale vždycky možné uložit data na disk tímto způsobem. Souborům, které jsou na disku uložené na několika místech, se říká **fragmentované**.

Počet vrstev (resp. hlav, což je to samé), cylindrů a sektorů se dost liší. Specifikace jejich počtu se nazývá **geometrií** pevného disku. Geometrie je obvykle uložena ve zvláštní bateriemi zálohované oblasti paměti, které se říká **CMOS RAM**. Z paměti CMOS RAM si geometrii disku operační systém načítá vždy během zavádění nebo inicializace ovladače disku.

Na neštěstí má BIOS[2] omezení, které neumožňuje adresovat v CMOS RAM počet stop větší, než 1024. To je pro pevné disky velké kapacity příliš málo. Uvedené omezení lze obejít tak, že řadič pevného disku nebude říkat ostatním prvkům výpočetního systému pravdu o jeho skutečné geometrii a bude **překládat adresy**, které zbytek systému požaduje, do něčeho, co odpovídá realitě. Mějme například pevný disk, jenž má 8 hlav, 2048 stop a 35 sektorů na stopu[3]. Řadič tohoto disku bude zbytku systému „lhát“ a tvrdit, že má 16 hlav, 1024 stop a 35 sektorů na stopu, což nepřekračuje omezení v počtu stop. Pak bude při každém požadavku systému na přístup k disku překládat adresu, kterou dostane tak, že počet hlav vydělí dvěma a počet stop dvěma vynásobí. Matematické úpravy budou v praxi složitější, protože skutečná čísla nejsou tak „pěkná“, jako v uvedeném příkladě. Ale nezbývá než zopakovat, že detaily nejsou tak důležité pro pochopení principu. Překládání adres zkresluje pohled operačního systému na to, jak je disk ve skutečnosti organizovaný. To je nepraktické, protože nelze pro snížení přístupové doby a zvýšení výkonu použít „trik“ s ukládáním všech souvisejících dat na jeden cylindr.

2 Systém BIOS (Basic Input Output System) je software uložený v paměti ROM. Kromě jiného má na starost úvodní procedury zavádění systému.

3 Tato čísla jsou zcela smyšlená.

Překlady adres jsou výlučně problémem disků typu IDE. Disky typu SCSI používají sekvenční čísla sektorů. To znamená, že řadič disku SCSI překládá každé sekvenční číslo sektoru na uspořádanou trojici [hlava, cylindr, sektor]. Navíc používá úplně jinou metodu komunikace s CPU, a proto jsou disky SCSI těchto problémů ušetřeny. Uvědomte si ale, že ani u disků SCSI počítač nezná jejich skutečnou geometrii.

Vzhledem k tomu, že operační systém Linux obvykle nezná skutečnou geometrii disku, nebudou se ani jeho souborové systémy pokoušet ukládat soubory na stejné cylindry. Místo toho se pokusí souborům přidělit sekvenčně řazené sektory. Tato metoda téměř vždy zaručí podobný výkon, jakého lze dosáhnout při ukládání souvisejících dat na jeden cylindr. Celá problematika je o něco složitější, řadiče například využívají vlastní vyrovnávací paměti, nebo mechanismus automatického, řadičem řízeného „přednačítání“ sekvenčně řazených sektorů atd.

Každý pevný disk je v systému reprezentován samostatným speciálním souborem. Nejčastěji budou v systému buď dva, nebo čtyři pevné disky IDE. Zastupují je pak speciální soubory `/dev/hda`, `/dev/hdb`, `/dev/hdc`, případně `/dev/hdd`. Pevné disky SCSI reprezentují speciální soubory `/dev/sda`, `/dev/sdb` atd. Podobné konvence týkající se názvů speciálních souborů platí i pro pevné disky jiných typů. Podrobnější informace uvádí [Anv]. Pamatujte na to, že speciální soubory zastupující pevné disky umožňují přístup k disku jako celku, bez ohledu na jeho diskové oblasti (o těch bude řeč později). Není proto těžké při práci s disky pochybit. Neopatrnost může v tomto případě vést ke ztrátě dat. Speciální soubory disků se obvykle používají pouze pro přístup k zaváděcímu sektoru disku (angl. master boot record, zkráceně MBR), o kterých se také dozvíte víc v dalších částech této kapitoly.

4.3 Diskety

Disketa sestává z pružné membrány pokryté z jedné nebo obou stran podobnou magnetickou substancí, jako pevný disk. Pružný disk samotný nemá čtecí a zápisovou hlavu, ta je součástí disketové mechaniky. Disketa vlastně odpovídá jedné desce pevného disku, ale na rozdíl od pevného disku je vyměnitelná. Jednu disketovou mechaniku lze využít pro přístup k různým disketám, kdežto pevný disk je jedinou nedílnou jednotkou.

Podobně jako pevný disk se i disketa dělí na stopy a sektory. Dvě korespondující si stopy každé strany diskety tvoří cylindr. Počet stop a sektorů je ale pochopitelně o hodně nižší, než u pevného disku.

Disketová jednotka umí obvykle pracovat s několika různými typy disket. Například 3,5palcová disketová mechanika může pracovat jak s disketami o velikosti 720 kB, tak 1,44MB disketami. Vzhledem k tomu, že disketová jednotka musí zacházet s každým typem diskety trochu jinak, musí i operační systém vědět, který typ diskety je zrovna zasunutý v mechanice. Proto existuje pro jednotky pružných disků množství speciálních souborů, a to vždy jeden pro

každou kombinaci typu disketové jednotky a typu diskety. Takže soubor `/dev/fd0H1440` pak reprezentuje první disketovou jednotku (`fd0`). Musí to být 3,5palcová mechanika pracující s 3,5palcovými disketami vysoké hustoty záznamu (proto `H` v názvu zařízení) s kapacitou 1 440 kB (proto `1440` ve jménu speciálního souboru). To jsou běžné 3,5palcové diskety HD (High Density).

Podrobnější informace o konvencích pro pojmenovávání speciálních souborů reprezentujících disketové mechaniky uvádí [Anv].

Konstrukce tvorby názvů speciálních souborů zastupujících disketové jednotky je poměrně složitá. Proto má Linux pro diskety i zvláštní typy zařízení. Tato zařízení automaticky detekují typ diskety zasunuté v mechanice. Fungují tak, že se při požadavku na přístup pokouší přečíst první sektor vložené diskety, přičemž postupně zkouší jejich různé typy, až se jim podaří najít ten správný. Samozřejmě, podmínkou je, aby vložená disketa byla nejdřív naformátovaná. Automatická zařízení zastupují speciální soubory `/dev/fd0`, `/dev/fd1` atd.

Parametry, které tato automatická zařízení používají pro přístup k disketám, lze nastavit také programem `setfdprm`. To se může hodit jednak v případě, že používáte diskety, jež nemají běžnou kapacitu, to znamená, že mají neobvyklý počet sektorů, dále v případě, že autodetekce z neznámých důvodů selže a nebo když vlastní speciální soubor chybí.

Kromě toho, že Linux zná všechny standardní formáty disket, umí pracovat i s množstvím nestandardních formátů. Některé z nich ale vyžadují speciální formátovací programy. Touto problematikou se teď nebudeme zabývat a doporučíme projít si soubor `/etc/fdprm`. Ten blíže specifikuje nastavení, která program `setfdprm` podporuje.

Operační systém musí vědět o tom, že byla disketa v mechanice vyměněna. Jinak by totiž mohl například použít data z dříve vložené diskety, uložená ve vyrovnávací paměti. Bohužel vodič, jenž se používá k signalizaci výměny diskety, bývá někdy poškozený. Při používání disketové jednotky v systému MS-DOS nebude mechanika vůbec schopná indikovat systému výměnu média, což je ještě horší situace. Jestli jste se někdy setkali s podivnými, zdánlivě nevysvětlitelnými problémy při práci disketami, jejich příčinou mohla být právě nefunkční indikace výměny média. Jediným způsobem, jak lze tyto problémy odstranit, je nechat disketovou mechaniku opravit.

4.4 Jednotky CD-ROM

Jednotky CD-ROM čtou opticky data z plastických disků. Informace jsou zaznamenány na povrchu těchto disků[4] jako miniaturní „dolíčky“ seřazené v husté spirále, jež začíná uprostřed disku a končí na jeho okraji. Jednotka vysílá laserový paprsek, který čte data z disku

4 Přesněji, na povrchu *uvnitř* disků – tedy na tenkém kovovém disku uvnitř plastového pláště.

tak, že sleduje tuto spirálu. Od hladkého povrchu disku se odráží jinak, než když narazí na dolík. Tímto způsobem lze jednoduše kódovat binární informace. Ostatní je prostě pouhá mechanika.

Ve srovnání s pevnými disky jsou jednotky CD-ROM pomalé. Pevné disky mají průměrnou přístupovou dobu typicky menší než 15 milisekund, kdežto rychlá jednotka CD-ROM bude data vyhledávat s přístupovou dobou řádově v desetinách sekundy. Skutečná rychlost přenosu dat je ale celkem vysoká, zhruba kilobajt za sekundu. To, že je jednotka CD-ROM „pomalá“, znamená, že ji nelze pohodlně využít jako alternativu pevných disků, i když to samozřejmě možné je. Některé distribuce Linuxu totiž nabízejí tzv. „live“ souborové systémy na CD-ROM. Ty fungují tak, že část souborů se při instalaci nekopíruje na pevný disk a v případě potřeby se čtou přímo z kompaktního disku. Instalace je pak jednodušší, méně časově náročná a ušetří se také značná část diskového prostoru. Jednotky CD-ROM lze s výhodou využít při instalaci nového programového vybavení, protože při instalování není vysoká rychlost určujícím faktorem.

Je několik způsobů jak se data na disky CD-ROM ukládají. Nejrozšířenější z nich upravuje mezinárodní standard ISO 9660. Tato norma definuje minimální souborový systém, který je ještě o něco primitivnější, než systém souborů používaný systémem MS-DOS. Na druhou stranu je souborový systém ISO 9660 tak jednoduchý, že by s ním měly – jako se svým původním – umět bez problémů pracovat všechny operační systémy.

Pro běžnou práci v Unixu je ale souborový systém ISO 9660 prakticky nepoužitelný. Proto se zavedlo rozšíření tohoto standardu, kterému se říká „Rock Ridge extension“. Rock Ridge například umožňuje používat dlouhá jména souborů, symbolické linky a mnoho dalších vymožeností, díky kterým se disk CD-ROM tváří víceméně jako unixový souborový systém. Navíc, rozšíření Rock Ridge zachovává kompatibilitu se souborovým systémem ISO 9660, takže je použitelné i v jiných než unixových systémech. Operační systém Linux podporuje jak ISO 9660, tak Rock Ridge extension. Rozšíření rozezná a dále používá zcela automaticky.

Souborový systém je ale pouze jednou stranou mince. Většina disků CD-ROM často obsahuje data, ke kterým lze přistupovat výhradně pomocí zvláštních programů. Bohužel většina těchto programů není určena pro Linux (možná s výjimkou některých programů, jež běží pod `dosemu`, linuxovým emulátorem systému MS-DOS).

K jednotce CD-ROM se také přistupuje prostřednictvím odpovídajícího speciálního souboru. Je několik způsobů jak připojit jednotku CD-ROM k počítači: buď rozhraním SCSI, nebo pomocí zvukové karty, nebo rozhraním EIDE. Popis dalších podrobností technického řešení jed

notlivých způsobů připojení jednotky CD-ROM jdou nad rámec této knihy. Pro vás je podstatné, že podobně jako u jednotek pružných disků určuje způsob připojení vždy jiný speciální soubor. Další podrobnosti, které by mohly více objasnit tuto problematiku, uvádí [Anv].

4.5 Pásky

Magnetopáskové jednotky používají pásky podobné[5] těm, které se používají pro záznam zvuku na magnetofonových kazetách. Páska je již svou povahou sériovým zařízením – aby bylo možné dostat se k některé její části, je nejdřív nutné projít všechny předchozí záznamy. K údajům na disku lze přistupovat náhodně, takže je možné „skočit" přímo na kterékoliv místo na něm. Sériový přístup k datům na páskách je příčinou toho, že jsou pomalé. Na druhou stranu jsou relativně levné, což kompenzuje nedostatky v rychlosti. Bez problému je lze vyrobit dost dlouhé, takže je na ně možno uložit velké množství dat. To vše předurčuje pásková zařízení k archivaci a zálohování, u nichž se nevyžaduje vysoká rychlost, ale naopak, využívá se nízkých nákladů a velké kapacity.

4.6 Formátování

Formátování je procedura zápisu značek, které se používají na označení stop a sektorů na magnetickém médiu. Předtím, než je disk naformátován, jsou magnetické signály na jeho povrchu neuspořádané, chaotické. Formátování do tohoto chaosu vnáší určitý řád. Načrtnou se – obrazně řečeno – linie, ve kterých vedou stopy a ty se pak rozdělí na sektory. Skutečné podrobnosti jsou trochu jiné, ale to není pro tuto chvíli podstatné. Důležité je to, že disk nelze používat bez toho, že by byl naformátován.

Pokud jde o formátování, je terminologie mírně zavádějící. v systému MS-DOS se pod pojmem „formátování" rozumí i proces vytváření souborového systému (o němž bude řeč později). Takže se obě tyto procedury označují jediným pojmem – obzvlášť u disket. Když je nutné je rozlišit, používá se pro formátování v pravém slova smyslu termín **nízkoúrovňové formátování** (angl. **low-level formatting**) a pro vytvoření souborového systému označení **vysokoúrovňové formátování** (angl. **high-level formatting**). Terminologie používaná v unixovém světě označuje obě tyto činnosti tradičními pojmy „formátování" a „vytvoření souborového systému". Nejinak tomu bude i v této knize.

5 Ale jinak, pochopitelně, úplně jiné.

Disky IDE a některé disky SCSI jsou formátovány ve výrobě a není třeba je po připojení formátovat opakovaně. Možná proto se o formátování disků málokdo zajímá. Ale právě nesprávné formátování pevného disku může být skutečnou příčinou toho, že disk nepracuje správně (některé disky vyžadují zvláštní způsob formátování, který pak umožňuje například automatické přemísťování vadných sektorů).

Navíc disky, které je potřeba (a které lze) formátovat často, vyžadují speciální formátovací programy, protože rozhraní formátovací logiky zabudované v jednotce se liší od jednoho typu disku k druhému. Takovéto formátovací programy jsou často buď součástí řadiče BIOS, nebo běží pouze pod systémem MS-DOS. Ani jeden z těchto prostředků tedy nelze bez problémů použít v systému Linux.

V průběhu formátování můžete narazit na chybná místa na disku, kterým se říká **vadné bloky** nebo **vadné sektory**. S vadnými bloky si někdy poradí samotný řadič pevného disku, ale v případě, že se jich objeví víc, je potřeba nějakým opatřením zamezit možnému použití těchto vadných částí disku. Mechanismus, který problém vadných bloků řeší, je součástí souborového systému. Způsob, jakým systém souborů pracuje s informacemi o vadných sektorech, je popsaný v dalších částech této kapitoly. Jinou alternativou je vytvoření malé diskové oblasti (partition), která by obsahovala pouze vadné bloky disku. Takovýto alternativní postup je vhodný zejména v případě, že je rozsah vadných sektorů velmi velký. Souborové systémy totiž mohou mít s rozsáhlými oblastmi vadných bloků problémy.

Diskety se formátují programem `fdformat`. Speciální soubor, který se má formátovat, se zadává jako jeho parametr. Následujícím příkazem bychom například zformátovali 3,5palcovou disketu s vysokou hustotou záznamu, vloženou do první disketové jednotky:

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$
```

Pamatujte si, že když chcete použít zařízení s autodetekcí (např. `/dev/fd0`), *musíte* nejdříve nastavit parametry zařízení pomocí programu `setfdprm`. Stejný výsledek jako v prvním příkladě mají příkazy:

```
$ setfdprm /dev/fd0 1440/1440
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
```

```
Verifying ... done
$
```

Je obvykle výhodnější vybrat správný speciální soubor, jenž odpovídá typu diskety. Zapamatujte si, že není rozumné formátovat disketu na vyšší kapacitu než je ta, pro kterou je určena.

Program `fdformat` zároveň prověří disketu – zkontroluje, zda neobsahuje vadné bloky. Pokud narazí na vadný blok, opakovaně se pokusí vadný blok použít (obvykle to uslyšíte – zvuky, které jednotka při formátování vydává, se dramaticky změní). Je-li na disketě pouze logická chyba (špatná úroveň signálu po zápisu znečištěnou zápisovou hlavou), program `fdformat` si nebude „stěžovat“. Naopak skutečná chyba – fyzicky poškozený sektor, přeruší proces kontroly povrchu diskety a jejího formátování. Jádro systému zapíše hlášení do log-souboru po každé vstupně/výstupní chybě, na kterou při formátování narazí. Tato hlášení se zároveň objeví na konzole. Když běží program `syslog`, zapisují se tato hlášení také do souboru `/var/log/messages`. Samotným programem `fdformat` uživatel nezjistí, kde přesně se vadný blok nachází (obvykle to ani nikoho nezajímá – diskety jsou tak levné, že se ty vadné automaticky vyhazují).

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... read: Unknown error
$
```

Vadné bloky na disku nebo diskové oblasti (i disketě) lze vyhledat pomocí příkazu `badblocks`. Ten ale neumí disk zformátovat, takže jej lze použít jenom ke kontrole existujících souborových systémů. Níže uvedený příklad hledá chyby na 3,5palcové disketě se dvěma vadnými bloky:

```
$ badblocks /dev/fd0H1440 1440
718
719
$
```

Výstupem příkazu `badblocks` jsou čísla vadných sektorů. Většina souborových systémů umí takovéto vadné bloky označit jako nepoužitelné. Systémy souborů udržují seznam, tabulku vadných sektorů, která se zakládá, když se systém souborů vytváří. Tento seznam pak lze kdykoliv měnit. První kontrola vadných bloků se dělá příkazem `mkfs`, jimž se vytváří souborový systém. Další kontroly se dělají programem `badblocks` a nové chybné bloky se přidávají do seznamu vadných bloků příkazem `fsck`. Příkazy `mkfs` a `fsck` popíšeme později.

Řada moderních disků umí automaticky zjistit výskyt vadných bloků a pokouší se „opravit“ je tak, že místo nich použije k těmto účelům zvlášť vyhrazené správné sektory. Mechanismus náhrady chybného bloku správným je pro operační systém transparentní. Takováto vlastnost diskové jednotky by měla být dokumentovaná v jejím manuálu. V něm byste měli najít podrobnější informace o způsobu jak zjistit, jestli se na disku, který používáte, vyskytují vadné bloky, které řadič disku popsaným způsobem nahradil správnými sektory. Avšak i disky tohoto typu by teoreticky mohli selhat, kdyby počet vadných bloků výrazně vzrostl. Nicméně je pravděpodobnější, že než by se tak stalo, byl by disk již tak opotřebovaný, že by byl prakticky nepoužitelný.

4.7 Diskové oblasti

Pevný disk může být rozdělen na několik **diskových oblastí,** neboli segmentů (angl. **partitions**). Každá oblast se chová tak, jako by byla samostatným diskem. Využití diskových oblastí má význam v případě, že máte jeden pevný disk a chcete na něm používat například dva operační systémy. Disk můžete rozdělit na dva segmenty. Každý operační systém pak bude používat vlastní diskovou oblast a nebude zasahovat do druhé. Takto mohou oba operační systémy v klidu a míru koexistovat na jediném disku. Kdybyste nepoužili rozdělení disku na samostatné diskové oblasti, museli byste zakoupit pevný disk pro každý z operačních systémů.

Diskety se na segmenty nedělí. Z technického hlediska to možné je, ale vzhledem k tomu, že mají malou kapacitu, by oblasti byly prakticky využitelné jenom v ojedinělých případech. Ani disky CD-ROM se obvykle nedělí na oblasti, protože je praktičtější je používat jako jeden velký disk a skutečně málokdy je potřeba mít na jednom disku CD-ROM uloženo několik operačních systémů.

4.7.1 Zaváděcí sektor disku, zaváděcí sektory operačních systémů a tabulka oblastí

Informace o tom, jak je pevný disk rozdělen na diskové oblasti, je uložena v prvním sektoru (to jest, prvním sektoru první stopy první vrstvy disku). První sektor je tzv. **zaváděcí sektor disku** (angl. **master boot record**, zkráceně MBR). Je to sektor, který se načítá a spouští systémem BIOS pokaždé, když se počítač spustí. Zaváděcí sektor disku obsahuje krátký program, jenž načte tabulku rozdělení disku na oblasti (angl. partition table) a zjistí, která oblast disku je aktivní (tedy označená jako zaváděcí). Pak přečte první sektor této oblasti, neboli **zaváděcí sektor** (angl. **boot sector**) této oblasti. (MBR je také zaváděcí sektor, ale má zvláštní postavení, a proto i zvláštní označení.) Zaváděcí sektor diskového segmentu obsahuje další krátký program, který načítá první část operačního systému, jenž je na této diskové oblasti uložený (samozřejmě za předpokladu, že je daná oblast označena jako aktivní – zaváděcí) a spouští jej.

Metoda dělení disku na diskové oblasti není hardwarově implementovaná a není ani součástí systému BIOS. Jde čistě o nepsaná pravidla podporovaná i jinými operačními systémy. Ale ne všechny operační systémy se chovají podle těchto konvencí, jsou i výjimky. Některé operační systémy sice podporují dělení disku na diskové oblasti, ale obsadí jeden diskový segment a v rámci této oblasti používají svou vlastní interní metodu dělení. Avšak i tyto typy operačních systémů mohou bez jakýchkoliv zvláštních prostředků spolupracovat s jinými systémy (včetně operačního systému Linux). Naopak, takový operační systém, jenž rozdělení disku na diskové oblasti nepodporuje, nemůže koexistovat na stejném pevném disku s jiným operačním systémem, který tyto konvence podporuje.

Je dobré si na kousek papíru vypsat tabulku rozdělení disku na oblasti. Kdyby pak náhodou v budoucnu došlo k poškození některé oblasti, nemusíte díky tomuto jednoduchému bezpečnostnímu opatření přijít o všechna data. (Poškozenou tabulku oblastí lze opravit programem `fdisk`.) Některé důležité informace získáte příkazem `fdisk -l`:

```
$ fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes

   Device Boot Begin   Start     End     Blocks   Id    system
/dev/hda1          1       1      24     10231+   82    Linux swap
/dev/hda2         25      25      48      10260   83    Linux native
/dev/hda3         49      49     408     153900   83    Linux native
/dev/hda4        409     409     790     163305    5    Extended
/dev/hda5        409     409     744    143611+   83    Linux native
/dev/hda6        745     745     790     19636+   83    Linux native
$
```

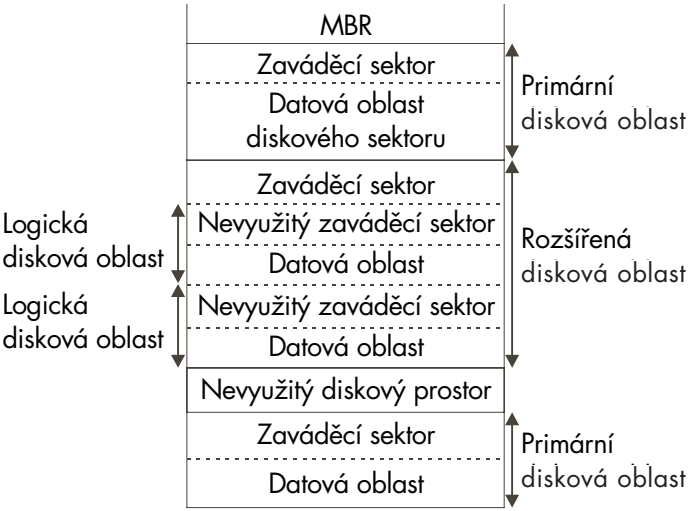
4.7.2 Rozšířené a logické diskové oblasti

Původní schéma dělení disků počítačů PC na diskové oblasti umožňuje vytvořit pouze čtyři diskové segmenty. To se záhy v praxi ukázalo jako nedostatečné. Zčásti například proto, že někteří uživatelé chtěli mít na svém počítači i víc než čtyři operační systémy (Linux, MS-DOS, OS/2, Minix, FreeBSD, NetBSD nebo Windows/NT, a to jsme vyjmenovali jenom některé), ale především proto, že je výhodné mít několik diskových oblastí i pro jeden operační systém. Například z důvodů zlepšení odezvy operačního systému je lepší nevyužívat pro odkládací prostor systému Linux hlavní diskovou oblast operačního systému, ale mít prostor pro „swap“ na samostatném diskovém segmentu (podrobnosti uvádíme v dalších částech manuálu).

Aby bylo možné omezení počtu diskových oblastí obejít, byly zavedeny tzv. **rozšířené diskové oblasti** (angl. **extended partitions**). Tento trik umožňuje rozdělit **primární diskové oblasti** (angl. **primary partitions**) na podoblasti. Takto rozdělená primární oblast je onou rozšířenou oblastí a její části (podoblasti) jsou tzv. **logické oblasti** (angl. **logical partitions**).

Chovají se jako primární[6], ale jsou vytvořené jiným způsobem. Není mezi nimi ale žádný rozdíl v rychlosti.

Struktura oblastí pevného disku by mohla vypadat například tak, jak je uvedeno na obrázku 4.2. Disk je rozdělen na tři primární diskové oblasti. Druhá z nich je rozdělena na dvě logické diskové oblasti. Část disku nepatří vůbec žádné oblasti. Disk jako celek a každá primární oblast má svůj zaváděcí sektor.



Obrázek 4.2
Příklad rozdělení pevného disku na diskové oblasti

4.7.3 Typy diskových oblastí

Tabulky oblastí disku (jedna v MBR a další na rozšířených diskových oblastech) mají vyhrazený jeden bajt pro každou oblast, jež identifikuje její typ. Typ diskové oblasti určuje operační systém, který daný segment využívá, případně jiný účel, pro který tuto oblast operační systém používá (např. odkládací prostor „swap“). Informační bajt by měl zamezit tomu, aby mohly být v počítačovém systému nainstalovány dva operační systémy, které by náhodně využí-

6 Nelogické?

valy stejnou oblast disku. Avšak ve skutečnosti většina operačních systémů označení typu diskové oblasti ignoruje. I Linux se například vůbec nezajímá o to, jak je určitá disková oblast označená. Dokonce – což je ještě horší, některé operační systémy tento bajt používají nesprávně. Například přinejmenším některé verze systému DR-DOS ignorují významnější část tohoto bajtu, kdežto zbylé ne.

Žádný z úřadů pro normalizaci nespecifikoval, co která hodnota bajtu znamená. Některé obecně přijaté hodnoty a odpovídající typy uvádí tabulka 4.1. Stejný seznam vypíše linuxový program `fdisk`.

4.7.4 Dělení pevného disku na diskové oblasti

Je mnoho programů, které umí vytvářet a mazat diskové oblasti. Součástí většiny operačních systémů je nějaký nástroj pro práci s diskovými segmenty. Je vždy lepší používat program, jenž je součástí operačního systému, v prostředí kterého se segmenty pracujete. Pouze v případě, že by se tento program choval nezvykle, se doporučuje použít jiný. Většinou se tyto programy jmenují `fdisk` (včetně toho, který je součástí distribuce systému Linux) nebo nějak podobně. Detaily týkající se možností linuxového programu `fdisk` podrobně popisuje jeho manuálová stránka. Podobný je příkaz `cfdisk`, který má hezčí, celoobrazovkové uživatelské rozhraní.

0	Nevyužitá oblast	40	Venix 80286	94	Amoeba BBT
1	DOS (12bit. FAT)	51	Novell?	a5	BSD/386
2	XENIX (oblast „root“)	52	Microport	b7	BSDI fs
3	XENIX (oblast „usr“)	63	GNU HURD	b8	BSDI swap
4	DOS (16bit) < 32M	64	Novell	c7	Syrinx
5	Rozšířená oblast	75	PC/IX	db	CP/M
6	DOS (16bit) > 32M	80	Starší MINIX	e1	DOS (přístupná)
7	OS/2 HPFS	81	Linux/MINIX	e3	DOS (pouze pro čtení)
8	AIX	82	Linux swap	f2	DOS (sekundární)
9	AIX zaváděcí	83	Linux nativní	ff	BBT
a	OS/2 Boot Manager	93	Amoeba		

Tabulka 4.1
Typy diskových oblastí (výstup programu `fdisk` pro Linux)

V případě, že používáte disky IDE, musí být celá zaváděcí disková oblast (tedy segment, na kterém jsou uloženy soubory obrazů jádra) na prvních 1024 cylindrech. To proto, že při zavádění operačního systému (předtím, než systém přechází do chráněného režimu) používá disk systém BIOS a ten neumí pracovat s více než 1024 cylindry. V některých případech je

možné používat zaváděcí diskovou oblast, která leží na prvních 1024 cylindrech, jenom zčásti. To je možné jenom když budou uloženy na prvních 1024 cylindrech všechny soubory, které při inicializaci čte systém BIOS. Vzhledem k tomu, že takovéhoto uspořádání je velmi obtížné dosáhnout, *je lepší jej vůbec nepoužívat* – nikdy si totiž nemůžete být jistí, zda změna parametrů jádra systému nebo defragmentace disku nezpůsobí, že systém nebude vůbec možné zavést. Proto se raději vždy ujistěte, že je zaváděcí oblast vašeho systému celá na prvních 1024 cylindrech. Některé nové verze systémů BIOS a disků IDE již umí pracovat i s disky, které mají více než 1024 cylindrů. Máte-li takovýto systém, můžete na tento problém zapomenout. Jestli si v tom nejste zcela jisti, umístněte raději podle doporučení zaváděcí diskovou oblast na prvních 1024 cylindrů.

Každá disková oblast by měla mít sudý počet sektorů, protože souborové systémy Linuxu používají bloky velikosti 1 kB, tedy dva sektory. Lichý počet sektorů diskové oblasti způsobí, že poslední sektor bude nevyužitý. Nemělo by to způsobit žádné zvláštní problémy, ale není to příliš elegantní. Některé verze programu `fdisk` vás budou na tento stav upozorňovat.

Při změně velikosti diskové oblasti je nejlepší vytvořit zálohu všeho, co chcete z tohoto segmentu disku uchovat (pro každý případ raději celý disk nebo oblast), pak tento segment smazat, vytvořit nový a obnovit na nové diskové oblasti soubory ze zálohy. Chcete-li zvětšit velikost diskové oblasti, budete muset pravděpodobně upravit i velikosti (tedy vytvořit zálohy a obnovit soubory) sousedních diskových oblastí.

Vzhledem k tomu, že změny velikostí diskových oblastí jsou dost pracné, je lepší nastavit je správně hned napoprvé. Jinak budete potřebovat efektivní a jednoduchý systém zálohování. Instalujete-li poprvé systém z médií, jež nevyžadují časté zásahy obsluhy (například z CD-ROM – protikladem jsou diskety), je často jednodušší si nejdřív pohrát s různými konfiguracemi. Jelikož zatím na disku nemáte žádná data, která by bylo potřeba zálohovat, není natolik bolestné několikrát změnit velikosti diskových oblastí.

Existuje program pro systém MS-DOS, který se jmenuje `fips`, a ten umí změnit velikosti diskových oblastí systému MS-DOS bez toho, že by bylo potřeba zálohovat, mazat a obnovovat soubory z těchto segmentů. Pro jiné souborové systémy je to zatím nadále nevyhnutné.

4.7.5 Speciální soubory a diskové oblasti

Každá disková oblast a rozšířená disková oblast má svůj vlastní speciální soubor. Podle konvence pro konstrukci názvů speciálních souborů se číslo diskové oblasti připojí za jméno celého disku s tím, že 1–4 budou primární diskové oblasti (podle toho kolik primárních oblastí bylo vytvořeno) a 5–8 jsou logické diskové oblasti (podle toho, na které primární oblasti jsou

vytvořené). Například `/dev/hda1` je první primární disková oblast prvního pevného disku IDE a `/dev/sdb7` je třetí rozšířená disková oblast na druhém pevném disku SCSI. Více informací najdete v seznamu zařízení, jenž je uveden v [Anv].

4.8 Souborové systémy

4.8.1 Co jsou souborové systémy?

Souborový systém tvoří metody a struktury dat, pomocí kterých operační systém udržuje záznamy souborů na disku nebo diskové oblasti. Jde tedy o způsob, jakým jsou soubory na disku organizované. Tento termín se ale používá i ve vztahu k diskové oblasti nebo disku, na kterém se ukládají soubory, nebo ve významu typu souborového systém. Takže když někdo řekne „mám dva souborové systémy“, může mít namysli to, že má disk rozdělen na dvě diskové oblasti, nebo to může znamenat, že používá „rozšířený souborový systém“, tedy určitý typ souborového systému.

Rozdíl mezi diskem či diskovou oblastí a souborovým systémem, který je na nich vytvořený, je dost podstatný. Některé programy (mezi nimi – zcela logicky – programy, pomocí kterých se souborové systémy vytváří) pracují přímo se sektory disku nebo diskového segmentu. Jestli na nich byl předtím vytvořený systém souborů, bude po spuštění takovýchto programů zničený nebo vážně poškozený. Většina aplikací ale pracuje se souborovým systémem. Proto je nelze použít na diskové oblasti, na které není vytvořený žádný systém souborů (nebo na oblasti, na které je vytvořený souborový systém nesprávného typu).

Při tím, než bude možné diskovou oblast nebo disk použít, je potřeba je inicializovat – musí se na ně zapsat určité účetní datové struktury. Tento proces se označuje jako **vytvoření souborového systému**.

Většina unixových souborových systémů má podobnou obecnou strukturu. v dalších podrobnostech se ale celkem dost liší. Mezi ústřední pojmy patří **superblok**, **i-uzel**, **datový blok**, **adresářový blok** a **nepřímý blok**. Superblok obsahuje informace o souborovém systému jako celku, například jeho velikost (zrovna u této položky závisí přesná hodnota na konkrétním souborovém systému). I-uzel obsahuje všechny informace o souboru, kromě jeho jména. Jméno souboru je uloženo v adresáři, společně s odpovídajícím číslem i-uzlu. Adresářová položka obsahuje jména souborů a čísla i-uzlů, které tyto soubory reprezentují. I-uzel dále obsahuje čísla datových bloků, v nichž jsou uložená data souboru, který daný i-uzel zastupuje. V i-uzlu je ale místo jenom pro několik čísel datových bloků. Když je jich potřeba víc, je dynamicky alokováno víc místa pro další ukazatele na datové bloky. Tyto dynamicky alokované bloky jsou tzv. nepřímé bloky. Jak jejich název naznačuje, v případě, že je potřeba najít datový blok, musí se nejdřív najít jeho číslo v nepřímém bloku.

Unixové souborové systémy obvykle umožňují vytvořit v souboru „**díru**“ (angl. **hole**), a to pomocí programu `lseek` (podrobnosti uvádí příslušná manuálová stránka). Vypadá to tak, že souborový systém pouze „předstírá“, že je na konkrétním místě souboru uloženo nula bajtů dat, takže pro toto místo nejsou vyhrazeny žádné sektory pevného disku (to znamená, že takovýto soubor zabírá o něco méně diskového prostoru). S tím se můžete setkat zvlášť často u malých binárních souborů, sdílených knihoven Linuxu, některých databází a v několika dalších speciálních případech. (Tato „prázdná místa“ jsou implementována tak, že se jako adresa datového bloku v nepřímém bloku nebo i-uzlu uloží zvláštní hodnota. Tato zvláštní adresa znamená, že pro určitou část souboru není alokován žádný datový blok, tedy že je v tomto souboru „díra“.)

Prázdná místa v souborech lze využít. Na autorově systému bylo jednoduchými prostředky dokázáno, že tímto způsobem lze potencionálně ušetřit asi 4 MB z celkových 200 MB diskového prostoru. A to je na tomto systému poměrně málo programů a vůbec žádné databázové soubory. Uvedené diagnostické nástroje jsou popsané v příloze A.

4.8.2 Široká paleta souborových systémů

Linux podporuje několik typů souborových systémů. Mezi nejdůležitější patří:

minix — Je nejstarší a je považovaný za nejspolehlivější. Má ale několik omezení – chybí časová razítka (angl. time stamps), jména souborů mohou být nejvíce 30 znaků dlouhá, souborový systém může mít maximálně 64 MB a další.

xia — Modifikovaná verze souborového systému minix. Nemá omezení v délce jmen souborů a velikosti souborového systému, jinak nepřináší žádné nové rysy. Mezi uživateli není velmi oblíbený, ale jinak má pověst velmi spolehlivého systému.

ext2 — Souborový systém, který má nejvíce různých možností ze všech zde uvedených původních souborových systémů pro Linux. V současnosti je také nejpopulárnější. Byl navržen tak, aby byl zpětně kompatibilní, takže nové verze kódu souborového systému nevyžadují nové, opakované vytváření již existujících souborových systémů.

ext — Starší verze `ext2`, která nebyla zpětně kompatibilní. Lze ji jenom stěží používat v nových instalacích Linuxu – většina uživatelů již přešla na systém souborů `ext2`.

Kromě toho je podporováno několik souborových systémů jiných operačních systémů, což umožňuje přenášet soubory mezi různými operačními systémy. Tyto cizí, nepůvodní souborové systémy jinak fungují jako původní systémy souborů pro Linux, ale obvykle postrádají některé rysy typické pro Unix, případně mají některá neobvyklá omezení nebo jiné zvláštnosti.

msdos Souborový systém kompatibilní se souborovým systémem FAT operačního systému MS-DOS (také OS/2 a Windows NT).

umsdos Rozšiřuje možnosti ovladače souborového systému `msdos` pro systém Linux. Umí pracovat s dlouhými názvy souborů, zná vlastníky souborů, přístupová práva, linky a speciální soubory. To umožňuje používat běžný souborový systém `msdos` jako by byl původním linuxovým souborovým systémem. Rovněž není potřeba mít oddělené diskové oblasti pro systémy Linux a MS-DOS.

iso9660 Standardní souborový systém disků CD-ROM. Oblíbené rozšíření „Rock Ridge extension" tohoto standardu automaticky zavádí delší jména souborů a další možnosti.

nfs Síťový souborový systém, který umožňuje sdílení souborových systémů mezi větším počtem počítačů a jednoduchý přístup k souborům každého z nich.

hpfs Souborový systém operačního systému OS/2.

sysv Souborové systémy operačních systémů SystemV/386, Coherent a Xenix.

Výběr konkrétního souborového systému závisí na dané situaci. V případě, že jsou kompatibilita nebo jiná omezení nutnou podmínkou výběru některého souborového systému jiného operačního systému, nezbývá než použít tento nepůvodní systém souborů. Jestli nejste ve výběru omezení, bude pravděpodobně nejrozumnější používat ext2, protože má ze všech uvedených systémů souborů nejvíce možností a nemá nedostatky z hlediska výkonu.

Dalším ze souborových systémů je systém souborů `proc`, nejčastěji dostupný prostřednictvím adresáře `/proc`. Ve skutečnosti ale není souborovým systémem v pravém slova smyslu, i když tak na první pohled vypadá. Systém souborů `proc` umožňuje přístup k určitým datovým strukturám jádra systému, například k seznamu procesů (odtud jeho jméno). Přizpůsobuje tyto datové struktury tak, že se navenek chovají jako soubory souborového systému. K systému souborů `proc` pak lze přistupovat všemi běžnými nástroji, které se soubory běžně pracují. Takže kdybyste chtěli znát například seznam všech procesů, zadali byste příkaz

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root       root         0 Jan 31 20:37 1
dr-xr-xr-x   4 liw        users        0 Jan 31 20:37 63
dr-xr-xr-x   4 liw        users        0 Jan 31 20:37 94
dr-xr-xr-x   4 liw        users        0 Jan 31 20:37 95
dr-xr-xr-x   4 root       users        0 Jan 31 20:37 98
dr-xr-xr-x   4 liw        users        0 Jan 31 20:37 99
-r--r--r--   1 root       root         0 Jan 31 20:37 devices
```

```
-r--r--r--   1 root       root          0 Jan 31 20:37 dma
-r--r--r--   1 root       root          0 Jan 31 20:37 filesystems
-r--r--r--   1 root       root          0 Jan 31 20:37 interrupts
-r--------   1 root       root    8654848 Jan 31 20:37 kcore
-r--r--r--   1 root       root          0 Jan 31 11:50 kmsg
-r--r--r--   1 root       root          0 Jan 31 20:37 ksyms
-r--r--r--   1 root       root          0 Jan 31 11:51 loadavg
-r--r--r--   1 root       root          0 Jan 31 20:37 meminfo
-r--r--r--   1 root       root          0 Jan 31 20:37 modules
dr-xr-xr-x   2 root       root          0 Jan 31 20:37 net
dr-xr-xr-x   4 root       root          0 Jan 31 20:37 self
-r--r--r--   1 root       root          0 Jan 31 20:37 stat
-r--r--r--   1 root       root          0 Jan 31 20:37 uptime
-r--r--r--   1 root       root          0 Jan 31 20:37 version
$
```

(v seznamu bude vždy několik souborů, které neodpovídají žádným procesům. Výstup ve výše uvedeném příkladu byl zkrácen.)

Je důležité si uvědomit, že i když se pro systém `proc` používá označení „souborový systém“, žádná z jeho částí neleží na žádném z disků. Existuje jenom „v představách“ jádra systému. Kdykoliv k některé části souborového systému `proc` přistupuje uživatel systému nebo některý z procesů, jádro „předstírá“, že je tato část uložená na disku, ale ve skutečnosti tomu tak není. Takže i když je v souborovém systému `proc` uložený několikamegabajtový soubor `/proc/kcore`, ve skutečnosti nezabírá žádné místo na disku.

4.8.3 Který souborový systém použít?

Obvykle není moc důvodů používat několik různých souborových systémů. V současné době je nejoblíbenějším souborový systém ext2fs a to je současně pravděpodobně ta nejrozumnější volba. Ve vztahu k zmiňovaným účetním strukturám, rychlosti, (pochopitelně) spolehlivosti, kompatibilitě a vzhledem k různým jiným důvodům by mohlo být vhodné zvolit i jiný systém souborů. Takovéto požadavky je třeba posuzovat případ od případu.

4.8.4 Vytváření souborového systému

Souborové systémy se vytváří a inicializují příkazem `mkfs`. Je to vlastně vždy jiný program pro každý typ souborového systému. Program `mkfs` je jenom tzv. „front end“, tedy program, který spouští některé další programy, podle typu požadovaného souborového systému. Typ souborového systému se volí přepínačem `-t fstype`.

Programy volané příkazem `mkfs` mají nepatrně odlišné rozhraní příkazové řádky. Běžně používané a nejdůležitější volby jsou uvedené níže. Více informací najdete v manuálových stránkách.

`-t` *typ_fs* Výběr typu souborového systému.

`-c` Hledání vadných bloků a aktualizace jejich seznamu.

`-l` *jméno_souboru* Načtení seznamu vadných bloků ze souboru *jméno_souboru*.

Kdybyste chtěli vytvořit souborový systém ext2 na disketě, zadali byste následující příkazy:

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 / bad-blocks
$ mkfs -t ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

V prvním kroku se disketa formátuje (volba `-n` zakáže její validaci, tedy kontrolu vadných sektorů). Vadné bloky pak vyhledává program `badblocks`, a to s výstupem přesměrovaným do souboru `bad-blocks`. Nakonec se vytvoří souborový systém a mezi účetní struktury se uloží seznam vadných bloků, ve kterém budou všechny vadné sektory, jež našel program `badblocks`.

Místo příkazu `badblocks` je možno použít argument `-c` programu `mkfs` a název souboru tak, jak to uvádí následující příklad:

```
$ mkfs -t ext2 -c /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
```

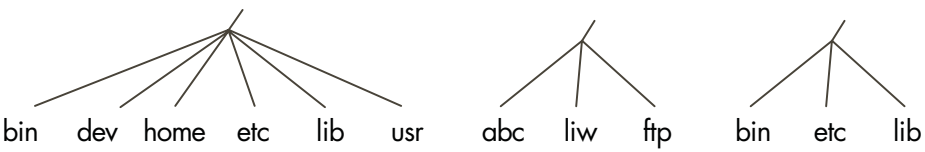
```
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group
Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

Volba `-c` programu `mkfs` je sice vhodnější, než použití programu `badblocks`, ale příkaz `badblocks` je vždy nutné použít pro kontrolu vadných bloků po vytvoření souborového systému. Postup, kterým se vytváří souborové systémy na pevných discích a diskových oblastech, je stejný jako naznačený postup inicializace souborového systému na disketě, až na to, že není nutné je formátovat.

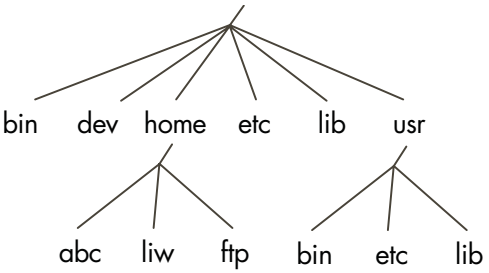
4.8.5 Připojení a odpojení

Souborový systém se musí před použitím **připojit**, namontovat. Po připojení systému souborů dělá operační systém některé účetní operace, kterými se ověřuje funkčnost připojení. Protože všechny soubory v Unixu jsou součástí jediného hierarchického adresářového stromu, operace připojení souborového systému začlení obsah připojovaného souborového systému do některého z adresářů dříve připojeného systému souborů.

Na obrázku 4.3 jsou například zobrazeny tři samostatné souborové systémy, každý se svým vlastním kořenovým adresářem. Když budou poslední dva souborové systémy připojeny pod adresář `/home` a `/usr` prvního systému souborů, dostaneme jeden adresářový strom, který je vyobrazen na obrázku 4.4.



Obrázek 4.3
Tři samostatné souborové systémy



Obrázek 4.4
Připojené souborové systémy `/home` a `/usr`

Souborový systém lze připojit například zadáním následujících příkazů:

```
$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$
```

Příkaz `mount` má dva argumenty. Prvním je speciální soubor odpovídající disku nebo diskové oblasti, na které leží připojovaný souborový systém. Druhým parametrem je adresář, pod nímž bude souborový systém připojen. Po provedení těchto příkazů vypadá obsah obou připojovaných systémů souborů tak, jako by byl součástí hierarchického stromu adresářů `/home` a `/usr`. Říká se, že „`/dev/hda2` je připojený na adresář `/home`“ a to samé platí i o adresáři `/usr`. Když si pak chcete prohlédnout některý ze souborových systémů, procházíte strukturou adresářů, ke kterým jsou připojené, jako by to byly jakékoliv jiné adresáře. Je důležité si uvědomit rozdíl mezi speciálním souborem `/dev/hda2` a adresářem `/home`, ke kterému je systém souborů připojen. Speciální soubor umožňuje přístup k „syrovému“ obsahu pevného disku, kdežto prostřednictvím adresáře `/home` se přistupuje k souborům, které jsou na tomto disku uložené. Adresář, ke kterému je souborový systém připojený, se nazývá **přípojný bod** (angl. **mount point**).

Operační systém Linux podporuje mnoho typů souborových systémů. Příkaz `mount` se vždy pokusí rozeznat typ připojovaného systému souborů. Lze také použít přepínač `-t` *typ_fs*, který přímo specifikuje typ připojovaného souborového systému. Někdy je to potřeba, protože heuristika programu `mount` nemusí vždy pracovat správně. Chcete-li například připojit disketu systému MS-DOS, zadáte příkaz:

```
$ mount −t msdos /dev/fd0 /floppy
$
```

Adresář, ke kterému se souborový systém připojuje, nemusí být prázdný, ale musí existovat. Soubory, jež byly původně uloženy v adresáři, ke kterému je připojený nový souborový systém, budou nepřístupné, a to až do jeho odpojení. (Soubory, které byly v době připojení otevřené, budou nadále přístupné. Soubory, které jsou nalinkované z jiných adresářů, budou přístupné prostřednictvím těchto původních souborů.) Nehrozí přitom žádné nebezpečí poškození těchto souborů a někdy to navíc může být i užitečné. Někteří uživatelé například rádi používají synonyma `/tmp` a `/var/tmp`. Proto si dělají symbolický link adresáře `/tmp` do adresáře `/var/tmp`. Předtím, než se při zavádění systému připojí souborový systém `/usr`, je adresář `/var/tmp` v kořenovém souborovém systému. Poté, co se připojí systém souborů `/usr`, bude adresář `/var/tmp` v kořenovém souborovém systému nepřístupný. V případě, že by adresář `/var/tmp` v souborovém systému „root“ neexistoval, bylo by použití dočasných souborů před připojením systému souborů `/var` nemožné.

Jestli nemáte v úmyslu v připojovaném souborovém systému cokoliv zapisovat, zadejte programu `mount` přepínač `-r`. Tím se vytvoří **připojení pouze pro čtení**. Jádro systému pak zabrání každému pokusu o zápis do tohoto souborového systému. Rovněž jádro samotné má zakázáno aktualizovat položku času posledního přístupu v i-uzlech souborů. Připojení systému souborů pouze pro čtení je podmínkou u médií, na která nelze zapisovat, např. u disků CD-ROM.

Pozorný čtenář si již uvědomil drobný logistický problém. Jakým způsobem se připojuje první souborový systém, tedy kořenový svazek (angl. **root filesystem**), někdy označovaný jako **kořenový souborový systém** (obsahuje kořenový adresář), když zjevně nemůže být připojený na jiný souborový systém? Odpověď je jednoduchá – je to „kouzlo“.[7] Kořenový souborový systém se „magicky“ připojuje při zavádění systému a lze se spolehnout na to, že bude připojený vždy. Kdyby z nějakých důvodů souborový systém „root“ nebylo možné připojit, systém se vůbec nezavede. Jméno souborového systému, jenž se magicky připojuje jako kořenový, je buď zkompilované v jádře systému, nebo se nastavuje zavaděčem LILO, případně programem `rdev`. Kořenový svazek se pokaždé obvykle nejdříve připojí pouze pro čtení. Pak inicializační skripty spustí program `fsck`, který jej zkontroluje. Když se neobjeví žádný problém, kořenový svazek se připojí znovu, a to tak, že na něj bude možno zapisovat. Program `fsck` se nesmí spouštět na připojeném souborovém systému s možností zápisu, protože jakékoliv změny v souborovém systému, ke kterým by došlo při kontrole (a opravách chyb v souborovém systému) *způsobí* potíže. Když je kořenový souborový systém při kontrole připojený pouze pro čtení, program `fsck` může bez obav opravovat všechny chyby, protože operace opětovného připojení souborového systému „spláchne“ všechna metadata, která má souborový systém uložená v paměti.

7 Víc informací najdete ve zdrojovém kódu jádra systému nebo v Průvodci jádrem systému.

V některých systémech se používají i jiné souborové systémy, které lze automaticky připojit při zavádění systému. Jsou specifikované v souboru `/etc/fstab`. Podrobnosti týkající se formátu tohoto souboru najdete v manuálové stránce k souboru `fstab`. Přesné detaily procesu připojování dalších souborových systémů závisí na množství faktorů a je-li potřeba, může je správce systému nastavit. K dalšímu rozšíření znalostí o připojování souborových systémů při zavádění systému lze doporučit závěr kapitoly, jež pojednává o zavádění operačního systému.

Když už není třeba mít souborový systém připojený, je možno jej odpojit příkazem `umount`[8]. Program `umount` má jediný argument, buďto speciální soubor, nebo bod přístupu. Například odpojení adresářů připojených v předchozím příkladu lze provést příkazy

```
$ umount /dev/hda2
$ umount /usr
$
```

Podrobnější instrukce jak používat příkaz `umount` najdete v manuálové stránce k tomuto programu. Důležitý imperativ, na který je potřeba upozornit – připojenou disketovou jednotku je vždy potřeba odpojit. *Nestačí jenom vytáhnout disketu z mechaniky!* Když systém používá vyrovnávací paměť, v okamžiku odpojení jednotky nemusí být data ze zásobníku paměti „cache“ zapsaná na disketu! Předčasné vytažení diskety z mechaniky by mohlo způsobit poškození jejího obsahu. Když z diskety pouze čtete, není její poškození moc pravděpodobné, ale když zapisujete – navíc náhodně – důsledky mohou být katastrofální.

Připojování a odpojování souborových systémů vyžaduje oprávnění superuživatele, takže ho může dělat pouze uživatel `root`. Je tomu tak například proto, že kdyby měl kterýkoliv z uživatelů právo připojit si jednotku pružných disků na libovolný adresář, nebylo by velmi těžké připojit disketu s virem typu trojského koně „přestrojeného“ za program `/bin/sh`, nebo jiný často používaný příkaz. Avšak dost často je potřeba, aby uživatelé mohli s disketami pracovat. Je několik způsobů, jak jim to umožnit:

- Sdělit uživatelům heslo superuživatele. To je z hlediska bezpečnosti zjevně špatné, avšak to nejjednodušší řešení. Funguje dobře v případě, že se vůbec není potřeba zabývat zabezpečením systému. To se týká řady osobních systémů – tedy systémů, které nejsou připojené do počítačové sítě.

[8] Logicky správný tvar je pochopitelně `unmount`, ale písmenko „n“ v sedmdesátých letech záhadně zmizelo a od té doby jej již nikdo víckrát nespatřil. Jestli jej náhodou najdete, vraťte jej laskavě do Bell Labs, NJ.

- Používat programy jako je `sudo`, jenž umožní uživatelům používat příkaz `mount`. Toto je z hlediska bezpečnosti rovněž špatné řešení, avšak tímto způsobem přímo nepředáváte privilegia superuživatele každému.[9]

- Umožnit uživatelům používat balík programů `mtools`, jenž umožňuje manipulovat se souborovými systémy MS-DOS bez toho, že by je bylo potřeba připojovat. Řešení funguje dobře pouze v případě, že jsou diskety systému MS-DOS vším, co budou uživatelé systému potřebovat. V ostatních případech je nevhodné.

- Zapsat všechny disketové jednotky a pro ně přípustné přípojné body spolu s dalšími vhodnými volbami do seznamu připojovaných systémů v souboru `/etc/fstab`.

Posledně uvedenou alternativu lze realizovat přidáním níže uvedeného řádku do souboru `/etc/fstab`:

```
/dev/fd0 /floppy msdos user,noauto 0 0
```

Význam jednotlivých položek (zleva doprava): speciální soubor, který se má připojit; adresář, na který se má toto zařízení namontovat (přípojný bod); typ souborového systému; některé další volby; četnost zálohování (používá program `dump`); posledním parametrem je pořadí kontroly programem `fsck` (určuje pořadí, ve kterém by měly být souborové systémy prověřovány při startu systému; 0 znamená nekontrolovat).

Přepínač `noauto` zakáže automatické připojení při startu systému (tj. zakáže připojení příkazem `mount -a`). Argument `user` umožní kterémukoliv uživateli připojit si souborový systém. Z důvodů bezpečnosti zamezí možnosti spouštět programy (jak běžné, tak programy s příznakem setuid) a interpretaci speciálních souborů z připojeného souborového systému. Pak si každý uživatel může namontovat disketovou jednotku se souborovým systémem `msdos` tímto příkazem:

```
$ mount /floppy
$
```

Disketu můžete (samozřejmě pokaždé musíte) odpojit odpovídajícím příkazem `umount`.

Chcete-li umožnit přístup k několika různým typům disket, musíte zadat několik přípojných bodů. Nastavení pro každý přípojný bod mohou být různá. Například přístup k disketám s oběma typy souborových systémů – MS-DOS i ext2 – byste umožnili přidáním těchto řádků do souboru `/etc/fstab`:

9 Uživatelé by nad tím pravděpodobně museli několik sekund přemýšlet.

```
/dev/fd0 /dosfloppy msdos user,noauto 0 0
/dev/fd0 /ext2floppy ext2 user,noauto 0 0
```

U souborových systémů MS-DOS (nejenom na disketách) budete pravděpodobně vyžadovat omezený přístup s využitím parametrů `uid`, `gid` a `umask`. Ty jsou podrobně popsané v manuálové stránce příkazu `mount`. Následkem neopatrnosti při připojování souborového systému MS-DOS může být totiž to, že kterýkoliv uživatel získá oprávnění číst v připojeném systému souborů kterékoliv soubory, což není moc dobré.

4.8.6 Kontrola integrity souborového systému programem fsck

Souborové systémy jsou poměrně složité struktury a tím také mají sklony k chybovosti. Správnost a platnost souborového systému se kontroluje příkazem `fsck`. Lze jej nastavit tak, aby při kontrole automaticky opravoval méně závažné chyby a varoval uživatele na výskyt chyb, které nelze odstranit. Naštěstí je kód, kterým se implementují souborové systémy, poměrně efektivně odladěný, takže problémy vznikají velmi zřídka a jsou obvykle zapříčiněny výpadkem napájecího napětí, selháním technického vybavení nebo chybou obsluhy, například nesprávným vypnutím systému. Většina systémů je nastavena tak, že spouští program `fsck` automaticky při zavádění systému, takže jakékoliv chyby jsou detekovány (a většinou i odstraněny) předtím, než systém přechází do běžného pracovního režimu. Používáním poškozeného systému souborů se totiž jeho stav obvykle ještě více zhoršuje. Jsou-li v nepořádku datové struktury souborového systému, práce se systémem je pravděpodobně poškodí ještě víc, důsledkem čeho mohou být ztráty dat většího rozsahu.

Jenomže kontrola velkých souborových systémů programu `fsck` chvíli trvá. Když se ale systém vypíná správně, chyby souborových systémů se vyskytují jenom velmi zřídka. Lze pak využít několika triků, díky kterým se lze kontrole (velkých) souborových systémů vy-hnout. První z nich: existuje-li soubor `/etc/fastboot`, neprovádí se žádná kontrola. Druhý: souborový systém ext2 má ve svém superbloku speciální příznak, jehož hodnota je nastavena podle toho, jestli byl souborový systém po předchozím připojení odpojen správně, či nikoliv. Podle tohoto příznaku pozná pak program `e2fsck` (verze příkazu `fsck` pro souborový systém ext2), jestli je nutné provádět kontrolu tohoto systému souborů, či nikoliv. V případě, že podle hodnoty příznaku byl daný systém souborů odpojený korektně, kontrola se neprovádí. Předpokládá se, že řádné odpojení systému zároveň znamená, že v souborovém systému nevznikly žádné defekty. To, zda se při proceduře zavádění systému vynechá proces validace systému souborů v případě, že soubor `/etc/fastboot` existuje, záleží na nastavení spouštěcích skriptů systému. Trik s příznakem souborového systému ext2 ale funguje vždy, když systém souborů kontrolujete programem `e2fsck`. Kdybyste totiž chtěli programu `e2fsck` přikázat, aby prověřil i korektně odpojený systém souborů, museli byste tuto podmínku explicitně zadat odpovídajícím přepínačem. (Podrobnosti o tom jak uvádí manuálová stránka programu *e2fsck*.)

Automatické prověřování správnosti souborových systémů se provádí jenom u systémů souborů, které se připojují automaticky při startu operačního systému. Manuálně lze program `fsck` použít pro kontrolu jiných souborových systémů, např. na disketách.

Najde-li program `fsck` neodstranitelné chyby, připravte se na to, že budete potřebovat buď hluboké znalosti obecných principů fungování souborových systémů a konkrétního typu poškozeného souborového systému zvlášť, nebo dobré zálohy. Druhou možnost lze jednoduše (ačkoli někdy dost pracně) zajistit. Nemáte-li potřebné „know-how“ sami, mohou první alternativu v některých případech zajistit vaši známí, dobrodinci z diskusních skupin o Linuxu na Internetu, popřípadě jiný zdroj technické podpory. Rádi bychom vám poskytli víc informací týkající se této problematiky, bohužel nám v tom brání nedostatek podrobných znalostí a zkušeností. Jinak užitečný by pro vás mohl být program `debugfs`, jehož autorem je Theodore T'so.

Program `fsck` může běžet pouze na odpojených souborových systémech, v žádném případě ne na připojených (s výjimkou kořenového souborového systému připojeného při zavádění systému pouze pro čtení). To proto, že program přistupuje k „syrovému“ disku, a tak může modifikovat systém souborů bez toho, že by využíval operační systém. Když se vám podaří operační systém tímto způsobem „zmást“, téměř určitě můžete očekávat problémy.

4.8.7 Kontrola chyb na disku programem badblocks

Je vhodné pravidelně kontrolovat výskyt vadných bloků na disku. Dělá se to příkazem `badblocks`. Jeho výstupem je seznam čísel všech vadných bloků, na které program narazil. Tímto seznamem pak můžete „nakrmit“ program `fsck`, který podle něj provede záznamy do datových struktur souborového systému. Podle informací těchto účetních struktur se řídí operační systém a když pak ukládá na disk data, nepokouší se využívat v seznamu uvedené vadné bloky. v následujícím příkladu je naznačen celý postup.

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.
```

```
/dev/fd0H1440: ***** FILE system WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

Je-li v seznamu vadných bloků uveden blok, který je již některým ze souborů využíván, program `e2fsck` se pokusí tento sektor přemístit na jiné místo disku. Když je blok skutečně vadný a nejde jenom o logickou chybu vzniklou při zápisu, bude obsah souboru s největší pravděpodobností poškozený.

4.8.8 Boj s fragmentací

Když se soubor ukládá na disk, nemůže být vždy zapsaný do po sobě jdoucích bloků. Soubor, jenž není uložen do sekvenční řady za sebou jdoucích bloků je **fragmentovaný**. Trvá pak déle takovýto fragmentovaný soubor načíst, protože čtecí hlava disku se musí při čtení víc pohybovat. Proto je nežádoucí připustit fragmentaci souborů, i když ta není až tak velkým problémem pro systémy s velkou vyrovnávací pamětí. Pomocí paměti „cache“ lze totiž načítat data napřed.

Souborový systém ext2 se snaží udržet fragmentaci souborů na minimu. I v případě, že všechny bloky jednotlivých souborů nelze uložit do sektorů, jež jdou po sobě, je ukládá tak, aby byly co nejvíce pohromadě. Systém souborů ext2 navíc vždy efektivně alokuje volné sektory, které jsou nejblíže k zbylým blokům ukládaného souboru. Používáte-li proto systémy souborů ext2, nemusíte se fragmentace obávat. Přesto existuje program, jenž umí defragmentovat tento souborový systém – viz v bibliografii uvedený odkaz [TV].

Pro systém MS-DOS existuje celá řada defragmentačních programů. Ty přesouvají bloky v souborovém systému tak, aby fragmentaci odstranily. v ostatních souborových systémech se musí defragmentace dělat tak, že se vytvoří záloha souborového systému, který se znovu vytvoří a ze zálohy se obnoví soubory. Doporučení zálohovat souborový systém před jeho defragmentací se týká všech souborových systémů, protože v průběhu procesu defragmentace může dojít k poškození systému souborů i dalším chybám.

4.8.9 Další nástroje pro všechny souborové systémy

Existují i některé další nástroje užitečné pro správu souborových systémů. Program `df` ukazuje volný diskový prostor v jednom či několika souborových systémech. Program `du` ukazuje, kolik diskového prostoru zabírá adresář a všechny soubory v něm uložené. Lze jej s výhodou využít při „honu“ na uživatele, kteří zabírají svými (mnohdy zbytečnými) soubory nejvíc místa na disku.

Program `sync` zapíše všechny neuložené bloky z vyrovnávací paměti (viz podkapitolu 5.6) na disk. Jen zřídkakdy je potřeba zadávat jej ručně, automaticky to totiž dělá démon `update`. Má to význam především v případě nečekaných událostí, například když jsou procesy `update` nebo jejich pomocný proces `bdflush` nečekaně ukončeny, nebo v případě, že musíte ihned vypnout napájení a nemůžete čekat, než se opět spustí program `update`.

4.8.10 Další nástroje pro souborový systém ext2

Kromě programu, kterým se tento systém souborů vytváří (`mke2fs`), a programu pro kontrolu jeho integrity (`e2fsck`), jež jsou přístupné přímo z příkazové řádky, případně přes program „front end" nezávislý na typu souborového systému, zná souborový systém ext2 některé další nástroje, jež mohou být užitečné při správě systému.

Program `tune2fs` umí upravit parametry souborového systému. Uvedeme některé z těch významnějších:

- Maximální počet připojení programem `mount`. Příkaz `e2fsck` kontroluje, zda nebyl souborový systém připojený vícekrát, než je povoleno, a to i v případě, že je nastavený příznak „clean". U systémů, které jsou určeny k vývoji nebo testování, je rozumné tento limit snížit.
- Maximální čas mezi kontrolami integrity. Příkaz `e2fsck` hlídá maximální periodu mezi dvěma kontrolami, opět i v případě, že je nastavený příznak „clean" a souborový systém nebyl připojen vícekrát, než je povoleno. Opakované kontroly lze zakázat.
- Počet bloků vyhrazených pro kořenový souborový systém. Souborový systém ext2 rezervuje některé bloky pro kořenový svazek. Když se pak souborový systém jako celek zaplní, není potřeba nic mazat a systém lze v omezené míře spravovat. Rezervovaný počet bloků pro souborový systém „root" je primárně nastaven na 5 %, což u většiny disků stačí k tomu, aby se zamezilo jejich přeplnění. Nemá smysl rezervovat bloky pro kořenový systém souborů na disketách.

```
dumpe2fs 0.5b, 11-Mar-95 for EXT2 FS 0.5a, 94/10/23
Filesystem magic number: 0xEF53
Filesystem state: clean
Errors behavior: Continue
Inode count: 360
Block count: 1440
Reserved block count: 72
Free blocks: 1133
Free inodes: 326
```

```
First block: 1
Block size: 1024
Fragment size: 1024
Blocks per group: 8192
Fragments per group: 8192
Inodes per group: 360
Last mount time: Tue Aug 8 01:52:52 1995
Last write time: Tue Aug 8 01:53:28 1995
Mount count: 3
Maximum mount count: 20
Last checked: Tue Aug 8 01:06:31 1995
Check interval: 0
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)

Group 0:
Block bitmap at 3, Inode bitmap at 4, Inode table at 5
1133 free blocks, 326 free inodes, 2 directories
Free blocks: 307-1439
Free inodes: 35-360
```

Obrázek 4.5
Příklad výstupu programu `dumpe2fs`

Více informací najdete v manuálové stránce programu *tune2fs*.

Program `dumpe2fs` vypisuje informace o daném souborovém systému typu ext2, většinou z jeho superbloku. Na obrázku 4.5 je příklad výstupu tohoto programu. Některé z těchto informací jsou ryze technické a vyžadují hlubší pochopení problematiky fungování tohoto souborového systému. Většina údajů je ale snadno pochopitelná i pro většinu správců – laiků.

Program `debugfs` je nástroj pro ladění souborového systému. Umožňuje přímý přístup k strukturám dat souborového systému uloženým na disku, a lze jej proto použít při opravách defektů na disku, které nelze opravit automaticky programem `fsck`. Lze jej též využít k obnově smazaných souborů. Když ale chcete použít program `debugfs`, je velmi důležité, abyste skutečně věděli, co děláte. V případě, že zcela neporozumíte některé jeho funkci, se totiž může stát, že všechna svá data zničíte.

Programy `dump` a `restore` lze použít při zálohování souborového systému ext2. Jsou to specifické verze tradičních nástrojů pro zálohování používaných v systému Unix, určené speciálně pro systém souborů ext2. Víc informací o zálohování najdete v kapitole 10.

4.9 Disky bez souborových systémů

Ne na všech discích nebo diskových oblastech se vytváří souborové systémy. Například disková odkládací oblast (swap) nebude používat žádný souborový systém. Dalším příkladem jsou diskety, jejichž mechaniky se mnohdy používají pro emulaci páskové jednotky. Program `tar` nebo jiný archivační nástroj pak zapisuje přímo na „syrový“ disk bez souborového systému. Zaváděcí diskety systému Linux rovněž nemají souborový systém, pouze „syrové“ jádro systému.

To, že se na disku (disketě) nevytvoří souborový systém, má výhodu v tom, že lze využít větší část kapacity disku, protože každý souborový systém má vždy nějaké účetní režijní náklady. Navíc lze takto dosáhnout větší kompatibility disků s jinými operačními systémy, například soubor s formátem, jenž používá program `tar`, má stejnou strukturu ve všech operačních systémech, kdežto souborové systémy samotné jsou ve většině operačních systémů různé. Další výhodou je, že disky bez souborových systémů lze v případě potřeby použít velmi rychle (odpadá operace vytváření a validace souborového systému). Zaváděcí diskety Linuxu také nutně nemusí obsahovat souborový systém, ačkoliv to možné je.

Dalším z důvodů proč používat „syrová“ zařízení je možnost vytvořit přesný zrcadlový obraz – kopii disku. Když například disk obsahuje částečně poškozený souborový systém, je vhodné předtím, než se pokusíte chybu opravit, udělat přesnou kopii poškozeného disku. Pak není problém začít znova v případě, že při neúspěšném pokusu opravit chybu poškodíte systém souborů ještě víc. Jedním ze způsobů, jak zrcadlovou kopii disku udělat, je použít program `dd`:

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$ dd if=floppy-image of=/dev/fd0H1440
2880+0 records in
2880+0 records out
$
```

První příkaz `dd` uloží přesný obraz diskety do souboru `floppy-image`, druhý zapíše tento obraz na další disketu. (Samozřejmě se předpokládá, že uživatel před zadáním druhého příkazu diskety v mechanice vyměnil. Jinak by samozřejmě byly tyto příkazy k ničemu.)

4.10 Přidělování diskového prostoru

4.10.1 Způsoby rozdělování disku na diskové oblasti

Rozdělit disk na oblasti tím nejlepším možným způsobem není vůbec jednoduché. A co je nejhorší – neexistuje univerzálně správný způsob, jak toho dosáhnout. Celý problém totiž komplikuje příliš mnoho různých faktorů.

„Tradiční“ variantou je mít (relativně) malý souborový systém „root“, jenž obsahuje adresáře `/bin`, `/etc`, `/dev`, `/lib`, `/tmp`, další programy a konfigurační soubory, které jsou potřeba k tomu, aby se systém spustil a běžel. Vše, co je třeba k tomu, aby mohl být systém uveden do chodu, je právě kořenový souborový systém (na vlastní oblasti nebo na samostatném disku). Důvodem tohoto „tradičního“ schématu je fakt, že když je kořenový svazek malý a méně často používaný, je menší pravděpodobnost, že se v případě havárie systému poškodí. S takovýmto uspořádáním je také jednodušší odstranit případné problémy způsobené havárií systému. V dalším kroku se pak vytvoří samostatné diskové oblasti (nebo použijí další disky) pro stromovou strukturu svazku `/usr`, domovské adresáře uživatelů (nejčastěji pod adresářem `/home`) a pro odkládací prostor (swap). Tím, že se vyčlení vlastní disková oblast (disk) domovským adresářům, v nichž jsou uloženy soubory jednotlivých uživatelů, se zjednoduší zálohování, protože programy, které jsou uložené v adresáři `/usr`, obvykle není potřeba zálohovat. V síťovém prostředí je navíc možné adresář `/usr` sdílet mezi několika počítači (např. využitím NFS) a tím snížit celkovou potřebu diskového prostoru. Ta by jinak dosahovala několika desítek či stovek megabajtů, násobeno počtem stanic v síti.

Problém několika samostatných diskových oblastí je v tom, že rozdělují celkové množství volného diskového prostoru na mnoho malých částí. v současnosti, kdy jsou disky a (snad) i operační systémy spolehlivější, stále více uživatelů preferuje možnost mít jenom jednu oblast, ve které jsou uložené všechny soubory. Na druhou stranu zálohování (a obnovování) malých diskových oblastí je ale méně pracné.

U malých pevných disků (předpokládá se, že neděláte zrovna něco jako vývoj jádra systému), je obvykle nejlepší mít jenom jednu diskovou oblast. U velkých pevných disků je pro změnu výhodnější rozdělit je na několik větších oblastí, pouze pro případ, že by se stalo něco, s čím jste při instalaci systému nepočítali. (Uvědomte si ale, že pojmy „malý“ a „velký“ se zde používají v relativním smyslu, jedině vaše konkrétní potřeby diskového prostoru rozhodnou o tom, kde bude ležet jejich hranice.)

Máte-li k dispozici několik disků, je vhodné umístnit kořenový souborový systém (včetně adresáře `/usr`) na jeden a domovské adresáře uživatelů na druhý disk.

Je dobré připravit se na malé „experimentování“ s různými způsoby rozdělení disku na oblasti – kdykoliv, ne pouze při první instalaci systému. Je s tím sice dost práce, protože nezbytnou podmínkou je opakovaná instalace systému poté, co se některý pokus nezdaří. Nicméně je to jediný způsob, jak si ověřit správnost rozdělení disku.

4.10.2 Nároky na diskový prostor

Distribuce Linuxu, kterou budete instalovat, vám obvykle nějakým způsobem sdělí, kolik diskového prostoru je potřeba pro různé konfigurace operačního systému. Programy, které se instalují dodatečně, se budou většinou chovat stejně. Díky tomu si můžete udělat představu o nárocích na diskový prostor a naplánovat si jeho rozdělení. Měli byste se ale připravit i na budoucnost a vyhradit si nějaké místo navíc pro věci, na které si vzpomenete později.

Prostor, který byste měli vyčlenit pro soubory uživatelů systému, závisí na tom, co budou dělat. Většina lidí totiž obvykle spotřebuje tolik diskového prostoru, kolik je jenom možné. Nicméně množství, které jim skutečně stačí, je velmi různé. Někteří uživatelé vystačí s psaním textů a spokojeně přežijí i s několika megabajty. Jiní dělají složitou editaci grafických souborů a budou potřebovat gigabajty volného prostoru.

Mimochodem, když budete při odhadech nároků na diskový prostor srovnávat velikosti souborů v kilobajtech nebo megabajtech a diskový prostor v megabajtech, uvědomte si, že tyto dvě jednotky mohou být různé. Někteří výrobci disků rádi tvrdí, že kilobajt je 1 000 bajtů a megabajt je 1 000 kilobajtů, kdežto zbytek počítačového světa používá pro oba koeficienty číslo 1 024. Proto váš 345MB pevný disk bude mít ve skutečnosti pouze 330 MB.[10]

O přidělování odkládacího prostoru pojednává odstavec 5.5.

4.10.3 Příklady rozvržení diskového prostoru

Autor manuálu dlouho používal 109MB pevný disk. V současné době má k dispozici pevný disk o velikosti 330 MB. V dalším textu bude vysvětleno, jak a proč byly tyto disky rozděleny na jednotlivé diskové oblasti.

Disk o velikosti 109 MB byl velmi často „přerozdělován“ mnoha různými způsoby podle toho, jak se měnily konkrétní potřeby a používané operační systémy. Popíšeme dva typické scénáře. Při prvním z nich byly na jednom disku instalovány operační systémy MS-DOS a Linux. První disková oblast o velikosti asi 20 MB byla vyhrazena pro systém MS-DOS, některý z kompilátorů jazyka C, textový editor, několik dalších utilit a rozpracovaný program. Několik megabajtů volného diskového prostoru zbylo proto, aby nevznikaly pocity klaustrofo-

[10] Sic transit discus mundi.

bie. Odkládacímu prostoru (swap) systému Linux bylo vyhrazeno 10 MB na samostatné diskové oblasti a zbytek, tedy 79 MB, na další diskové oblasti byl vyčleněn pro soubory operačního systému Linux. Rozdělovat takovýto prostor na samostatné oblasti pro souborové systémy „root“, `/usr` a domovské adresáře `/home` nemá praktický význam.

Když pak nebude třeba systém MS-DOS, lze změnit rozdělení disku tak, že se vyhradí 12 MB pro odkládací prostor Linuxu a zbytek bude opět jako jeden souborový systém.

Disk o velikosti 330 MB lze rozdělit na několik diskových oblastí následujícím způsobem:

5 MB	kořenový souborový systém
10 MB	oblast pro odkládací prostor (swap)
180 MB	souborový systém `/usr`
120 MB	souborový systém `/home`
15 MB	neobsazená disková oblast

Neobsazenou diskovou oblast lze využívat na různé „experimenty“, při nichž je potřeba mít samostatný diskový segment, např. při testování různých distribucí Linuxu, nebo srovnávání rychlosti souborových systémů a podobně. Jinak lze neobsazenou diskovou oblast rovněž využít jako odkládací prostor (hlavně v případě, že máte rádi hodně otevřených oken).

4.10.4 Zvětšování diskového prostoru pro Linux

Zvětšení diskového prostoru vyhrazeného pro Linux je jednoduché. Především v případě, že se instalují nové pevné disky (popis instalace disků jde ale nad rámec této knihy). Je-li to nutné, je potřeba disky zformátovat. Pak se podle výše uvedeného postupu vytvoří diskové oblasti a souborový systém a přidají se odpovídající řádky do souboru `/etc/fstab` kvůli tomu, aby se nové disky připojovaly automaticky.

4.10.5 Tipy jak ušetřit místo na disku

Nejlepším způsobem jak ušetřit diskový prostor je vyvarovat se instalování nepotřebných programů. Většina distribucí Linuxu při instalaci nabízí možnost výběru balíků programů, které se mají instalovat. Podle této nabídky byste měli být schopni analyzovat své potřeby a pak pravděpodobně zjistíte, že většinu z nich nebudete potřebovat. Tím se ušetří hodně místa na disku, protože některé z programů a aplikačních balíků jsou dost objemné. I když zjistíte, že budete některou aplikaci nebo i celý balík programů potřebovat, určitě nebudete muset

instalovat všechny jejich součásti. Obvykle není potřeba instalovat například „on-line“ dokumentaci, stejně jako některé ze souborů Elisp pro verzi GNU programu Emacs, některé z fontů pro X11 či některé knihovny programovacích jazyků.

V případě, že nemůžete některé balíky programů trvale odinstalovat, můžete využít kompresi. Komprimační programy jako `gzip` nebo `zip` spakují (a rozbalí) jednotlivé soubory, případně celé skupiny souborů. Systém `gzexe` transparentně komprimuje a dekomprimuje programy tak, že to uživatel v běžném provozu nepostřehne (nepoužívané programy se komprimují a rozbalí se, až když jsou potřeba). V současné době se testuje systém `DouBle`, který transparentně komprimuje všechny soubory v souborovém systému. (Systém `DouBle` pracuje na stejném principu jako program Stacker pro operační systém MS-DOS, který možná znáte.)*

* Poznámka korektora: Dalším používaným programem pro komprimaci je `bzip2`, pomocí něhož jsou komprimovány i zdrojové texty jádra operačního systému Linux.

Kapitola

5

Správa paměti

Minnet, jag har tappat mitt minne,
är jag svensk eller finne
kommer inte ihĺg. . .
(Bosse Österberg)

V této kapitole jsou popsány hlavní rysy systému správy paměti operačního systému Linux, tedy subsystémy virtuální paměti a diskové vyrovnávací paměti (angl. disk buffer cache). Kapitola dále popisuje jejich účel, funkce a další podrobnosti, které správce systému musí vzít v úvahu.

5.1 Co je virtuální paměť?

Operační systém Linux podporuje systém **virtuální paměti**, to znamená, že používá disk jako rozšíření paměti RAM. Tím se efektivní velikost využitelné paměti odpovídajícím způsobem zvětší. Jádro systému zapisuje obsah právě nevyužívaných paměťových bloků na pevný disk a paměť se tak může využívat pro jiné účely. Když pak přijde požadavek na její původní obsah, bloky z disku se načtou zpět do paměti RAM. To vše probíhá z pohledu uživatele zcela transparentně. Programy běžící pod Linuxem si zjišťují pouze velikost dostupné paměti RAM a nestarají se o to, že její část je občas uložená na disku. Přirozeně, čtení a zápis na pevný disk je pomalejší (zhruba o tři řády), než využití reálné paměti, takže programy neběží tak rychle. Část disku, která se využívá jako virtuální paměť, se nazývá **odkládací prostor** (angl. **swap space**).

Linux může použít pro „swap“ jak normální soubor uložený v souborovém systému, tak zvláštní diskovou oblast. Předností samostatného diskového segmentu je rychlost, výhodou odkládacího souboru je jednodušší možnost změny celkové velikosti odkládacího prostoru. Není přitom totiž potřeba měnit rozdělení celého pevného disku na oblasti, kdy navíc hrozí nutnost kompletní reinstalace systému (při případném nezdaru této operace). Víte-li, jak velký odkládací prostor budete potřebovat, zvolte odkládací prostor na zvláštní oblasti disku. Pokud si nároky na „swap“ nejste zcela jisti, zvolte nejdřív odkládací prostor v souboru. Když budete systém nějakou dobu používat, budete schopni odhadnout, kolik odkládacího prostoru skutečně potřebujete. Až budete mít ohledně předpokládané velikosti požadovaného odkládacího prostoru jasno, vytvoříte pro něj zvláštní diskovou oblast.

Měli byste rovněž vědět, že Linux umožňuje využívat současně několik odkládacích oblastí, případně několik odkládacích souborů. Takže když potřebujete pouze příležitostně větší množství odkládacího prostoru, je lepší (místo trvale vyhrazené rezervy) nastavit další soubor pro „swap“ navíc.

Poznámka k terminologii používané v oblasti operačních systémů: v odborných kruzích se obvykle rozlišuje mezi odkládáním (angl. swapping), tedy zapsáním celého procesu do odkládacího prostoru na disku, a stránkováním (angl. paging), tedy zapisováním pouhých částí pevné velikosti (obvykle několik kilobajtů) najednou. Stránkování je obecně výkonnější a je to metoda, kterou používá i operační systém Linux. Tradiční terminologie systému Linux ale používá pojem „odkládání“ (swapping).[1]

5.2 Vytvoření odkládacího prostoru na disku

Odkládací soubor (swap) je běžný soubor a není pro jádro systému ničím zvláštní. Jediná vlastnost, která má pro jádro význam, je, že odkládací soubor nemá tzv. „prázdná místa“ (angl. holes) a že je připraven pro použití programem `mkswap`. Musí být navíc (z důvodů implementace) uložený na lokálním disku, takže nemůže být uložen v souborovém systému, který je připojen pomocí NFS.

Zmínka o „dírách“ je důležitá. Odkládací soubor si rezervuje určitý diskový prostor, takže jádro systému pak může rychle odložit stránku paměti bez toho, že by muselo absolvovat celou proceduru alokace diskového prostoru, která se používá pro běžný soubor. Jádro využívá pouze ty sektory, které byly pro odkládací soubor vyhrazeny. Protože „prázdné místo“ v souboru znamená, že tomuto místu souboru nejsou přiděleny žádné diskové sektory, nebylo by pro jádro systému dobré je využívat.

[1] Což pokaždé zcela zbytečně hrozným způsobem rozladí množství počítačových odborníků.

Jeden ze způsobů, kterým lze vytvořit odkládací soubor bez prázdných míst, je:

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

kde `/extra-swap` je jméno odkládacího souboru, jehož velikost je uvedená za parametrem `count=`. Ideální je zvolit velikost jako násobek 4, protože jádro systému zapisuje do odkládacího prostoru **stránky paměti**, které jsou 4 kilobajty velké. Nebude-li velikost násobkem 4, může být poslední pár kilobajtů nevyužitý.

Disková oblast pro „swap“ rovněž není ničím neobvyklým. Vytvoří se stejně, jako každá jiná disková oblast. Jediným rozdílem je, že se používá jako „syrové“ zařízení, tedy bez souborového systému. Je dobré označit ji jako typ 82 („Linux swap“). I když to jádro systému striktně nevyžaduje, vnese to do seznamu diskových oblastí řád.

Poté, co byli vytvořeny diskový segment pro odkládací prostor nebo odkládací soubor, je potřeba zapsat na jejich začátek odpovídající označení, které obsahuje některé informace významné z hlediska správy systému a informace, jež využívá jádro systému. Provede se to příkazem `mkswap` tímto způsobem:

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

Je důležité si uvědomit, že odkládací prostor systém zatím nevyužívá. Sice existuje, ale jádro systému jej jako virtuální paměť zatím nezná.

Při zadávání příkazu `mkswap` byste měli být velice opatrní, protože program nekontroluje, zda se soubor nebo disková oblast nevyužívá k jiným účelům. *Příkazem **mkswap** proto můžete lehce přepsat důležité soubory nebo celé diskové segmenty!* Naštěstí budete tento příkaz potřebovat pouze když instalujete operační systém.

Linuxový manažer paměti omezuje velikost každého z odkládacích prostorů na asi 127 MB (z různých technických důvodů je současný limit (4096-10) x 8 x 4 096=133 890 048 bajtů neboli 127,6875 MB).* Avšak současně můžete využívat až 16 odkládacích prostorů, tedy celkem téměř 2 GB.[2]

* Poznámka korektora: Dnes už existují i záplaty (ale nikoli chybové) do jádra, které umožňují využívat odkládací soubory o velikosti až do 2GB.

2 Gigabajt sem, gigabajt tam, o reálné paměti jsme začali mluvit dost brzo.

5.3 Využívání odkládacího prostoru

Využívání nově vytvořeného odkládacího prostoru lze zahájit příkazem `swapon`. Příkaz sdělí jádru systému, že odkládací prostor, jehož úplná cesta se zadává jako argument příkazu, lze od této chvíle používat. Takže když budete chtít začít využívat dočasný soubor jako odkládací prostor, zadáte tento příkaz:

```
$ swapon /extra-swap
$
```

Odkládací prostory lze využívat automaticky poté, co budou zapsány v souboru `/etc/fstab`, například:

```
/dev/hda8    none    swap    sw    0    0
/swapfile    none    swap    sw    0    0
```

Spouštěcí skripty vykonávají příkaz `swapon -a`, jenž zahájí odkládání do všech odkládacích prostorů uvedených v souboru `/etc/fstab`. Takže příkaz `swapon` se často používá jenom když je potřeba použít odkládací prostor navíc.

Příkazem `free` lze monitorovat využívání odkládacích prostorů. Příkaz zobrazí celkové množství odkládacího prostoru, který je v systému využíván:

```
$ free

             total       used       free     shared    buffers
Mem:         15152      14896        256      12404       2528
-/+ buffers:            12368       2784
Swap:        32452       6684      25768
$
```

v prvním řádku výstupu (položka `Mem:`) se zobrazuje velikost fyzické paměti. Sloupec `total` neukazuje skutečnou velikost fyzické paměti, kterou využívá jádro systému, ta má obvykle asi jeden megabajt. v sloupci `used` je zobrazeno množství využívané paměti (v druhém řádku chybí údaj o zásobnících vyrovnávací paměti). Sloupec `free` udává celkové množství nevyužité paměti. Sloupec `shared` ukazuje paměť sdílenou několika procesy – platí čím více, tím lépe. v sloupci `buffers` je zobrazena aktuální velikost vyrovnávací diskové paměti.

v posledním řádku (`Swap:`) jsou analogické informace, jež se týkají odkládacích prostorů. Když jsou v tomto řádku samé nuly, není odkládací prostor systému aktivovaný.

Stejné informace lze získat příkazem `top`, nebo z údajů v souborovém systému `proc`, přesněji v souboru `/proc/meminfo`. Obvykle je ale dost obtížné získat informace o využití jednoho konkrétního odkládacího prostoru.

Odkládací prostor lze vyřadit z činnosti příkazem `swapoff`. Příkaz pravděpodobně využijete pouze pro vyřazení dočasných odkládacích prostorů. Všechny stránky, které jsou uloženy v odkládacím prostoru, se po zadání příkazu `swapoff` nejdříve načtou do paměti. Když není dostatek fyzické paměti, do které by se načetly, budou uloženy do některého z jiných odkládacích prostorů. Když není ani dostatek virtuální paměti na odložení všech načítaných stránek odkládacího prostoru, jenž má být vyřazen z činnosti, začnou problémy. Po delší době by se operační systém měl zotavit, ale mezitím bude prakticky nepoužitelný. Proto byste měli předtím, něž vyřadíte některý odkládací prostor z činnosti, zkontrolovat (např. příkazem `free`), jestli máte dostatek volné paměti.

Všechny odkládací prostory, které se používají automaticky po zadání příkazu `swapon -a`, lze vyřadit z činnosti příkazem `swapoff -a`. Příkaz vyřadí z činnosti pouze odkládací prostory uvedené v souboru `/etc/fstab`. Odkládací prostory přidané ručně zůstanou nadále v činnosti.

Někdy mohou nastat situace, že se využívá příliš mnoho odkládacího prostoru, i když má systém dostatek volné fyzické paměti. Může se to stát například v situaci, kdy jsou v jednom okamžiku velké nároky na virtuální paměť, ale po chvíli je ukončen některý větší proces, který využívá větší část fyzické paměti, a ten paměť uvolní. Odložená data se ale nenačítají do paměti automaticky a zůstanou uložena na disku až do doby, než budou potřeba. Fyzická paměť by tak mohla zůstat dost dlouho volná, nevyužitá. Není potřeba se tím znepokojovat, ale je dobré vědět, co se v systému děje.

5.4 Sdílení odkládacího prostoru s jinými operačními systémy

Virtuální paměť používá mnoho operačních systémů. Vzhledem k tomu, že každý ze systémů využívá svůj odkládací prostor jenom když běží (tedy nikdy ne několik systémů současně), odkládací prostory ostatních operačních systémů pouze zabírají místo na disku. Pro operační systémy by bylo efektivnější sdílet jediný odkládací prostor. I to je možné, ale je potřeba se tomu trochu více věnovat. V publikaci „Tips – HOWTO“ je uvedeno několik praktických rad, jak systém sdílení odkládacího prostoru realizovat.

5.5 Přidělování odkládacího prostoru

Možná někdy narazíte na radu, že máte přidělovat dvakrát tolik odkládacího prostoru, než máte fyzické paměti. To je pouhá pověra. Uvádíme proto správný postup:

1. Zkuste odhadnout, jaké jsou vaše celkové potřeby paměti, tedy největší množství paměti, které budete pravděpodobně v jednom okamžiku potřebovat. To je dané součtem paměťových nároků všech programů, které budou (pravděpodobně vždy) běžet současně.

   Když například chcete, aby běželo grafické uživatelské rozhraní X Window, měli byste mu přidělit asi 8 MB paměti RAM. Kompilátor gcc vyžaduje několik megabajtů (kompilace některých souborů by mohla mít neobvykle velké nároky na paměť, jež by mohly dosáhnout až několika desítek megabajtů, ale běžně by měly stačit zhruba čtyři megabajty). Jádro samotné bude využívat asi jeden megabajt paměti, běžné interprety příkazů a jiné menší utility několik stovek kilobajtů (řekněme asi jeden megabajt dohromady). Není potřeba být v odhadech úplně přesný, stačí udělat velmi hrubý odhad, ale ten by měl být spíše pesimistický.

   Je důležité si uvědomit, že když bude systém současně používat více uživatelů, budou paměť RAM potřebovat všichni. Avšak budou-li současně ten samý program používat dva uživatelé, celková potřeba paměti obvykle nebude dvojnásobná, protože stránky kódu a sdílené knihovny budou v paměti pouze jednou.

   Pro správný odhad nároků na paměť jsou užitečné příkazy `free` a `ps`.

2. K odhadu podle kroku 1 připočítejte nějakou rezervu. To proto, že odhady paměťových nároků programů budou velmi pravděpodobně nedostatečné – je možné, že na některé aplikace, které budete chtít používat, zapomenete. Takto budete mít jistotu, že pro tento případ máte nějaké to místo navíc. Mělo by stačit pár megabajtů. (Je lepší vyčlenit příliš mnoho, než moc málo odkládacího prostoru. Ale není potřeba to přehánět a alokovat celý disk, protože nevyužitý odkládací prostor zbytečně zabírá místo. V dalších částech této kapitoly bude uvedeno, jak přidat další odkládací prostor.) Vzhledem k tomu, že se lépe počítá s celými čísly, je dobré hodnoty zaokrouhlovat směrem nahoru, řádově na další celé megabajty.

3. Na základě těchto výpočtů budete vědět, kolik paměti budete celkem potřebovat. Takže když odečtete velikost fyzické paměti od celkových nároků na paměť, dozvíte se, kolik odkládacího prostoru musíte celkem vyčlenit. (U některých verzí Unixu se musí vyčlenit i paměťový prostor pro obraz fyzické paměti, to znamená, že velikost paměti vypočítaná podle kroku 2 představuje skutečné nároky na paměť a není třeba odečítat velikost fyzické paměti od celkových nároků na paměťový prostor.)

4. Je-li vypočítaná velikost odkládacího prostoru o hodně větší, než dostupná fyzická paměť (více, než dvakrát), měli byste raději více investovat do fyzické paměti. Jinak bude výkon systému příliš nízký.

Vždy je dobré mít v systému alespoň nějaký odkládací prostor, i když podle výpočtů žádný nepotřebujete. Linux používá odkládací prostor poněkud agresivněji, snaží se mít tolik volné fyzické paměti, kolik je jenom možné. Linux navíc odkládá paměťové stránky, které se nepoužívají, i když se fyzická paměť zatím nevyužívá. Tím se totiž zamezí čekání na případné odkládání – data lze takto odložit dřív, v době, kdy je disk jinak nevyužitý.

Odkládací prostor lze rozdělit mezi několik disků. To může v některých případech zlepšit výkon systému, jenž v tomto směru závisí na relativních rychlostech disků a jejich přístupových modelech. Lze samozřejmě experimentovat s několika variantami, ale mějte vždy na paměti to, že je velmi lehké u takovýchto pokusů pochybit. Neměli byste také věřit tvrzením, že některá z možností je lepší než ostatní, protože to nemusí být vždy pravda.

5.6 Disková vyrovnávací paměť

Čtení z disku[3] je ve srovnání s přístupem k (reálné) paměti velmi pomalé. Navíc se v běžném provozu velmi často z disku načítají stejná data několikrát během relativně krátkých časových intervalů. Například v případě elektronické pošty se nejdříve musí načíst došlá zpráva, když na ni chcete odpovědět, načte se ten samý dopis do editoru, pak stejná data načte i poštovní program, který je kopíruje do souboru s došlou, případně odeslanou poštou a podobně. Zkuste si představit, jak často se může zadávat příkaz `ls` na systému s mnoha uživateli. Jediným načtením informací z disku a jejich uložením do paměti do doby, než je nebude potřeba, lze zrychlit především operace čtení z disku. Paměť vyhrazená pro tyto účely, tedy pro ukládání z disku načítaných a na disk zapisovaných dat (angl. **disk buffering**), se nazývá **disková vyrovnávací paměť** (angl. **buffer cache**).

Protože paměť je naneštěstí omezený a navíc „vzácný“ systémový zdroj, nemůže být vyrovnávací paměť obvykle příliš velká (nemohou v ní být totiž uložena úplně všechna data, která by chtěl někdo používat). Když se vyrovnávací paměť zaplní, data, jež se nepoužívala nejdéle, se odloží na disk a takto uvolněná vyrovnávací paměť se využije na ukládání dalších dat.

Vyrovnávací paměť funguje při zápisu na disk. Jednak proto, že data, která se zapisují na disk, se velmi často brzo opakovaně načítají (např. zdrojový kód je ukládán do souboru a pak jej opět načítá kompilátor), takže je výhodné uložit data zapisovaná na disk i do vyrovnávací pa-

3 Pochopitelně s výjimkou disku RAM.

měti. Jednak již pouhým uložením dat do paměti „cache“ a tím, že se nezapisují na disk ihned, se zlepší odezva programu, jenž data zapisuje. Zápis dat na disk pak může probíhat na pozadí bez toho, že by se tím zpomalovaly ostatní programy.

Diskové vyrovnávací paměti používá většina operačních systémů (i když je možné, že se jim říká nějak jinak). Ne všechny ale pracují podle výše uvedených principů. Část z nich se označuje jako **vyrovnávací paměti s přímým zápisem** (angl. **write-through**). Zapisují data na disk ihned (data přirozeně zůstanou rovněž uložena ve vyrovnávací paměti). Je-li zápis odložen na pozdější dobu, označují se tyto vyrovnávací paměti jako **paměti s odloženým zápisem** (angl. **write-back cache**). Posledně uvedený systém je zřejmě efektivnější, než prvně jmenovaný, je ale také o něco více náchylný k chybám. V případě havárie systému, případně výpadku proudu v nevhodném okamžiku, či vytažení diskety z disketové mechaniky předtím, než jsou data čekající ve vyrovnávací paměti na uložení zapsána, se změny uložené ve vyrovnávací paměti obvykle ztratí. Navíc by takováto událost mohla zapříčinit chyby v souborovém systému (je-li na disku či disketě vytvořen), jelikož mezi nezapsanými daty mohou být i důležité změny účetních informací samotného souborového systému.

Proto by se nikdy nemělo vypínat napájení počítače dřív, než správně proběhla procedura zastavení systému (viz kapitola 6). Rovněž by se neměla vytahovat disketa z mechaniky předtím, než byla odpojena příkazem `umount` (byla-li předtím připojená), případně předtím, než program, jenž přistupoval na disketu, signalizuje, že práci s disketovou mechanikou ukončil a než přestane svítit kontrolní dioda LED mechaniky pružného disku. Příkaz `sync` **vyprázdní** vyrovnávací paměť, tedy uloží všechna nezapsaná data na disk. Lze jej použít vždy, když si nejste zcela jisti, jestli se obsah vyrovnávací paměti bezpečně uložil na disk. V tradičních systémech Unix existuje program `update`, který běží na pozadí a spouští příkaz `sync` každých 30 sekund. V těchto systémech obvykle není potřeba příkaz `sync` používat ručně. V systému Linux běží další démon `bdflush`, jenž dělá něco podobného jako program `sync`, ale častěji a ne v takovém rozsahu. Navíc se tímto způsobem vyhnete náhlému „zamrznutí“ systému, které může někdy příkaz `sync` při větším rozsahu vstupně-výstupních operací způsobit.

V systému Linux se program `bdflush` spouští příkazem `update`. V případě, že je démon `bdflush` z jakéhokoliv důvodu neočekávaně ukončen, jádro systému o tom podá varovné hlášení. Obvykle ale není důvod se toho obávat. Proces `bdflush` lze spustit ručně (příkazem `/sbin/update`).

Ve skutečnosti se obvykle do vyrovnávací paměti neukládají soubory, ale bloky, což jsou nejmenší jednotky při vstupně-výstupních diskových operacích (v Linuxu mají nejčastěji velikost 1 kB). Tímto způsobem se do vyrovnávací paměti ukládají i adresáře, superbloky, další účetní data souborového systému a data z disků bez souborových systémů.

O efektivitě vyrovnávací paměti prvotně rozhoduje její velikost. Malá vyrovnávací paměť je téměř nepoužitelná. Bude v ní uloženo tak málo dat, že se vyrovnávací paměť vždy vyprázdní předtím, než mohou být data použita. Kritická velikost záleží na tom, jaké množství dat se z disků čte a na disky zapisuje a jak často se k těmto údajům přistupuje. Jediným způsobem jak to zjistit, je experimentovat.

Má-li vyrovnávací paměť pevnou velikost, není výhodné ji mít příliš velkou. To by mohlo kriticky zmenšit dostupnou paměť a zapříčinit časté odkládání (jež je navíc velmi pomalé). Aby byla reálná paměť využívána co nejefektivněji, Linux automaticky využívá veškerou volnou paměť RAM jako vyrovnávací paměť. Naopak, operační systém vyrovnávací paměť automaticky zmenšuje, když běžící programy požadují víc fyzické paměti.

V systému Linux není potřeba dělat nic zvláštního pro to, aby bylo možné vyrovnávací paměť využívat. Systém ji totiž používá zcela automaticky. Kromě správného postupu při zastavení systému a vyjímání disket z mechaniky se prakticky o vyrovnávací paměť vůbec nemusíte starat.

Kapitola

6

Zavádění systému a ukončení jeho běhu

Spusť mě
Ó... nesmíš... nesmíš
Nikdy, nikdy, nikdy nesmíš zastavit
Rozjeď to
Ó... rozjeď to, nikdy, nikdy, nikdy
Nezklameš,
nezklameš
(Rolling Stones)

Tato kapitola popisuje, co se děje po tom, co je systém Linux spuštěn, a poté, co je zastaven. Dále uvádí správný postup procedur zavedení a zastavení systému. Když se tyto postupy nedodrží, mohou se některé soubory poškodit anebo ztratit.

6.1 Zavádění a ukončení práce systému - přehled

Zapnutí počítače, po němž se zavede operační systém[1], se říká **zavedení systému** (angl. **booting**). Tento původní termín vznikl z anglické fráze „pull yourself UP by your own bootstraps“, tedy vytáhnout sebe sama za jazyk vlastních bot, což obrazně vystihuje proces, který probíhá při zapnutí počítače.

[1] První počítače nestačilo pouze zapnout. Bylo ještě potřeba manuálně zavést operační systém. To až tyhle nově opentlovaná „udělátka“ zařídí všechno sami.

V průběhu zaváděcích sekvencí počítač nejdříve zavede krátký kód, kterému se říká **zavaděč** (angl. **bootstrap loader**), jenž pak zavede a spustí samotný operační systém. Zavaděč je obvykle uložen na předem určeném místě pevného disku nebo diskety. Důvodem pro rozdělení procedury zavádění systému do dvou kroků je to, že samotný operační systém je velký a složitý, kdežto část kódu, kterou počítač zavádí jako první, je velmi krátká (má několik stovek bajtů). Tím se zamezí nežádoucímu komplikování firmwaru.

Různé typy počítačů provádí úvodní sekvence zavádění systému různě. Pokud jde o počítače třídy PC, ty (přesněji jejich systém BIOS) načítají první sektor pevného disku nebo diskety, kterému se říká **zaváděcí sektor** (angl. **boot sector**). Zavaděč je uložen v tomto prvním – zaváděcím sektoru. Zavaděč pak načítá operační systém z jiného místa na disku, případně i z nějakého jiného média.

Poté, co se operační systém Linux zavede, inicializují se technické prostředky výpočetního systému a ovladače jednotlivých zařízení. Pak se spustí proces `init`, který spouští další procesy, umožňující uživatelům přihlásit se do systému a pracovat v něm. O podrobnostech této části zaváděcího procesu se pojednává níže.

Aby bylo možné systém Linux zastavit, je nejdříve nutné požádat všechny procesy, aby ukončily činnost (zavřely všechny otevřené soubory, popřípadě zařídily další důležité věci – obrazně řečeno „uklidily“ po sobě). Potom se odpojí souborové systémy, odkládací prostory a nakonec se na konzole objeví zpráva, že lze počítač vypnout. Jestli se nedodrží správný postup, mohou se stát (a obvykle se stanou) dost nepříjemné věci. Nejzávažnějším problémem je v tomto případě nevyprázdněná vyrovnávací disková paměť souborového systému. Všechna data ve vyrovnávací paměti se totiž ztratí, takže souborový systém na disku bude nekonzistentní a pravděpodobně nepoužitelný.

6.2 Proces zavádění systému při pohledu z blízka

Systém Linux lze zavést buď z diskety, z pevného disku nebo i z CD. Příslušná kapitola příručky „Průvodce instalací a začátky“ ([Wel]), jež podrobně pojednává o instalaci operačního systému Linux, uvádí návod jak systém instalovat oběma z výše uvedených způsobů.

Když počítač PC startuje, systém BIOS provádí různé testy a kontroluje, zda je vše v pořádku[2]. Až pak zahájí skutečné zavádění operačního systému. Vybere zaváděcí diskovou jednotku. Typicky první disketovou mechaniku – je-li v mechanice zasunutá disketa, jinak první pevný disk – nejde-li o bezdiskový počítač. Pořadí prohledávání lze konfigurovat. Poté se na-

2 Říká se tomu **test systému při zapnutí** (angl. **power on self test**, zkráceně **POST**).

čte první sektor tohoto disku, kterému se říká **zaváděcí sektor** (angl. **boot sector**). U pevných disků se označuje jako **zaváděcí sektor disku** (angl. **master boot record**), protože na pevném disku může být několik diskových oblastí, každá se svým vlastním zaváděcím sektorem.

Zaváděcí sektor obsahuje krátký program (dostatečně krátký na to, aby se vešel do jednoho bloku), úkolem kterého je načíst z disku operační systém a spustit jej. Pokud jde o zavádění systému z diskety – zaváděcí sektor pružného disku obsahuje kód, který načte jenom prvních několik stovek bloků (v závislosti na aktuální velikosti jádra) do předem určeného místa v paměti. Na linuxové zaváděcí disketě totiž obvykle nebývá vytvořen souborový systém, je na ní uloženo jenom jádro systému v několika po sobě jdoucích sektorech, což proces zavádění systému značně zjednodušuje. Systém lze zavést rovněž z diskety se souborovým systémem, a to pomocí zavaděče systému Linux (angl. LInux LOader, zkráceně LILO).

Když se systém zavádí z pevného disku, kód obsažený v zaváděcím sektoru disku si prohlédne tabulku rozdělení disku, která je v tomto sektoru také uložená. Pak zaváděcí kód identifikuje aktivní diskovou oblast (oblast, která je označená jako zaváděcí), načte zaváděcí sektor aktivní oblasti a spustí kód uložený v tomto zaváděcím sektoru. Kód v zaváděcím sektoru diskové oblasti dělá v podstatě to samé, co kód obsažený v zaváděcím sektoru diskety. Načte jádro systému z aktivní diskové oblasti a spustí jej. Avšak v detailech se tyto procedury poněkud liší, protože obecně není výhodné mít zvláštní diskovou oblast čistě pro obraz jádra systému, proto kód v zaváděcím sektoru diskové oblasti nemůže jednoduše sekvenčně číst data z disku, jako při zavádění systému z diskety. Musí hledat příslušné sektory všude, kam je souborový systém uložil. Existuje několik způsobů řešení tohoto problému. Nejběžnější možností je použít zavaděč operačního systému Linux LILO. (Další detaily tohoto postupu nejsou v této chvíli podstatné. Více informací najdete v dokumentaci zavaděče LILO, která je v tomto směru dokonalejší.)

Když se zavádí operační systém pomocí zavaděče LILO, pokračuje se dál v zaváděcí sekvenci, takže zavaděč LILO načte a zavede implicitně vybrané jádro Linuxu. Je ale možné nakonfigurovat jej tak, aby zavedl některý z více obrazů jádra systému, nebo i jiný operační systém, než Linux. Uživatel systému si tak při zavádění může vybrat, které jádro, případně operační systém, se implicitně zavede při spuštění počítače. Zavaděč LILO lze nakonfigurovat i tak, že při zmáčknutí kláves Alt, Shift, nebo Ctrl v okamžiku zavádění systému (tj. při spouštění LILO) se nebude zavádět systém ihned. Místo toho se zavaděč zeptá, který operační systém se bude zavádět. LILO lze nastavit i tak, že se bude při zavádění systému ptát na požadovaný systém, ale s volitelným časovým prodlením, po kterém se zavede implicitně určené jádro.

Zavaděč LILO rovněž umožňuje předat jádru systému argumenty příkazové řádky, které se zadávají za jménem jádra nebo operačního systému, který se zavádí.

META: Existují i jiné zavaděče, než LILO. Informace o nich budou přidány v některé z dalších verzí manuálu. `loadlin`.

Zavádění systému z diskety i z pevného disku má své výhody. Obecně je zavádění z disku příjemnější, uživatel je ušetřen zbytečných nepříjemností v případě, že v disketách „zcela náhodou“ zavládne nepořádek. Kromě toho je zavádění z disku rychlejší. Naopak je výrazně pracnější konfigurovat operační systém tak, aby se zaváděl z disku. Proto mnoho uživatelů dává v první fázi přednost zavádění systému z diskety a až pak, když bude nainstalovaný systém fungovat, doinstalují zavaděč LILO a budou Linux zavádět z pevného disku.

Až poté, co se jádro Linuxu některým z výše uvedených způsobů načte do paměti, je skutečně spuštěno. Probíhají zhruba tyto věci:

- Jádro Linuxu se instaluje v komprimovaném tvaru, takže se před samotným zavedením samo dekomprimuje. Stará se o to krátký program, jenž je obsažen v začátku obrazu jádra.

- Rozezná-li systém kartu Super VGA, která má nějaké zvláštní textové režimy (jako například 100 sloupců na 40 řádků), zeptá se vás, který z režimů budete používat. V průběhu kompilace jádra systému lze video-mód implicitně nastavit – pak se systém při startu na video-režim nedotazuje. Stejného efektu lze dosáhnout konfigurací zavaděče systému LILO, nebo příkazem `rdev`.

- v dalším kroku jádro zkontroluje, jaké další hardwarové komponenty jsou k dispozici (pevné disky, diskety, síťové adaptéry atd.) a odpovídajícím způsobem nastaví některé ze svých ovladačů zařízení. V průběhu této „inventury“ vypisuje jádro zprávy o tom, která zařízení byla nalezena. Například při zavádění systému, jenž používá autor, se vypisují tyto hlášky:

```
LILO boot:
Loading linux.
Console: colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3.94 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
lp_init: lp1 exists (0), using polling driver
Memory: 7332k/8192k available (300k kernel code, 384k reserved,
  176k data)
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
Loopback device init
Warning WD8013 board not found at i/o = 280.
Math coprocessor using irq13 error reporting.
```

```
Partition check:
  hda: hda1 hda2 hda3
VFS: Mounted root (ext filesystem).
Linux version 0.99.pl9-1 (root@haven) 05/01/93 14:12:20
```

Přesný formát výstupů se na různých systémech liší, a to v závislosti na hardwarové konfiguraci výpočetního systému, verzi operačního systému a jeho konkrétním nastavení.

- Potom se jádro Linuxu pokusí připojit kořenový svazek. Přípojné místo lze nastavit při kompilaci jádra, později pak příkazem `rdev`, případně zavaděčem LILO. Typ souborového systému je detekován automaticky. Jestli připojení souborového systému selže (například proto, že jste při kompilaci jádra systému zapomněli uvést odpovídající ovladač souborového systému), jádro zpanikaří a systém se v tomto kroku zastaví (beztak mu nic jiného ani nezbývá). Kořenový svazek se obvykle připojuje pouze pro čtení (to lze nastavit stejným způsobem, jako místo připojení). Díky tomu se může provést kontrola souborového systému při jeho připojování. Není vhodné prověřovat systém souborů, jenž je připojen pro čtení i zápis.

- Poté spustí jádro systému na pozadí program `init`, jenž se nachází v adresáři `/sbin`. Program `init` bude vždy procesem číslo 1 a jeho úkolem jsou různé „úklidové práce“ spojené se startem systému. Přesný postup toho, co jádro systému v tomto kroku dělá, závisí na tom, jak je konfigurováno. Více informací uvádí kapitola 7. Jádro systému v této fázi přinejmenším spustí na pozadí některé nezbytné démony.

- Proces `init` pak přejde do víceuživatelského režimu, spustí procesy `getty` pro virtuální konzolu a sériové linky. Proces `getty` je program, který umožňuje uživatelům přihlásit se prostřednictvím virtuální konzoly a sériových terminálů do systému. Proces `init` může rovněž spouštět některé další programy, a to podle toho, jak je konfigurován.

- Poté je zavádění systémů ukončeno a systém normálně běží.

6.3 Podrobněji o zastavení systému

I při zastavování operačního systému Linux je důležité dodržovat správný postup. Jestli se správná procedura zastavení systému nedodrží, budou souborové systémy pravděpodobně „na vyhození“. Obsah jednotlivých souborů se totiž může náhodně promíchat. To proto, že operační systém využívá diskovou vyrovnávací paměť, jež nezapisuje některé změny na disk ihned, ale v určitých časových intervalech. To sice významně zvyšuje výkon systému, ale následkem toho, že se z ničeho nic vypne napájení ve chvíli, kdy paměť „cache“ obsahuje množství nezapsaných změn, mohou být data na disku nekonzistentní a souborový systém zcela nefunkční, protože se na disk zapsala jenom některá změněná data.

Dalším z argumentů proti pouhému „cvrknutí“ do síťového vypínače je to, že v systému se souběžným zpracováním úloh (multitaskingem) může běžet hodně programů na pozadí. Vypnutí ze sítě by pak mohlo mít katastrofální důsledky. Tím, že se při zastavení systému dodržuje korektní postup, se zajistí, že všechny procesy běžící na pozadí svá data včas uloží.

Běh systému Linux se správně ukončí příkazem `shutdown`. Zadává se obvykle jedním ze dvou způsobů.

Když jste přihlášeni v systému jako jediný uživatel, je potřeba před zadáním příkazu `shutdown` ukončit všechny běžící programy, odhlásit se ze všech virtuálních konzolí a přihlásit se na poslední z nich jako superuživatel. Jestli jste přihlášeni jako uživatel `root`, je potřeba se vrátit do kořenového adresáře. Vyhnete se tak problémům při odpojení souborových systémů. Pak můžete zadat příkaz `shutdown -h now`. Mezi zadáním příkazu `shutdown` a zastavením systému bude určité časové prodlení, když nahradíte argument `now` znaménkem plus a počtem minut (přece jenom – běžně nejste jediným uživatelem systému).

Druhou alternativou – je-li v systému přihlášeno více uživatelů – je použití příkazu `shutdown -h +`*`time message`*, kde *`time`* je čas v minutách zbývající do zastavení systému a *`message`* je stručné sdělení důvodu tohoto opatření.

```
# shutdown -h +10 'Bude instalován nový disk. Systém by měl být
> opět spuštěn za tři hodiny.'
#
```

Takto můžete každého uživatele varovat, že systém bude za deset minut zastaven a že bude lepší se odpojit, než ztratit neuložená data. Varování se zobrazí na každém terminálu, na kterém je někdo přihlášený, včetně všech terminálů xterm systému X Window:

```
Broadcast message from root (ttyp0) Wed Aug 2 01:03:25 1995...
Bude instalován nový disk. Systém by měl být
opět spuštěn za tři hodiny.
The system is going DOWN for system halt in 10 minutes !!
```

Varování se automaticky opakuje několik minut před zastavením systému v kratších a kratších intervalech, až stanovený čas vyprší.

Když se pak po určeném časovém prodlení rozjede procedura skutečného zastavení systému, odpojí se nejdříve všechny souborové systémy (kromě systémového svazku), uživatelské procesy (je-li někdo stále přihlášen) se ukončí, běžící démoni se zastaví, všechny připojené svazky se odpojí, obrazně řečeno – všechno se utiší. Poté proces `init` vypíše zprávu, že lze počítač vypnout. Až pak *a jenom pak* lze šáhnout na síťový vypínač.

v některých případech (zřídkakdy u správně nakonfigurovaného systému) není možné zastavit systém správně. Například když jádro systému zpanikaří, havaruje, hoří a vůbec jinak zlobí, může být zcela nemožné zadat jakýkoliv další příkaz. Pochopitelně, v takovéto situaci je i správné zastavení systému poněkud obtížné. Nezbývá než doufat, že se neuložené soubory až tak moc nepoškodí a vypnout napájení. Když nejsou potíže až tak vážné (řekněme, že někdo jenon sekerou rozsekl klávesnici vašeho terminálu) a jádro systému i program `update` stále normálně běží, je obvykle dobré pár minut počkat (dát tím programu `update` šanci vyprázdnit zásobník vyrovnávací paměti) a potom jednoduše vypnout proud.

Někteří uživatelé rádi používají při zastavení systému příkaz `sync`[3] zadaný třikrát po sobě, pak počkají, než se ukončí diskové vstupně-výstupní operace a vypnou napájení. Neběží-li žádné programy, je tento postup téměř ekvivalentní zadání příkazu `shutdown` s tím rozdílem, že se neodpojí žádný ze souborových systémů, což ale může způsobit problémy s nastavením příznaku „clean filesystem“ u souborových systémů ext2fs. Proto se metoda trojího zadání příkazu `sync` *nedoporučuje*.

(Možná vás nenapadá proč zrovna *trojí* zadání příkazu `sync` – v ranných dobách Unixu se všechny příkazy musely „naťukat“ zvlášť, takže bylo obvykle dost času na to, aby se ukončila většina diskových vstupně-výstupních operací.)

6.4 Znovuzavedení systému

Pod znovuzavedením operačního systému (angl. rebooting) se rozumí jeho opakované zavedení, tj. jeho správné zastavení, vypnutí a opětovné zapnutí napájení počítače. Jednodušší cestou je žádost programu `shutdown` o znovuzavedení systému (místo jeho pouhého zastavení), a to použitím parametru `-r`, tedy například zadáním příkazu `shutdown -r now`.

Většina systémů Linux vykonává příkaz `shutdown -r now` i v případě, že se na systémové klávesnici současně zmáčknou klávesy Ctrl+Alt+Del. Tato trojkombinace obvykle vyvolá znovuzavedení operačního systému. Ale to, co se stane po zmáčknutí kláves Ctrl+Alt+Del lze v systému Linux nastavit při jeho konfiguraci. Na víceuživatelském počítači by například bylo lepší před znovuzavedením systému povolit určité časové prodlení. Naopak systémy, které jsou fyzicky přístupné komukoliv, by bylo lepší nastavit tak, aby se při zmáčknutí kombinace kláves Ctrl+Alt+Del nedělo vůbec nic.

[3] Příkaz `sync` vyprázdní zásobník diskové vyrovnávací paměti.

6.5 Jednouživatelský režim

Příkaz `shutdown` lze použít i při přechodu systému do jednouživatelského režimu. V něm se do systému nemůže přihlásit nikdo jiný, než superuživatel, jenž může používat konzolu systému. Jednouživatelský mód lze s výhodou využít při plnění některých úkolů spojených se správou systému, které nelze dělat, systém běží v normálním (víceuživatelském) režimu.

6.6 Záchranné zaváděcí diskety

Občas se stává, že není možné při zapnutí počítače zavést systém z pevného disku. Například tím, že uděláte chybu při nastavování parametrů zavaděče systému LILO, můžete zavinit, že systém Linux nebude možné zavést. v těchto situacích by přišel vhod nějaký jiný způsob zavedení systému, jenž by navíc fungovat vždy (když samozřejmě funguje hardware). Pro počítače PC je takovou alternativou zavádění systému z diskety.

Většina distribucí operačního systému Linux umožňuje vytvořit tzv. **záchrannou zaváděcí disketu** již při instalaci systému. Doporučujeme to udělat. Avšak některé takovéto záchranné diskety obsahují pouze jádro systému. Předpokládá se, že při odstraňování vzniklých problémů budete používat programy uložené na instalačních médiích distribuce systému. Někdy ale tyto programy nestačí. Například v případě, že budete muset obnovit některé soubory ze záloh, jež jste dělali programem, který není na instalačních discích distribuce systému Linux.

Proto by si správce měl vytvořit vlastní zaváděcí diskety, přizpůsobené konkrétním potřebám. Pokyny jak na to jsou obsaženy v příručce „HOWTO – Bootdisk“ od Grahama Chapmana ([Cha]). Musíte mít přirozeně neustále na paměti, abyste měli záchranné zaváděcí a „rootovské“ diskety stále aktuální.

Disketovou mechaniku, která se využívá pro připojení superuživatelské zaváděcí diskety, nebudete moci použít k čemukoliv jinému. Tento problém může být obzvlášť nepříjemný v případě, že máte jenom jednu disketovou mechaniku. Když ale máte dostatek paměti, můžete nastavit zavádění z diskety tak, aby se obsah tohoto superuživatelského zaváděcího disku načítal do virtuálního „ramdisku“ (je proto nutné zvlášť nakonfigurovat jádro systému na zaváděcí disketě). Když se podaří načíst superuživatelskou zaváděcí disketu na ramdisk, lze disketovou mechaniku využít pro připojení jiných disket.

Kapitola

7

Proces init

Uuno on numero yksi

Tato kapitola popisuje proces `init`, jenž je vždy prvním procesem uživatelské úrovně, který spouští jádro systému. Proces `init` má mnoho důležitých povinností. Spouští například program `getty` umožňující uživatelům přihlásit se do systému, implementuje úrovně běhu systému, stará se o osiřelé procesy atd. V této kapitole bude vysvětleno, jak se proces `init` konfiguruje a jak se zavádí různé úrovně běhu systému.

7.1 Proces init přichází první

Proces `init` je jedním z programů, jenž jsou sice absolutně nezbytné k tomu, aby operační systém Linux fungoval, ale kterým obvykle nemusíte věnovat přílišnou pozornost. Součástí každé dobré distribuce systému je i předem nastavená konfigurace procesu `init`, která vyhovuje většině systémů. Správce pak už obvykle nemusí kolem procesu `init` nic dělat. O proces `init` většinou „zavadíte“ jenom pokud připojujete nové sériové terminály, modemy pro příchozí volání, tzv. „dial-in“ (nikoliv tedy modemy, pomocí kterých se budete připojovat do jiných systému, tzv. „dial-out“) a když potřebujete změnit implicitní úroveň běhu systému.

Když se zavede jádro systému (načte se do paměti, spustí se, inicializuje ovladače zařízení, datové struktury atd.), ukončí svou roli v proceduře zavádění operačního systému tím, že spustí první program uživatelské úrovně – proces `init`. Takže program `init` je vždy prvním spuštěným procesem, má tedy vždy číslo procesu 1.

Jádro systému hledá (z historických důvodů) proces `init` na několika místech. Jeho správné umístnění v systému Linux je `/sbin/init`. Nenajde-li jádro program `init`, pokusí se spustit program `/bin/sh` a pokud neuspěje, skončí neúspěšně i celá procedura startu systému.

Když proces `init` startuje, ukončí se proces zavedení systému tím, že se provede několik administrativních úkolů, například kontrola souborových systémů, „úklid“ v adresáři `/tmp`, start obsluhy různých služeb a spuštění procesu `getty` pro každý terminál nebo virtuální konzolu, prostřednictvím nichž se uživatelé mohou přihlašovat do systému atd. (viz kapitola 8).

Po správném zavedení systému proces `init` po každém odhlášení uživatele restartuje procesy `getty` pro příslušný terminál. Umožní tím případné další přihlášení jiných uživatelů. Program `init` si také osvojuje všechny „osiřelé“ procesy. Když některý z procesů spustí další proces (svého potomka) a později ukončí svou činnost dřív, než některý z potomků, sirotci se okamžitě stanou potomky procesu `init`. Adopce sirotků má význam především z různých technických důvodů, je ale dobré o tom vědět, protože je pak snadnější pochopit význam položek seznamu procesů a grafů hierarchického stromu běžících procesů.[1]

Správce systému má na výběr několik verzí programu `init`. Většina distribucí operačního systému Linux používá program `sysvinit` (autorem programu je Miquel van Smoorenburg), jehož předlohou je program `init` pro Unix System V. Unix verze BSD používá odlišný program `init`. Primárním rozdílem mezi uvedenými verzemi programů jsou úrovně běhu systému. Unix System V je implementuje, kdežto verze BSD nikoliv (alespoň pokud jde o jejich tradiční verze). Tento rozdíl není podstatný, a tak se podíváme pouze na program `sysvinit`.

7.2 Konfigurace procesu init pro spouštění programu getty - soubor /etc/inittab

Když proces `init` startuje, načítá konfigurační soubor `/etc/inittab`. Když pak systém Linux běží, proces `init` tento konfigurační soubor opakovaně načítá pokaždé, když přijme signál HUP[2]. Tato vlastnost umožňuje měnit konfiguraci programu `init` a zajistit, aby se takováto změna projevila i bez toho, že by bylo nutné znovu zavést systém.

Soubor `/etc/inittab` je trochu složitější. Začneme tedy s jednoduchým příkladem konfigurace řádku programu `getty`. Jednotlivé řádky souboru `/etc/inittab` sestávají ze čtyř polí oddělených dvojtečkou:

1 Proces `init` samotný nelze ukončit. Nelze jej „zabít“ ani odesláním signálu SIGKILL.

2 Například zadáním příkazu `kill -HUP 1` jako uživatel „`root`“.

```
id:úrovně_běhu:akce:proces
```

Uvedené položky budou popsán níže. Soubor `/etc/inittab` může kromě řádkových záznamů obsahovat i prázdné řádky a ty, jež začínají znakem „#“. Ty program `init` ignoruje.

id — První položka identifikuje každý z řádků konfiguračního souboru. Řádky procesů `getty` blíže specifikují terminál, který daný proces obsluhuje (rozlišuje se písmeny následujícími příslušný název speciálního souboru `/dev/tty`). V řádcích pro ostatní procesy nemá toto pole žádný význam (kromě jeho omezení v délce), avšak mělo by být v celém souboru jedinečné.

úrovně_běhu — Úrovně běhu systému, které připadají pro daný řádek (proces) v úvahu. Úrovně se zadávají jako číslice bez oddělovače. Jednotlivé úrovně běhu systému budou popsány v následujícím odstavci.

akce — Akce, jež se má provést, např. `respawn` (opakovaně spustí příkaz, jenž je uveden v dalším poli pokaždé, když je z různých důvodů ukončen), nebo `once` (spustí daný příkaz jenom jednou).

proces — Příkaz, který se má vykonat.

Chcete-li spustit programu `getty` pro první virtuální terminál (`/dev/tty1`) ve všech běžných víceuživatelských úrovních běhu (2–5), vložte do souboru `/etc/inittab` tento řádek:

```
1:2345:respawn:/sbin/getty 9600 tty1
```

První pole identifikuje řádek pro zařízení `/dev/tty1`. Druhá položka určuje, že proces uvedený v posledním poli lze spouštět na úrovni běhu systému číslo 2, 3, 4 a 5. Třetí pole znamená, že tento příkaz by měl být vždy opakovaně spuštěn poté, co se ukončí (aby se po odhlášení uživatele mohl přihlásit kdokoliv jiný). v posledním poli je příkaz, který spouští proces `getty` pro první virtuální terminál.[3]

Když budete potřebovat přidat do systému další terminály nebo modemové linky pro příchozí volání (dial-in), měli byste rozšířit soubor `/etc/inittab` o další řádky, vždy jeden pro každý terminál, resp. modemovou linku. Více informací najdete v manuálových stránkách *init*(8), *inittab*(5) a *getty*(8).

Nespustí-li se úspěšně příkaz uvedený v souboru `/etc/inittab` a je-li v konfiguraci procesu `init` nastavený jeho restart (akce `respawn`), bude proces zabírat značnou část systémových zdrojů. Proces `init` totiž tento proces spustí, ten se neprovede úspěšně a ukončí se,

[3] Různé verze programu `getty` se chovají různě. Ověřte si vlastnosti programu `getty` v jeho manuálové stránce, ale nezapomeňte se také ujistit, že si čtete v správné manuálové stránce odpovídající verzi programu.

`init` jej opakovaně spustí, proces se ukončí, `init` jej spustí, proces se ukončí a tak dál, až do nekonečna. Této situaci se předchází tím, že si proces `init` vede záznamy o tom, jak často se pokoušel určitý příkaz spustit. Je-li frekvence opakovaných pokusů o spuštění procesu příliš vysoká, `init` před dalším pokusem o provedení příkazu vyčká pět minut.

7.3 Úrovně běhu systému

Úrovní běhu systému se rozumí určitý stav procesu `init` i celého systému. Tento stav určuje, které ze služeb se nabízí. Jednotlivé úrovně se rozlišují čísly, uvedenými v tabulce 7.1. Pokud jde o uživatelsky definované úrovně (2 až 5), neexistují obecně platná pravidla pro jejich používání. Někteří správci systémů pomocí těchto úrovní určují, které subsystémy se spustí, zdali například poběží X, jestli bude systém připojený k síti atd. Jiní dávají přednost inicializaci všech subsystémů na všech uživatelsky definovaných úrovních, popřípadě některé ze subsystémů spouští a zastavují samostatně bez toho, že by se měnily úrovně běhu systému. To proto, že počet úrovní je poměrně malý a řízení konfigurace takovýchto systémů pomocí jednotlivých uživatelsky definovaných úrovní běhu systému je tedy příliš hrubé. Správce systému se v této otázce musí rozhodnout sám, avšak nejjednodušší bude řídit se způsobem, jenž implementuje distribuce, ze které byl systém Linux instalován.

0	Zastavení systému.
1	Jednouživatelský režim (pro zvláštní úkoly, spojené s administrací systému).
2–5	Běžný provoz (uživatelsky definovaný).
6	Znovuzavedení systému.

Tabulka 7.1
Čísla úrovní běhu

Úrovně běhu systému se konfigurují v souboru `/etc/inittab` řádkem podobným tomuto:

```
l2:2:wait:/etc/init.d/rc 2
```

v prvním poli je uvedeno libovolné návěstí, druhé pole znamená, že se tento záznam (řádek) uplatní při úrovni běhu systému číslo 2. Třetí položka (pole `wait`) říká procesu `init`, aby spustil příkaz uvedený ve čtvrtém poli jenom jednou, a to při startu dané úrovně běhu systému a pak vyčkal, než se příkaz provede. Samotný příkaz `/etc/init.d/rc` spustí všechny procesy a příkazy, které jsou potřebné pro spuštění a ukončení služeb, jimiž se implementuje úroveň běhu číslo 2.

Všechnu „dřinu“ spojenou s nastavováním určité úrovně běhu dělá samotný příkaz, uvedený ve čtvrtém poli záznamu. Spouští sadu služeb, které zatím neběží, a pozastaví obsluhu těch, které by na nové úrovni běhu již neměly být poskytovány. Záleží na konkrétní distribuci operačního systému Linux, který z příkazů bude v posledním políčku souboru `/etc/inittab` přesně uveden a které úrovně běhu budou v této konfiguraci implementovány.

Když proces `init` startuje, hledá ten řádek v souboru `/etc/inittab`, jenž specifikuje implicitní, obecnou úroveň běhu systému:

```
id:2:initdefault:
```

Proces `init` lze také požádat o to, aby se spustil v jiné než běžné úrovni běhu, a to tak, že předáte jádru systému argument příkazové řádky `single` nebo `emergency`.[4] Tím si v podstatě vyberete jednouživatelský režim (1. úroveň běhu systému), jenž je popsán v odstavci 7.5.

Když už systém běží, lze změnit aktuální úroveň běhu příkazem `telinit`. V případě, že se úroveň běhu systému mění, proces `init` spouští ten příkaz v souboru `/etc/inittab`, jenž odpovídá nové úrovni běhu systému.

7.4 Zvláštní konfigurace v souboru /etc/inittab

Soubor `/etc/inittab` má některé zvláštní rysy, jež umožňují procesu `init` reagovat i na některé zvláštní situace. Tyto speciální vlastnosti procesu `init` určují zvláštní klíčová slova ve třetím poli záznamu konfiguračního souboru. Několik příkladů:

`powerwait` — Umožní procesu `init` v případě výpadku napájení zastavit systém. Předpokládá se, že systém používá záložní zdroj UPS a software, jenž sleduje UPS a informuje proces `init` o případném výpadku napájení.

`ctrlaltdel` — Umožňuje procesu `init` znovu zavést systém, když uživatel současně zmáčkne klávesy Ctrl+Alt+Del. Uvědomte si, že správce systému může reakci na tuto kombinaci nastavit tak, že se místo „rebootu“ provede nějaká jiná akce, že se například – zvlášť když má k systému přístup širší veřejnost – tato klávesová zkratka ignoruje.[5]

`sysinit` — Příkaz, jenž se provede při zavádění systému. Tímto způsobem se například obvykle mažou soubory v adresáři `/tmp`.

4 Parametry příkazové řádky jádra systému lze zadat například pomocí zavaděče LILO. Viz odstavec 7.5.

5 Nebo se spustí program `nethack`.

Výše uvedený seznam klíčových slov není úplný. Další možnosti i podrobnosti týkající se těch, o kterých se zmiňujeme, uvádí manuálová stránka souboru *inittab*(5).

7.5 Zavádění systému v jednouživatelském režimu

Důležitou úrovní běhu systému je tzv. jednouživatelský režim (úroveň běhu číslo 1), v němž může počítač používat pouze správce systému. V tomto režimu běží jenom minimum systémových služeb (včetně možnosti přihlášení do systému). Jednouživatelský režim je nutný pro některé úlohy spojené s údržbou systému.[6] Například kontrola konzistence svazku `/usr` programem `fsck` vyžaduje, aby byla disková oblast se souborovým systémem odpojená. Toho ale nelze dosáhnout, pokud nejsou ukončeny téměř všechny systémové služby.

Běžící systém lze přepnout do jednouživatelského režimu příkazem `telinit` a požadavkem na přechod do úrovně běhu 1. Při zavádění systému lze do jednouživatelského režimu přejít zadáním parametru `single` nebo `emergency` na příkazové řádce jádra systému. Jádro předá parametry příkazové řádky procesu `init`. Program `init` podle tohoto parametru pozná, že nemá použít implicitní úroveň běhu. (Způsob, kterým se zadává parametr příkazové řádky jádra systému, je podmíněn způsobem, jakým se zavádí operační systém.)

Někdy je potřeba zavést systém v jednouživatelském režimu například proto, aby bylo možné ručně spustit program `fsck` dřív, než se systém pokusí připojit poškozený systém souborů `/usr`, nebo se s ním pokusí jiným způsobem manipulovat. Jakékoliv aktivity na defektním svazku jej s největší pravděpodobností poškodí ještě více, proto by se měla kontrola programem `fsck` udělat co nejdřív.

Zaváděcí skripty, které spouští proces `init`, automaticky přechází do jednouživatelského režimu pokaždé, když automatická kontrola programu `fsck` při zavádění systému neproběhne úspěšně. Pokouší se tak zabránit systému použít souborový systém, jenž je poškozený natolik, že jej nelze automaticky opravit zmiňovaným programem `fsck`. Takovéto poškození svazku je relativně málo frekventované a jeho příčinou bude pravděpodobně mechanické poškození pevného disku, případně nějaká chyba v experimentální verzi jádra systému. Bude ale lepší, když budete jako správce systému připraven i na tuto situaci.

Správně nakonfigurovaný systém se před spuštěním shellu v jednouživatelském režimu zeptá na přístupové heslo superuživatele. Je to důležité bezpečnostní opatření, protože jinak by bylo možné jednoduše zadat vhodný parametr příkazové řádky zaváděči systému LILO a dostat se tak k účtu a oprávněním superuživatele. (Takto nastavený systém se samozřejmě nezavede, když bude důsledkem defektů na systémovém svazku soubor `/etc/passwd` poškozený. Pro tento případ je dobré mít někde po ruce zaváděcí diskety.)

6 Pravděpodobně jej nebude možno použít k hraní hry `nethack`.

Kapitola

8

Přihlašování do systému a ukončování sezení

Tato kapitola by potřebovala citát. Má někdo nějaký návrh?

Tato část knihy popisuje, co se v systému děje poté, co se do něj uživatel přihlásí a zahájí sezení a následně se ze systému odhlásí. Dále budou detailněji popsán různé interakce některých procesů běžících na pozadí, při zahajování a ukončování sezení používané „log“-soubory, konfigurační soubory a další.

8.1 Přihlašování přes terminály

Na obrázku 8.1 je graficky zobrazený algoritmus přihlášení uživatele do systému přes terminál. V prvním kroku si proces `init` ověří, zda běží program `getty` pro dané terminálové spojení (nebo konzolu). Program `getty` sleduje terminál a čeká na uživatele, jenž by mu sdělil, že se chce přihlásit do systému (obvykle tím, že stiskne některou klávesu na klávesnici terminálu). Když proces `getty` zjistí, že uživatel něco napsal na klávesnici, vypíše na obrazovku uvítací zprávu. Ta je uložená v souboru `/etc/issue`. Pak vyzve uživatele, aby zadal své uživatelské jméno a nakonec spustí program `login`. Program `login` dostane zadané uživatelské jméno jako parametr a následně vyzve uživatele, aby zadal přístupové heslo. Je-li heslo zadáno správně, program `login` spustí příkazový interpret vybraný podle nastavení konfigurace pro přihlášeného uživatele. V opačném případě se program `login` jednoduše ukončí, a tím se ukončí i celý proces přihlašování (většinou až poté, co uživatel dostane další možnost zadat správné uživatelské jméno a přístupové heslo). Proces `init` rozpozná, že byla procedura přihlašování ukončena a spustí pro daný terminál novou instanci programu `getty`.

Je důležité si uvědomit, že jediným novým procesem je ten, jenž vytvoří program `init` (použitím systémového volání `fork`). Procesy `getty` a `login` nahrazují právě tento nový proces (použitím volání systému `exec`).

V případě přihlašování po sériových linkách se pro sledování aktivity uživatelů používá zvláštní program proto, že někdy může být (a tradičně bývá) poměrně složité zjistit, kdy je terminál po nečinnosti opět aktivní. Program `getty` rovněž přizpůsobuje přenosové rychlosti a další nastavení pro konkrétní spojení. Takovéto změny parametrů připojení jsou obzvlášť důležité v případě, že systém odpovídá na příchozí modemové žádosti o připojení. V tomto případě se totiž přenosové parametry běžně mění případ od případu.

V současnosti se používá několik různých verzí programů `getty` a `init`. Mají samozřejmě své výhody i nevýhody. Je dobré si přečíst dokumentaci k verzím, které jsou součástí vašeho systému, ale rozhodně neuškodí ani informace o jiných verzích. Další dostupné verze programu lze vyhledat pomocí „Mapy programového vybavení pro Linux“ (The Linux Software Map). V případě, že nemusíte obsluhovat příchozí volání se žádostmi o přihlášení, nebudete se pravděpodobně muset programem `getty` zabývat, avšak podrobnější informace o programu `init` pro vás budou i nadále důležité.

8.2 Přihlášení prostřednictvím sítě

Dva počítače, které jsou zapojené v jedné síti, jsou obvykle propojeny jediným fyzickým kabelem. Když spolu stanice prostřednictvím sítě komunikují, programy, které běží na každé z nich a podílejí se na vzájemné komunikaci, jsou propojeny **virtuálními spojeními**, tedy jakousi sadou imaginárních kabelů. Když spolu aplikace na obou koncích virtuálního spojení komunikují, mají pro sebe vyhrazenou vlastní „linku“. Proto, že tato linka není skutečná, pouze imaginární, mohou operační systémy na obou počítačích vytvořit i několik virtuálních spojení sdílejících tutéž fyzickou linku. Takto spolu může s využitím jediného kabelu komunikovat několik programů bez toho, že by o ostatních spojeních věděly, nebo se o ně nějakým jiným způsobem staraly. Stejné fyzické médium může být sdílelno i několika počítači. Když pak existuje virtuální spojení mezi dvěma stanicemi, další počítače, které se komunikace neúčastní a sdílí tutéž fyzickou linku, toto spojení ignorují.

Toto byl komplikovaný a možná až příliš odtažitý popis reality. Měl by ale stačit k pochopení důležitého rozdílu mezi přihlášením prostřednictvím sítě a normálním přihlášením přes terminál. Virtuální spojení se vytvoří v případě, že existují dva programy na různých stanicích a přejí si spolu komunikovat. Vzhledem k tomu, že je principiálně možné připojit se z kteréhokoliv počítače v síti na kterýkoliv jiný, existuje velké množství potenciálních virtuálních spojení. Díky tomu není praktické spouštět proces `getty` pro každé potenciální síťové přihlášení do systému.

Proto také existuje jediný proces `inetd` (odpovídající procesu `getty`), který obsluhuje *všechna* síťová připojení. V případě, že proces `inetd` zaregistruje žádost o připojení ze sítě (tedy zaregistruje navázání nového virtuálního spojení s některým jiným počítačem zapojeným v síti), spustí nový proces obsluhující toto jediné přihlášení. Původní proces je nadále aktivní a dále čeká na nové požadavky o připojení.

Aby to nebylo až tak jednoduché, existuje pro připojení ze sítě víc komunikačních protokolů. Dva nejvýznamnější jsou `telnet` a `rlogin`. Kromě připojení do systému existuje i mnoho dalších druhů virtuálních spojení, která lze mezi počítači v síti navázat (síťové služby FTP, Gopher, HTTP a další). Bylo by neefektivní mít zvláštní proces, jenž by sledoval žádosti o navázání spojení pro každý typ připojení (službu). Místo toho existuje jediný proces, který umí rozeznat typ spojení a spustit správný program, jenž pak poskytuje odpovídající služby. Tímto procesem je právě proces `inetd`. Podrobnější informace uvádí „Průvodce správce sítě systému Linux“.

8.3 Co dělá program login

Program `login` se stará o autentizaci uživatele (kontroluje, zda bylo zadáno správné uživatelské jméno a přístupové heslo) a počáteční nastavení uživatelského prostředí nastavením oprávnění pro sériovou linku a spuštěním interpretu příkazů.

Částí procedury úvodního nastavení uživatelského prostředí je i vypsání obsahu souboru `/etc/motd` (zkratka pro „message of the day“ – zprávu pro tento den) a kontrola nově příchozí elektronické pošty. Tyto kroky lze zakázat vytvořením souboru nazvaného `.hushlogin` v domovském adresáři uživatele.

Existuje-li soubor `/etc/nologin`, jsou přihlášení do systému zakázána. Tento soubor je typicky vytvářen příkazem `shutdown` nebo příbuznými programy. Program `login` kontroluje, jestli tento soubor existuje, a v případě, že je tomu tak, odmítne akceptovat přihlášení a předtím, než se definitivně ukončí, vypíše obsah tohoto souboru na terminál.

Program `login` rovněž zapisuje všechny neúspěšné pokusy o přihlášení do systémového „log“-souboru (pomocí programu `syslog`). Rovněž zaznamenává úspěšné i neúspěšné pokusy o přihlášení superuživatele. Oba druhy záznamů jsou užitečné při pátrání po případných „vetřelcích“.

Současně přihlášení uživatelé jsou zapsáni v seznamu `/var/run/utmp`. Tento soubor je platný jenom do dalšího znovuzavedení nebo zastavení systému, protože v průběhu zavádění systému se jeho obsah vymaže. Jinak jsou v souboru `/var/run/utmp` kromě seznamu

všech přihlášených uživatelů a používaných terminálů (nebo síťových spojení), uvedené i další užitečné informace. Příkazy `who`, `w` a další podobné se dívají právě do souboru `/var/run/utmp` a zjišťují, kdo je k systému připojený.

Všechna úspěšná přihlášení jsou zaznamenána do souboru `/var/log/wtmp`. Tento soubor se může bez omezení zvětšovat, proto je potřeba jej pravidelně mazat (například po týdnu) zadáním úkolu démonu `cron`.[1] Soubor `wtmp` lze procházet příkazem `last`.

Oba soubory `utmp` i `wtmp` mají binární formát (viz manuálová stránka *utmp*), takže je nelze prohlížet bez speciálních programů.

8.4 X a xdm

META: X implementují přihlášení prostřednictvím procesu `xdm` a rovněž `xterm -ls`.

8.5 Řízení přístupu

Databáze uživatelů je tradičně uložena v souboru `/etc/passwd`. Některé systémy používají tzv. **stínová hesla**. Přesouvají uživatelská přístupová hesla ze souboru `/etc/passwd` do souboru `/etc/shadow`. Sítě s velkým počtem počítačů, ve kterých se informace o uživatelských účtech sdílí pomocí systému NIS nebo nějakou jinou metodou, mohou databázi uživatelů automaticky kopírovat z jediného centrálního počítače na všechny ostatní stanice.

Databáze uživatelů obsahuje nejenom hesla, ale i některé další informace o uživatelích, například jejich skutečná jména, domovské adresáře a interprety příkazů, jenž se implicitně spouští po přihlášení atd. Je potřeba, aby byly tyto informace o uživatelích v systému obecně dostupné a aby si je mohl každý přečíst. Kvůli tomu se heslo ukládá v zakódovaném tvaru. Má to ale jeden háček. Kdokoliv, kdo má přístup k databázi uživatelů, může s pomocí různých kryptografických metod hesla rozšifrovat i bez toho, že by se musel přihlásit k hostitelskému počítači. Systém stínových hesel se snaží zamezit možnosti prolomení přístupových hesel tím, že se přesouvají do jiného souboru, jenž je přístupný pouze superuživateli (hesla se i tak ukládají v zakódovaném tvaru). Avšak s pozdější instalací systému stínových hesel na systému, který jej nepodporuje, mohou vznikat různé potíže.

Ať tak nebo onak, z bezpečnostních důvodů je důležité pravidelně ověřovat, jestli jsou všechna v systému používaná přístupová hesla netriviální, tedy taková, aby nebylo lehké je uhodnout. Lze použít například program `crack`, jenž zkouší hesla v `/etc/passwd` dekódovat. Heslo, které se mu podaří uhodnout, nelze podle výše uvedeného považovat za spolehlivé.

[1] Dobré distribuce operačního systému Linux to zajistí automaticky.

Program `crack` mohou samozřejmě zneužít i případní vetřelci, ale správci systému může jeho pravidelné používání pomoci preventivně omezit výběr nevhodných přístupových hesel. K volbě netriviálního přístupového hesla lze uživatele donutit i programem `passwd`. To je metoda, která je efektivnější především z hlediska zatížení procesoru, protože zpětná analýza zašifrovaných hesel programem `crack` je výpočetně o hodně náročnější.

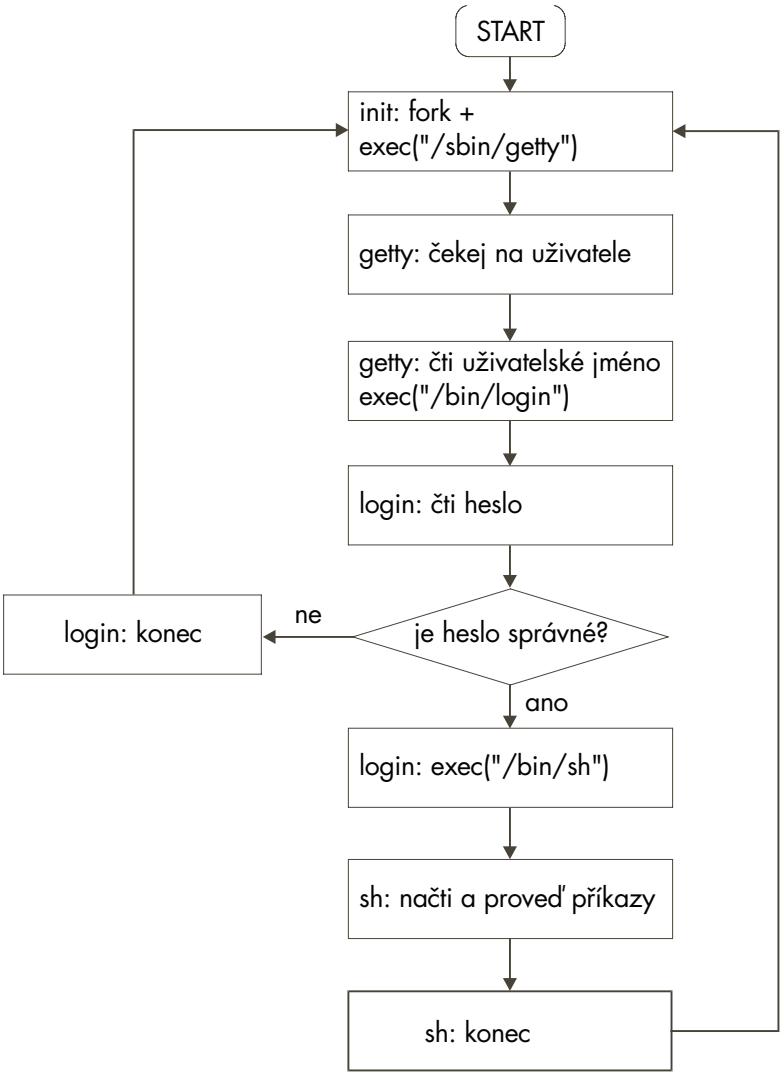
Databáze skupin uživatelů je uložena v souboru `/etc/group`, u systémů se stínovými hesly v souboru `/etc/shadow.group`.

Uživatel `root` se obvykle nemůže přihlásit z kteréhokoliv terminálu nebo počítače v síti, pouze z terminálu uvedeného v seznamu `/etc/securetty`. Pak je nutné mít k některému z těchto terminálů fyzický přístup. Avšak takovéto bezpečnostní opatření nelze považovat za dostatečné, protože je možné přihlásit se z kteréhokoliv jiného terminálu jako běžný uživatel a pro změnu uživatelských oprávnění použít příkaz `su`.

8.6 Spouštění interpretu příkazů

Při startu každý interaktivní příkazový interpret (shell) automaticky spouští jeden či více předem určených souborů. Různé interprety spouští různé konfigurační soubory. Podrobnější informace najdete v dokumentaci k jednotlivým typům interpretů.

Většina shellů nejdříve spustí některý globální soubor, například interpret Bourne shell (`/bin/sh`) a jeho klony spouští soubor `/etc/profile`, až poté spustí soubor `.profile`, jenž je uložen v uživatelově domovském adresáři. Soubor `/etc/profile` umožňuje správci systému nastavit běžné, implicitní uživatelské prostředí, například nastavením proměnné PATH, tak, aby zahrnovalo kromě obvyklých i lokální adresáře s příkazy. Soubor `.profile` zase umožňuje každému z uživatelů upravil si předem nastavené běžné prostředí podle svého vlastního vkusu.



Obrázek 8.1
Připojení přes terminály: vztahy mezi procesy `init`, `getty`, `login` a interpretem příkazů

Kapitola

9

Správa uživatelských účtů

Jaký je rozdíl mezi správcem systému a překupníkem drog?
Žádný, oba měří své prostředky v kilech a oba mají své klienty.
(Starý a otřelý počítačový vtip.)

Tato kapitola popisuje způsob vytváření nových uživatelských účtů, změny vlastností těchto účtů a způsoby jejich odstraňování. Různé distribuce systému Linux používají pro tyto úkoly různé nástroje.

9.1 Co je to účet?

Používá-li počítač více lidí, je obvykle nutné mezi jednotlivými uživateli rozlišovat. Například proto, aby jejich osobní soubory a data byly osobními v pravém smyslu slova. Identifikace uživatelů je ale důležitá i v případě, že systém využívá pouze jedna osoba, což se týká převážné většiny mikropočítačů.[1] Proto má každý uživatel systému přiděleno jednoznačné uživatelské jméno, které zadává při každém přihlášení.

Avšak pojem „účet“ je poněkud širší a zahrnuje – kromě uživatelského jména – i některé další informace o uživateli. Pojmem **uživatelský účet** se označují všechny soubory, zdroje a informace, jež se vztahují k danému uživateli. Běžně se termín „účet“ spojuje s bankovním sektorem. V komerčním výpočetním systému se kolem každého účtu skutečně točí nějaké peníze. Ty mohou z „účtu“ mizet různou rychlostí, podle toho, jak moc uživatel systém zatěžuje. Tak například diskový prostor lze ohodnotit cenou za megabajt uložených dat a den, čas procesoru může mít určitou cenu za sekundu využití a podobně.

1 Dost by mi například vadilo, kdyby si má sestra četla v mé milostné korespondenci.

9.2 Vytváření uživatelských účtů

Samotné jádro systému Linux považuje uživatele systému za pouhé číslo. Každého uživatele lze totiž identifikovat podle jednoznačného celého čísla, tzv. **identifikačního čísla uživatele** (angl. **user ID**, zkráceně **UID**). Je tomu tak proto, že počítač umí zpracovat čísla rychleji a jednodušeji než jména v textové formě. Jména v textové podobě a přidělená **uživatelská jména** se udržují ve zvláštní databázi mimo vlastní jádro. Tato databáze obsahuje též další informace o uživatelích systému.

Když potřebujete vytvořit nový uživatelský účet, musíte přidat informace o novém uživateli do uživatelské databáze a vytvořit pro něj vlastní domovský adresář. Kromě toho by měl každý nový uživatel absolvovat školení. Je též vhodné nastavit pro nové uživatele přiměřené počáteční nastavení prostředí.

Většina distribucí systému Linux se dodává s programy pro vytváření uživatelských účtů. Správce má dokonce k dispozici hned několik takovýchto programů. Dvě varianty, jejichž uživatelským rozhraním je příkazová řádka, jsou programy `adduser` a `useradd`. Existují i utility s grafickým uživatelským rozhraním. Ať už se jako správce systému rozhodnete pro kterýkoliv z programů, oceníte jejich hlavní přínos, tedy to, že omezují manuální práci s nastavováním na minimum, či dokonce úplně. I když na vás při administraci uživatelských účtů čeká mnoho dost spletitých detailů, díky těmto nástrojům vypadá jednoduše. Postup při „ručním“ zakládání nových uživatelských účtů uvádí odstavec 9.2.4.

9.2.1 Soubor /etc/passwd a další informační soubory

Základní databází uživatelů v systému Unix je textový soubor `/etc/passwd` (angl. **password file**), v němž jsou uvedeny platná uživatelská jména a další k nim přidružené informace. Každému uživateli odpovídá v souboru jeden záznam – řádek, který je rozdělen na sedm polí, jejichž oddělovačem je dvojtečka. Význam jednotlivých položek je následující:

1. Uživatelské jméno.
2. Heslo v zakódované podobě.
3. Identifikační číslo uživatele.
4. Identifikační číslo pracovní skupiny (angl. group ID, zkráceně GID).
5. Skutečné jméno uživatele, případně popis účtu.
6. Domovský adresář.
7. Příkazový interpret (nebo program), který se spustí po přihlášení.

Formát jednotlivých políček je podrobněji popsán v manuálové stránce *passwd*(5).

Každý uživatel systému má k souboru `/etc/passwd` přístup (může jej číst). Může tedy například zjistit přihlašovací jména ostatních uživatelů. To ale znamená, že jsou všem přístupná i hesla ostatních uživatelů (druhé pole každého záznamu). Hesla uložená v souboru `/etc/passwd` jsou zakódovaná, takže teoreticky nevzniká žádný problém. Avšak použitý kódovací algoritmus lze prolomit, zvlášť je-li zvolené heslo jednoduché (např. krátké slovo, jenž lze najít v nějakém slovníku, jméno nebo příjmení uživatele atd.). Proto z hlediska bezpečnosti není dobré mít hesla uložená přímo v souboru `/etc/passwd`.

Řada systémů Linux používá systém tzv. **stínových hesel** (angl. **shadow passwords**). Jde o alternativní způsob uložení uživatelských přístupových hesel, jež se zašifrovaná ukládají do jiného souboru (`/etc/shadow`), jenž může číst pouze superuživatel. Soubor `/etc/passwd` pak obsahuje v druhém poli pouze speciální znak. Program, který si potřebuje ověřit totožnost uživatele a má propůjčená přístupová práva vlastníka souboru (pomocí volání jádra setuid), může soubor `/etc/shadow` číst. Běžné programy, které používají pouze některé z dalších položek souboru `/etc/passwd`, přístup k heslům uživatelů systému nemají.[2]

9.2.2 Výběr čísel uživatelského ID a ID skupiny

Ve většině systémů nezáleží na tom, jaké jsou hodnoty UID a GID. Když ale používáte síťový souborový systém NFS, musíte mít stejná UID a GID na všech systémech v síti. To proto, že i systém NFS identifikuje uživatele podle hodnoty UID. Jestliže systém NFS nepoužíváte, můžete vybírat identifikační čísla uživatele a skupiny podle automatického návrhu některého z nástrojů pro správu uživatelských účtů.

Když v síti využíváte NFS, budete si muset zvolit některý z mechanismů synchronizace informací o uživatelských účtech. Jednou z možností je systém NIS – Network Information Service (viz [Kir]).

META: možná na nesprávném místě? Měli byste se také vyvarovat opakovanému přidělování stejných uživatelských čísel (i textových uživatelských jmen), protože nový vlastník UID (případně uživatelského jména) by tak měl přístup k souborům bývalého vlastníka, k jeho elektronické poště a dalším informacím.

[2] Ano, to znamená, že soubor s uživatelskými hesly `/etc/passwd` obsahuje všechny informace o uživatelích, kromě jejich hesel. Překvapení, která přináší vývoj.

9.2.3 Nastavení uživatelského prostředí: adresář /etc/skel

Když jste již pro nového uživatele vytvořili vlastní domovský adresář, můžete nastavit vlastnosti uživatelského prostředí tak, že do domovského adresáře nového uživatele nakopírujete některé soubory z adresáře `/etc/skel`. Správce systému si totiž může v adresáři `/etc/skel` vytvořit konfigurační soubory, jež novým uživatelům vytvoří příjemné základní uživatelské prostředí. Administrátor může například vytvořit soubor `/etc/skel/.profile`, v němž lze nastavením proměnné prostředí `EDITOR` vybrat některý z textových editorů, jenž by měl pro méně zkušené uživatele přátelské ovládání.

Avšak obvykle se doporučuje mít v adresáři `/etc/skel` co nejméně souborů, protože by jinak bylo téměř nemožné upravit na větších víceuživatelských systémech při každé změně konfigurační soubory v již existujících uživatelských adresářích. Když se například změní název onoho uživatelsky příjemného editoru, všichni současní uživatelé si musí upravit svůj vlastní soubor `.profile`. Správce systému by se to mohl pokusit udělat automaticky, například pomocí skriptu, ale takovéto akce téměř pravidelně končí tak, že se nechtěně přepíše či poškodí nějaký jiný soubor.

Vždy když to situace dovolí, je lepší nastavovat globální konfigurace v globálních souborech, jako je `/etc/profile`. Pak lze nastavení pohodlně upravovat bez toho, že by se muselo měnit vlastní nastavení jednotlivých uživatelů systému.

9.2.4 Manuální vytváření uživatelských účtů

Nový uživatelský účet lze vytvořit ručně tímto postupem:

1. Upraví se soubor `/etc/passwd`, například příkazem `vipw`(8), tak, že se do souboru hesel přidá další řádek nového uživatelského účtu. Je nutné dodržovat správnou syntaxi. *Není vhodné upravovat tento soubor přímo běžným editorem!* Program `vipw` soubor `/etc/passwd` uzamkne, takže se ostatní programy nebudou pokoušet jej změnit. Do pole pro heslo se vloží znak „`*`“, takže zatím nebude možné se prostřednictvím tohoto účtu do systému přihlásit.

2. Jestli je potřeba vytvořit i novou pracovní skupinu, upraví se podobným způsobem i soubor `/etc/group`, a sice programem `vigr`.

3. Příkazem `mkdir` se pro nového uživatele vytvoří domovský adresář.

4. Do nově vytvořeného domovského adresáře se nakopírují konfigurační soubory z adresáře `/etc/skel`.

5. Příkazy `chown` a `chmod` se upraví jejich vlastnická a přístupová práva. Užitečný je v tomto případě jejich parametr `-R`. Správná přístupová práva se mohou trochu lišit, a to podle typického využití toho kterého systému, nicméně příkazy v níže uvedeném příkladu obvykle vyhovují většině případů:

```
cd /home/newusername
chown -R username.group .
chmod -R go=u,go-w .
chmod go= .
```

6. Zadejte heslo programem `passwd`(1).

Poté, co bylo v posledním kroku nastaveno heslo, bude nový účet přístupný. Heslo by se skutečně mělo nastavovat až v posledním kroku, jinak by se mohl uživatel nepozorovaně přihlásit do systému například v době, kdy kopírujete konfigurační soubory.

Někdy je potřeba vytvořit „falešný“ účet, který se nebude používat pro přihlašování běžných uživatelů[3]. Například při konfigurování anonymního serveru FTP je výhodné vytvořit účet s uživatelským jménem `ftp`. Pak si ze serveru může stahovat soubory kdokoliv a pro budoucí (anonymní) klienty se nemusí zřizovat vlastní uživatelské účty. V takovýchto případech obvykle není třeba nastavovat heslo (vynechá se poslední krok výše uvedeného postupu). Je vskutku lepší zřizovat takovéto anonymní účty bez hesla. Jinak by k nim měl přístup pouze uživatel, který by předtím musel získal oprávnění superuživatele, protože pouze uživatel `root` má přístup k ostatním uživatelským účtům.

9.3 Změny vlastností uživatelských účtů

Je několik příkazů, kterými lze měnit různé vlastnosti uživatelských účtu (tedy příslušných položek v souboru `/etc/passwd`):

`chfn` Mění pole, ve kterém je uloženo skutečné jméno uživatele.

`chsh` Mění nastavený příkazový interpret.

`passwd` Mění přístupové heslo.

3 Surreální uživatelé?

Superuživatel může pomocí těchto programů změnit vlastnosti kteréhokoliv účtu. Ostatní neprivilegovaní uživatelé mohou měnit pouze vlastnosti svého vlastního účtu. V některých případech je vhodnější běžným uživatelům zakázat možnost používat uvedené příkazy (programem `chmod`), například v případě, že systém používá větší počet méně zkušených začátečníků.

Ostatní změny položek souboru `/etc/passwd` se musí dělat ručně. Když například potřebujete změnit uživatelské jméno, musíte přímo upravit databázi uživatelů `/etc/passwd` (opakovaně připomínáme, že pouze příkazem `vipw`). Analogicky, když potřebujete přidat či odebrat uživatele z nebo do některé z pracovních skupin, musíte upravit soubor `/etc/group` (příkazem `vigr`). Avšak takovéto úkoly se dělají zřídka a musí se dělat opatrně, protože když například změníte některému z uživatelů jeho uživatelské jméno, nebude mu docházet elektronická pošta a musíte pro něj vytvořit „přezdívku“, tedy alias.[4]

9.4 Zrušení uživatelského účtu

Potřebujete-li zrušit uživatelský účet, smažte nejdříve všechny soubory, které patří uživateli rušeného účtu, včetně poštovní schránky, aliasů pro elektronickou poštu, tiskových úloh, úkolů spouštěných démony `cron` a `at` a všechny další odkazy na tohoto uživatele. Pak odstraňte odpovídající řádek v souborech `/etc/passwd` a `/etc/group` (nezapomeňte odstranit uživatelské jméno i ze všech skupin, do nichž byl uživatel zařazen). Předtím, než začnete mazat vše ostatní, je lepší zakázat přístup k rušenému účtu (viz níže). Uživatel tak nebude mít možnost se připojit do systému v době, kdy je jeho účet odstraňován.

Nezapomeňte, že uživatelé systému mohou mít některé soubory uloženy i mimo svůj domovský adresář. Najdete je pomocí příkazu `find`:

```
find / -user username
```

Pamatujte na to, že když má váš systém velké disky, poběží výše uvedený příkaz dost dlouho. V případě, že máte v souborovém systému připojeny síťové disky (viz odstavec 2.3.8), musíte dávat pozor, abyste tím nezpůsobili problémy s odezvou v síti nebo na serveru.

Součástí některých distribucí systému Linux jsou i zvláštní příkazy, které lze použít při rušení uživatelských účtů. Zkuste na vašem systému vyhledat programy `deluser` či `userdel`. Každopádně není zase tak složité to udělat ručně, navíc tyto programy nemusí najít a odstranit všechny souvislosti.

[4] Uživatelské jméno se může změnit například z důvodů sňatku. Uživatelka by mohla chtít používat uživatelské jméno, které odpovídá jejímu novému příjmení.

9.5 Dočasné zablokování uživatelského účtu

Občas je potřeba dočasně zablokovat přístup k některému z účtů bez toho, že by bylo nutné jej smazat. Například když uživatel nezaplatil poplatky za využívání systému, nebo když má správce systému podezření, že neznámý „hacker“ prolomil přístupové heslo uživatele některého účtu.

Nejvhodnějším způsobem jak zamezit přístup k podezřelému účtu, je zaměnit nastavený příkazový interpret zvláštním programem, který na terminál vypíše určitou hlášku. Když se pak kdokoliv pokusí přihlásit do systému přes zablokovaný účet, neuspěje a dozví se proč. Zprávou lze sdělit uživateli, že se má spojit se správcem systému a domluvit se s ním na řešení vzniklého problému.

Alternativou je změna uživatelského jména, případně hesla. Uživatel se tak ale nedozví, co se vlastně děje. A zmatení uživatelé znamenají i více práce pro správce systému.[5]

Nejjednodušší způsob jak vytvořit program, který by blokoval přístup k účtu, je napsat skript pro program `tail`:

```
#!/usr/bin/tail +2
Tento účet byl z důvodů porušení bezpečnostních opatření zablokován.
Volejte prosím číslo 555-1234 a vyčkejte příjezdu mužů v černém.
```

Podle prvních dvou znaků („#!“) pozná jádro systému, že zbytek řádku je příkaz, který je třeba spustit, aby se skript provedl. V tomto případě je to příkaz `tail`, jenž vypíše vše, kromě prvního řádku, na standardní výstup.

Je-li uživatel `billg` podezřelý, že porušuje bezpečnostní opatření, může správce systému udělat něco jako:

```
# chsh -s /usr/local/lib/no-login/security billg
# su - tester
Tento účet byl z důvodů porušení bezpečnostních opatření zablokován.
Volejte prosím číslo 555-1234 a vyčkejte příjezdu mužů v černém.
#
```

5 Máte-li pověst zlomyslného BOFH, tak vám možná budou připadat celkem legrační.*

* Poznámka překladatele: Za zkratkou BOFH (The Bastard Operator From Hell) se skrývá imaginární postava zlotřilého, vychytralého a zlomyslného správce systému (pohybujícího se obvykle v prostředí univerzitních počítačových učeben plných usilovně pracujících studentů – jinak počítačově nevelmi zdatných, avšak o to aktivnějších laiků), který se náramně baví tím, že si ze svých všetečných uživatelů, kteří jej neustále (a velmi často bezdůvodně) otravují se svými problémy nebo pseudoproblémy, na oplátku (ale velmi často i bezdůvodně) dělá legraci, tahá je za nos a „školí“ je kanadskými žertíky nejhrubšího zrna. Sarkastické příběhy si pochopitelně vymýšlí a v elektronických magazínech na Internetu publikují sami operátoři (správci systémů).

Pomocí příkazu `su` ve výše uvedeném příkladu se pochopitelně pouze testuje, zda je změna původně nastaveného interpretu funkční.

Takovéto skripty pro `tail` by měly být uloženy ve zvláštním adresáři, aby jejich jména nekolidovala s příkazy jednotlivých uživatelů.

Kapitola

10

Zálohování

Hardware je indeterministicky spolehlivý.
Software je deterministicky nespolehlivý.
Lidé jsou indeterministicky nespolehliví.
Příroda je deterministicky spolehlivá.

Tato kapitola vysvětluje proč, jak a kdy zálohovat a jak ze záloh obnovit data.

10.1 O důležitosti zálohování

Vaše data mají určitou cenu. Jejich cena je daná hodnotou vašeho času a cenou úsilí data v případě ztráty znovu vytvořit. To vše lze vyjádřit v penězích, nebo přinejmenším vyvážit zármutkem a slzami. Někdy již totiž ztracené informace nelze obnovit, například když je jejich ztráta důsledkem různých experimentů. Vzhledem k tomu, že informace a data jsou v jistém slova smyslu investicí, měli byste tuto investici chránit a učinit kroky, jež by zabránily jejímu znehodnocení.

v zásadě existují čtyři důvody, jež by mohly vést ke ztrátě dat: závady hardwaru, chyby v programech, jednání lidí nebo přírodní katastrofy.[1]

1 Pátým důvodem je „něco dalšího“.

I když je v současnosti hardware relativně spolehlivý, ještě stále se může zdánlivě spontánně porouchat. Pokud jde o ukládání dat, je nejkritičtější součástí výpočetního systému pevný disk. Ve světě plném elektromagnetického šumu se bláhově spoléhá na to, že miniaturní magnetická políčka na povrchu disku, ve kterých jsou cenné informace uloženy, zůstanou v neporušeném stavu.

Vývoj u programů už vůbec nesměřuje ke spolehlivosti. Robustní a spolehlivý software je spíše výjimkou, než pravidlem.

I lidé jsou dost nespolehliví. Buďto udělají nějakou chybu, nebo jsou zlomyslní a zničí data úmyslně.

Přírodu obecně bychom neměli považovat za zlo, ale i když je na nás hodná, může způsobit zkázu.

Takže je malým zázrakem, když všechno funguje jak má.

Řekli jsme, že zálohování je způsobem ochrany investic do dat a informací. Máte-li několik kopií dat, nestane se zase až tak moc, když se některá z verzí zničí (cenou za takovou ztrátu je pouze to, že data musíte obnovit ze zálohy).

Je důležité zálohovat správně. Jako všechno ostatní v reálném světě, dříve nebo později může selhat i zálohování. Součástí správného zálohování je i to, že se kontroluje, jestli vůbec funguje. Jistě byste si nechtěli jednoho dne „všimnout“, že se zálohy nedělaly správně.[2]

Neštěstí v neštěstí – může se stát, že dojde k nějaké vážné havárii právě v okamžiku, kdy zálohujete a máte pouze jediné zálohovací médium. To se může rovněž poškodit a z úmorné práce zbude (někdy doslova) hromádka popela.[3] Může se také stát, že si v momentě, kdy se pokoušíte obnovit data ze záloh, uvědomíte, že jste zapomněli zálohovat něco důležitého, například databázi uživatelů systému, která může mít třeba 15 000 položek. To „nejlepší“ nakonec – všechny zálohovací dávky mohou fungovat bezchybně, ale v poslední známé páskové jednotce, jež ještě umí přečíst typ pásky, který používáte, je vědro vody. Když totiž dojde až na zálohy, je paranoia v popisu práce.

10.2 Výběr média pro zálohování

Co se týče zálohování, je nejdůležitějším rozhodnutím výběr médií. Musíte zvážit náklady, rychlost, dostupnost a použitelnost.

2 Nesmějte se. Několika lidem se to stalo.

3 Radši u toho nebýt...

Cena je poměrně důležitá, protože byste měli mít raději několikrát větší kapacitu zálohovacích médií, než ve skutečnosti potřebujete. Levné médium je obvykle nezbytnost.

Extrémně důležitá je spolehlivost, protože poškozená záloha by rozbrečela i dospělého. Data uložená na zálohovacích médiích musí vydržet bez poškození i několik let. I způsob, kterým médium používáte, má vliv na jeho spolehlivost z hlediska zálohování. Například pevný disk je typicky velmi spolehlivé médium. Nelze ale prohlásit totéž, když hodnotíme spolehlivost z hlediska zálohování a když je tento disk v tom samém počítači jako disk, který zálohujeme.

Jestli lze data zálohovat tak, že nedochází ke kolizím s jinými programy, není rychlost většinou příliš důležitá. Když nemusíte na zálohování dohlížet, pak nevadí, že trvá dejme tomu dvě hodiny. Ale naopak, nelze-li provést zálohování v době, ve které je počítač jinak nevyužitý (například v průběhu noci), pak může být i rychlost problémem.

Dostupnost je zjevně důležitá, protože nemůžete používat zálohovací médium, které není k dostání. Méně zjevný je důležitý požadavek, aby bylo zálohovací médium dostupné i v budoucnu a případně i pro jiné počítače než ty, které používáte dnes. Jinak se může stát, že nebudete schopni zálohy po nečekané události obnovit.

Použitelnost je nejvíce závislá na tom, jak často se zálohování provádí. Čím jednodušší je zálohování, tím lépe. Zálohování na zvolené médium nesmí být složité, pracné nebo otravné.

Typickými alternativami jsou diskety a pásky. Diskety jsou velmi levné, celkem spolehlivé, ne velmi rychlé, velmi dostupné, ale ne příliš použitelné v případě velkého množství dat. Magnetické pásky jsou levné i dražší, celkem spolehlivé, celkem rychlé, dost dostupné a – podle toho, jaká je jejich kapacita – z hlediska použitelnosti i docela pohodlné.

Existují i jiné možnosti. Obvykle nejsou nejlepší, pokud jde o jejich dostupnost, ale vznikne-li problém, oceníte je více, než ostatní možnosti. Řeč je o magneto-optických discích, jež kombinují dobré vlastnosti disket (jejich náhodný, nesekvenční přístup k souborům, možnost rychlého obnovování jednotlivých souborů) a magnetických páskových médií (zálohování velkého objemu dat).

10.3 Výběr nástroje pro zálohování

Existuje bezpočet nástrojů, jichž lze při zálohování využít. Mezi ty tradiční, používané při zálohování v systémech Unix, patří programy `tar`, `cpio` a `dump`. Kromě toho lze sáhnout i po velkém množství balíků programů třetích výrobců (šířených zdarma jako freeware i komerčně). Výběr médií pro zálohování má často vliv i na výběr nástroje.

Programy `tar` a `cpio` jsou podobné a z hlediska zálohování povětšině stejné. Oba programy umí ukládat soubory na pásky a obnovovat je, oba jsou schopné využívat téměř všechna média, protože díky ovladačům zařízení jádra systému, které mají na starost obsluhu nízkoúrovňových zařízení, se takováto zařízení chovají vůči programům uživatelské úrovně stejně. Některé unixové verze programů `tar` a `cpio` mohou mít problémy s neobvyklými soubory (symbolickými linky, speciálními soubory, soubory s velmi dlouhou cestou a podobně), avšak verze pro operační systém Linux by měly pracovat se všemi soubory korektně.

Program `dump` se liší v tom, že přistupuje k souborovému systému přímo a ne jeho prostřednictvím.

Navíc byl napsán speciálně pro zálohování, kdežto programy `tar` a `cpio` jsou primárně určeny pro archivaci souborů, i když je lze s úspěchem použít i jako zálohovací nástroje.

Přímý přístup k souborovému systému má několik výhod. Umožňuje zálohovat soubory bez toho, že by to mělo vliv na jejich časové razítko (angl. time stamp). U programů `tar` a `cpio`, byste museli nejdřív připojit souborový systém pouze pro čtení. Přímé čtení dat ze souborového systému je také efektivnější v případech, kdy je potřeba zálohovat všechna data na disku, protože v tomto případě proběhne zálohování s výrazně menším pohybem čtecí hlavy disku. Největší nevýhodou přímého přístupu je, že zálohovací program, který jej používá, je specifický pro každý typ souborového systému. Program `dump` pro Linux rozumí pouze souborovému systému ext2.

Program `dump` má navíc zabudovanou podporu tzv. úrovní zálohování (angl. backup levels), o kterých bude řeč později. U programů `tar` a `cpio` musí být implementovány pomocí jiných nástrojů.

Srovnávání zálohovacích nástrojů třetích výrobců jde nad rámec této knihy. „Mapa programů pro Linux“ (The Linux Software Map) uvádí výčet těch, které jsou jako freeware šířeny zdarma.

10.4 Jednoduché zálohování

Při proceduře jednoduchého zálohování se v prvním kroku vytvoří zálohy všech dat a v dalších krocích se pak zálohuje jenom to, co se od posledního zálohování změnilo. Prvnímu kroku se říká **úplné zálohování** (angl. **full backup**) a v dalších krocích se tvoří tzv. **inkrementální,** neboli **přírůstkové zálohy** (angl. **incremental backups**). Úplné zálohování je obvykle pracnější, než inkrementální, protože se na médium zapisuje víc dat, a proto se úplná záloha nemusí vždy vejít na jednu pásku (nebo disketu). Naopak, obnovování dat z přírůstkových záloh bývá pochopitelně mnohokrát pracnější, než obnovování z úplných záloh. Nicméně, ob-

novování dat z přírůstkových záloh lze optimalizovat tak, že vždy zálohujete všechny změny od poslední úplné zálohy. Takto je sice o něco pracnější proces zálohování, ale pak by nemělo být nikdy potřeba obnovovat víc, než některou úplnou a inkrementální zálohu.

Uveďme příklad, kdy chcete data zálohovat každý den a máte k dispozici šest páskových kazet, můžete první z nich použít (řekněme v pátek) pro uložení první úplné zálohy a pásky 2 až 5 pro další přírůstkové zálohování (od pondělí do čtvrtka). Pak vytvoříte novou úplnou zálohu na pásku číslo 6 (druhý pátek) a začnete znovu s inkrementálním zálohováním na pásky 2 až 5. Nepřepisujte pásku číslo jedna do doby, než se ukončí nové úplné zálohování – v jeho průběhu by se totiž mohlo stát něco neočekávaného. Poté, co vytvoříte novou úplnou zálohu na pásce číslo 6, měli byste uschovat pásku 1 někam jinam pro případ, že by se ostatní zálohovací pásky zničily, např. při požáru. Takto vám v případě neočekávané události zůstane alespoň něco. Když pak potřebujete vytvořit další úplnou zálohu, dojdete pro pásku číslo 1 a pásku číslo 6 necháte na jejím místě.

Máte-li víc, než šest kazet, můžete ukládat na ty, jež jsou navíc, další úplné zálohy. Pokaždé, když budete dělat úplnou zálohu, použijete tu nejstarší pásku. Takto budete mít úplné zálohy z několika předchozích týdnů, což je dobré, když potřebujete obnovit například některý starší, omylem smazaný soubor, případně starší verzi nějakého souboru.

10.4.1 Zálohování programem tar

Pomocí programu `tar` lze jednoduše vytvořit úplnou zálohu tímto způsobem:

```
# tar -create -file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
#
```

Ve výše uvedeném příkladu byla použita syntaxe programu `tar` verze GNU s jeho dlouhými tvary přepínačů. Tradiční verze programu `tar` rozumí pouze jednopísmenným volbám. Verze GNU ale umí pracovat i se zálohami, které se nevejdou na jednu pásku nebo disketu, a s velmi dlouhými cestami k zálohovaným souborům. Ne všechny klasické verze programu `tar` tyto věci zvládají. (Operační systém Linux používá výhradně program `tar` ve verzi GNU.)

Jestli se záloha nevejde na jednu pásku, je potřeba zadat přepínač `-multi-volume` (`-M`):

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the archive
Prepare volume #2 for /dev/fd0H1440 and hit return:
#
```

Nezapomeňte, že před zálohováním je potřeba diskety zformátovat. Stačí, když to uděláte v jiném okně, případně na jiném virtuálním terminálu ve chvíli, kdy program `tar` čeká na vložení další diskety.

Pokaždé, když ukončíte zálohování, byste měli zkontrolovat, zda proběhlo správně. Zadejte příkaz `tar` s parametrem `-compare` (`-d`):

```
# tar -compare -verbose -f /dev/ftape
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
....
#
```

Je-li kontrola záloh neúspěšná, nebudete moci v případě potřeby původní data z takovýchto záloh obnovit.

Přírůstkové zálohování dělá programem `tar` s parametrem `-newer` (`-N`):

```
# tar -create -newer '8 Sep 1995' -file /dev/ftape /usr/src -verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/modules/
usr/src/linux-1.2.10-includes/include/asm-generic/
usr/src/linux-1.2.10-includes/include/asm-i386/
usr/src/linux-1.2.10-includes/include/asm-mips/
usr/src/linux-1.2.10-includes/include/asm-alpha/
usr/src/linux-1.2.10-includes/include/asm-m68k/
usr/src/linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz
#
```

Bohužel program `tar` neumí zjistit změny informací v i-uzlu souboru, například změny bitu vyhrazeného pro přístupová práva nebo změny jména souboru. Ke změnám v i-uzlech se lze dopracovat pomocí programu `find` a pak srovnávat stav dat uložených v souborovém systému se seznamem souborů, které byly posledně zálohovány. Skripty a programy, které implementují tento způsob zálohování, najdete na linuxových uzlech FTP.

10.4.2 Obnovování souborů programem tar

Zálohované soubory lze obnovit příkazem `tar` s parametrem `-extract` (`-x`):

```
# tar -extract -same-permissions -verbose -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

Můžete rovněž obnovovat jenom vybrané soubory či adresáře (a všechny soubory a podadresáře, které jsou v nich uloženy) tak, že je uvedete v příkazové řádce:

```
# tar xpvf /dev/fd0H1440
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
#
```

Když vás zajímá jenom to, které soubory záloha obsahuje, zadejte příkaz `tar` s parametrem `-list` (`-t`):

```
# tar -list -file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

Uvědomte si, že program `tar` čte zálohu vždy sekvenčně, takže je při práci s většími objemy dat dost pomalý. Ale jestli používáte páskové jednotky nebo jiná média se sekvenčním přístupem, stejně nemůžete využít různé databázové techniky, jež využívají náhodný přístup.

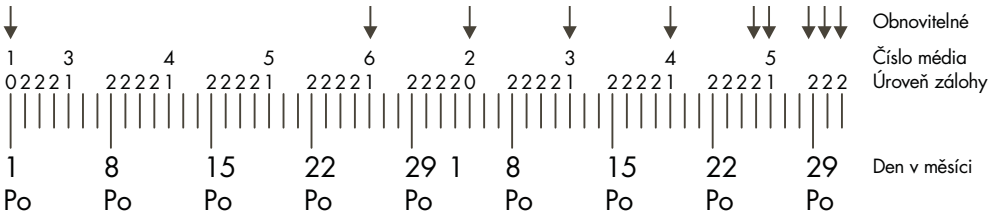
Program `tar` neumí správně zacházet se smazanými soubory. Potřebujete-li obnovit souborový systém z úplné a přírůstkové zálohy a mezi těmito zálohami jste některý ze souborů smazali, po obnovení dat ze záloh bude smazaný soubor znovu existovat. To může být dost závažný problém například v případě, že soubor obsahuje citlivá data, která by již neměla být k dispozici.

10.5 Víceúrovňové zálohování

Metoda jednoduchého zálohování, jež byla načrtnuta v předchozím odstavci, je vhodná pro osobní potřebu nebo systémy s malým počtem uživatelů. Pro náročnější podmínky je vhodnější víceúrovňové zálohování.

Metoda jednoduchého zálohování má dvě úrovně: úplné a přírůstkové zálohování. Ty lze zobecnit do libovolného počtu dalších úrovní. Úplná záloha by mohla být úrovní 0 a další různé přírůstkové zálohy úrovněmi 1, 2, 3 atd. Na každé úrovni přírůstkového zálohování se zálohuje vše, co se změnilo od poslední zálohy stejné nebo nižší úrovně.

Účelem víceúrovňového zálohování je levněji prodloužit historii zálohování dat. V příkladu v předchozím odstavci šla historie zálohování jenom k poslední úplné záloze. Kdybyste měli víc médií, mohli byste ji prodloužit, ale s každou další novou páskou jenom o týden, a to by bylo příliš nákladné. Delší historie vytváření záloh je užitečná, protože omylem smazaných nebo poškozených souborů si často všimnete až po delší době. Navíc jakákoliv verze souboru, i když není zrovna aktuální, je obvykle lepší, než žádná.



Obrázek 10.1
Příklad časového rozvrhu víceúrovňového zálohování

Víceúrovňovým zálohováním lze historii vytváření archivů prodloužit levněji. Když si například koupíte deset kazet, můžete používat pásky 1 a 2 na měsíční zálohy (první pátek každého měsíce), pásky 3 až 6 na týdenní zálohy (všechny ostatní pátky – uvědomte si, že v některém měsíci může být pět pátků, takže potřebujete o čtyři pásky víc) a pásky 7 až 10 na denní zálohy (od pondělí do čtvrtka). Takto jste schopni – pouhým zakoupením čtyř dalších pásek

navíc – prodloužit historii zálohování ze dvou týdnů (s využitím všech denních záloh) na dva měsíce. Pravda, během těchto dvou měsíců nemůžete obnovit všechny verze zálohovaných souborů, avšak to, co obnovit lze, obvykle postačí.

Z obrázku 10.1 je patrné, která úroveň zálohování se používá v ten který den a z kterých záloh je možno na konci měsíce data obnovit.

Víceúrovňové zálohování navíc snižuje i čas potřebný k obnovení celého souborového systému. Když potřebujete obnovit celý systém souborů a máte mnoho inkrementálních záloh s monotónně rostoucí řadou úrovní, musíte data pracně obnovovat postupně ze všech médií. Když ale budete používat méně záloh a větší počet úrovní, snížíte počet úrovní potřebných k obnovení celého svazku.

Chcete-li snížit počet médií potřebných k obnovení dat, používejte pro přírůstkové zálohy úrovně menšího rozsahu. Ale i tak se čas zálohování zvýší, protože při každém zálohování se ukládá vše, co se změnilo od předchozí úplné zálohy. O něco lepší plán zálohování se uvádí v manuálové stránce programu `dump`. Jeho schéma je patrné z tabulky 10.2. Zkuste použít posloupnost úrovní zálohování: 3, 2, 5, 4, 7, 6, 9, 8, 9 atd. Snížíte tím čas zálohování i obnovování dat. Zálohovat byste měli vždy maximálně po dvou dnech práce. Počet pásek, z nichž se budou obnovovat data, závisí na intervalu mezi úplnými zálohami, ale je rozhodně nižší, než u procedury jednoduchého zálohování.

Páska	Úroveň (dny)	Záloha (pásky)	Obnovování
1	0	-	1
2	3	1	1, 2
3	2	2	1, 3
4	5	1	1, 2, 4
5	4	2	1, 2, 5
6	7	1	1, 2, 5, 7
7	6	2	1, 2, 5, 7, 8
8	9	1	1, 2, 5, 7, 8
9	8	2	1, 2, 5, 7, 9
10	9	1	1, 2, 5, 7, 9, 10
11	9	1	1, 2, 5, 7, 9, 10, 11
...	9	1	1, 2, 5, 7, 9, 10, 11, ...

Obrázek 10.2
Efektivnější postup zálohování s větším počtem úrovní

Tyto sofistikované postupy zálohování obvykle redukují pracnost, ale na druhou stranu je víc věcí, které jako správce systému musíte uhlídat. Sami se musíte rozhodnout, jestli se vám to vyplatí.

Program `dump` má zabudovanou podporu víceúrovňového zálohování. U příkazů `tar` a `cpio` je potřeba víceúrovňové zálohování implementovat pomocí skriptů.

10.6 Co zálohovat

Je pochopitelné, že budete chtít zálohovat vše, co se vám na zálohovací média vejde. Významnou výjimkou je software, jenž lze bezbolestně obnovit z instalačních médií.[4] Ale aplikace mohou používat mnoho různých konfiguračních souborů a ty je lepší zálohovat, protože si tím ušetříte nelehkou práci, kterou zabere jejich opakované nastavování. Další významnou výjimkou je souborový systém `/proc`. Obsahuje pouze data, která jádro vytváří automaticky, a proto *nemá žádný smysl* je zálohovat. Zvláště nežádoucí je soubor `/proc/kcore`, protože je to pouze aktuální obraz fyzické paměti, takže je dost velký.

Jakousi „šedou zónou“ jsou zprávy „news“, poštovní schránky, log-soubory a mnoho dalších dat uložených v adresáři `/var`. Sami se musíte rozhodnout, co pokládáte za důležité.

Typickým příkladem toho, co se musí zálohovat, jsou uživatelské soubory (adresář `/home`) a systémové konfigurační soubory (adresář `/etc` a další konfigurační tabulky, které jsou často rozeseté po celém souborovém systému).

10.7 Komprimované zálohy

Zálohy zabírají hodně místa na disku, což může stát dost peněz. Nároky na diskový prostor lze minimalizovat komprimováním záloh. Existuje několik způsobů, jak to udělat. Některé programy přímo podporují kompresi, například po zadání parametru `-gzip` (`-z`) programu `tar` verze GNU se jeho výstup spojí pomocí roury s kompresním programem `gzip`. Záloha se pak komprimuje před tím, než se zapíše na zálohovací médium.

Bohužel komprimované zálohy mohou způsobit vážné komplikace. Z povahy toho, jak komprimace funguje, vyplývá, že když se zapíše jediný bit špatně, bude nepoužitelný i celý zbytek komprimovaných dat. Některé zálohovací programy sice používají zabudované opravné algoritmy, ale žádná z těchto metod si neumí poradit s větším počtem chyb. Je-li tedy záloha

[4] Musíte se rozhodnout, co je jednodušší. Jsou lidé, kteří tvrdí, že je jednodušší instalovat programy z tuctů instalačních disket.

komprimovaná způsobem, jakým to dělá program `tar` verze GNU, který svůj výstup komprimuje jako celek, může jediná chyba způsobit poškození celé zálohy. Stěžejním rysem zálohování musí být spolehlivost, takže tato metoda komprese není příliš dobrá.

Alternativou je komprimace jednotlivých záloh (souborů). Když se pak stane, že bude některý z nich poškozen, můžete použít jinou zálohou. Pravděpodobnost, že data, která tím ztratíte, se mohou poškodit i jiným způsobem, je stejná, takže na tom nebudete o moc hůř, než kdyby se nezálohovalo vůbec. Metodu komprimace jednotlivých souborů využívá např. program `afio` (varianta programu `cpio`).

Komprese o něco prodlouží proces zálohování. Může to způsobit, že zálohovací program nebude schopen zapisovat na pásku dostatečně rychle.[5] Tento problém lze řešit využitím vyrovnávacích pamětí pro výstup zálohovacích programů (buď interních, je-li zálohovací program natolik „chytrý“, nebo pomocí jiných programů). Ale i tak by se mohlo stát, že zálohování nebude fungovat správně. S tímto problémem byste se mohli setkat u velmi pomalých počítačů.

5 Nepřenáší-li se na páskovou mechaniku data dostatečně rychle, musí se zastavit. To zálohování ještě víc zpomaluje a není to dobré ani pro pásku a ani pro páskovou mechaniku.

Kapitola

11

Udržování správného času

Čas je iluze. Čas oběda dvojnásob.
(Douglas Adams)

v této kapitole bude vysvětleno, jakým způsobem systém Linux udržuje správný čas a co je potřeba dělat, abyste potížím s nesprávným systémovým časem předešli. Obvykle nebudete muset dělat se systémovým časem nic, ale je užitečné porozumět jeho principům.

11.1 Časové zóny

Měření času je založeno na převážně pravidelných přírodních jevech jako je střídání světla a tmy, jehož příčinou je rotace planety. Celkový čas mezi dvěma po sobě následujícími periodami je stejný, ale délka denní a noční doby se mění. Jedinou konstantou je poledne.

Poledne je denní doba, ve které se Slunce nachází na své nejvyšší pozici. Vzhledem k tomu, že se Země otáčí,[1] je poledne na různých místech zeměkoule v jinou dobu. Z toho vychází představa **lokálního**, **místního času**. Lidstvo měří čas v různých jednotkách, jejichž většina je rovněž svázána s přírodními jevy, jako je ona kulminace Slunce v poledne. Pokud se nacházíte na stejném místě, nezáleží na tom, že je lokální čas na jiných místech jiný.

Když ale potřebujete komunikovat se vzdálenějšími místy, uvědomíte si zároveň, že potřebuzjete jakýsi „společný“ čas. V dnešní moderní době potřebuje hodně míst komunikovat s různými jinými místy na celé planetě. Proto byl zaveden společný standard měření času. Tomuto „společnému“ času se říká **univerzální čas** (angl. **universal time**, zkráceně UT nebo

[1] Dle výsledků posledních výzkumů.

UTC), dříve označovaný jako greenwichský čas (angl. Greenwich Mean Time nebo GMT), protože je místním časem ve městě Greenwich v Anglii. Když spolu potřebují komunikovat lidé, kteří mají různý lokální čas, mohou používat společný univerzální čas. Nevzniká tak zmatek kolem toho, kdy se která věc stala nebo měla stát.

Místním časům se říká časové zóny. I když se z pohledu geografie zdá logické, aby byla místa, která mají poledne ve stejnou dobu, začleněna do stejného časového pásma, neumožňují to hlediska politická. Mnoho zemí z různých důvodů zavádí **letní čas** (angl. **daylight savings time**, zkráceně DST). Posouvají ručičky hodinek tak, aby měly více denního světla v době, kdy se pracuje, pak je v zimě zase přetáčí zpět. Jiné státy to tak pro změnu nedělají. No a ty, které tak činí, nejsou zajedno v tom, kdy má být čas posunutý a mění pravidla z roka na rok. To je důvod proč nemohou být posuny časových pásem triviální.

Časová pásma je nejlepší určovat podle polohy nebo podle rozdílu mezi místním a univerzálním časem. Ve Spojených státech a některých dalších zemích mají místní časové zóny své jméno a odpovídající třípísmennou zkratku. Ale tato zkratka není jedinečná a neměla by se používat bez současného označení země. Je lepší mluvit o lokálním čase například v Helsinkách, než o východoevropském čase (zkratka EET, East European Time), protože ne všechny státy východní Evropy leží ve stejném pásmu a nemusí mít stejná pravidla přechodu na letní čas.

Linux má balík programů pro práci s časovými pásmy. Tento software zná všechny existující časové zóny a navíc jej lze jednoduše přizpůsobit v případě, že se změní pravidla určování času. Takže jediné, co musí správce systému udělat, je vybrat správné časové pásmo. Také každý z uživatelů si může nastavit své vlastní časové pásmo – to je důležité proto, že mnoho z nich používá prostřednictvím sítě Internet počítače v různých zemích. Když se ve vašem místním časovém pásmu změní pravidla přechodu na letní čas, budete muset upgradovat jenom tomu odpovídající část subsystému pro určování času. Stačí pak nastavit časové pásmo systému a upravit datové soubory zvoleného pásma, není potřeba se trápit s nastavováním systémového času.

11.2 Hardwarové a softwarové hodiny

Osobní počítač má hardwarové systémové hodiny napájené z baterií. Díky těmto bateriím hodiny fungují, i když je zbytek počítače bez proudu. Hardwarové systémové hodiny lze seřizovat jako jednu z položek na obrazovce pro nastavení systému BIOS, nebo z běžícího operačního systému.

Jádro systému Linux udržuje vlastní čas nezávisle na hardwarových hodinách. Při zavádění operačního systému Linux nastaví své vlastní softwarové hodiny na stejný čas, jaký je v tom momentě na systémových hardwarových hodinách. Pak běží oboje hodiny nezávisle. Systém Linux udržuje vlastní čas pomocí softwarových hodin proto, že využívání systémových hardwarových hodin je dost pomalé a složité.

Hodiny jádra systému ukazují vždy univerzální čas. Proto jádro nemusí vůbec nic vědět o časových zónách – jednoduchost přispívá k vyšší spolehlivosti a ulehčuje možnost změny informací o vybraném časovém pásmu. Každý proces si převádí časová pásma sám (pomocí standardních nástrojů, jež jsou součástí balíku pro konverze časových zón).

Hardwarové systémové hodiny mohou být nastaveny na místní či univerzální čas. Je obvykle lepší mít nastavený univerzální čas, protože pak není potřeba měnit nastavení hardwarových systémových hodin na začátku a konci období letního času (UTC nemá DST). Bohužel některé operační systémy pro PC – včetně systémů MS-DOS, Windows, OS/2 – počítají s tím, že hardwarové hodiny ukazují lokální čas. Systém Linux si umí poradit s oběma způsoby nastavení, ale v případě, že hardwarové hodiny ukazují místní čas, bude potřeba měnit jeho nastavení při přechodu z a na letní čas (jinak by softwarové hodiny neukazovaly místní čas).

11.3. Zobrazení a nastavování času

v systému Debian je systémové časové pásmo určeno symbolickým linkem `/etc/localtime`. Tento link odkazuje na datový soubor pro dané časové pásmo, jež popisující místní časovou zónu. Jednotlivé datové soubory pro časová pásma jsou uloženy v adresáři `/usr/lib/zoneinfo`. Jiné distribuce Linuxu mohou parametry pro časová pásma nastavovat odlišně.

Uživatel může změnit své vlastní časové pásmo nastavením proměnné prostředí TZ. Není-li hodnota TZ nastavena, předpokládá se, že uživatel používá nastavení časového pásma systému. Syntaxe nastavení hodnoty proměnné TZ je popsána v manuálové stránce příkazu *tzset*(3).

Příkaz `date` ukáže aktuální datum a čas.[2] Například:

```
$ date
Sun Jul 14 21:53:41 EET DST 1996
$
```

2 Pozor na příkaz `time`, jenž neukazuje aktuální čas.

To znamená, že je neděle, 14. července 1996, asi za deset minut deset večer, to všechno v časovém pásmu označeném „EET DST“, což by mohlo v angličtině znamenat „East European Daylight Savings Time“, tedy východoevropský letní čas. Příkazem `date` můžeme rovněž zjistit univerzální čas:

```
$ date -u
Sun Jul 14 18:53:42 UTC 1996
$
```

Příkazem `date` se také nastavují softwarové hodiny jádra systému:

```
# date 07142157
Sun Jul 14 21:57:00 EET DST 1996
# date
Sun Jul 14 21:57:02 EET DST 1996
#
```

Více podrobností hledejte v manuálové stránce příkazu `date` – syntaxe příkazu je tak trochu „tajemná“. Čas může nastavovat pouze superuživatel. I když může mít každý z uživatelů nastaveno své vlastní časové pásmo, systémový čas je pro všechny stejný.

Příkaz `date` ukazuje nebo nastavuje jenom softwarové hodiny. Příkaz `clock` synchronizuje systémové hardwarové a softwarové hodiny. Spouští se při zavádění systému, kdy se zjišťuje nastavení hardwarových systémových hodin. Podle nich se pak nastavují hodiny softwarové. Potřebujete-li nastavit oboje, nastavte nejprve softwarové hodiny příkazem `date`, následně hardwarové příkazem `clock -w`.

Parametrem `-u` sdělíte programu `clock`, že hardwarové hodiny ukazují univerzální čas. Přepínač `-u` je potřeba používat správně. V opačném případě bude mít váš systém mírný zmatek v tom, jaký je vlastně přesný čas.

Nastavení hodin se musí dělat opatrně. Mnoho částí operačního systému Unix spoléhá na to, že hodiny fungují správně. Například démon `cron` spouští pravidelně různé příkazy. Změníte-li nastavení hodin, může se stát, že nebude vědět, zda je potřeba některý z programů spustit, či nikoliv. Když na některém starším unixovém systému někdo nastavil hodiny o dvacet let dopředu, démon `cron` se snažil spustit všechny periodicky vykonávané příkazy za celých dvacet let naráz. Aktuální verze programu `cron` si s tímto problémem umí poradit. Přesto byste měli být při změnách času opatrní. Velké časové „skoky“ dopředu nebo posuny vzad jsou nebezpečnější, než menší změny a posuny dopředu.

11.4 Co když jdou hodiny špatně

Softwarové hodiny systému Linux nejdou vždy přesně. V chodu je udržují pravidelná **přerušení časovače** generované hardwarem PC. Běží-li v systému příliš mnoho procesů, může trvat obsluha přerušení časovače příliš dlouho a softwarové hodiny se začnou zpožďovat. Hardwarové systémové hodiny běží nezávisle na operačním systému a jsou obvykle přesnější. Jestli často vypínáte a zapínáte počítač (to je případ většiny systémů, které neslouží jako servery), budete mít obvykle i přesnější systémový čas.

Potřebujete-li upravit nastavení hardwarových hodin, je obyčejně nejjednodušší restartovat systém, spustit obrazovku pro nastavení systému BIOS a udělat to tam. Tak lze předejít všem potížím, které by mohly být zapříčiněny změnou systémového času za běhu systému. Nemáte-li možnost upravit čas v nastaveních systému BIOS, nastavte jej příkazy `date` a `clock` (v tomto pořadí), ale připravte se na to, že se možná některé části systému budou chovat podivně, takže budete muset systém opět restartovat.

Počítač připojený k síti (i když jenom pomocí modemu) může automaticky kontrolovat správnost nastavení vlastních hodin tak, že je bude srovnávat s nastavením hodin nějakého jiného počítače. Jestli se o tomto jiném počítači ví, že jeho hodiny jdou velmi přesně, budou pak mít po připojení přesný čas nastavené oba systémy. Systémové časy dvou systémů lze seřídit příkazy `rdate` a `netdate`. Oba uvedené programy kontrolují čas na vzdáleném počítači (příkaz `netdate` i na několika vzdálených stanicích) a podle toho nastaví místní čas lokálního počítače. Počítač, na kterém se bude pravidelně spouštět některý z těchto příkazů, bude mít tak přesný čas, jak přesné budou hodiny vzdáleného počítače.

A

Zjišťování „prázdných míst“ v souborech

v příloze A je uvedena zajímavá část programu, jenž se používá k zjišťování potenciálu „děr“ v souborovém systému. Originální distribuce této knihy obsahuje celý zdrojový kód (`sag/measure-holes/measure-holes.c`).

```
int process(FILE *f, char *filename) {
static char *buf = NULL;
static long prev_block_size = -1;
long zeroes;
char *p;

if (buf == NULL || prev_block_size != block_size) {
free(buf);
buf = xmalloc(block_size + 1);
buf[block_size] = 1;
prev_block_size = block_size;
}
zeroes = 0;
while (fread(buf, block_size, 1, f) == 1) {
for (p = buf; *p == '\0'; )
++p;
if (p == buf+block_size)
zeroes += block_size;
}
if (zeroes > 0)
```

```
printf("%ld %s\n", zeroes, filename);
if (ferror(f)) {
errormsg(0, -1, "read failed for ‘%s’", filename);
return -1;
}
return 0;
}
```

B

Slovníček

Knihovník Unseenské univerzity se rozhodl sám napomoci obecnému porozumění tak, že napíše orangutansko-lidský slovník.
Pracoval na něm celé tři měsíce.
Nebylo to lehké. Musel se dostat až k slůvku „oook“.
(Terry Pratchett, „Muži ve zbrani“)

Zde je stručný seznam slovních definicí pojmů, spojovaných s operačním systémem Linux, zvláště pak se správou systému.

ambice Sepsání několika zábavných vět v naději, že se dostanou mezi linuxové soubory „cookie“.

aplikační program Software, který dělá něco užitečného. Výsledek používání aplikačního programu je v podstatě důvodem zakoupení počítače. Viz také systémový program, operační systém.

démon Proces číhající v pozadí – obvykle nenápadně – až do chvíle, než jej něco „rozjede“. Například démon `update` se probudí každých třicet sekund (nebo tak nějak) a vyprázdní buffer vyrovnávací paměti, démon `sendmail` se probere pokaždé, když někdo odesílá poštu.

jádro systému Část operačního systému, která implementuje interakce s technickými prostředky výpočetního systému a sdílení zdrojů. Viz také systémový program.

operační systém Software, jenž umožňuje sdílení zdrojů počítačového systému (procesor, paměť, diskový prostor, síť a tak dál) mezi uživateli a aplikačními programy, které uživatelé spouští. Nabízí zabezpečení systému řízením přístupu k němu. Viz také jádro systému, systémový program, aplikační program.

slovníček Seznam slov a vysvětlení jejich významu. Nezaměňovat se slovníkem, jenž je rovněž seznamem slov a jejich vysvětlení.

souborový systém Metoda a datové struktury, které používá operační systém pro ukládání záznamů souborů na disk či diskovou oblast; způsob, kterým jsou soubory na disku organizován. Pojem se používá též pro označení diskové oblasti či disku, kam se soubory ukládají, případně pro určení typu souborového systému.

systémový program Programy, které implementují vyšší úrovně funkcionality operačního systému, tedy ty vlastnosti systému, které nejsou přímo závislé na hardwaru. Někdy mohou vyžadovat ke spuštění zvláštní oprávnění (např. doručování elektronické pošty), ale velmi často jsou považovány za běžnou součást systému (např. kompilátor). Viz také aplikační program, jádro systému, operační systém.

volání systému Služby, které poskytuje aplikačním programům jádro systému, a způsob, jímž jsou volány. Viz sekci 2 manuálových stránek.

LINUX

Příručka správce sítě

Kapitola

1

Úvod do vytváření sítí

1.1 Historie

Myšlenka vytváření sítí je pravděpodobně stejně stará jako vlastní telekomunikace. Uvažte lidi žijící v době kamenné, kteří si možná mezi sebou předávali zprávy pomocí bubnů. Dejme tomu, že jeskynní člověk A chtěl pozvat jeskynního člověka B na zábavnou hru, která spočívala v tom, že po sobě házeli kamením, ale žil příliš daleko na to, aby B slyšel jeho obrovský buben. Jaké měl tedy A volby? Mohl 1) dojít za B, 2) sehnat si větší buben nebo 3) požádat C, který žil v polovině vzdálenosti jejich obydlí, aby zprávu předal. V případě poslední možnosti se jedná o vytvoření sítě.

Samozřejmě, že od primitivních snah a prostředků našich předků jsme urazili již dlouhou cestu. Když si dnes chceme domluvit schůzku na sobotní fotbalový zápas, máme počítače, které spolu komunikují prostřednictvím soustavy drátů, optických vláken, mikrovln apod.[1] V následujícím textu se budeme zabývat způsoby a metodami, kterými je to realizováno, opustíme však záležitosti kolem drátů i kolem fotbalu.

V tomto průvodci si popíšeme dva typy sítí: sítě založené na protokolu UUCP a sítě založené na protokolu TCP/IP. Jde o sady protokolů a softwarové balíky, které obstarávají přenos dat mezi počítači. V této kapitole se podíváme na oba typy sítí a probereme si jejich základní principy.

Síť definujeme jako soubor *hostitelů*, kteří spolu mohou vzájemně komunikovat a často se přitom spoléhají na služby vyhrazených hostitelů, přenášejících data mezi jednotlivými účastníky. Hostitelé jsou často počítače, ale neplatí to vždy; někdy tak nazýváme i X-terminály nebo inteligentní tiskárny. Menším seskupením hostitelů říkáme *místa (site)*.

---

1 Původní duch, který se v Evropě stále ještě zjevuje.

Komunikace by nebyla možná bez určitého druhu jazyka nebo kódu. V počítačových sítích se těmto jazykům souhrnně říká *protokoly*. Neměli byste si však představovat písemné protokoly, ale spíše vysoce formalizovaný kód chování, které lze pozorovat například při setkání hlav států. Podobně protokoly používané v počítačových sítích nejsou ničím jiným, než pravidly pro výměnu zpráv mezi dvěma hostiteli.

1.2 Sítě UUCP

UUCP je zkratka spojení Unix-to-Unix Copy. Původně to byl balík programů pro přenos souborů po sériových linkách, plánování těchto přenosů a spouštění programů na vzdálených hostitelích. Od své první implementace koncem sedmdesátých let prodělal tento protokol velké změny, ale co se týče služeb, které nabízí, je stále poněkud nekomfortní. Hlavní využití má stále v sítích WAN založených na vytáčených (dial-up) telefonních linkách.

Protokol UUCP byl vytvořen v Bellových laboratořích v roce 1977, kde měl sloužit ke komunikaci mezi místy podílejícími se na vývoji Unixu. V polovině roku 1978 spojovala tato síť již přes 80 míst. Používala se zde elektronická pošta i vzdálený tisk. Ovšem hlavní využití systému spočívalo v šíření nového softwaru a opravách chyb.[2] Dnes již není protokol UUCP omezen jen na prostředí Unixu. Existují jak volně šířitelné, tak i komerční porty na různé platformy, včetně AmigaOS, DOS, Atari TOS atd.

Jednou z hlavních nevýhod sítí UUCP je malá šířka pásma. Na jedné straně telefonní vybavení úzce omezuje maximální přenosovou rychlost. Na druhé straně jsou spojení prostřednictvím protokolu UUCP jen zřídka trvalá; hostitelé se místo toho spojují v pravidelných intervalech. Proto většinu času, který zabere doručení poštovní zprávy v síti UUCP, zpráva jen nečinně „dřepí“ na disku nějakého hostitele a čeká na nadcházející navázání spojení.

Navzdory těmto omezením stále funguje po celém světě spousta sítí UUCP, o něž pečují zejména nadšenci, kteří za rozumné ceny nabízí soukromým uživatelům přístup k síti. Hlavním důvodem oblíbenosti protokolu UUCP je skutečnost, že ve srovnání s připojením počítače pevnou linkou je téměř „za babku“. K tomu, abyste z vašeho počítače udělali uzel UUCP, potřebujete jen modem, fungující implementaci protokolu UUCP a jiný uzel UUCP, který bude ochotný vás zásobovat poštou a newsy.

2 Ne že by se od té doby časy příliš změnily…

1.2.1 Jak používat protokol UUCP

Myšlenka stojící v pozadí protokolu UUCP je velice jednoduchá: jak již jeho název napovídá, kopíruje vlastně soubory z jednoho hostitele na druhý. Kromě toho ale umožňuje provádět na vzdáleném hostiteli také určité akce.

Dejme tomu, že váš počítač má povolen přístup k hypotetickému hostiteli jménem **swim** a může na něm spouštět tiskový příkaz `lpr`. Potom byste mohli na hostiteli **swim** vytisknout tuto knihu za pomoci následujícího příkazu zadaného v příkazové řádce:[3]

```
$ uux -r swim!lpr !netguide.dvi
```

Příkaz `uux` ze sady protokolu UUCP naplánuje úlohu pro hostitele **swim**. Tuto úlohu formuluje vstupní soubor `netguide.dvi` a požadavek o předání tohoto souboru příkazu `lpr`. Příznak `-r` říká příkazu `uux`, aby nevolal vzdálený systém ihned, ale aby úlohu pozdržel, dokud nebude při příští příležitosti navázáno spojení. Tomuto postupu říkáme *spooling*.

Další vlastností protokolu UUCP je, že umožňuje doručovat úlohy a soubory přes několik hostitelů, za předpokladu, že tyto spolupracují. Dejme tomu, že hostitel **swim** z předchozích příkladů má UUCP-spojení s hostitelem **groucho**, který spravuje rozsáhlý archiv unixových aplikací. Ke stažení souboru *tripwire-1.0.tar.gz* můžete použít následující příkaz:

```
$ uucp  -mr  swim!groucho!~/security/tripwire-1.0.tar.gz trip.tgz
```

Vytvořená úloha požádá hostitele **swim**, aby získal příslušný soubor od hostitele **groucho** a poslal ho zpět vašemu počítači, přičemž ho protokol UUCP uloží do souboru `trip.tgz` a oznámí vám poštou jeho doručení. Výše uvedený proces proběhne ve třech krocích. Nejprve pošle váš počítač úlohu hostiteli **swim**. Jakmile tento hostitel naváže příští spojení s hostitelem **groucho**, nahraje příslušný soubor. Závěrečným krokem je vlastní přenos souboru z hostitele **swim** na váš počítač.

Nejdůležitější služby, které dnes poskytují sítě založené na protokolu UUCP, jsou elektronická pošta a síťové news. K těmto tématům se ještě vrátíme později, takže nyní si je zde jen nastíníme.

Elektronická pošta – zkráceně e-mail – umožňuje uživatelům předávání zpráv s jinými uživateli pracujícími na vzdálených hostitelích, aniž by bylo nutné znát cestu k těmto hostitelům. Proces směrování zprávy od vašeho počítače k cílovému počítači je plně v režii systému pro správu pošty. V prostředí UUCP jsou poštovní zprávy zpravidla předávány sousednímu hosti-

[3] Při použití příkazového interpretu *bash*, GNU Bourne Again Shell, možná bude nutné zrušit význam znaku vykřičník, protože tento znak se zde používá jako znak historie.

teli za pomoci příkazu `rmail`, kterému je předána adresa příjemce a vlastní zpráva. Příkaz `rmail` potom přesměruje zprávu na jiného hostitele atd., dokud nedorazí k cílovému hostiteli. Na doručování zpráv se podrobněji podíváme v kapitole 13.

Síťové news lze nejlépe přirovnat k jistému druhu distribuované BBS (Bulletin Board System – Elektronický konferenční systém). Nejčastěji se tento termín používá pro „usenetové news," což je nejznámější síť pro výměnu news, u které se odhaduje, že má 120 000 účastníků*. Počátky Usenetu se datují do roku 1979, kdy krátce po vydání protokolu UUCP, jakožto součásti Unixu V7, přišli tři absolventi univerzity s myšlenkou všeobecné výměny informací v rámci unixové komunity. Dali dohromady několik skriptů, čímž vznikl první systém síťových news. V roce 1980 se k této síti připojily servery **duke**, **unc** a **phs** a dvě univerzity v Severní Karolíně. I bez toho se však Usenet konečně začal rozrůstat. I když se původně jednalo o síť založenou na protokolu UUCP, dnes se neomezuje pouze na tento jediný typ sítě.

Základní jednotkou informace je zde článek, který lze zaslat do hierarchie diskusních skupin vyhrazených určitému tématu. Většina míst dostává pouze výběr některých diskusních skupin, což se v průměru rovná asi 60 MB článků denně.

Ve světe UUCP jsou newsy posílány prostřednictvím UUCP- spojení tím způsobem, že se vezmou všechny články požadované diskusní skupiny a sbalí se do několika *dávek (batches)*. Tyto se pak pošlou příslušnému počítači, který je rozbalí a dále zpracuje za pomoci příkazu `rnews`.

UUCP je konečně také médiem archivů umožňujících přístup přes vytáčenou linku a nabízejících veřejný přístup. Přístup k nim znamená připojení prostřednictvím protokolu UUCP, přihlášení se jako uživatel-host a stažení souborů z veřejně dostupné archivní oblasti. Tyto účty mají zpravidla jméno a heslo typu **uucp/nuucp** nebo jim podobné.

1.3 Sítě TCP/IP

I když se protokol UUCP hodí pro levné vytáčené síťové linky, v mnoha situacích je metoda store-and-forward (ulož a předej) příliš málo pružná, například v sítích LAN (Local Area Networks). Tyto sítě jsou obvykle tvořeny malým počtem počítačů, umístěných v jedné budově či dokonce na stejném podlaží a vzájemně propojených, čímž vznikne homogenní pracovní prostředí. Typicky budete chtít sdílet soubory v rámci těchto hostitelů nebo na různých počítačích spouštět distribuované aplikace.

* Poznámka korektora: Dnes je to již podstatně více.

Tyto úlohy vyžadují kompletně odlišný přístup k vytváření sítí. Místo přenášení celých souborů společně s popisem úlohy jsou všechna data rozdělena na malé kousky (pakety), které jsou ihned předány cílovému hostiteli, který je zase zpětně spojí. Tomuto typu sítí říkáme *packed-switched* (přenos paketů). Kromě jiného tak lze prostřednictvím sítě spouštět interaktivní aplikace. Cenou je samozřejmě výrazně složitější software.

Řešení, které přijal systém Unix (a také spousta neunixových platforem) se nazývá TCP/IP. V této stati se podíváme na základní pojetí tohoto protokolu.

1.3.1 Úvod do sítí TCP/IP

Protokol TCP/IP byl vyvinut v rámci výzkumného projektu financovaného americkou společností DARPA (Defense Advanced Research Projects Agency) v roce 1969. Jednalo se o experimentální síť zvanou ARPANET, která byla uvedena do provozu v roce 1975, poté co se ukázalo, že by mohla mít úspěch.

V roce 1983 byla standardizována sada protokolů TCP/IP, kterou museli používat všichni hostitelé v této síti. Když se ze sítě ARPANET nakonec vyvinul Internet (přičemž síť ARPANET sama o sobě zanikla v roce 1990), rozšířilo se používání protokolu TCP/IP i v sítích mimo vlastní Internet. Nejpozoruhodnější jsou lokální sítě na bázi Unixu, ale v předvečer nástupu rychlých telefonních zařízení typu IDSN má slibnou budoucnost i jako přenosový protokol pro vytáčené sítě.

Abychom si ukázali konkrétnější využití protokolu TCP/IP, který budeme probírat v následujících statích, vezmeme si za příklad univerzitu GMU (Groucho Marx University), která se nachází někde v Fredlandu. Většina kateder zde má vlastní lokální sítě, některé je sdílejí a jiné jich zase mají více. Všechny jsou vzájemně propojeny a k Internetu jsou připojeny jednou vysokorychlostní linkou.

Dejme tomu, že váš stroj pracující pod Linuxem je připojen k lokální síti unixových hostitelů na katedře matematiky a jmenuje se **erdos**. Budete-li se chtít připojit k hostiteli na katedře fyzyky, který se jmenuje řekněme **quark**, zadáte následující příkaz:

```
$ rlogin quark.physics
Welcome to the Physics Department at GMU
(ttyq2) login:
```

Na výzvu zadáte vaše uživatelské jméno, třeba **andres,** a vaše heslo. Pak se vám spustí příkazový interpret na hostiteli **quark**, se kterým můžete pracovat stejným způsobem, jako byste seděli u konzoly tohoto systému. Právě jste použili jednu z okamžitých interaktivních aplikací, které poskytuje protokol TCP/IP: vzdálené přihlášení.

Jakmile jste připojeni ke stroji **quark**, budete asi chtít spouštět také nějaké aplikace systému X11, například tiskový program nebo prohlížeč postscriptových dokumentů. Aby příslušná aplikace věděla, že chcete mít její okna zobrazena na obrazovce vašeho hostitele, je třeba nastavit proměnnou prostředí *DISPLAY*:

```
$ export DISPLAY=erdos.maths:0.0
```

Když nyní tuto aplikaci spustíte, spojí se místo stroje **quark** s X-serverem a zobrazí všechna svá okna na vaší obrazovce. K tomu je samozřejmě nutné, aby na počítači **erdos** běžel systém X11. Protokol TCP/IP umožňuje hostitelům **quark** a **erdos** vzájemnou výměnu paketů, což budí dojem, že pracujete v jednom systému. Síť je zde takřka transparentní.

Další velice důležitou vlastností sítí TCP/IP je souborový systémy NFS (Network File Systém). Také on přispívá k transparentnosti sítě, protože umožňuje připojovat adresářové struktury z jiných hostitelů, které se pak jeví jako lokální souborové systémy. Například všechny domovské adresáře uživatelů mohou být na centrálním serveru, ze kterého si je připojují všichni ostatní hostitelé v lokální síti. V důsledku toho se uživatelé mohou přihlásit na libovolný stroj a získají vždy stejný domovský adresář. Podobně je možné nainstalovat aplikace, které vyžadují velký diskový prostor (například TEX), pouze na jednom počítači a tyto soubory pak na ostatní počítače exportovat. K souborovému systému NFS se ještě vrátíme v kapitole 11.

Samozřejmě, že to jsou jen příklady využití sítí založených na protokolu TCP/IP. Možnosti jsou takřka neomezené.

Nyní se podrobněji podíváme na způsob, jakým funguje protokol TCP/IP. Jeho znalost vám pomůže lépe pochopit jak a proč máte nakonfigurovat váš počítač. Začneme hardwarem a pomalu se dostaneme až k tomuto protokolu.

1.3.2 Ethernety

Typu hardwaru, který se nejčastěji používá v lokálních sítích, říkáme *Ethernet*. Je tvořen jedním kabelem, ke kterému jsou jednotliví hostitelé připojeni prostřednictvím konektorů, odboček nebo transceiverů. Jednoduché Ethernety nejsou příliš drahé, což je společně s jejich přenosovou rychlostí 10 Mbitů za sekundů důvodem jejich oblíbenosti.

Ethernety existují ve třech provedeních, které nazýváme *tlustý (thick)*, *tenký (thin)* a *kroucená dvojlinka (twisted pair)*. Tlustý a tenký Ethernet používají koaxiální kabel, který se liší šířkou a způsobem, jakým k němu lze hostitele připojit. Tenký Ethernet používá konektor „BNC“ ve tvaru písmene T, který vložíte do kabelu a zapojíte do zadní stěny vašeho počítače. Tlustý Ethernet vyžaduje vyvrtání malé díry do kabelu a připojení transceiveru za pomocí „vampire tap“. Tenký a tlustý Ethernet může být dlouhý maximálně 200 a 500 metrů, a pro-

to se jim také říká 10base-2 a 10base-5. Kroucená dvojlinka je kabel tvořený dvěma měděnými dráty, které se vyskytují také v obyčejných telefonních linkách, ale zpravidla vyžaduje ještě dodatečný hardware. Také se mu říká 10base-T.

I když je připojení hostitele k tlustému Ethernetu poněkud trnité, není přitom třeba odpojovat síť. Pro připojení hostitele k tenkému Ethernetu je třeba alespoň na několik minut přerušit síťové spojení, protože musíte kabel přeříznout, abyste do něho mohli vložit konektor.

Většina lidí upřednostňuje tenký Ethernet, protože je velice levný: síťové karty seženete už za 500 korun a kabel stojí jen několik korun za metr. Nicméně pro rozsáhlejší instalace je vhodnější tlustý Ethernet, aby nemusel být při každém připojování nového hostitele přerušován síťový provoz.

Jednou z nevýhod ethernetové technologie je omezená délka kabelu, což vylučuje jeho použití pro jiné než lokální sítě. Nicméně pomocí repeaterů, mostů a směrovačů může být pospojováno několik ethernetových segmentů. Repeatery prostě jen kopírují signály mezi dvěma nebo více segmenty, takže všechny segmenty dohromady působí dojmem, jako by se jednalo o jediný Ethernet. Dalším omezením Ethernetu je časování, kdy mezi dvěma hostiteli v síti nesmí být více něž čtyři repeatery. Mosty a směrovače jsou propracovanější. Analyzují příchozí data a doručí je pouze v případě, kdy příjemce není připojen lokálním Ethernetem.

Ethernet funguje jako sběrnice, po které může hostitel posílat pakety (nebo *rámce*) až do velikosti 1500 bajtů jinému hostiteli ve stejném Ethernetu. Adresa hostitele je dlouhá šest bajtů a je napevno zakódována do firmwaru jeho ethernetové desky. Takovéto adresy jsou zpravidla zapisovány ve tvaru dvojic hexadecimálních číslic oddělených dvojtečkami, například: **aa:bb:cc:dd:ee:ff**.

Rámec poslaný jednou stanicí vidí všechny připojené stanice, ale pouze cílová stanice si ho vyzvedne a zpracuje. Pokud se ve stejnou dobu pokusí vysílat dvě stanice, doje ke *kolizi*, kterou vyřeší tak, že vysílání přeruší a za nějakou dobu to zkusí znovu.

1.3.3 Další typy hardwaru

U větších instalací, jako je například Groucho Marx University, se zpravidla kromě Ethernetu používá ještě jiný typ vybavení. Na této univerzitě jsou všechny lokální sítě jednotlivých kateder připojeny k univerzitní páteři, což je optický kabel s rozhraním FDDI (*Fiber Distributed Data Interface*). Toto rozhraní používá při přenosu dat úplně jiný přístup, který spočívá v posílání několika *tokenů* a data může vysílat pouze ta stanice, která zachytí nějaký token. Hlavní výhodou rozhraní FDDI je rychlost až 100 Mbps a maximální délka kabelu až 200 km.

Pro dálková síťová spojení se často používají jiné typy vybavení, vycházející ze standardu X.25. Tuto službu nabízí mnoho takzvaných veřejných datových sítí, jakou je v USA například Tymnet nebo v Německu Datex-P.* Standard X.25 vyžaduje speciální hardware, jmenovitě Packet Assembler/Disassembler (*PAD).X.25* definuje sadu vlastních síťových protokolů, ale přesto je často používáno k propojování sítí, které používají protokol TCP/IP nebo jiné. Protože IP pakety nelze jednoduše namapovat na pakety X.25 (a naopak), jsou IP pakety před odesláním jednoduše zapouzdřeny do paketů X.25.

Radioamatéři často používají své vybavení k vzájemnému propojení svých počítačů; tomu říkáme *paketové rádio* nebo tzv. *ham radio*. Protokol, který používají *paketová rádia*, se nazývá AX.25 a byl odvozen od protokolu X.25.

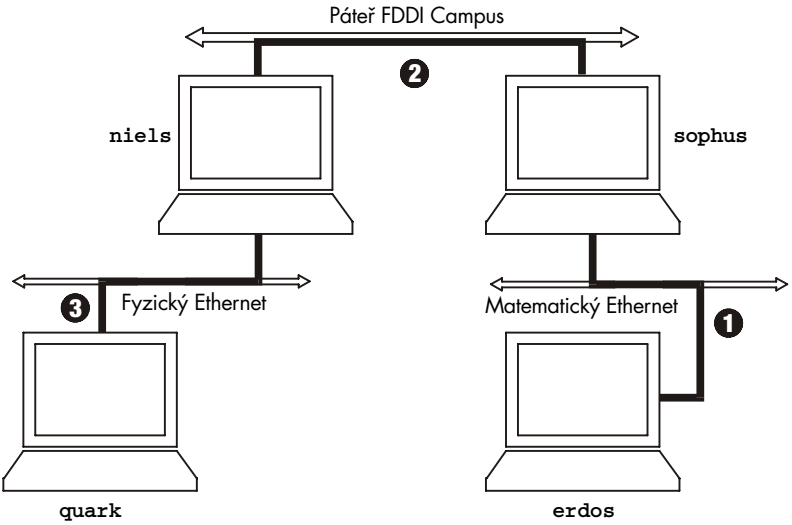
Kromě toho se používají sice pomalé, ale levné sériové linky pro vytáčený (dial-up) přístup. To vyžaduje ještě jeden protokol pro přenos paketů, například SLIP nebo PPP, které si popíšeme dále.

1.3.4 Internetový protokol

Samozřejmě nechcete, aby byla vaše síť omezena jen na Ethernet. V ideálním případě budete chtít používat síť bez ohledu na hardware a počet jednotek, ze kterých je vystavěna. V rozsáhlých sítích, jakou mají například na Groucho Marx University, existuje zpravidla několik oddělených Ethernetů, které je třeba nějakým způsobem pospojovat. Na GMU fungují na katedře matematiky dva Ethernety: první je tvořen rychlými počítači, které využívají profesoři a absolventi, a druhý spojuje pomalejší počítače, na kterých pracují studenti. Obě sítě jsou za pomoci rozhraní FDDI připojeny k univerzitní páteřní síti.

Toto připojení zajišťuje vyhrazený hostitel, takzvaná *brána*, která obsluhuje příchozí a odchozí pakety tím způsobem, že je kopíruje mezi dvěma Ethernety a kabelem s optickými vlákny. Sedíte-li například na katedře matematiky a chcete se z vašeho linuxového stroje připojit k počítači **quark**, který je připojen do lokální sítě na katedře fyziky, nemůže síťový software poslat pakety přímo hostiteli **quark** na katedře fyziky, protože není připojen do stejného Ethernetu. Musí se spolehnout na bránu jakožto na dopravce. Brána (její jméno je **sophus**) pak za pomoci páteřní sítě tyto pakety předá bráně **niels** stejné úrovně na katedře fyziky, která je doručí cílovému počítači. Na obrázku 1.1 je ukázán tok dat mezi bránou **erdos** a **quark** (omlouvám se Guy L. Steeleovi).

* Poznámka korektora: V České republice je to např. společnost Telecom.



Obrázek 1.1
Tři kroky posílání datagramu z počítače **erdos** počítači **quark**

Tomuto schématu doručování dat ke vzdálenému hostiteli se říká *směrování* a paketům v tomto kontextu říkáme *datagramy*. Aby to bylo jednodušší, je výměna datagramů řízena protokolem, který je nezávislý na použitém hardwaru: IP, neboli *Internetový protokol*. Tímto protokolem a otázkami s ním souvisejícími se budeme podrobněji zabývat v kapitole 2.

Hlavní užitek protokolu IP tkví v tom, že spojuje fyzicky rozdílné sítě do jedné zjevně homogenní sítě. Tomu říkáme internetworking (spojování více počítačových sítí do jedné) a výsledné „metasíti“ říkáme *internet*. Všimněte si zde jemného rozdílu mezi slovy *internet* a *Internet*. Druhé z nich označuje konkrétní globální internet.

Samozřejmě, že protokol IP vyžaduje adresové schéma, které je nezávislé na hardwaru. Toho dosáhneme tím, že každému hostiteli přidělíme jedinečné 32bitové číslo nazývané *IP adresa*. IP adresa se zpravidla zapisuje jako čtyři desítková čísla (každé z nich udává 8bitovou část) oddělená tečkami. Například počítač **quark** by mohl mít IP adresu **0x954C0C04**, což bychom zapsali jako **149.76.12.4**. Tomuto formátu zápisu také říkáme *tečková notace*.

Nyní tedy máme tři různé typy adres: nejprve je zde název hostitele, **quark**, potom jeho IP adresa a konečně hardwarová adresa typu 6bajtové ethernetové adresy. Všechny tyto adresy si musí určitým způsobem odpovídat, takže když zadáte `rlogin quark`, může být síťovému softwaru předána IP adresa hostitele **quark** a když protokol IP doručí data do Ethernetu katedry fyziky, musí nějakým způsobem zjistit, která ethernetová adresa této IP adrese odpovídá. Což vypadá poněkud zmateně.

Teď se tím nebudeme zabývat, ale k tomuto tématu se vrátíme v kapitole 2. Nyní nám bude stačit, když si zapamatujeme, že tomuto procesu získávání adresy mapováním názvu hostitele na IP adresu říkáme *rozlišování názvu hostitele* (*hostaname resolution*) a mapování na hardwarovou adresu říkáme *rozlišování adresy* (*address resolution*).

1.3.5 IP přes sériové linky

Na sériových linkách je de facto standardem protokol SLIP, neboli *Serial Line IP*. Modifikovaný typ protokolu SLIP se nazývá CSLIP, neboli *compressed SLIP*, a provádí kompresi IP hlaviček, aby je bylo možné lépe používat v relativně malé šířce pásma, kterou poskytují sériové linky.[4] Odlišným sériovým protokolem je PPP, neboli *Point-to-Point Protocol*. Tento protokol má mnohem více funkcí než protokol SLIP, včetně fáze vyjednávání spojení. Jeho hlavní výhodou oproti protokolu SLIP je, že může přenášet libovolné datagramy, nejenom IP.

1.3.6 Transmission Control Protocol

Samozřejmě, že odesláním datagramů z jednoho hostitele na druhý celý proces nekončí. Přihlásíte-li se k počítači **quark**, chcete, aby bylo spojení mezi vaším procesem *rlogin* na hostiteli **erdos** a příkazovým interpretem na hostiteli **quark** bezpečné. Proto musí odesílatel informaci posílanou z jednoho počítače na druhý rozdělit na pakety a příjemce pak z nich musí znovu sestavit proud znaků. Vypadá to sice jednoduše, ale celý tento proces znamená provedení několika nepříjemných úkolů.

Předně je důležité vědět, že cílem protokolu IP nebylo zajistit bezpečnost. Představte si, že by deset lidí připojených k vaší ethernetové síti začalo stahovat poslední verzi programu Xfree86 z FTP serveru GMU. Síťový provoz, který by tyto operace vyvolaly, by brána nemusela zvládnout, protože je příliš pomalá a příliš těsně svázaná s pamětí. Když teď pošlete z počítače **quark** nějaký paket, nemusí mít brána **sophus** dostatek místa ve vyrovnávací paměti a tím pádem ho nebude moci předat dál. Protokol IP řeší tento problém tak, že příslušný paket zruší. Paket je tak neodvolatelně ztracen. Proto závisí na komunikujících hostitelích, aby kontrolovali integritu a úplnost dat a v případě výskytu chyby přenos opakovali.

K tomu se používá další protokol zvaný TCP, neboli *Transmission Control Protocol*, který vytváří spolehlivou službu nad protokolem IP. Základní vlastností protokolu TCP je, že za pomoci protokolu IP vytváří iluzi jediného spojení mezi příslušnými dvěma procesy na vašem hostiteli a vzdáleném počítači, takže se nemusíte starat jakým způsobem a po jaké trase vaše data ve skutečnosti putují. TCP spojení v podstatě funguje jako dvousměrná roura, do které mohou oba procesy zapisovat i z ní číst. Lze to přirovnat k telefonnímu hovoru.

[4] Protokol SLIP popisuje specifikace RFC 1055. Protokol CSLIP, který provádí kompresi hlaviček, je popisován specifikací RFC 1144.

Protokol TCP rozpozná koncové body takového spojení podle IP adresy příslušných dvou hostitelů a číslo takzvaného *portu* na každém z hostitelů. Na porty je možné se dívat jako na přípojky síťových spojení. Máme-li znovu použít podobnost s telefonním spojením, lze IP adresu přirovnat ke směrovému číslu (směrují čísla na určité město) a čísla portů pak k místním telefonním číslům (směrují čísla na jednotlivé telefony).

V našem příkladu otevře klientská aplikace (`rlogin`) port na hostiteli **erdos** a spojí se s portem 513 na hostiteli **quark**, který odposlouchává server `rlogind`. Tím je navázáno TCP spojení. Za pomoci tohoto spojení provede server `rlogind` autorizaci uživatele a potom spustí příkazový interpret. Standardní vstup a výstup tohoto příkazového interpretu jsou přesměrovány na TCP spojení, takže cokoliv napíšete na vašem počítači, bude proudem TCP přeneseno a předáno na standardním vstupu příkazového interpretu.

1.3.7 User Datagram Protocol

Samozřejmě, že protokol TCP není jediným uživatelským protokolem v sítích TCP/IP. I když je vhodný pro aplikace typu *rlogin,* jeho vysoká režie neumožňuje používat aplikace typu NFS. Místo toho se používá spřízněný protokol UDP, neboli *User Datagram Protocol*. Podobně jako TCP, i protokol UDP dovoluje aplikaci kontaktovat službou na určitém portu na vzdáleném počítači, ale nenaváže s ní spojení. Místo toho lze pomocí tohoto protokolu poslat cílové službě samostatné pakety – odtud je odvozen název tohoto protokolu.

Předpokládejme, že máte připojenu adresářovou strukturu programu **TeX** z centrálního serveru NFS vaší katedry, který se jmenuje ***galois***, a chcete si prohlédnout dokument popisující používání programu LaTeX. Spustíte editor, který nejprve načte celý soubor. Avšak navázání TCP spojení s hostitelem ***galois***, poslání souboru a jeho uvolnění by trvalo příliš dlouho, takže místo toho pošlete hostiteli ***galois*** požadavek, aby poslal příslušný soubor jako dvojici UDP-paketů, což je mnohem rychlejší. Ovšem protokol UDP nebyl vystavěn tak, aby uměl obsloužit ztrátu nebo poškození paketů. Záleží na aplikaci (v tomto případě NFS) aby se o to postarala.

1.3.8 Více o portech

Na porty je možné nahlížet jako na přípojky síťových spojení. Pokud chce nějaká aplikace nabízet určitou službu, připojí sama sebe k určitému portu a čeká na klienty (říká se tomu také *odposlouchávání* portu). Klient, který chce tuto službu využívat, si alokuje nějaký port na lokálním hostiteli a připojí se k příslušnému portu serveru na vzdáleném hostiteli.

Důležitou vlastností portů je, že jakmile bylo navázáno spojení mezi klientem a serverem, může se k portu serveru připojit jiná kopie serveru a odposlouchávat další klienty. To umožňuje například současné přihlášení ke stejnému hostiteli při využití stejného portu

číslo. 513. Protokol TCP je schopen tato spojení rozlišit, protože všechny přicházejí z různých portů nebo hostitelů. Pokud se například dvakrát přihlásíte k hostiteli **quark** z počítače **erdos**, potom bude první klient `rlogin` využívat port 1023 a druhý klient port 1022. Oba však budou připojeni ke stejnému portu 522 na hostiteli **quark**.

Tento příklad ukazuje využití portů jako přístupových bodů, kde se klient připojuje ke specifickému portu, aby získal určitou službu. Aby klient znal správné číslo portu, musí mezi administrátory obou systémů existovat určitá dohoda ohledně přiřazování těchto čísel. U služeb, které jsou široce používány, například `rlogin`, musí být tato čísla spravována centrálně. O to se stará organizace IETF (*Internet Engineering Task Force*), která pravidelně vydává RFC nazvané *Přiřazená čísla* (*Assigned Numbers*). Kromě jiného popisuje čísla portů přiřazená *všeobecně známým službám*. Linux používá službu pro mapování názvů služeb na čísla, která se nazývá `/etc/services`. Je popisována ve stati *Soubory služeb a protokolů*.

Je třeba poznamenat, že i když spojení využívající protokoly TCP i UDP spoléhají na porty, nedochází mezi nimi ke konfliktům. To znamená, že se například TCP port 513 liší od UDP-portu 513. Ve skutečnosti slouží tyto porty jako vstupní body pro dvě různé služby, jmenovitě `rlogin` (TCP) a `rwho` (UDP).

1.3.9 Knihovna socketů

V operačních systémech Unix je software, který obsluhuje veškeré výše popsané úkoly a protokoly, většinou součástí jádra, což platí i pro Linux. Ve světě Unixu je nejběžnějším programovým rozhraním knihovna *Berkeley Socket Library*. Její název je odvozen od oblíbené analogie, která nazírá na porty jako na sockety a připojování k portu chápe jako zapojování. Voláním příkazu (`bind(2)`) je specifikován vzdálený hostitel, přenosový protokol a služba, ke které se program může připojit nebo ji odposlouchávat (pomocí příkazů `connect(2)`, `listen(2)` a `accept(2)`). Knihovna socketů je však obecnější v tom, že poskytuje nejenom třídu socketů založených na protokolu TCP/IP (sockety *AF_INET*), ale také třídu, která obsluhuje spojení s lokálním počítačem (třída *AF_UNIX*). Některé implementace mohou obsluhovat také jiné třídy, jako například protokol XNS (*Xerox Networking Systém*) nebo X.25.

V Linuxu je knihovna socketů součástí standardní knihovny jazyka C zvané *libc*. V současné době podporuje pouze sockety *AF_INET* a *AF_UNIX*, ale pracuje se na podpoře pro síťové protokoly firmy Novell, čehož výsledkem bude začlenění jednoho nebo více tříd socketů tohoto druhu.

1.4 Vytváření sítí v Linuxu

Jakožto výsledek soustředěného úsilí programátorů z celého světa by Linux nemohl existovat bez globální počítačové sítě. Takže nikoho nepřekvapí, že již v raných stádiích vývoje začalo několik lidí pracovat na síťových schopnostech tohoto operačního systému. Implementace protokolu UUCP běžela v Linuxu již od samého začátku a práce na začlenění protokolu TCP/IP začala na podzim roku 1992, když Ross Biro a další vytvořili projekt, který získal označení Net-1.

Poté co Ross v květnu 1993 opustil práci na vývoji tohoto protokolu, začal Fred van Kempen pracovat na nové implementaci, přičemž přepracoval hlavní části kódu. Výsledkem byla verze Net-2. První veřejná verze Net-2d byla hotová v létě 1992 (jako součást jádra verze 0.99.10) a od té doby byla udržována a rozšiřována různými lidmi, zejména Alanem Coxem, který stál u zrodu verze Net-2Debugged. Po důkladném otestování a mnoha vylepšeních kódu přišla po vydání Linuxu verze 1.0 změna názvu na Net-3. Právě tato verze síťového kódu je obsažena v oficiální verzi jádra.

Net-3 nabízí ovladače zařízení pro širokou paletu ethernetových karet, stejně jako SLIP (pro síťový provoz po sériových linkách) a PLIP (pro paralelní linky). Díky Net-3 má Linux implementaci protokolu TCP/IP, která se chová velmi dobře v prostředí lokální sítě, dobu provozu, která překonává některé komerční Unixy pro PC. Vývoj teď směřuje k zajištění nezbytné stability pro běh na internetových hostitelích.

Kromě těchto výhod probíhá několik projektů, které zvýší univerzálnost Linuxu. Ovladač pro protokol PPP (point-to-point protokol je nový způsob síťového přenosu po sériových linkách) je ve stádiu beta-verze a ovladač pro protokol X.25 pro amatérské rádio existuje ve verzi alfa. Alan Cox také implementoval ovladač pro novellovský protokol IPX, ale úsilí o vytvoření kompletní sady nástrojů kompatibilních s produkty firmy Novell bylo nyní pozastaveno, protože firma Novell nepříliš ochotně poskytuje nezbytnou dokumentaci. Dalším velice slibným projektem je *samba*, volně šiřitelný server NetBIOS pro Unices, který napsal Andrew Tridgell.[5]

1.4.1 Různé směry vývoje

Mezitím Fred pokračoval ve vývoji a přešel na verzi Net-2e, která měla přepracovanou strukturu síťové vrstvy. V době psaní této knihy byla Net-2e stále ve stádiu beta-verze. Největším přínosem Net-2e je zařazení rozhraní DDI (*Device Driver Interface*). Toto rozhraní nabízí jednotný přístup a konfiguraci všech síťových zařízení a protokolů.

5 NetBIOS je protokol, na kterém jsou založeny aplikace typu *lanmanager* a Windows for Workgroups.

Další implementace protokolu TCP/IP pochází od Matthiase Urlichse, který napsal ovladač ISDN pro Linux a FreeBSD. Za tím účelem integroval do jádra Linuxu část síťového kódu BSD.

Lze předpokládat, že v brzké době se objeví verze Net-3.* Alan v současné době pracuje na implementaci protokolu AX.25, který používají ham-radioamatéři. Plánovaný „modulový" kód jádra nepochybně vnese nové podněty i do síťového kódu. Moduly umožní připojování ovladačů k jádru za běhu.

I když se tyto různé síťové implementace snaží poskytovat stejné služby, existují mezi nimi na úrovni jádra a ovladačů velké rozdíly. Proto nelze nakonfigurovat systém, na kterém běží jádro Net-2e za pomocí utilit z Net-2d nebo Net-3 a naopak. To se ale týká jen příkazů, které úzce spolupracují s jádrem; aplikace a běžné síťové příkazy, jakými jsou například `rlogin` nebo `telnet`, fungují na obou.

Přesto však by vás všechny tyto různé síťové verze neměly trápit. Pokud se nepodílíte na aktivním vývoji, nemusí vás zajímat, jakou verzi TCP/IP kódu používáte. Oficiální vydání jádra budou vždy obsahovat sadu síťových nástrojů, které jsou kompatibilní se síťovým kódem obsaženým v jádru.

1.4.2 Kde získáte kód

Poslední verzi síťového kódu Linuxu lze získat prostřednictvím anonymního FTP z různých míst. Oficiální FTP server pro Net-3 je **sunacm.swan.ac.uk**, který je zrcadlen na **sunsite.unc.edu**. Poslední verze oprav a spustitelných souborů pro Net-2e najdete na **ftp.aris.com**. Síťový kód odvozený od BSD, jehož autorem je Matthias Urlichs, najdete na **ftp.ira.uka.de** v adresáři `/pub/system/linux/netbsd`.

Poslední verze jádra lze získat na adrese **nic.funet.fi** v adresáři `/pub/OS/Linux/PEOPLE/Linux`, jejíž zrcadla jsou **sunsite** a **tsx-11.mit.edu**.**

1.5 Údržba vašeho systému

V průběhu celé knihy se budeme zabývat především otázkami instalace a konfigurace. Správa však znamená mnohem víc – po zavedení služby je třeba ji také udržovat v provozu. Většina služeb vyžaduje jen nepatrnou údržbu, ale u některých z nich, jako je pošta a news, vyžaduje zachování aktuálnosti systému provádění rutinních úkolů. Tyto úkoly budeme probírat v dalších kapitolách.

* Poznámka korektora: Tato verze dnes již existuje.

** Poznámka korektora: V poslední době je oficiálním archívem zdrojových textů jádra server **ftp.kernel.org.**

Absolutní minimum údržby spočívá v pravidelné kontrole systémových a aplikačních log-souborů, zda neobsahují chybové stavy nebo neobvyklé události. Běžně se to dělá za pomoci dvojice administrativních skriptů, které se pravidelně spouští z `cron`. Tyto skripty obsahují distribuce zdrojových kódů některých hlavních aplikací, jako je například `smail` nebo C news. Je třeba si je jen upravit tak, aby vyhovovaly vašim potřebám a preferencím.

Výstup každé úlohy `cron` by měl být poslán poštou na administrativní účet. Mnoho aplikací implicitně posílá chybové zprávy, statistiky využití nebo souhrny log-souborů na účet **root**. To má smysl, pokud se často přihlašujete jako uživatel **root**, ale výhodnější je za pomocí poštovního aliasu, který popisujeme v kapitole 14, přesměrovat poštu pro uživatele **root** na váš osobní účet.

Bez ohledu na to, jak pečlivě váš systém nastavíte, se v důsledku existence Murphyho zákonů zaručeně vynoří nějaký problém. Proto znamená údržba systému i neustálou připravenost řešit stížnosti. Lidé zpravidla očekávají, že správce systému bude k zastižení přinejmenším prostřednictvím účtu **root**, ale pro kontaktování lidí, kteří jsou odpovědní za určitý aspekt údržby, jsou běžně používány také jiné adresy. Například stížnosti na špatně fungující poštu budou adresovány uživateli **postmaster** a problémy se systémem news je možné oznamovat na adresách **newsmaster** nebo **usenet**. Pošta na účet **hostmaster** by měla být přesměrována člověku, který má na starosti základní síťové služby, a pokud provozujete jmenný server (name-server), také jmennou službu DNS.

1.5.1 Bezpečnost systému

Dalším velice důležitým aspektem správy systému v síťovém prostředí je ochrana systému a uživatelů před vetřelci. Nedbale spravované systémy jsou vhodným cílem pro škodolibé uživatele, jejichž rozsah útoků je poměrně široký: od uhodnutí hesla až po sledování Ethernetu a způsobené škody mohou sahat od padělaných poštovních zpráv až po ztrátu dat nebo narušení soukromí vašich uživatelů. Když budeme hovořit o souvislostech, za kterých k nim může dojít, zmíníme se o některých konkrétních problémech a běžných ochranách proti nim.

Tato stať probírá některé příklady a základní postupy zajištění systémové bezpečnosti. Samozřejmě, že zde uvedená témata nemohou důkladně postihnout všechny bezpečností problémy, s nimiž se můžete setkat; slouží především k ilustraci problémů, které se mohou vyskytnout. Proto je nutné přečíst si nějakou dobrou knihu pojednávající o bezpečnosti, zejména pak v systému propojeném do sítě. Doporučujeme knihu „Practical UNIX Security“, jejímž autorem je Simon Gafinkel (viz [Spaf93]).

Základem bezpečnosti systému je dobrá systémová administrace. Ta zahrnuje kontrolu vlastnictví a přístupových práv ke všem životně důležitým souborům a adresářům, monitorování používání privilegovaných účtů atd. Program COPS například ověří, zda váš systém souborů

(file system) a běžné konfigurační soubory neobsahují neobvyklá přístupová práva nebo jiné anomálie. Také je rozumné používat sadu programů pro správu hesel, které prosadí jistá pravidla pro zadávání hesel takovým způsobem, aby bylo těžké je uhodnout. Stínová sada programů pro správu hesel například vyžaduje, aby bylo heslo dlouhé minimálně pět písmen a obsahovalo, kromě malých a velkých písmen, i číslice.

Při zpřístupňování určité služby přes síť jí nezapomeňte přidělit „nejnižší práva“, což znamená, že jí nedovolíte provádět věci, které pro své fungování nepotřebuje. Programům byste měli například přidělit UID **root** nebo některý jiný privilegovaný účet, jen pokud ho opravdu potřebují. Také v případě, kdy chcete používat některou službu jen pro omezenou aplikaci, ji neváhejte nakonfigurovat s co největším omezením práv, jak to dovolí vaše aplikace. Chcete-li například povolit bezdiskovým hostitelům zavádění z vašeho počítače, musíte zprovoznit službu TFTP (trivial file transver service), aby si tito hostitelé mohli z adresáře `/boot` nahrát základní konfigurační soubory. Pokud byste ale používali službu TFTP bez omezení, umožnili byste komukoliv na světě nahrát si z vašeho systému libovolný soubor, ke kterému má přístupová práva pro čtení. Proč tedy neomezit služby TFTP jen na adresář `/boot`?[6]

Při stejném směru myšlení možná budete chtít omezit určité služby poskytované určitými hostiteli, řekněme z vaší lokální sítě. V kapitole 9 si představíme nástroj `tcpd`, který to umožňuje pro různé síťové aplikace.

Dále je nutné se vyhnout „nebezpečným“ aplikacím. Samozřejmě, že každý software, který používáte, může být svým způsobem nebezpečný, protože může obsahovat chyby, jichž mohou chytří lidé využít k získání přístupu k vašemu systému. K takovýmto věcem dochází, a proto neexistuje úplná ochrana. Tento problém se týká volně šiřitelného softwaru stejně jako komerčních produktů.[7] Nicméně programy, které vyžadují speciální přístupová práva jsou dědičně více nebezpečné než jiné, protože každá díra v nich může mít drastické důsledky.[8] Pokud instalujete program setuid pro síťové účely, pak dávejte dvojnásobný pozor na to, abyste nezapomněli na nic z dokumentace a náhodou tak nevytvořili bezpečnostní trhlinu.

Nikdy nelze vyloučit, že vaše opatření selžou, bez ohledu na to, jak opatrní jste byli. Proto byste měli zajistit včasné odhalení vetřelců. Kontrola systémových log-souborů je dobrým výchozím bodem, ale vetřelec bude zřejmě natolik chytrý, že po sobě smaže všechny zřetelné stopy. Existují však nástroje typu `tripwire`[9], které umožňují kontrolovat, zda nedošlo ke změně obsahu nebo přístupových práv životně důležitých systémových souborů. Program

6 K tomuto tématu se ještě vrátíme v kapitole 9.

7 Existovaly jisté komerční subjekty, kterým jste zaplatili spoustu peněz za setuid-skripty, jež využitím jednoduchého standardního triku dovolovaly uživatelům získat práva **root**.

8 V roce 1988 způsobil červ RTM problémy velké části Internetu tím, že zneužil díry v programu *sendmail*. Tato bezpečnostní díra již byla od té doby odstraněna.

9 Jeho autorem je Gene Kim a Gene Spafford.

`tripwire` provede různé kontrolní součty nad těmito soubory a uloží je do databáze. Později jsou kontrolní součty vypočítány znovu a porovnány s hodnotami uloženými v databázi, čímž se zjistí jakékoliv případné modifikace.

1.6 Přehled následujících kapitol

Několik dalších kapitol se bude zabývat konfigurací Linuxu pro sítě TCP/IP a provozem některých základních aplikací. Dříve než se pustíme do úpravy souborů a věcí okolo si v kapitole 2 podrobněji popíšeme protokol IP. Pokud již ovládáte způsob, jakým funguje IP směrování a jak probíhá rozpoznávání adres, můžete tuto kapitolu přeskočit.

Kapitola 3 se zabývá základními spornými body konfigurace, jako je kompilace jádra a nastavení ethernetové karty. Konfigurace sériových portů je probírána v samostatné kapitole, protože se netýká jen protokolu TCP/IP, ale také protokolu UUCP.

Kapitola 5 vám pomůže zprovoznit váš počítat v síti TCP/IP. Obsahuje rady k instalaci samostatných hostitelů, kteří mají povolenu jen zpětnou vazbu, a hostitelů připojeným k Ethernetu. Také se zde seznámíte s několika užitečnými nástroji, jichž lze využít při testování a ladění vašeho nastavení. Další kapitola probírá konfiguraci rozpoznávání hostitelů a vysvětluje jak nastavit jmenný server.

Potom následují dvě kapitoly, které se zabývají konfigurací a používáním protokolů SLIP a PPP. Kapitola 7 vysvětluje jak navázat spojení pomocí protokolu SLIP a najdete zde také podrobný popis nástroje `dip`, který umožňuje zautomatizovat většinu nezbytných kroků. Kapitola 8 popisuje protokol PPP a démona `pppd`, který k němu potřebujete.

Kapitola 9 je krátkým úvodem do zavádění některých nejdůležitějších síťových aplikací, například `rlogin`, `rcp` atd. Dozvíte se zde také, jak jsou za pomocí *ihetd* spravovány služby a jak lze určité služby, u kterých je důležitá bezpečnost, omezit jen na důvěryhodné hostitele.

V následujících dvou kapitolách je probírán systém NIS, Network Information System, a NFS, Network File System. NIS je užitečný nástroj pro šíření administrativních informací typu uživatelských hesel v lokální počítačové síti. NFS umožňuje sdílet systémy souborů mezi různými hostiteli v síti.

Kapitola 12 je obsáhlým úvodem do správy Taylor UUCP-sady programů protokolu UUCP, která je zdarma.

Zbytek knihy je věnován podrobnému popisu elektronické pošty a usenetových news. V kapitole 13 se dozvíte základní pojmy týkající se elektronické pošty – jak vypadá poštovní adresa a jak systém pro obsluhu pošty zajistí, aby se vaše zpráva dostala k adresátovi.

Kapitoly 14 a 15 popisují nastavení programů `smail` a `sendmail`, dvou agentů pro přenos pošty, jichž lze v Linuxu využívat. Tato kniha popisuje oba dva, protože instalace agenta `smail` je pro začátečníka jednodušší, zatímco agent `sendmail` si lze lépe přizpůsobit.

Kapitoly 16 a 17 objasňují způsob, jakým jsou na Usenetu spravovány síťové news a jak lze nainstalovat C news, oblíbený softwarový balík pro správu usenetových news. Kapitola 18 stručně vysvětluje nastavení démona NNTP, který vaší síti zprostředkuje přístup pro čtení news. A konečně v kapitole 19 se dozvíte, jak nastavit a udržovat různé programy pro čtení news.

Kapitola

2

Problémy sítí na bázi TCP/IP

Nyní se podíváme na problémy, se kterými se můžete setkat při připojení vašeho linuxového počítače k síti, jež využívá protokol TCP/IP. Vyřešíme problémy týkající se IP adres, názvů hostitelů a některé problémy směrování. Základy, které získáte v této kapitole, se vám budou hodit při konfigurování linuxového počítače. Další kapitoly se budou zabývat nástroji, které k tomu budete potřebovat.

2.1 Síťová rozhraní

Aby se vyřešila rozmanitost zařízení, která mohou být používána v síťovém prostředí, definuje protokol TCP/IP abstraktní *rozhraní*, přes které je přistupováno k hardwaru. Toto rozhraní nabízí množinu operací, která je stejná pro všechny typy hardwaru a zabezpečuje posílání a přijímání paketů.

Každé periferní zařízení, které chcete použít v síti, musí mít v jádru operačního systému příslušné rozhraní. Například v Linuxu se rozhraní Ethernet nazývají *eth0* a *eth1* a rozhraní SLIP pak *sl0*, *sl1* atd. Tyto názvy rozhraní se používají při konfiguraci, když chcete v jádru systému pojmenovat konkrétní fyzické zařízení. Jinde nemají žádný význam.

Aby bylo rozhraní použitelné v sítích na bázi TCP/IP, musí mu být přiřazena IP adresa, která slouží jako identifikace při komunikaci se zbytkem světa. Tato adresa je odlišná od názvu rozhraní, o kterém jsme mluvili výše; pokud bychom rozhraní přirovnali ke dveřím, pak adresa odpovídala štítku, který je na nich připevněn.

Samozřejmě lze nastavovat i další parametry; jedním z nich je maximální velikost datagramů, které mohou být zpracovány danou částí hardwaru. Tento parametr se také nazývá *maximální přenosová jednotka* (Maximal Transfer Unit), neboli MTU. Další atributy budou představeny později.

2.2 IP adresy

Již v předchozí kapitole jsme se zmínili o tom, že adresy, kterým rozumí síťový protokol IP, jsou 32bitová čísla. Každému počítači musí být vzhledem k síťovému prostředí přiděleno jedinečné číslo. Pokud provozujete lokální síť, která s ostatními sítěmi nekomunikuje na bázi protokolu TCP/IP, můžete tato čísla přidělit podle vlastního uvážení. Nicméně systémům připojeným k Internetu jsou čísla přidělována centrálně, konkrétně síťovým informačním centrem (Network Information Center), zkráceně NIC.[1]

IP adresy jsou pro přehlednost rozděleny do čtyř 8bitových čísel, kterým říkáme *oktety*. Například adresa **quark.physics.groucho.edu** má IP adresu **0x954C0C04**, která je zapsána jako **149.76.12.4**. Tomuto formátu se také někdy říká *tečková notace*.

Dalším důvodem této symboliky je rozdělení IP adres na číslo *sítě*, které je obsaženo v levých dvou oktetech, a na číslo *hostitele*, které je zapsáno v pravých dvou oktetech. Když požádáte centrum NIC o přidělení IP adres, nebude vám přidělena adresa pro každého hostitele, kterého hodláte používat. Místo toho obdržíte jediné číslo sítě a pak na základě vlastního uvážení můžete přidělovat hostitelům ve vaší síti všechny IP adresy v rámci získaného rozsahu IP adres.

V závislosti na velikosti sítě je nutné mít i různě velkou část hostitele. Protože mají různí uživatelé různé nároky na velikost sítí, existuje několik tříd sítí, které definují různá rozdělení IP adres.

Třída A Třída A obsahuje sítě od **1.0.0.0** do **127.0.0.0**. Číslo sítě je obsaženo v prvním oktetu. Takto je k dispozici 24bitová část hostitele, která umožňuje až 1,6 milionů hostitelů.

Třída B Třída B obsahuje sítě od **128.0.0.0** do **191.255.0.0**; číslo sítě je obsaženo v prvních dvou oktetech. To umožňuje 16 320 sítí, z nichž každá může mít až 65 024 hostitelů.

[1] Často vám jsou IP-adresy přidělovány poskytovatelem, od kterého si IP-spojení koupíte. Můžete však také požádat o přidělení IP-adresy pro vaši síť přímo NIC. Provedete to tak, že pošlete e-mail na adresu **hostmaster@internic.net**.

Třída C Třída C zahrnuje sítě od **192.0.0.0** do **223.255.255.0**, kde číslo sítě je obsaženo v prvních třech oktetech. To umožňuje vytvořit téměř 2 miliony sítí s až 254-mi hostiteli.

Třída D, E a F Adresy, které spadají do intervalu od **224.0.0.0** do **254.0.0.0,** jsou buď experimentální, nebo jsou rezervovány pro budoucí použití. Nespecifikují žádnou síť.

Vrátíme-li se zpět k příkladu z minulé kapitoly, zjistíme, že adresa **149.76.12.4**, což je adresa **quark**, spadá do třídy sítí B **149.76.0.0** a odkazuje na hostitele **12.4**.

Ve výše uvedeném seznamu jste si možná všimli, že v každém oktetu nebyly v příslušné části hostitele povoleny všechny hodnoty. To proto, že hostitelé, jejichž čísla oktetů jsou rovna **0** nebo **255,** jsou rezervováni pro speciální účely. Adresa, ve které jsou všechny bity hostitelské části nulové, odkazuje na síť a adresa, v níž jsou všechny bity hostitelské části rovny jedné, se nazývá vysílací adresa (broadcast address). Vysílací adresa současně odkazuje na všechny hostitele v konkrétní síti. Tedy adresa **149.76.255.255** není koréktní hostitelská adresa, ale odkazuje na všechny hostitele v síti **149.76.0.0**.

Dále existují ještě dvě rezervované síťové adresy, a to **0.0.0.0** a **127.0.0.0**. První z nich se nazývá *implicitní směr* (default route), druhá se nazývá *zpětnovazebná adresa* (loopback address). Implicitní směr souvisí se způsobem směrování IP datagramů, které budeme probírat dále.

Síť **127.0.0.0** je rezervována pro místní IP dopravu k vašim hostitelům. Adresa **127.0.0.1** je obvykle přiřazena speciálnímu rozhraní na vašem hostiteli, které se někdy také nazývá *zpětnovazebné rozhraní* (loopback interface) a chová se jako uzavřený obvod. Každý IP paket předaný protokolem TCP nebo UDP je vrácen zpět, jakoby právě přišel z nějaké sítě. To umožňuje vyvíjet a testovat síťový software, aniž byste používali „skutečnou“ síť. Další užitečnou aplikací je případ, kdy chcete použít síťový software na samostatném hostiteli. Není to zase tak neobvyklé, jak by se mohlo zdát; například mnoho systémů UUCP nemá vůbec žádné IP propojení, ale přesto chtějí provozovat systém news INN. Aby systém INN správně fungoval pod Linuxem, vyžaduje zpětnovazebné rozhraní.

2.3 Rozlišování adres

Když teď víte, jak se vytváří IP adresy, bude vás asi zajímat, jak se s jejich pomocí v Ethernetu adresují různí hostitelé. Protokol Ethernet identifikuje hostitele číslem složeným ze šesti oktetů, které nemá nic společného s IP adresou.

Proto potřebujeme mechanismus, jenž ethernetovým adresám přiřadí IP adresy. Tento mechanismus se nazývá protokol pro rozlišování adres (Address Resolution Protocol), zkráceně ARP, ktrý není omezen pouze na Ethernet, ale používá se i u jiných typů sítí, příkladem je ham-radio. Idea protokolu ARP je stejná jako když hledáte pana X. Představte si tlačenici asi 150-ti lidí, kteří budou chodit pořád dokola, volat jeho jméno a spoléhat na to, že jim v případě, že je přítomen, odpoví.

Když chce ARP najít ethernetovou adresu odpovídající příslušné IP adrese, použije tzv. „vysílání“ (broadcasting), při kterém je datagram současně adresován na všechny stanice v síti. Vyslaný datagram, který odešle protokol ARP, obsahuje dotaz na příslušnou IP adresu. Každý hostitel, který přijímá, ji porovná se svou vlastní IP adresou a pokud souhlasí, vrátí dotazujícímu se hostiteli odpověď pomocí protokolu ARP. Nyní může dotazující se hostitel vytáhnout z odpovědi ethernetovou adresu zasilatele.

Možná se ptáte, jak hostitelé vědí, na kterém z miliard světových Ethernetů lze požadovaného hostitele nalézt a proč to má být zrovna Ethernet? Všechny tyto otázky souvisí se směrováním, tedy nalezením fyzického místa hostitele v síti. To bude předmětem dalšího výkladu.

Podívejme se nyní na protokol ARP. Když hostitel získá ethernetovou adresu, uloží ji do vyrovnávací paměti protokolu ARP, takže když bude dotyčnému hostiteli posílat zpět nějaký datagram, nemusí se na ni znovu dotazovat. Tuto informaci by však nebylo rozumné uchovávat příliš dlouho; ethernetová karta vzdáleného hostitele může být například z důvodu technických problémů vyměněna, čímž by byla data protokolu ARP neplatná. Abychom si vynutili další dotazy na IP adresu, dochází po určité době k mazání vstupů z vyrovnávací paměti protokolu ARP.

Někdy je také nutné nalézt IP adresu, která odpovídá určité ethernetové adrese. To je případ, kdy se bezdiskový počítač pokouší zavádět operační systém ze síťového serveru. V lokálních sítích je to docela běžná situace. Bezdiskový klient však o sobě nemůže poskytnout skoro žádné informace – kromě své ethernetové adresy. Jediné co může udělat je poslat spouštěcímu serveru zprávu se žádostí o sdělení jeho IP adresy. Pro tento účel existuje další protokol, který se nazývá Protokol pro zpětné rozlišení adres (*Reverse Adress Resolution Protocol*), zkráceně RARP. Společně s protokolem BOOTP slouží k definování procedury zavádění operačního systému bezdiskových klientů v síti.

2.4 Směrování IP

2.4.1 Sítě IP

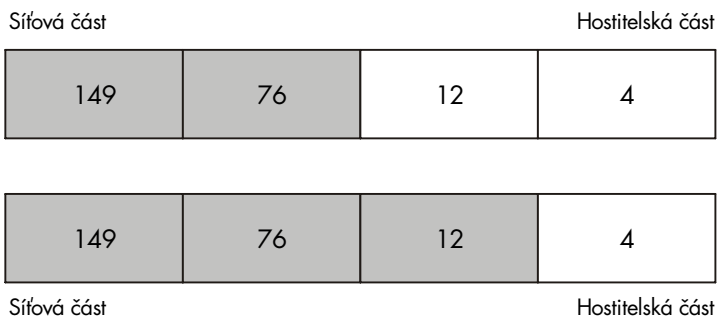
Když někomu píšete dopis, uvedete obvykle na obálce kompletní adresu, která obsahuje název země, stát, kraj, PSČ atd. Poté co ho vhodíte do poštovní schránky, ho pošta doručí na cílové místo: pošle ho do vyznačené země, zde ho národní pošta odešle do příslušného kraje atd. Výhoda tohoto hierarchického schématu je poměrně zřejmá: Ať už pošlete dopis kamkoliv, bude vždy místní poštmistr zhruba vědět směr, kterým má dopis poslat dál, a přitom se nemusí starat o to, po jaké trase poputuje dopis v rámci cílové země.

Sítě IP jsou strukturovány obdobným způsobem. Celý Internet se skládá z mnoha sítí, kterým říkáme *autonomní systémy* (autonomous systems). Každý takovýto systém provádí veškeré směrování v rámci svých členských hostitelů, takže se úkol doručení datagramu zredukuje na hledání cesty k síti cílového hostitele. To znamená, že jakmile datagram zachytí *libovolný* hostitel v cílové síti, bude další proces prováděn výhradně touto sítí.

2.4.2 Podsítě

Tato struktura vyplývá z rozdělení IP adres na část hostitelskou a část síťovou, viz výše uvedený popis. Cílová síť je implicitně odvozena ze síťové části IP adresy. Proto by se hostitelé, kteří mají identická síťová IP čísla, měli nacházet ve stejné síti, což platí i obráceně.[2]

Obdobné schéma má smysl i *uvnitř* sítě, protože vlastní síť se může skládat ze stovek menších sítí, kde jsou nejmenšími jednotkami fyzické sítě, například Ethernety. Protokol IP umožňuje rozdělit sít IP na několik *podsítí*.



Obrázek 2.1
Podsítě sítě třídy B

[2] Nicméně autonomní systémy jsou o něco obecnější. Mohou se skládat z více než jedné sítě IP.

Podsíť na sebe přebírá (od sítě IP, které je součástí) odpovědnost za doručení datagramů pro daný rozsah IP adres. K její identifikaci slouží síťová část IP adresy, jako tomu bylo u tříd A, B nebo C. Nyní je však síťová část rozšířena tak, aby obsahovala i některé bity z hostitelské části. Počet bitů, které jsou interpretovány jako číslo podsítě, udává tzv. *maska podsítě* (*subnet mask*) neboli *síťová maska* (*netmask*). Jedná se také o 32bitové číslo, které určuje bitovou masku části sítě IP adresy.

Příkladem takové sítě je školní síť Groucho Marx University. Má síťové číslo třídy B **149.76.0.0**, a proto je její síťová maska **255.255.0.0**.

Vnitřně je školní síť GMU složena z několika menších sítí, které tvoří například lokální sítě různých kateder. Takže rozsah IP adres je rozložen na 254 podsítí, od adresy **149.76.1.0** po adresu **149.76.254.0**. Například katedra teoretické fyziky má přidělenu adresu **149.76.12.0**. Páteří školní sítě je v pravém slova smyslu síť s adresou **149.76.1.0**. Tyto podsítě sdílí stejné číslo sítě IP, zatímco třetí oktet slouží k jejich rozlišení. Používají tedy masku podsítě **255.255.255.0**.

Obrázek 2.1 ukazuje rozdílnou interpretaci čísla **149.76.12.4**, což je adresa **quark.** Je-li v prvním případě považována adresa za adresu běžné sítě třídy B, pak ve druhém případě je považována za adresu podsítě.

Je dobré vědět, že vytváření podsítí slouží pouze k *vnitřnímu dělení* sítě. Podsítě jsou generovány vlastníkem sítě (nebo jejími správci). Sítě jsou často vytvářeny kvůli hranicím, například fyzickým (mezi dvěma Ethernety), administrativním (mezi dvěma katedrami) nebo geografickým, a pravomocemi nad těmito podsítěmi je pověřena nějaká kontaktní osoba. Tato struktura však odráží pouze vnitřní chování sítě a pro okolní svět je neviditelná.

2.4.3 Brány

Vytváření podsítí nemá jen organizační výhody, je často přirozeným důsledkem hranic hardwaru. „Výhled“ hostitele dané fyzické sítě, jako například Ethernetu, je velmi omezující: jedinými hostiteli, se kterými lze komunikovat přímo, jsou hostitelé v dané síti. Ke všem ostatním hostitelům může být přistupováno jen s pomocí tzv. *bran*. Brána je hostitel, který je současně spojen se dvěma nebo více fyzickými sítěmi a který je nastaven pro přenášení paketů mezi nimi.

Aby protokol IP poznal, zda se hostitel nachází v příslušné lokální fyzické síti, musí různé fyzické sítě náležet k odlišným sítím IP. Například síťové číslo **149.76.4.0** je rezervováno pro hostitele lokální sítě katedry matematiky. Když se posílá datagram na adresu **quark**, síťový software na serveru **erdos** okamžitě z IP adresy **149.76.12.4** pozná, že cílový hostitel je připojen k jiné fyzické síti, a proto je dosažitelný pouze pomocí brány (implicitní bránou je **sophus**).

Vlastní brána **sophus** je propojena se dvěma rozdílnými podsítěmi: s katedrou matematiky a s páteří školní sítě. Ke každé z nich přistupuje za pomoci různých rozhraní *eth0*, resp. *fddi0*. Jakou adresu mu ale přidělíme? Máme mu dát adresu podsítě **149.76.1.0** nebo **149.76.4.0**?

Odpověď zní: obě. Při komunikaci s hostitelem v lokální síti katedry matematiky by měla brána **sophus** použít IP adresu **149.76.4.1** a při komunikaci s hostitelem v páteřní síti by měla použít adresu **149.76.1.4**.

Bráně je tedy přidělena jedna IP adresa pro každou síť s níž je spojena. Tyto adresy – společně s odpovídajícími síťovými maskami – jsou spojeny s rozhraním, ke kterému přistupuje podsíť. Mapování rozhraní a adres v rámci brány **sophus** by tedy mělo vypadat následovně:

Rozhraní	Adresa	Síťová maska
Eth0	**149.76.4.1**	**255.255.255.0**
Fddi0	**149.76.1.4**	**255.255.255.0**
Lo	**127.0.0.1**	**255.0.0.0**

Poslední řádek popisuje zpětnovazebné rozhraní *lo*, které bylo popsáno výše.

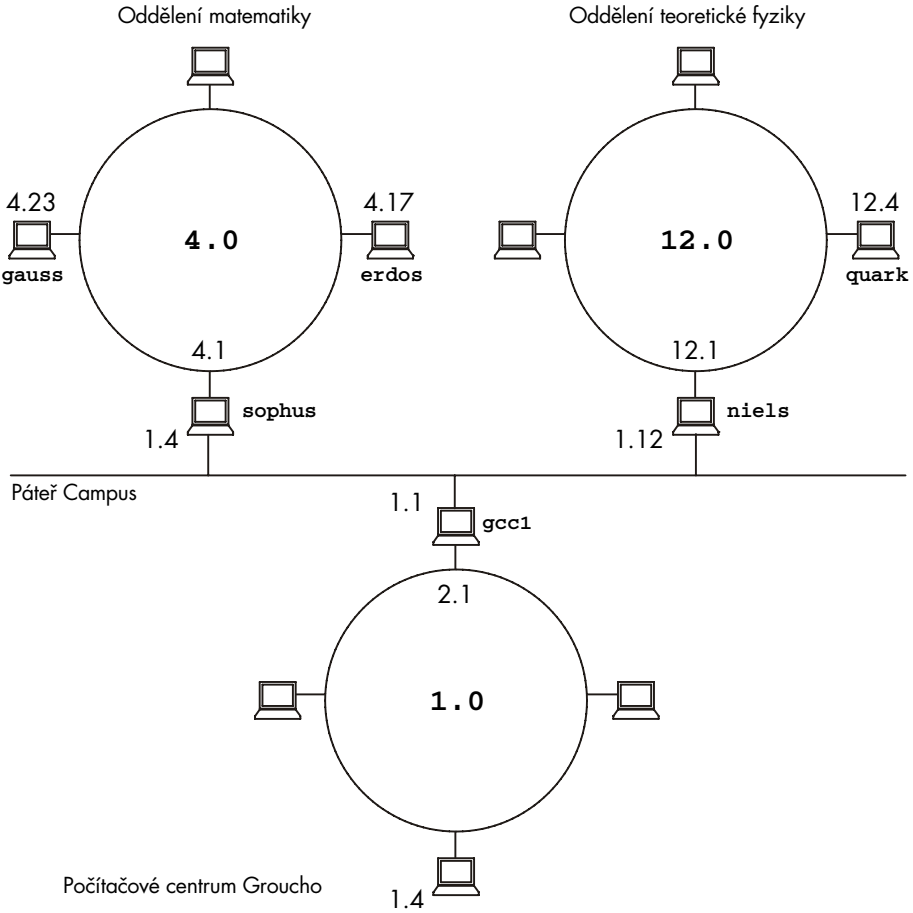
Obrázek 2.2 ukazuje část síťové topologie na Groucho Marx University (GMU). Hostitelé, kteří jsou v daném okamžiku ve dvou podsítích, jsou zobrazeni s oběma adresami.

Jemnou odlišnost mezi přidělením adresy hostiteli nebo jeho rozhraní je možné ignorovat. Na hostitele, kteří se nachází v jedné síti, například hostitel **erdos**, se budete většinou odkazovat pomocí IP adresy, i když třeba tato adresa odpovídá ethernetovému rozhraní. Tento rozdíl je ale důležitý pouze v případě, kdy se odkazujete na bránu.

2.4.4 Směrovací tabulka

Nyní se zaměříme na způsob, jakým protokol IP vybírá bránu, kterou použije k doručení datagramu do vzdálené sítě.

Ukázali jsme si, že u datagramu pro server **quark** zkontroluje hostitel **erdos** cílovou adresu, načež zjistí, že se nenachází v lokální síti. Proto ho pošle na implicitní bránu **sophus**, která je postavena před stejný úkol. Brána **sophus** pozná, že hostitel **quark** není na žádné ze sítí, s nimiž je přímo spojena, takže musí najít další bránu, na kterou by datagram poslala. Správnou volbou by byla brána ***niels***, což je brána katedry fyziky. Proto potřebuje brána **sophus** příslušné informace, aby mohla přiřadit cílovou síť vhodné bráně.



Obrázek 2.2
Část ze síťové topologie Groucho Marx University

Směrovací informace, které protokol IP pro tento účel používá, jsou tvořeny tabulkou, jež přiřazuje jednotlivé sítě branám, s nimiž jsou spojeny. Tabulka musí být doplněna o data typu zachyť-vše (catch all) (pro *implicitní směr*); je to brána sdružená se sítí **0.0.0.0**. Všechny pakety pro neznámou síť jsou poslány implicitním směrem. Pro bránu **sophus** by mohla tabulka vypadat následovně.

Síť	Brána	Rozhraní
149.76.1.0	**-**	**fddi0**
149.76.2.0	**149.76.1.2**	**fddi0**
149.76.3.0	**149.76.1.3**	**fddi0**
149.76.4.0	**-**	**eth0**
149.76.5.0	**149.76.1.5**	**fddi0**
…	**…**	**…**
0.0.0.0	**149.76.1.2**	**fddi0**

V sloupci brána je symbol „-“ proto, že se směruje do sítě, s níž je brána **sophus** přímo spojena, tudíž není žádná brána vyžadována.

Směrovací tabulky mohou být vytvářeny různým způsobem. U malých sítí LAN je většinou nejefektivnější vytvořit je ručně a při procesu zavádění je naplnit IP adresami pomocí příkazu `route` (viz. kapitola 5). V rozsáhlejších sítích jsou vytvářeny a přizpůsobovány v čase spuštění pomocí *směrovacích démonů*; démoni běží na centrálních hostitelích sítě a vyměňují si směrovací informace, čímž určí „optimální“ směrování mezi členskými sítěmi.

V závislosti na velikosti sítě budou použity různé směrovací protokoly. Ke směrování uvnitř autonomních systémů (jako je školní síť Groucho Marx) budou použity *vnitřní směrovací protokoly*. Nejvýznamnějším z nich je protokol RIP, což je směrovací informační protokol (Routing Information Protocol), který je implementován do *BSD-démona* `routed`. Ke směrování mezi autonomními systémy by měly být použity *externí směrovací protokoly*, jako je například EGP (vnější bránový protokol (External Gateway Protocol)) nebo BGP (hraniční bránový protokol (Border Gateway Protocol)); tyto protokoly (stejně tak i RIP) byly implementovány do *démona* `gated`.[3]

2.4.5 Metrické hodnoty

Dynamické směrování založené na protokolu RIP vybírá nejlepší směrování k cílovému hostiteli nebo síti na základě počtu „skoků“ (hops), to znamená počtu bran, kterými musí datagram projít, než dosáhne cíle. Čím kratší je směr, tím rychlejší je RIP. Velmi dlouhé směry s 16-ti nebo více skoky jsou považovány za nepoužitelné a jsou zrušeny.

[3] *Démona* `routed` považuje mnoho lidí za překonaný. Protože *démon* `gated` samozřejmě podporuje i protokol RIP, je lepší ho používat místo *démona* `routed`.

Chcete-li použít protokol RIP na vnitřní správu směrovacích informací ve vaší místní síti, musíte mít na všech hostitelích spuštěn program `gated`. Při procesu zavádění zkontroluje program `gated` všechna aktivní síťová rozhraní. Pokud existuje více než jedno aktivní rozhraní (nepočítaje zpětnovazebné rozhraní), bude předpokládat, že hostitelé posílají pakety mezi několika sítěmi a bude aktivně vyměňovat a vysílat směrovací informace. V opačném případě bude jen pasivně přijímat všechny aktualizace z protokolu RIP a aktualizovat místní směrovací tabulku.

Při vysílání informace z místní směrovací tabulky vypočte program `gated` délku směrování z tzv. *metrické hodnoty*, která je ve směrovací tabulce součástí dat jednotlivých položek. Tato metrická hodnota je při konfiguraci směrování nastavena systémovým správcem a měla by odrážet aktuální váhu použití daného směru. Proto by měla být metrická hodnota směrování k podsíti, se kterou je hostitel přímo propojen, vždy rovna nule, zatímco směr procházející dvěma bránami by měl mít metrickou hodnotu rovnu dvěma. S těmito metrickými hodnotami si však nemusíte dělat starosti, pokud nebudete používat protokol RIP nebo program `gated`.

2.5 Internetový protokol na řízení zpráv

Protokol IP obsahuje doprovodný protokol, o kterém jsme se zatím nezmínili. Jedná se o tzv. *internetový protokol na řízení zpráv* (*Internet Control Message Protocol*) (ICMP), který využívá síťový kód jádra operačního systému k předávání chybových zpráv ostatním hostitelům apod. Předpokládejme například, že jste znovu na hostiteli **erdos** a chcete se za pomoci `telnetu` připojit na port 12 345 serveru **quark**, ale na tomto portu neexistuje žádný odposlech. Pokud dorazí na tomto portu na server **quark** první paket, síťová hladina to pozná a okamžitě vrátí za pomoci protokolu ICMP hostiteli **erdos** zprávu, v níž bude uvedeno „Port je nedostupný“ (Port Unreachable).

Protokol ICMP rozumí poměrně velkému počtu zpráv, z nichž mnohé počítají s chybovými stavy. Mezi nimi existuje jedna velice zajímavá zpráva Přesměrování zprávy (Redirect message). Generuje je směrovací modul v případě, že zjistí, že ho již jako bránu používá jiný hostitel, i když ve skutečnosti existuje i mnohem kratší cesta. Například po restartu systému může být směrovací tabulka brány **sophus** neúplná, obsahujíc směrování na síť katedry matematiky, na páteř FDDI a implicitní směr na centrální bránu Groucho Computing Center (**gcc1**). Proto bude každý paket pro server **quark** poslán na bránu **gcc1** místo aby byl poslán přímo na bránu **niels**, což je brána katedry fyziky. Při přijetí takového datagramu si brána **gcc1** všimne, že jde o špatnou volbu, pošle paket na bránu **niels** a zároveň vrátí bráně **sophus** pomocí protokolu ICMP zprávu o přesměrování, která bude obsahovat výhodnější cestu.

Teď to vypadá, že jde o velmi chytrý způsob, jak se vyhnout manuálnímu nastavování všech směrů s výjimkou těch nejzákladnějších. Ale ne vždy je dobré spoléhat na dynamická směrovací schémata, jako jsou protokoly RIP nebo ICMP se zprávou o přesměrování. Zpráva o přesměrování u protokolu ICMP nebo protokol RIP nabízí jen malou, případně nulovou kontrolu toho, zda je směrovací informace skutečně autentická. Takto mohou zlomyslní uživatelé přerušit dopravu v celé síti nebo případně provést i něco horšího. Proto existují určité verze linuxového síťového kódu, které zacházejí se zprávami o přesměrování, jež ovlivňují směrování sítí, jakoby šlo jen o zprávy o přesměrování hostitelů.

2.6 Systém DNS

2.6.1 Rozlišení názvu hostitele

Jak jsme řekli, je adresování v sítích na bázi protokolu TCP/IP prováděno prostřednictvím 32bitových čísel. Pokud byste si však chtěli zapamatovat o něco více, strávili byste nad nimi poměrně hodně času. Proto se hostitelé označují „běžnými“ názvy, jako je hostitel **gauss** nebo hostitel **strange**. Povinností aplikace je, aby našla IP adresu odpovídající danému názvu. Tento proces se označuje jako *rozlišení názvu hostitele*.

Aplikace, která chce najít IP adresu příslušného názvu hostitele, nemusí obsahovat vlastní rutiny pro hledání hostitelů a jejich IP adres. Místo toho se spoléhá na skupinu funkcí z knihoven, které tuto operaci provádějí transparentně. Funkce se nazývají *gethostbyname(3)* a *gethostbyaddr(3)*. Obvykle jsou tyto funkce společně s mnoha dalšími příbuznými procedurami seskupeny v oddělené knihovně, v tzv. knihovně resolveru; v Linuxu jsou součástí standardní knihovny *libc*. Z toho důvodu se tato sbírka funkcí označuje jako tzv. „*resolver*.“

U malých sítí, jako je Ethernet, nebo dokonce jeho části, je správa tabulek s názvy hostitelů společně s jejich adresami příliš složitá. Tato informace je obvykle uchovávána v souboru s názvem `/etc/hosts`. Při přidávání nebo odebírání hostitele nebo opětovném přidělování adresy stačí na všech hostitelích aktualizovat soubor `hosts`. Tento proces se komplikuje u sítí, které jsou složeny z většího počtu počítačů.

Jedním z řešení tohoto problému je systém NIS, *Síťový informační systém* (*Network Information System*) vyvinutý firmou Sun Microsystems, hovorově se mu také říká YP, nebo-li *Žluté stránky* (*Yellow Pages*). Systém NIS ukládá soubor `hosts` (a další informace) do databáze na hlavním hostiteli, odkud ho mohou klienti dle potřeby získat. Přesto je tento přístup vhodný pouze pro středně velké sítě, jako jsou sítě LAN, protože vyžaduje centrální správu celé databáze `hosts` a její distribuci na všechny servery.

V Internetu byly adresy zprvu uchovávány v jediné databázi nazvané `HOSTS.TXT`. Tento soubor byl spravován síťovým informačním centrem (NIC) a všechny zúčastněné systémy si ho musely stáhnout a nainstalovat. Jak síť rostla, objevilo se v souvislosti s tímto schématem několik problémů. Kromě administrativního přetížení, které bylo způsobeno pravidelnou instalací souboru `HOSTS.TXT`, se příliš zvětšovalo i zatížení serverů, které ho distribuovaly. Mnohem horší ale bylo, že všechny názvy musely být registrovány v centru NIC. To proto, aby se některý název neobjevil dvakrát.

Proto došlo v roce 1984 k adaptaci nového schématu pro rozlišování názvů. Byl jím tzv. *Doménový jmenný systém* (*Domain Name System*). Systém DNS byl navržen Paulem Mockapetrisem a oba tyto problémy řeší současně.

2.6.2 Data DNS

Systém DNS organizuje názvy hostitelů do hierarchie domén. Doména je skupina systémů, mezi nimiž je nějaký vztah – buď proto, že tvoří vlastní síť (například všechny univerzitní počítače nebo všichni hostitelé v síti BITNET), nebo proto, že všechny patří určité organizaci (jako je například vláda Spojených států), nebo zkrátka tvoří geografické celky. Například univerzity jsou seskupeny do domény **edu**, každá univerzita nebo vysoké učení používá svoji *subdoménu*, pod kterou jsou sloučeni její hostitelé. Groucho Marx University má přidělenu doménu **groucho.edu** a síť LAN katedry matematiky má přidělenu subdoménu **maths.groucho.edu**. Hostitelé v síti katedry by měli tento název domény připojen ke svému názvu hostitele; takže hostitel **erdos** bude známý jako hostitel **erdos.maths.groucho.edu**. Tato symbolika je označována jako *plně kvalifikovaný název domény* (*fully qualified domain name*), nebo zkráceně FQDN, a jednoznačně identifikuje tohoto hostitele po celém světě.

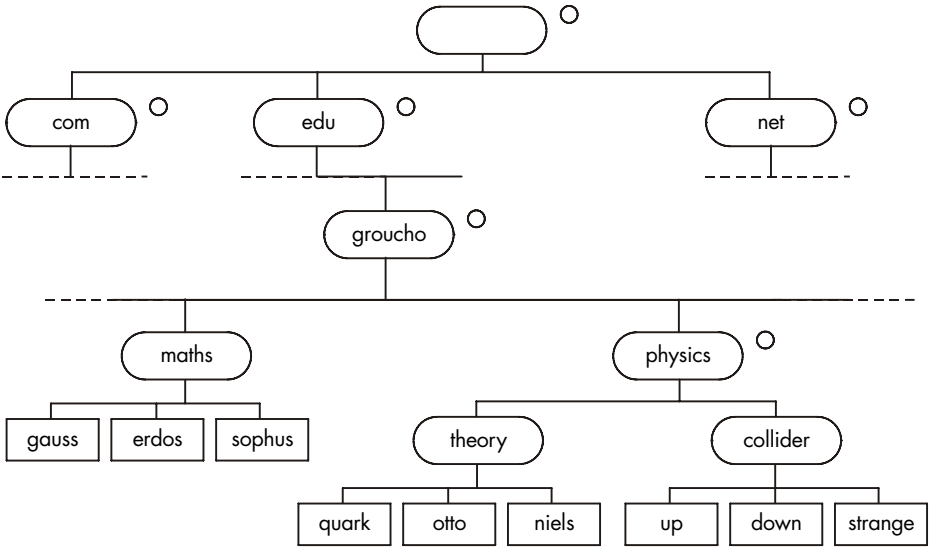
Obrázek 2.3 ukazuje část prostoru s názvy domén. Vstup u kořene tohoto stromu, který je označen malou tečkou, je poměrně výstižně nazván *kořenovou doménou* (*root domain*) a obklopuje všechny další domény. Aby se naznačilo, že název hostitele je plně kvalifikovaným názvem domény a nikoliv relativním názvem nějaké (implicitní) místní domény, píše se někdy s postfixovou tečkou. Ta udává, že název poslední části představuje kořenovou doménu.

V závislosti na jejím umístění v hierarchii názvů může být doména nazvána doménou nejvyšší úrovně, druhé úrovně nebo třetí úrovně. Více úrovní rozdělení sice je možných, ale vyskytují se jen řídce. Následuje několik domén nejvyšší úrovně, s nimiž se můžete často setkat:

edu Vzdělávací instituce (většinou v USA), jako jsou univerzity atd.

com Komerční organizace, společnosti.

org Nekomerční organizace. Často jsou v této doméně soukromé sítě typu UUCP.



Obrázek 2.3
Část prostoru s názvy domén

net Brány a další administrativní hostitelé na síti.

mil Vojenské instituce Spojených států amerických.

gov Vládní instituce Spojených států amerických.

uucp Oficiálně byly do této domény přesunuty názvy všech systémů, které byly dříve používány jako názvy UUCP bez domén.

Z technického hlediska patří první čtyři domény k části Internetu Spojených států amerických, ale i v těchto doménách můžete nalézt neamerické systémy. To platí zejména pro doménu **net**. Avšak domény **mil** a **gov** jsou používány výhradně Spojenými státy americkými.

Obecně platí, že každá země mimo území USA používá vlastní doménu nejvyšší úrovně, která je tvořena dvoupísmenným kódem země dle definice standardu ISO-3166. Například Finsko používá doménu **fi**, doména **fr** je přidělena Francii, doména **de** Německu, doména **au** patří Austrálii atd. Pod touto doménou nejvyšší úrovně může centrum NIC každé země libovolným způsobem organizovat názvy hostitelů. Například Austrálie má doménu druhé úrovně podobnou mezinárodním doménám nejvyšší úrovně, a sice **com.au**, **edu.au** atd. Ostatní země, jako je Německo, tuto speciální úroveň nepožívají, ale využívají raději delší názvy, které odkazují přímo na organizace, jež mají zvláštní doménu. Například není nic výjimečného setkat se s hostitelem, jehož název je například **ftp.informatik.uni-erlangen.de**. Zřejmě to nijak neovlivňuje německou výkonnost.

U těchto národních domén samozřejmě nemusí platit, že hostitel pod příslušnou doménou skutečně leží v dané zemi; pouze to signalizuje, že hostitel byl registrován centrem NIC dané země. Švédský výrobce může mít filiálku v Austrálii a přesto bude mít všechny své hostitele registrovány pod doménou nejvyšší úrovně **se**. Organizováním názvů do hierarchie názvů domén se tedy elegantně vyřeší problém jedinečnosti názvů; v systému DNS musí být název hostitele jedinečný pouze uvnitř vlastní domény, protože ta mu dává název odlišný od ostatních hostitelů po celém světě. Kromě toho jsou plně kvalifikované názvy poměrně lehce zapamatovatelné. Pak je velmi vhodné rozdělit rozsáhlou doménu na několik subdomén.

Ale systém DNS toho umí ještě mnohem více: umožňuje pověřit správou subdomény její správce. Například správci v instituci Groucho Computing Center mohou vytvořit subdoménu pro každý ústav. Již jsme se setkali se subdoménami **maths** a **physics**. Pokud se bude zdát správcům síť katedry fyziky pro správu zvenčí příliš rozsáhlá a chaotická (konec konců fyzici jsou známí jako neovladatelná skupina lidí), mohou jednoduše předat řízení nad doménou **physics.groucho.edu** správcům této sítě. Ti potom mohou používat libovolné názvy hostitelů a přidělovat jim IP-adresy své sítě, aniž by do toho zvenčí někdo zasahoval.

Na konci řetězce je prostor s názvy rozdělen na *zóny* (*zones*), z nichž každá je směrována na doménu. Všimněte si jemné odlišnosti mezi zónou a doménou: *doména* **groucho.edu** obsahuje všechny hostitele instituce Groucho Marx University, zatímco *zóna* **groucho.edu** obsahuje pouze hostitele, které přímo spravuje centrum Computing Center, například hostitele z katedry matematiky. Hostitelé z katedry fyziky patří do odlišné zóny, konkrétně **physics.groucho.edu**. Na obrázku 2.3 je začátek zóny označen malým kolečkem napravo od názvu domény.

2.6.3 Vyhledávání názvů s pomocí DNS

Na první pohled se může zdát, že u všech těchto rozsáhlých domén a zón je provedení rozlišení názvu nesmírně komplikované. Konec konců, neřídí-li názvy, které jsou přidělovány daným hostitelům, žádná centrální správa, jak je má potom aplikace na nižší úrovni uhodnout?

Nyní přichází na řadu bezelstná vlastnost DNS. Chcete-li nalézt IP-adresu serveru **erdos**, pak vám DNS odpoví, že se máte jít zeptat lidí, kteří ho spravují, a oni vám ji řeknou.

Systém DNS je de facto obrovskou distribuovanou databází. Je implementován za pomoci takzvaných jmenných serverů (name servers), které dodávají informace o dané doméně nebo o dané skupině domén. U každé zóny existují nejméně dva a nejvýše několik jmenných serverů, které uchovávají všechny závažné informace o hostitelích v příslušné zóně. Pokud chcete získat IP-adresu serveru **erdos**, pak se stačí spojit s jmenným serverem zóny **groucho.edu**, který vám následně vrátí požadovaná data.

Možná si myslíte, že se o tom mnohem snadněji mluví, ale hůře se to provádí. Takže, jak mám vědět, jak se spojit s jmenným serverem na Groucho Marx University? V případě, že váš počítač není vybaven nástrojem pro analýzu adres, provede to systém DNS za něj. Když chce vaše aplikace vyhledat informace o serveru **erdos**, spojí se s místním jmenným serverem, který aplikuje na dané informace tzv. iterační dotaz. Začne zasláním dotazu jmennému serveru v kořenové doméně, kde se zeptá na adresu **erdos.maths.groucho.edu**. Kořenový jmenný server pozná, že tento název nepatří do jeho správní zóny, ale patří do zóny pod doménou **edu**. Takže vám sdělí, že se máte spojit s jmenným serverem zóny **edu**, kde získáte více informací. Ke své odpovědi dále přibalí i seznam všech jmenných serverů domény **edu** společně s jejich adresami. Potom bude váš místní jmenný server pokračovat a pošle dotaz na některý z nich, například na **a.isi.edu**. Podobným způsobem jako u kořenového jmenného serveru se server **a.isi.edu** dozví, že lidé z **groucho.edu** mají svou vlastní zónu a odkáže vás na její vlastní servery. Místní jmenný server bude pokračovat v dotazu na server **erdos** na jednom z těchto serverů, který konečně zpozná, že daný server patří do jeho zóny, a vrátí jeho odpovídající IP-adresu.

Teď to možná vypadá, že kvůli vyhledání jedné mizerné IP-adresy bylo zapotřebí provést spoustu operací, ale ve skutečnosti je to v porovnání s množstvím dat, které by musely být přeneseny, kdyby se stále pracovalo se souborem `HOSTS.TXT`, minimum. Stále však existuje prostor, jak toto schéma vylepšit.

Aby zkrátil dobu odpovědi při dalších dotazech, uchovává jmenný server získané informace ve své místní *vyrovnávací paměti (cache)*. Takže když chce příště někdo z vaší lokální sítě vyhledat adresu hostitele v doméně **groucho.edu**, nemusí jmenný server znovu absolvovat celý výše popsaný proces, ale přímo se spojí s jmenným serverem pro doménu **groucho.edu**.[4]

Samozřejmě, že server nebude tyto informace uchovávat donekonečna, ale po určité době je smaže. Tato doba platnosti se nazývá *čas přežití* (*time to live*), zkráceně TTL. V databázi DNS přiděluje každé položce TTL správce, který je zodpovědný za danou zónu.

2.6.4 Servery systému DNS

Jmenné servery, které uchovávají všechny informace o hostitelích příslušné zóny, se nazývají *autoritativními jmennými servery* (*authoritative*) dané zóny a někdy jsou také označovány jako *hlavní jmenné servery* (*master name servers*). Jakýkoliv dotaz na hostitele uvnitř této zóny nakonec stejně skončí až u těchto hlavních jmenných serverů.

[4] Pokud tak neučiní, je systém DNS stejně špatný, jako ostatní metody, protože každý dotaz bude vyžadovat zapojení kořenových jmenných serverů.

Aby byl zachován spojitý obraz zóny, musí být její hlavní servery poměrně dobře synchronizovány. Toho dosáhneme tak, že jednoho ze serverů označíme jako *primární* (*primary*). Ten bude načítat informace o své zóně z datových souborů a ostatní servery budou označeny jako *sekundární* (*secondary*) a budou v pravidelných intervalech přenášet data z primárního serveru.

Jedním z důvodů, proč bychom měli mít několik jmenných serverů, je rozložení pracovní zátěže, dalším důvodem je pak redundance. Když přestane některý z jmenných serverů správně pracovat, například když spadne nebo ztratí síťové spojení, přejdou všechny dotazy na ostatní servery. Samozřejmě vás toto schéma neochrání před chybami serveru, jejichž důsledkem budou špatné odpovědi na všechny požadavky systému DNS, například z důvodu softwarových chyb ve vlastním programu serveru.

Samozřejmě můžete chtít provozovat i takový jmenný server, který nebude správcem žádné domény[5]. Přesto však je tento typ serveru důležitý, protože je stále schopen provádět DNS-dotazy pro aplikace, které jsou spuštěny v lokální síti, a dále může ukládat informace do vyrovnávací paměti. Proto se tento typ serveru nazývá server *pouze pro cachování* (*caching-only* server).

2.6.5 Databáze systému DNS

Ukázali jsme si, že systém DNS nejenom určuje IP-adresy hostitelů, ale také vyměňuje informace mezi jmennými servery. Ve skutečnosti může databáze DNS obsahovat celý svazek různých typů záznamů.

V databázi DNS se jednotlivým informacím říká *zdrojový záznam* (*resource record*), zkráceně RR. Každý záznam má přidělený typ, který popisuje druh dat, jež reprezentuje a třídu, určující typ sítě, na kterou se bude záznam vztahovat. Třída uspokojuje potřeby různých adresních schémat, jako jsou IP-adresy (třída IN) nebo adresy sítí Hesiod (používaných v MIT) a některé další. Vzorovým typem zdrojového záznamu je záznam A, který sdružuje plně kvalifikované názvy domény s IP-adresou. Samozřejmě, že hostitel může mít více než jeden název. Nicméně jeden z těchto názvů musí být označen jako oficiální, neboli *kanonický název hostitele* (*canonical host name*) a ostatní jsou chápány jako přezdívky oficiálního názvu. Rozdíl spočívá v tom, že kanonický název hostitele je takový, se kterým je sdružen záznam typu A, zatímco ostatní názvy mají pouze záznam typu CNAME, který odkazuje na kanonický název hostitele.

Nebudeme zde procházet všechny typy záznamů, to si schováme do příští kapitoly. Nyní vám raději poskytneme krátký příklad. Obrázek 2.4 ukazuje část databázové domény, kterou mají načtenu jmenné servery zóny **physics.groucho.edu**.

5 Většinou je to v pořádku. Jmenný server by měl přinejmenším poskytovat jmenné služby pro **místního hostitele** a vracet vyhledávání adresy **127.0.0.1**.

```
;
; Autoritativní informace o doméně physics.groucho.edu
@                        IN     SOA            {
                        niels.physics.groucho.edu.
                        hostmaster.niels.physics.groucho.edu.
                        1034                ; serial no
                        360000              ; refresh
                        3600                ; retry
                        3600000             ; expire
                        3600                ; default ttl
                      }
;
; Jmenné servery
                            IN     NS          niels
                            IN     NS          gauss.maths.groucho.edu.
gauss.maths.groucho.edu.    IN A               149.76.4.23
;
; Teoretická fyzika (podsíť 12)
niels                       IN     A           149.76.12.1
                            IN     A           149.76.1.12
nameserver                  IN     CNAME       niels
otto                        IN     A           149.76.12.2
quark                       IN     A           149.76.12.4
down                        IN     A           149.76.12.5
strange                     IN     A           149.76.12.6
...
; Collider Lab. (podsíť 14)
boson                       IN     A           149.76.14.1
muon                        IN     A           149.76.14.7
bogon                       IN     A           149.76.14.12
...
```

Obrázek 2.4
Výpis ze souboru `named.hosts` pro katedru fyziky

Kromě záznamů typů A a CNAME můžete v horní části souboru vidět speciální záznam, který je roztažen na několika řádcích. Je to zdrojový záznam SOA, což označuje *začátek správy* (*Start of Authority*), kde jsou umístěny všeobecné informace o zóně, kterou daný server spravuje. Je v něm například obsažen implicitní čas přežití pro všechny záznamy.

Všimněte si, že všechny názvy v ukázkovém souboru, které nekončí tečkou, by měly být interpretovány vzhledem k doméně **groucho.edu**. Speciální název „@“ použitý v záznamu typu SOA odkazuje na vlastní název domény.

Řekli jsme si, že jmenné servery domény **groucho.edu** musí nějakým způsobem vědět o zóně **physics**, aby mohly odkazovat dotazy na její jmenné servery. Toho se obvykle dosáhne pomocí dvojice záznamů: záznam typu NS, který poskytne FQDN název serveru, a záznam typu A, který tomuto názvu přidruží adresu. Protože právě tyto záznamy drží jmenný prostor pohromadě, označují se často jako tzv. *tmelící záznamy* (*glue records*). Jsou jedinými případy záznamů, kde rodičovská zóna skutečně uchovává informace o hostitelích podřazené zóny. Na obrázku 2.5 ukazují tmelící záznamy na jmenné servery **physics.groucho.edu**.

```
;
; Data zóny groucho.edu.
@                    IN        SOA           {
                      vax12.gcc.groucho.edu.
                      hostmaster.vax12.gcc.groucho.edu.
                      233               ; serial no
                      360000            ; refresh
                      3600              ; retry
                      3600000           ; expire
                      3600              ; default ttl
                     }
....
;
; Tmelící záznamy pro doménu physics.groucho.edu
physics              IN      NS         niels.physics.groucho.edu.
                     IN      NS         gauss.maths.groucho.edu.
niels.physics        IN      A          149.76.12.1
gauss.maths          IN      A          149.76.4.23
...
```

Obrázek 2.5
Výpis ze souboru `named.hosts` pro GMU

2.6.6 Zpětné vyhledávání

Kromě vyhledávání IP-adresy, která náleží hostiteli, je někdy žádoucí vyhledat také název kanonického hostitele, který odpovídá příslušné adrese. Tento proces se označuje jako tzv. *zpětné mapování* (*reverse mapping*) a některé síťové služby ho používají k ověřování identity klienta. Pokud se používá jediný soubor `hosts`, pak si zpětné vyhledávání vyžádá vyhledání souboru hostitele, který vlastní požadovanou IP-adresu. U DNS samozřejmě nepřipadá v úvahu úplné prohledání jmenného prostoru. Místo toho byla vytvořena speciální doména **in-addr.arpa**, která obsahuje všechny IP-adresy všech hostitelů v převrácené tečkové notaci. Například IP-adrese **149.76.12.4** odpovídá název **4.12.76.149.in-addr.arpa**. Záznam typu PTR je takový záznam, který tyto názvy sdružuje s názvy svých kanonických hostitelů.

Vytvoření správní zóny obvykle znamená, že její správci mají plnou kontrolu nad způsobem, jakým jsou adresy přiřazovány k názvům. Protože většinou obsluhují jednu nebo více sítí nebo podsítí IP, existuje mezi DNS-zónami a IP-sítěmi jedno či více mapování. Například katedra fyziky obsahuje podsítě **149.76.8.0**, **149.76.12.0** a **149.76.14.0**.

V důsledku toho musí být nové zóny v doméně **in-addr.arpa** vytvářeny společně se zónou **physics** a musí být postoupeny síťovým správcům kateder: **8.76.149.in-addr.arpa**, **12.76.149.in-addr.arpa** a **14.76.149.in-addr.arpa**. Jinak by instalace nového hostitele v Collider Lab vyžadovala, aby se správci spojili se svou nadřazenou doménou, kde se nová adresa zapíše do souboru zóny **in-addr.arpa**.

Na obrázku 2.6 je zobrazena databáze zóny pro podsíť 12. Odpovídající tmelící záznamy v databázi jejich rodičovské zóny vidíte na obrázku 2.7.

```
;
; doména 12.76.149.in-addr.arpa.
@                   IN      SOA    {
                    niels.physics.groucho.edu.
                    hostmaster.niels.physics.groucho.edu.
                    233 360000 3600 3600000 3600
                  }
2                   IN      PTR        otto.physics.groucho.edu.
4                   IN      PTR        quark.physics.groucho.edu.
5                   IN      PTR        down.physics.groucho.edu.
6                   IN      PTR        strange.physics.groucho.edu.
```

Obrázek 2.6
Výtah ze souboru `named.rev` pro podsíť 12

V důsledku toho mohou být zóny vytvářeny pouze jako nadmnožiny sítí IP-a co je ještě krutější, musí mít síťové masky na hranicích bajtu. Všechny podsítě v Groucho Marx University mají síťovou masku **255.255.255.0**, takže lze pro každou podsíť vytvořit zónu **in-addr.arpa**. Pokud by však existovala síťová maska **255.255.255.128**, nebylo by možné vytvořit zónu pro podsíť **149.76.12.128**, protože by neexistoval žádný způsob, jak sdělit systému DNS, že doména **12.76.149.in-addr.arpa** byla rozdělena na dvě zóny se samostatnou správou a s názvy hostitelů od **1** do **127**, resp. od **128** do **255**.

```
;
; the 76.149.in-addr.arpa domain.
@                   IN        SOA           {
                    vax12.gcc.groucho.edu.
                    hostmaster.vax12.gcc.groucho.edu.
                    233 360000 3600 3600000 3600
                    }
...
; podsíť 4: Katedra matematiky
1.4                 IN      PTR       sophus.maths.groucho.edu.
17.4                IN      PTR       erdos.maths.groucho.edu.
23.4                IN      PTR       gauss.maths.groucho.edu.
...
; podsíť 12: Katedra fyziky
12                  IN      NS        niels.physics.groucho.edu.
                    IN      NS        gauss.maths.groucho.edu.
niels.physics.groucho.edu. IN  A 149.76.12.1
gauss.maths.groucho.edu. IN  A   149.76.4.23
...
```

Obrázek 2.7
Výtah ze souboru `named.rev` pro síť **149.76**

Kapitola

3

Konfigurace síťového hardwaru

3.1 Zařízení, ovladače atd.

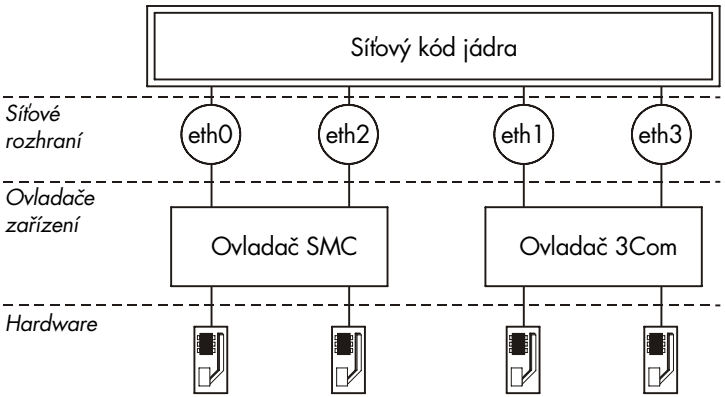
Až dosud jsme se bavili o síťových rozhraních a o všeobecných problémech protokolu TCP/IP, ale zatím jsme si neřekli, *co* se doopravdy děje, když „síťový kód“ jádra přistupuje k části hardwaru. Nyní si řekneme něco o koncepci rozhraní a ovladačů.

Nejprve máte samozřejmě vlastní hardware, například ethernetovou kartu: to je laminátová deska přecpaná spoustou mrňavých čipů, na kterých lze najít nějaká čísílka. Toto vše je umístěno ve slotu vašeho počítače a všeobecně se tomu říká zařízení.

Abyste vůbec mohli používat ethernetovou kartu, musí být v jádru Linuxu k dispozici speciální funkce, které rozumí způsobu, jakým je k tomuto zařízení přistupováno. Takovýmto funkcím říkáme ovladače zařízení. Například Linux obsahuje ovladače zařízení pro několik skupin ethernetových karet, které si jsou z funkčního hlediska hodně podobné. Jsou nazvány „Becker Series Drivers“ po svém autorovi Donaldu Beckerovi. Jiným příkladem je ovladač D-Link, který obsluhuje kapesní adaptér D-Link, jenž je připojen k paralelnímu portu.

Ale co myslíme tím, když říkáme, že ovladač se „stará“ o zařízení? Vraťme se zpět k ethernetové kartě, kterou jsme si popsali na začátku. Ovladač musí být nějakým způsobem schopen komunikovat s logikou hardwarové karty: musí jí posílat příkazy a data, a ta by mu měla na oplátku doručit všechna přijatá data.

U počítačů PC se tato komunikace odehrává v oblasti V/V paměti, která je namapována na registry karty. Všechny příkazy a data vyslané jádrem do karty musí těmito registry projít. V/V paměť je obecně určena zadáním počáteční, neboli *základní adresy* (*base address*). Typické základní adresy pro ethernetové karty jsou **0x300** nebo **0x360**.



Obrázek 3.1
Vztahy mezi ovladači, rozhraními a hardwarem

Obvykle není třeba si dělat příliš starosti s konfigurací hardwaru, jako je základní adresa, protože jádro se při zavádění pokusí detekovat pozici karty. Tento proces se nazývá automatické detekce, což znamená, že jádro přečte několik adresových míst a porovná přečtená data s tím, co by získal, kdyby na nich byla nainstalována určitá ethernetová karta. Mohou však existovat ethernetové karty, které automatickou detekci neumí; to bývá případ levných ethernetových karet, které nejsou dokonalými klony standardních karet jiných výrobců. Jádro systému také bude při zavádění zkoušet detekovat pouze jediné ethernetové zařízení. Pokud používáte více než jednu kartu, musíte jádru o této kartě explicitně říci.

Dalším takovým parametrem, který musíte jádru předat, je požadavek kanálu přerušení (interrupt request channel). Pokud potřebují hardwarové komponenty něco zařídit, obvykle předají jádru přerušení. Příkladem může býti obdržení dat nebo výskyt speciálních událostí. U počítačů PC se mohou přerušení vyskytnout na jednom z 15 kanálů přerušení, které jsou očíslovány 0, 1 a 3 až 15. Číslo přerušení, které je hardwarové komponentě přiděleno, se nazývá *číslo požadavku přerušení* (*interrupt request number*), zkráceně IRQ.[1]

V kapitole 2 jsme si řekli, že jádro přistupuje k zařízení pomocí tzv. rozhraní. Ta poskytují abstraktní množinu funkcí, jako je například posílání nebo příjem datagramu, které jsou pro všechny typy hardwaru totožné.

Rozhraní jsou identifikována prostřednictvím svého názvu. Tyto názvy jsou interně definovány v jádru, nejedná se však o soubory zařízení v adresáři `/dev`. Typickými názvy ethernetových rozhraní jsou *eth0*, *eth1* atd. Přiřazení rozhraní k jednotlivým zařízením obvykle závisí

[1] IRQ 2 a 9 jsou totožné, protože PC obsahuje dva procesory přerušení seřazené za sebou, z nichž každý má osm IRQ; druhý procesor je připojen na IRQ 2 prvního procesoru.

na pořadí, ve kterém jsou zařízení nakonfigurována; například první nainstalovaná ethernetová karta obdrží název *eth0*, další bude *eth1* atd. Jedinou výjimku z tohoto pravidla tvoří rozhraní SLIP, která jsou přidělována dynamicky; to znamená, kdykoliv je uskutečněno spojení SLIP, je rozhraní přiřazeno sériovému portu.

Schéma uvedené na obrázku 3.1 se pokouší ukázat vztahy mezi hardwarem, ovladači zařízení a rozhraními. Při zavádění zobrazí jádro detekovaná zařízení a rozhraní, která se mu podařilo nainstalovat. Následuje výpis typické obrazovky při zavádění systému:

```
.
.
This processor honours the WP bit even when in supervisor mode. Good.
Floppy drive(s): fd0 is 1.44M
Swansea University Computer Society NET3.010
IP Protocols: ICMP, UDP, TCP
PPP: version 0.2.1 (4 channels) OPTIMIZE_FLAGS
TCP compression code copyright 1989 Regents of the University
  of California
PPP line discipline registered.
SLIP: version 0.7.5 (4 channels)
CSLIP: code copyright 1989 Regents of the University of California
dl0: D-Link DE-600 pocket adapter, Ethernet Address: 00:80:C8:71:76:95
Checking 386/387 coupling...OK, fpu using exception 16 error reporting.
Linux version 1.1.11 (okir@monad) #3 Sat May 7 14:57:18 MET DST 1994
```

Výpis ukazuje, že jádro bylo zkompilováno se zapnutým protokolem TCP/IP, včetně ovladačů pro protokoly SLIP, CSLIP a PPP. Třetí řádka odspodu uvádí, že byl detekován kapesní adaptér D-Link a nainstalován jako rozhraní *dl0*. Máte-li jiný typ ethernetové karty, vypíše jádro zpravidla řádek začínající *eth0*, za nímž následuje typ detekované karty. Pokud máte nainstalovanou ethernetovou kartu, ale nevidíte žádnou takovou zprávu, znamená to, že jádro není schopno správně detekovat vaši kartu. Tímto problémem se budeme zabývat v další části.*

* Poznámka korektora: V současné době jsou podporována i tzv. modulární jádra, kdy kromě vlastního jádra existují ještě moduly, které mohou být v průběhu činnosti jádra „nataženy“ a používány stejně jako ovladače přímo v jádru.

3.2 Konfigurace kernelu (jádra)

Většina distributorů Linuxu dodává zaváděcí diskety, které pracují se všemi běžnými typy hardwaru počítačů PC. To znamená, že jádro na těchto disketách je zkompilováno se všemi skupinami ovladačů, které však nikdy nebudete potřebovat všechny a které plýtvají drahocennou pamětí, protože části jádra nelze z paměti zrušit. Proto si obvykle sestavíte své vlastní jádro, jež bude obsahovat pouze ovladače, které skutečně chcete nebo potřebujete.

Chcete-li používat systém Linux, měli byste ovládat sestavování jádra. Základy jsou vysvětleny v manuálu Matta Welshe „Installation and Getting Started“, který je součástí série Linux – dokumentační projekt. Proto se v této stati zmíníme pouze o těch konfiguračních volbách, které ovlivňují nastavení sítí.

Při spuštění příkazu `make config` budete nejprve dotazováni na všeobecné konfigurace, například zda chcete či nechcete jádro s emulací matematického koprocesoru atd. Jeden z těchto dotazů se bude týkat podpory pro sítě na bázi TCP/IP. Aby bylo vaše jádro schopno pracovat se sítěmi, musíte na tento dotaz odpovědět **y** (yes - ano).

3.2.1 Volby jádra Linuxu verze 1.0 a vyšší

Jakmile je dokončena obecná část voleb, bude konfigurace pokračovat dotazy na různé rysy systému, jako jsou ovladače SCSI apod. Následující seznam otázek se pak bude týkat podpory sítí. Přesná množina konfiguračních voleb se z důvodu probíhajícího vývoje stále mění. Typický seznam voleb, který nabízí většina verzí jádra mezi 1.0 a 1.1, vypadá následovně:

```
*
* Network device support
*
Network device support? (CONFIG_ETHERCARDS) [y]
```

Pokud chcete používat *nějaký* typ síťových zařízení, ať se už jedná o protokol Ethernet, SLIP nebo PPP, musíte na tuto otázku odpovědět **y** (bez ohledu na název makra, který je uveden v kulatých závorkách). Pokud na tuto otázku odpovíte kladně, bude automaticky povolena podpora ethernetových typů zařízení. Podpora ostatních typů síťových ovladačů musí být povolena samostatně:

```
SLIP (serial line) support? (CONFIG_SLIP) [y]
SLIP compressed headers (SL_COMPRESSED) [y]
PPP (point-to-point) support (CONFIG_PPP) [y]
PLIP (parallel port) support (CONFIG_PLIP) [n]
```

Tyto otázky se týkají spojení pomocí různých protokolů, které Linux podporuje. Protokol SLIP vám umožní posílat IP-datagramy po sériové lince. Možnost komprimovat hlavičku poskytuje podporu pro protokol CSLIP. Tato technika zkomprimuje hlavičky TCP/IP až na tři bajty. Všimněte si, že tato volba jádra nezapíná automaticky protokol CSLIP, pouze mu zprostředkovává nezbytné funkce jádra.

PPP je dalším protokolem, který umožňuje posílání síťových dat po sériové lince. Je mnohem pružnější, než protokol SLIP a není omezen jen na protokol IP, ale podporuje i protokol IPX, je-li tento nainstalován.

PLIP nabízí způsob, jak posílat IP-datagramy za pomoci spojení přes paralelní port. Většinou se používá ke komunikaci s počítači PC, na kterých běží operační systém DOS.

Následující otázky se budou zabývat ethernetovými kartami od různých výrobců. Protože se stále vyvíjí nové ovladače, může být tato část rozšířena o další otázky. Pokud chcete sestavit jádro, které budete používat na více strojích, měli byste povolit více než jeden ovladač.

```
NE2000/NE1000 support (CONFIG_NE2000) [y]
WD80*3 support (CONFIG_WD80x3) [n]
SMC Ultra support (CONFIG_ULTRA) [n]
3c501 support (CONFIG_EL1) [n]
3c503 support (CONFIG_EL2) [n]
3c509/3c579 support (CONFIG_EL3) [n]
HP PCLAN support (CONFIG_HPLAN) [n]
AT1500 and NE2100 (LANCE and PCnet-ISA) support (CONFIG_LANCE) [n]
AT1700 support (CONFIG_AT1700) [n]
DEPCA support (CONFIG_DEPCA) [n]
D-Link DE600 pocket adaptor support (CONFIG_DE600) [y]
AT-LAN-TEC/RealTek pocket adaptor support (CONFIG_ATP) [n]
*
* CD-ROM drivers
*
...
```

Nakonec se vás konfigurační script v části věnované systémům souborů zeptá, zda chcete podporu pro NFS, síťový souborový systém (network filesystem). NFS umožňuje exportovat souborové systémy na několik hostitelů, takže se vám bude zdát, že jsou soubory uloženy na běžném pevném disku, který je připojen k danému hostiteli.

```
NFS filesystem support (CONFIG_NFS_FS) [y]
```

3.2.2 Volby jádra Linuxu verze 1.1.14 a vyšší

Od verze 1.1.14, do které byla přidána alfa-verze protokolu IPX, se lehce změnila konfigurační procedura. Část s obecnými volbami se nyní zeptá, zda si přejete nějakou podporu sítí. Pak ihned následuje skupina dotazů týkající se síťových voleb.

```
*
* Networking options
*
TCP/IP networking (CONFIG_INET) [y]
```

Aby bylo možné používat sítě na bázi TCP/IP, musíte na tuto otázku odpovědět **y**. Pokud však odpovíte **n**, i nadále bude možné sestavit jádro s podporou protokolu IPX.

```
IP forwarding/gatewaying (CONFIG_IP_FORWARD) [n]
```

Tuto volbu byste měli povolit v případě, že váš systém bude vystupovat jako brána mezi dvěma Ethernety, nebo mezi Ethernetem a spojením SLIP atd. I když příliš nevadí, když tuto volbu implicitně povolíte, budete možná chtít v případě konfigurace hostitele jako tzv. firewall tuto volbu zakázat. Firewally jsou hostitelé, kteří jsou připojeni ke dvěma nebo více sítím, avšak nesměrují mezi nimi síťový provoz. Obecně poskytují uživatelům firemní sítě přístup k Internetu, a to s minimálními riziky pro vnitřní síť. Uživatelé se budou moci přihlašovat k firewallu a používat internetové služby, ale počítače ve firmě budou chráněny před vnějšími útoky, protože žádné příchozí spojení nemůže přes firewall přejít.

```
*
* (it is safe to leave these untouched)
*
PC/TCP compatibility mode (CONFIG_INET_PCTCP) [n]
```

Za pomoci této volby zabráníte nekompatibilitě s některými verzemi PC/TCP, což je komerční implementace TCP/IP u počítačů PC s operačním systémem DOS. Pokud tuto volbu povolíte, budete stále moci komunikovat s běžnými unixovými stroji, avšak u pomalých linek může dojít ke ztrátě výkonu.

```
Reverse ARP (CONFIG_INET_RARP) [n]
```

Tato funkce povolí RARP, protokol pro zpětné rozlišení adres. Protokol RARP je používán u bezdiskových klientů a X-terminálů, kde při zavádění slouží ke zjištění IP-adresy. Protokol RARP byste měli povolit jen v případě, že hodláte provozovat tento typ klientů. Poslední balík síťových utilit (*net-0.32d*) obsahuje malou utilitu s názvem *rarp*, která umožňuje přidávat systémy do vyrovnávací paměti RARP.

```
Assume subnets are local (CONFIG_INET_SNARL) [y]
```

Posíláte-li data přes TCP, musí je jádro před předáním protokolu IP rozdělit do několika IP-paketů. U hostitelů, kteří jsou dosažitelní po místní síti, jako například Ethernet, se použijí větší pakety než u hostitelů, pro jejichž dosažení musí urazit dlouhou cestu.[2] Pokud nepovolíte *SNARL*, bude jádro předpokládat, že lokální sítě jsou pouze takové, se kterými má skutečné rozhraní. Pokud se však podíváte na síť třídy B v Groucho Marx University, uvidíte, že celá síť třídy B je lokální, avšak většina hostitelů má rozhraní pouze s jednou nebo se dvěma podsítěmi. Jestliže *SNARL* povolíte, bude jádro předpokládat, že *všechny* podsítě jsou lokální a pro komunikaci se všemi hostiteli v rámci univerzity použije velké pakety.

Pokud chcete pro data, která jsou posílána konkrétním hostitelům, používat pakety o malé velikosti (protože například data putují po spojení SLIP), můžete tak učinit za pomoci volby *mtu* příkazu *route*; ta je stručně probrána na konci této kapitoly.

```
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
```

Pravidlo NAGLE je heuristické a zabraňuje posílání zvláště malých IP-paketů, někdy nazývaných minigramy (tinygrams). Minigramy jsou obvykle tvořeny nástroji pro interaktivní práci se sítí, které přenášejí jednotlivé stisky kláves, jako například *telnet* nebo *rsh*. Minigramy jsou však mimořádně nehospodárné u spojení s malou šířkou pásma, například pomocí protokolu SLIP. Algoritmus NAGLE se tomu může pokusit zabránit tak, že za určitých okolností na chvíli pozdrží vyslání dat za pomoci protokolu TCP. Máte-li vážné problémy se ztracenými pakety, pak můžete algoritmus NAGLE zakázat.

```
The IPX protocol (CONFIG_IPX) [n]
```

Tato volba povolí protokol IPX, což je transportní protokol, který používají sítě Novell. Jednou z výhod může být možnost výměny dat s utilitami DOS, které využívají protokol IPX, nebo směrování dopravy mezi vašimi sítěmi Novell prostřednictvím spojení PPP.

Od verze jádra 1.1.16 podporuje Linux ještě jeden typ ovladače, tzv. fiktivní ovladač. Následující otázka se týká počátku sekce, ve které se nastavují ovladače zařízení.

```
Dummy net driver support (CONFIG_DUMMY) [y]
```

Fiktivní ovladač toho příliš mnoho neumí, ale je poměrně užitečný u samostatných hostitelů nebo hostitelů SLIP. V podstatě jde o maskující zpětnovazebné rozhraní. Důvodem pro tento druh rozhraní, jsou hostitelé, kteří poskytují SLIP, ale nemají žádný Ethernet. Přesto však potřebujete rozhraní, které bude mít po celou dobu přiřazenu IP-adresu. Podrobněji je tento problém probírán ve stati Fiktivní rozhraní v kapitole 5.

---

2 To z důvodu, abychom se vyhnuli fragmentaci u těch spojení, u kterých lze nastavit maximální velikost paketu na velmi malou hodnotu.

3.3 Průvodce linuxovými síťovými zařízeními

Jádro Linuxu podporuje množství ovladačů hardwaru pro různé typy zařízení. Tato stať obsahuje krátký přehled dostupných rodin ovladačů a názvů rozhraní, které ovladače používají.

V Linuxu existuje množství standardních názvů rozhraní, jejichž výčet následuje. Většina ovladačů podporuje více než jedno rozhraní. V takovém případě jsou rozhraní číslována,např. *eth0*, *eth1* atd.

lo Lokální zpětnovazebné rozhraní. Je používáno za účelem testování, stejně jako dvojice síťových aplikací. Pracuje jako uzavřený obvod, kde všechny datagramy, které jsou na něj poslány, jsou okamžitě. V jádru existuje vždy jedno zpětnovazebné rozhraní. Menší nebo větší počet těchto rozhraní nemá smysl.

ethn Ethernetová karta číslo *n*. To je obecný název rozhraní pro většinu ethernetových karet.

dln Tato rozhraní přistupují ke kapesnímu adaptéru D-Link DE-600, což je další ethernetové zařízení. Jeho specialita spočívá v tom, že je řízeno přes paralelní port.

sln Rozhraní SLIP číslo *n*. Rozhraní SLIP jsou přidělena sériovým linkám v takovém pořadí, v jakém jsou sériové linky přiřazovány protokolu SLIP; například první sériová linka nakonfigurovaná pro protokol SLIP bude *sl0* atd. Jádro podporuje až čtyři rozhraní SLIP.

pppn Rozhraní PPP číslo *n*. Stejně jako rozhraní SLIP je i rozhraní PPP přiděleno sériové lince v okamžiku, kdy je sériová linka přepnuta do režimu PPP. V současné době jsou podporována až čtyři rozhraní.

plipn Rozhraní PLIP číslo *n*. Protokol PLIP dopravuje datagramy po paralelních linkách. Jsou podporována až tři rozhraní PLIP. Tato rozhraní jsou přidělována ovladačem PLIP při procesu zavádění systému a jsou mapována na paralelní porty.

Pro ostatní ovladače rozhraní, které budou připsány později, například ISDN nebo AX.25, budou zavedeny nové názvy.

V následujících odstavcích si popíšeme použití výše popsaných ovladačů.

3.4 Instalace Ethernetu

Aktuální síťový kód Linuxu podporuje různé skupiny ethernetových karet. Většinu ovladačů napsal Donaldem Becker (**becker@cesdis.gsfc.nasa.gov**), který je autorem rodiny ovladačů karet založených na čipu National Semiconductor 8390; tato skupina se proslavila pod názvem Becker Series Drivers. Dále existují ovladače pro několik produktů od firmy D-Link, mezi ně patří i kapesní adaptér D-Link, který umožňuje přistupovat k Ethernetu přes paralelní port. Tento ovladač napsal Bjřrn Ekwall (**bj0rn@blox.se**). Ovladač DEPCA vytvořil Davide C. Davies (**davies@wanton.lkg.dec.com**).*

3.4.1 Kabeláž Ethernetu

Pokud instalujete Ethernet poprvé v životě, pak se vám možná bude hodit krátký úvod do kabeláže. Ethernet je velmi vybíravý na správné kabely. Kabel musí být na obou koncích ukončen rezistorem o odporu 50 Ohmů a nesmí mít žádné větvení (například tři kabely spojené do hvězdy). Pokud používáte tenký koaxiální kabel s BNC-konektory tvaru T, musí být tyto konektory zacvaknuty přímo do konektoru karty; neměli byste vřazovat žádný kabelový segment.

Pokud používáte tlusté kabely, musíte svého hostitele připojit přes tzv. transceiver (někdy se také používá označení jednotka pro připojení Ethernetu). Transceiver můžete zapojit do 15pólového portu AUI, který najdete na vaší kartě. Alternativou je použití stíněného kabelu.

3.4.2 Podporované karty

Kompletní seznam podporovaných karet je k dispozici v dokumentu Ethernet HOWTO, jehož autorem je Paul Gotmaker.

Následuje seznam známějších karet podporovaných systémem Linux. Aktuální seznam, který najdete v dokumentu HOWTO, je asi třikrát delší. I když ale najdete v tomto seznamu svou kartu, podívejte se nejprve do dokumentu HOWTO. Někdy jsou tam totiž důležité informace o tom, jak tyto karty úspěšně zprovoznit. To se týká zejména některých ethernetových karet založených na kanálu DMA, které používají stejný DMA-kanál jako implicitně nastavený SCSI-řadič Adaptec 1542. Pokud nepřesunete jednu z karet na jiný DMA-kanál, bude ethernetová karta zapisovat data z paketu na libovolná místa na vašem pevném disku.

3Com EtherLink

Jsou podporovány karty 3c503 a 3c503/16, stejně tak i 3c507 a 3c509. To platí i pro kartu 3c501, která je ale příliš pomalá, než aby se její koupě vyplatila.

* Poznámka korektora: Linux dnes podporuje naprostou většinu síťových karet.

Novell Eagle

NE1000, NE2000 a spousta jejich klonů. Jsou podporovány i karty NE1500 a NE2100.

Western Digital/SMC

Podporovány jsou karty WD8003 a WD8013 (stejné jako SMC Elite a SMC Elite Plus) a také nová karta SMC Elite 16 Ultra.

Hewlett Packard

HP 27252, HP 27247B a HP J2405A.

D-Link

Kapesní adaptér DE-600, DE-100, DE-200 a DE-220-T. Existuje i záloha pro kartu DE-650-T, což je PCMCIA karta.[4]

DEC

DE200 (32K/64K), DE202, DE100 a DEPCA model E.

Allied Telesis

AT1500 a AT1700.

V Linuxu lze společně s jednou z těchto karet použít předkompilované jádro od jednoho z hlavních linuxových distributorů. Tato jádra mají zpravidla vestavěné ovladače pro všechny typy karet. Nicméně z dlouhodobého hlediska je lepší si sestavit své vlastní jádro a zkompilovat ho pouze s těmi ovladači, které budete skutečně potřebovat.

3.4.3 Automatická detekce Ethernetu

Při zavádění se ethernetový kód pokusí nalézt vaši kartu a určit její typ. Karty jsou detekovány na následujících adresách a v následujícím pořadí:

Karta	Sondované adresy
WD/SMC	0x300, 0x280, 0x380, 0x240
SMC 16 Ultra	0x300, 0x280
3c501	0x280

[4] Společně s ostatním materiálem pro přenosné počítače ji lze získat na adrese **tsx-11.mit.edu** v adresáři `packages/laptops`.

Karta	Sondované adresy
3c503	0x300, 0x310, 0x330, 0x350, 0x250, 0x280, 0x2a0, 0x2e0
Nex000	0x300, 0x280, 0x320, 0x340, 0x360
HP	0x300, 0x320, 0x340, 0x280, 0x2c0, 0x200, 0x240
DEPCA	0x300, 0x320, 0x340, 0x360

Pro kód automatické detekce existují dvě omezení. Za prvé, jádro nemusí korektně rozpoznat všechny karty. To platí zejména pro některé levnější klony běžných karet, ale také pro některé karty WD80x3. Druhým problémem je, že jádro nebude automaticky detekovat výskyt více než jedné karty. To je záměr, protože se předpokládá, že budete osobně chtít řídit, které rozhraní bude dané kartě přiřazeno.

Používáte-li více než jednu kartu nebo pokud se automatická detekce karty nezdaří, je třeba jádru explicitně říci základní adresu a název karty.

V Net-3 můžete k tomuto účelu použít dva různé postupy. Jeden spočívá v úpravě nebo doplnění informací do souboru `drivers/net/Space.c` se zdrojovým kódem jádra, který obsahuje veškeré informace o ovladačích. Tento způsob se doporučuje pouze v případě, že jste obeznámeni se síťovým kódem. Mnohem lepší je poskytnout jádru tyto informace při zavádění. Pokud pro zavádění systému použijete příkaz `lilo`, můžete předat parametry jádra za pomoci volby `append` v souboru `lilo.conf`. Chcete-li jádro informovat o ethernetovém zařízení, předejte mu následující parametr:

```
ether=irq,base_addr,param1,param2,name
```

První čtyři parametry jsou číselné, zatímco poslední představuje název zařízení. Všechny numerické hodnoty jsou nepovinné; pokud je vynecháte nebo uvedete nulové hodnoty, pokusí se jádro získat jejich hodnotu detekcí nebo místo nich použije implicitní hodnoty.

První parametr nastavuje IRQ, které bude přiděleno zařízení. Jádro se implicitně pokusí automaticky detekovat IRQ-kanál daného zařízení. Ovladač 3c503 disponuje speciální vlastností, která vybere volný IRQ-kanál ze seznamu 5, 9, 3, 4 a nakonfiguruje kartu tak, aby tento kanál používala.

Parametr **`base_addr`** udává V/V základní adresu karty. Zadáte-li hodnotu nula, pokusí se jádro zjistit adresu z výše uvedeného seznamu.

Zbývající dva parametry mohou různé typy ovladačů používat odlišným způsobem. U karet sdílejících paměť, jako například WD80x3, určují počáteční a koncovou adresu oblasti sdílené paměti. Ostatní karty používají zpravidla parametr **`param1`** k nastavení úrovně zobrazení ladicích informací. Hodnoty od 1 do 7 ukazují zvyšující se úroveň rozsahu výpisů, zatímco

hodnota 8 ji vypíná; 0 udává implicitní volbu. Ovladač 3c503 používá **param2** k výběru vnitřního transceiveru (implicitně) nebo vnějšího transceiveru (hodnota 1). Nulová hodnota způsobí použití BNC-konektoru umístěného na kartě; hodnota 1 vyvolá použití portu AUI.

Pokud máte dvě ethernetové karty, může Linux automaticky detekovat jednu z nich a parametry druhé karty mu pak předáte pomocí příkazu `lilo`. Je třeba se však ujistit, že ovladač náhodně nenašel nejprve druhou kartu. V tom případě by totiž druhou kartu vůbec nenašel. Lze tak učinit pomocí volby **reserve** v příkazu `lilo`, která explicitně řekne jádru, aby nedetekoval V/V-prostor obsazený druhou kartou.

Chcete-li například, aby Linux nainstaloval druhou ethernetovou kartu na adresu **0x300** jako *eth0*, musíte jádru předat následující parametry:

```
reserve=0x300,32 ether=0,0x300,eth1
```

Volba `reserve` zajistí, aby žádný ovladač nepřistupoval při detekci určitého zařízení k V/V-prostoru karty. Parametry jádra lze také použít k přesměrování automatické detekce *eth0*:

```
reserve=0x340,32 ether=0,0x340,eth0
```

Chcete-li úplně vypnout automatickou detekci, použijte argument **base_addr** s hodnotou –1:

```
ether=0,-1,eth0
```

3.5 Ovladač PLIP

Protokol PLIP znamená *IP-protokol po paralelní lince* (*Parallel Line IP*) a představuje levnou alternativu sítě, pokud chcete propojit pouze dva počítače. Používá paralelní port společně se speciálním kabelem a dosahuje rychlosti od 10 kBps do 40 kBps.

Protokol PLIP původně vyvinula firma Crynwr, Inc. Jeho návrh je poměrně prostý: po dlouhou dobu byly u počítačů PC paralelní porty používány pouze jako jednosměrné porty pro tiskárny; to znamená, že při posílání informací z počítače PC do periferního zařízení mohlo být použito pouze osm datových linek. Posílání nebylo možné opačným směrem. Protokol PLIP se s tím vypořádal tak, že u portu používá pro vstup pět stavových linek, což ale zpomaluje přenos dat, protože všechna data jsou přenášena pouze po částech (po polovinách bajtu). Tento pracovní režim se nazývá nulový režim protokolu PLIP. Dnes se zdá, že se již tyto jednosměrné porty vůbec nepoužívají. Proto existuje rozšíření protokolu PLIP zvané režim 1, které používá plné 8bitové rozhraní.

Na rozdíl od předchozích verzí kódu protokolu PLIP se současná verze snaží být kompatibilní s implementacemi protokolu PLIP firmy Crynwr. Totéž platí i pro ovladač protokolu PLIP v `telnetu` NCSA.[5] Ke spojení dvou počítačů za pomoci protokolu PLIP potřebujete speciální kabel, který seženete v některých obchodech pod označením kabel „Null Printer“ nebo „Turbo Laplink“. Poměrně jednoduše si ale můžete vyrobit kabel vlastní. Příloha A ukazuje, jak na to.

Linuxový ovladač protokolu PLIP je výsledkem práce obrovského množství lidí. Momentálně ho spravuje pan Niibe Yutaka. Pokud je ovladač protokolu PLIP zakompilován do jádra, nastaví následující síťové rozhraní každého z dostupných paralelních portů: rozhraní *plip0* bude odpovídat paralelnímu portu `lp0`, rozhraní *plip1* bude odpovídat paralelnímu portu `lp1` atd. Mapování rozhraní PLIP k paralelním portům je následující:

Rozhraní	V/V port	IRQ
plip0	0x3BC	7
plip1	0x378	7
plip2	0x278	5

Máte-li port pro tiskárnu nakonfigurován odlišným způsobem, je třeba upravit tyto hodnoty v souboru `drivers/net/Space.c`, ve zdrojovém kódu jádra Linuxu. Potom je třeba jádro znovu sestavit.

Nicméně toto mapování neznamená, že nemůžete používat paralelní porty obvyklým způsobem. Ovladač protokolu PLIP přistupuje k paralelním portům pouze v případě, že je k němu nakonfigurováno odpovídající rozhraní.

3.6 Ovladače SLIP a PPP

Protokoly SLIP (IP po sériové lince) a PPP (Protokol Point-to-Point) se hojně využívají k posílání IP-paketů po sériové lince. Připojení SLIP a přístup k internetovým počítačům za pomoci protkolu PPP nabízí množství firem. Tímto způsobem poskytují IP připojení soukromým osobám (jinak by pro ně bylo připojení cenově nedostupné).

Abyste mohli používat ovladače SLIP nebo PPP, nejsou nutné žádné hardwarové úpravy; stačí použít libovolný sériový port. Protože sériové porty nejsou pro komunikaci na bázi TCP/IP typické, bude jim věnována samostatná kapitola.Více informací tak najdete v kapitole 4.

5 `Telnet` NCSA je populární program pro DOS, který spouští protokol TCP/IP pomocí Ethernetu nebo protokolu PLIP a podporuje standardy `telnet` a FTP.

Kapitola

4

Nastavení sériového hardwaru

Povídá se, že v oblasti sítí stále ještě existují lidé, kteří vlastní pouze jediný počítač PC a kteří nemají dostatek peněz na internetové spojení typu T1. Aby však získali svou „každodenní dávku konferencí a pošty“, spoléhají na spojení pomocí protokolu SLIP, na sítě typu UUCP a na systémy BBS, které využívají veřejné telefonní sítě.

Tato kapitola má pomoci všem lidem, kteří jsou odkázáni pouze na modemy. Do přílišných podrobností se ale v této kapitole pouštět nemůžeme, například jak nakonfigurovat modem pro volání. Všechna tato témata budou zpracována v dokumentu Grega Hankinse Serial HOWTO[1].

4.1 Komunikační software pro modemová spojení

V Linuxu je k dispozici mnoho komunikačních balíků. Spoustu z nich tvoří *terminálové programy*, které umožňují uživateli propojení s jiným počítačem a které se tváří, jakoby uživatel seděl před jednoduchým terminálem. Tradičním unixovým terminálovým programem je `kermit`. Je však značně zastaralý. Dnes jsou dostupné mnohem komfortnější programy, které podporují adresář s telefonními čísly, skriptové jazyky, jež se používají pro volání a připojování ke vzdáleným počítačovým systémům atd. Jeden z nich se nazývá `minicom`. Je podobný některým terminálovým programům, které možná znají dřívější uživatelé operačního systému DOS. Existují také komunikační balíky na bázi X, například `seyon`.

Rovněž je k dispozici množství linuxových balíků pro BBS. Některé z těchto balíků je možné nalézt na FTP-serveru **sunsite.unc.edu** v adresáři `/pub/Linux/system/Network`.

---

1 Greg Hankins je k zastižení na e-mailu **gregh@cc.gatech.edu**.

Kromě terminálových programů existuje software, který při přenosu dat z vašeho nebo do vašeho počítače nevyužívá sériovou linku interaktivním způsobem. Výhodou této metody je výrazně kratší čas potřebný k automatickému stažení několika desítek kilobajtů. Typickým příkladem je on-line čtení pošty z poštovní schránky nebo hledání zajímavých článků v systému BBS. Na druhou stranu však tento způsob vyžaduje mnohem větší diskový prostor, protože načítá i bezcenné informace.

Typickým představitelem tohoto typu softwaru je UUCP. Jedná se o programový balík, který kopíruje data z jednoho hostitele na druhý, spouští programy na vzdáleném hostiteli atd. V soukromých sítích je často používán pro přenos pošty nebo konferencí. V následující kapitole bude popsán balík UUCP od Iana Taylora, který také běží v prostředí Linuxu. Další neinteraktivní komunikační software se používá například v sítích Fidonet. Jsou k dispozici i aplikace přenesené na platformu Linuxu, například `ifmail`.

Internetový protokol pro sériové linky SLIP se nachází někde na rozmezí, dovoluje jak interaktivní, tak i neinteraktivní použití. Mnoho lidí využívá protokol SLIP pro připojení ke školní síti nebo k některému jinému druhu veřejného SLIP-serveru. Mohou tak využívat služby protokolu FTP a jiné. Protokol SLIP lze ale použít také k propojení několika lokálních sítí pevnými nebo částečně pevnými linkami. Toto využití je zajímavé pouze pokud současně použijeme zařízení ISDN.

4.2 Úvod k sériovým zařízením

Zařízení, pomocí nichž umožňuje jádro Unixu přístup k sériovým zařízením, se obvykle nazývají *tty*. Je to zkratka názvu společnosti *Teletype*(TM), která bývala v počátcích unixové éry jedním z hlavních výrobců terminálů. V současnosti se tento termín používá pro jakékoliv znakově založené datové terminály. V průběhu této kapitoly budeme tento termín používat výhradně k označení zařízení jádra.

Linux rozlišuje tři třídy tty: (virtuální) konzoly, pseudoterminály (podobné obousměrnému potrubí, které používají aplikace jako je X11) a sériová zařízení. Posledně zmíněná třída je rovněž považována za tty, poněvadž umožňuje interaktivní spojení po sériové lince; sériová zařízení lze používat s využitím telefonní linky buďto z pevně připojeného terminálu, nebo ze vzdáleného počítače.

Zařízení tty mají spoustu konfiguračních parametrů, které lze nastavovat pomocí systémového volání *ioctl(2)*. Mnoho z nich se vztahuje pouze na sériová zařízení, protože ty vyžadují ke správě různých typů spojení výrazně vyšší flexibilitu.

Mezi nejvýznamnější parametry linek patří rychlost a parita. Existují však i tzv. registry, které řídí konverzi mezi malými a velkými písmeny, převod znaků CR (carriage return) na LF (linefeed) atd. Ovladač tty může také podporovat různé linkové disciplíny, které mohou zcela změnit chování ovladače zařízení. Například linuxový ovladač SLIP je implementován jako speciální linková disciplína.

Pro způsob měření rychlosti linky je příznačná určitá dvojznačnost. Správným termínem je tzv. *bitová rychlost*, která se vztahuje na rychlost přenosu linky a měří se v bitech za sekundu (zkráceně bps). Někdy lidé tento termín označují jako tzv. *přenosovou rychlost*, což není tak úplně správné. Tyto termíny totiž nelze zaměňovat. Přenosová rychlost se vztahuje k fyzickým charakteristikám daného sériového zařízení, konkrétně k taktovacímu kmitočtu, ve kterém jsou přenášeny pulsy. Bitová rychlost označuje spíše aktuální stav existujícího sériového spojení mezi dvěma body, konkrétně průměrný počet bitů přenesený za sekundu. Je důležité vědět, že tyto dvě hodnoty se obvykle liší, protože většina zařízení kóduje do elektrického impulsu více než jeden bit.

4.3 Přístup k sériovým zařízením

Stejně jako ke všem unixovým zařízením, je i k sériovým zařízením přistupováno pomocí speciálních souborů zařízení, které se nachází v adresáři `/dev`. K ovladačům sériových zařízení se vztahují dvě skupiny souborů zařízení. Každému portu odpovídá jeden soubor zařízení z každé skupiny souborů zařízení. Zařízení se bude chovat v závislosti na typu souboru, který se používá pro přístup k tomuto zařízení.

První skupina souborů zařízení se použije kdykoliv, kdy je k portu přistupováno ve směru do počítače; má hlavní číslo 4 a soubory se nazývají `ttyS0`, `ttyS1` atd. Druhá skupina se použije pro vytáčení směrem z počítače; soubory se nazývají `cua0` atd. a jejich hlavní číslo je rovno 5.

Vedlejší čísla jsou pro oba typy stejná. Máte-li modem připojen na jednom z portů *COM1* až *COM 4*, bude vedlejší číslo rovno číslu *COM*-portu plus 63. Pokud se vaše nastavení liší nebo používáte například kartu, která podporuje několik sériových linek, podívejte se, prosím, do dokumentu Serial HOWTO.

Budeme předpokládat, že váš modem je připojen na port *COM2*. Jeho vedlejší číslo tak bude mít hodnotu 65 a hlavní číslo bude rovno 5, protože modem budeme používat pro volání z počítače. Mělo by existovat zařízení `cva1`, které bude mít tyto hodnoty. Vypište si sériová zařízení tty z adresáře `/dev`. Sloupce 5 a 6 by měly ukazovat hlavní a vedlejší čísla v uvedeném pořadí:

```
$ ls -l /dev/cua*
crw-rw-rw-   1 root     root     5,  64 Nov 30 19:31 /dev/cua0
crw-rw-rw-   1 root     root     5,  65 Nov 30 22:08 /dev/cua1
crw-rw-rw-   1 root     root     5,  66 Oct 28 11:56 /dev/cua2
crw-rw-rw-   1 root     root     5,  67 Mar 19  1992 /dev/cua3
```

Jestliže takové zařízení neexistuje, musíte je vytvořit: přihlaste se jako superuživatel a napište:

```
# mknod -m 666 /dev/cua1 c 5 65
# chown root.root /dev/cua1
```

Někteří lidé doporučují vytvořit soubor `/dev/modem`, který by sloužil jako symbolické spojení s vaším modemovým zařízením, takže si příležitostní uživatelé nemusí pamatovat poněkud neintuitivní `cua1`. Nemůžete ale v jednom programu používat soubor `modem` a ve druhém skutečný název souboru zařízení. To proto, že programy používají tzv. *zamykací soubory (lock file)*, které signalizují, že dané zařízení je právě používáno. Podle úmluvy je například název zamykacího souboru pro `cua1` `LCK..cua1`. Použijete-li pro stejný port různé názvy souborů zařízení, pak programy správně nerozliší zamykací soubory a budou současně přistupovat ke stejnému zařízení. Výsledkem bude, že nebude pracovat ani jedna z aplikací.

4.4 Sériový hardware

V současné době podporuje Linux širokou škálu sériových karet, které využívají standard RS-232C, který je momentálně nejběžnějším pro sériovou komunikaci ve světě počítačů PC. K přenosu jednotlivých bitů a synchronizaci využívá několik elektronických obvodů. Další linky lze využít k signalizaci výskytu tzv. nosné (carrier) (kterou používají modemy) a k signalizaci počátku výměny informací, tzv. handshake (podání ruky).

Ačkoliv je hardwarový handshake volitelný, je velice užitečný. Umožňuje oběma stanicím vzájemné předávání informací o stavech - zda je jedna strana připravena na příjem dalších dat nebo zda by měla druhá strana počkat, až strana, která je na příjmu, dokončí zpracování příchozích dat. K tomuto účelu se používají tzv. linky „Clear to Send“ (CTS) a „Ready to Send“ (RTS), které provádějí cosi na způsob hardwarového handshake, konkrétně „RTS/CTS“.

[2] Existoval i čip NSC 16 550, ale jeho paměť FIFO snad vůbec nikdy nefungovala.

V počítačích PC je rozhraní RS-232 obvykle řízeno čipem UART, který je odvozen od čipu 16 450 firmy National Semiconductor nebo od jeho novější verze NSC 16 550A[2]. Některé skupiny zařízení (nejvýraznějšími z nich jsou interní modemy vybavené čipovou sadou firmy Rockwell) používají ale úplně odlišné čipy, které byly naprogramovány tak, aby se chovaly jako čip 16 550.

Hlavní odlišností čipů 16 450 a 16 550 je podpora vyrovnávací paměti FIFO o velikosti 16 bajtů, která je implementována v novějším čipu 16 550, zatímco čip 16 450 disponuje pouze vyrovnávací pamětí o velikosti 1 bajt. Tento rys omezuje použití čipu 16 450 pro maximální přenosové rychlosti 9 600 bps, ale čip kompatibilní s 16 550 lze použít i pro rychlosti vyšší. Kromě těchto čipů podporuje Linux i čip 8250, což byl původní čip počítačů PC AT.

Jádro implicitně kontroluje čtyři standardní sériové porty *COM1* až *COM4*. Těmto portům jsou přidělena vedlejší čísla v rozmezí od 64 do 67, viz výše uvedený popis.

Pro přesné nastavení sériových portů slouží příkaz *setserial,* jehož autorem je Ted Ts'o, a s ním i skript `rc.serial`. Tento skript lze vyvolat při zavádění systému z adresáře `/etc/rc`. Ke konfiguraci sériových zařízení jádra používá skript `rc.serial` příkaz `setserial`. Typický skript `rc.serial` vypadá následovně:

```
# /etc/rc.serial - Konfigurační skript sériové linky
#
# Detekce přerušení
/sbin/setserial -W /dev/cua*

# Konfigurace sériových zařízení
/sbin/setserial /dev/cua0 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua1 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua2 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua3 auto_irq skip_test autoconfig

# Zobrazení konfigurace sériových zařízení
/sbin/setserial -bg /dev/cua*
```

Vysvětlení parametrů najdete v dokumentaci příkazu `setserial`.

Pokud nebude detekována vaše sériová karta, nebo pokud příkaz `setserial -bg` zobrazí nesprávná nastavení, je třeba konfiguračnímu skriptu explicitně poskytnout správné hodnoty. Uživatelé interních modemů s čipovou sadou firmy Rockwell potvrzují, že se s tímto problémem již setkali. Pokud bude například čip UART detekován jako čip NSC 16 450, i když je ve skutečnosti kompatibilní s čipem NSC 16 550, je nutné změnit konfigurační příkaz na:

```
/sbin/setserial /dev/cua1 auto_irq skip_test autoconfig uart 16550
```

Podobné volby existují i pro specifikaci konkrétního *COM*-portu, základní adresy a nastavení IRQ. Podívejte se prosím do manuálu příkazu `setserial(8)`.

Pokud váš modem podporuje hardwarový handshake, měli byste se ujistit, že je opravdu povolen. Protože většina komunikačních programů předpokládá, že je hardwarový handshake povolen, nesnaží se ho implicitně zapínat; takže je třeba to provést ručně. Nejlépe je to provést ve skriptu `rc.serial` za pomoci příkazu *stty*:

```
$ stty crtscts < /dev/cua1
```

Ke kontrole aktivity hardwarového handshake použijte příkaz:

```
$ stty -a < /dev/cua1
```

Tento řádek vypíše stav všech registrů příslušného zařízení; znaménko minus předcházející zobrazenému registru tento registr vypíná, například **`-crtscts`**.

Kapitola

5

Konfigurace sítí na bázi TCP/IP

V této kapitole si projdeme kroky, které jsou nezbytné pro konfiguraci sítí na bázi protokolu TCP/IP. Začneme s přidělením IP-adresy, pak si projdeme nastavení rozhraní TCP/IP a představíme si několik užitečných nástrojů pro hledání problémů ve vaší síťové instalaci.

Většinu úkolů probraných v této kapitole budete muset obvykle provést pouze jedenkrát. Pozdější změny v konfiguračních souborech budou nutné jen v případech, kdy budete chtít do sítě přidat nový systém; nebo budete-li chtít systém kompletně překonfigurovat. Nicméně některé z příkazů pro konfiguraci protokolu TCP/IP musí být spouštěny pokaždé při zavádění systému. To se zpravidla provádí jejich voláním ze systémových skriptů v adresáři `/etc/rc`.

Část procedury týkající se sítí je většinou obsažena ve skriptu s názvem `rc.net` nebo `rc.inet`. Někdy se také můžete setkat se dvěma skripty s názvy `rc.inet1` a `rc.inet2`. První inicializuje síťovou část jádra systému, zatímco druhý spouští základní síťové služby a aplikace. V průběhu následujícího výkladu se budeme držet druhého zmíněného konceptu.

Dále si vysvětlíme akce prováděné skriptem `rc.inet1`, vlastní aplikace budou probrány v pozdějších kapitolách. Po přečtení této kapitoly byste měli umět napsat příslušnou posloupnost příkazů, která správně nakonfiguruje síť s protokolem TCP/IP. Dále byste měli nahradit všechny vzorové příkazy v souboru `rc.inet1` svými vlastními příkazy a ujistit se, že dochází ke spouštění tohoto skriptu při zavádění, a restartovat počítač. Síťové *rc*-skripty dodávané společně s oblíbenou verzí Linuxu vám mohou posloužit jako dobré příklady.

5.1 Nastavení souborového systému proc

Některé z konfiguračních nástrojů balíku Net verze 2 spoléhají při komunikaci s jádrem na souborový systém `proc`. Jedná se o rozhraní, které umožňuje přístup k aktuálním informacím jádra za pomoci mechanismu, jenž je podobný souborovému systému. Když je systém `proc` připojen, můžete stejně jako u ostatních souborových systémů vypisovat jeho soubory nebo zobrazovat jejich obsah. Typickými položkami jsou soubory *loadavg*, které obsahují průměrné zatížení systému, nebo soubor `meminfo`, který zobrazuje aktuální obsazení fyzické paměti a využití odkládacích souborů.

K tomuto účelu přidává síťový kód adresář `net`. Obsahuje množství souborů s informacemi jako například tabulky ARP jádra systému, stav spojení na bázi TCP a směrovací tabulky. Většina síťových administrativních nástrojů čerpá informace z těchto souborů.

Souborový systém `proc` (někdy se používá označení *procfs*) je obvykle připojen při zavádění systému jako adresář `/proc`. Nejlépe toho dosáhnete přidáním následujícího řádku do souboru `/etc/fstab`:

```
# připojovací bod procfs
none /proc proc defaults
```

Potom ze svého skriptu `/etc/rc` spusťte příkaz „**`mount / proc`**“.

V současné době je souborový systém *procfs* implicitně začleněn do většiny jader operačního systému. Není-li souborový systém *procfs* ve vašem jádru, objeví se například zpráva „**mount: fs type procfs not supported by kernel**“. V tom případě budete muset překompilovat jádro systému a na dotaz, zda si přejete nainstalovat podporu souborového systému *procfs*, odpovědět „yes“.

5.2 Instalace binárních souborů

Používáte-li jednu z předkompilovaných distribucí Linuxu, bude velmi pravděpodobně obsahovat hlavní síťové aplikace a utility, a také kompletní sadu vzorových souborů. Jediným případem, kdy budete chtít získat a nainstalovat nové utility, je instalace nové verze jádra operačního systému. Protože to občas vede ke změnám v síťové vrstvě jádra systému, budete muset aktualizovat i základní konfigurační nástroje. To znamená přinejmenším rekompilaci binárních souborů, ale někdy budete potřebovat i nejnovější sady binárních souborů. Ty jsou obvykle distribuovány společně s jádrem a zabaleny v archívu `net-XXX.tar.gz`, kde *XXX* označuje číslo verze. Verze odpovídající Linuxu 1.0 je 0.32b, avšak poslední jádro operačního systému (v době vzniku této publikace to byla verze 1.1.12) vyžaduje verzi 0.32d.

Pokud si chcete sami zkompilovat a nainstalovat standardní síťové aplikace na bázi protokolu TCP/IP, získáte jejich zdrojový kód na většině linuxových serverů. Jsou to více či méně opravené verze programů z balíku Net-BSD nebo z ostatních zdrojů. Ostatní aplikace, jako například `Xmosaic`, `xarchie` nebo Gopher a klienti IRC, je nutné získat samostatně.. Budete-li dodržovat příslušné instrukce, zkompiluje se většina z těchto aplikací takřkajíc po vytažení z krabice.

Oficiální systém pro balík Net-3 najdete na adrese **sunacm.swan.ac.uk**, jehož zrcadlem je **sunsite.unc.edu** a příslušný adresář se jmenuje `system/Network/sunacm`. Poslední aktualizace balíku Net-2e a jeho binární soubory jsou k dispozici na adrese **ftp.aris.com**. Balík BSD od Matthiase Urlichse – odvozený ze síťového kódu - můžete získát na adrese **ftp.ira.uka.de** v adresáři `/pub/system/linux/netbsd`.

5.3 Další příklad

Ve zbytku této knihy si dovolím uvést nový příklad, jenž je oproti případu Groucho Marx University méně složitý, ale může se více podobat problémům, s nimiž se ve skutečnosti setkáte. Vezměme si společnost Virtual Brewery, což je malá společnost, která, jak název napovídá, vaří virtuální pivo. Aby mohli virtuální pivovarníci efektivněji spravovat svůj obchod, chtějí mít své počítače propojeny do sítě. Všechny počítače jsou typu PC se šikovným operačním systémem Linux verze 1.0.

Na stejném podlaží na druhém konci budovy existuje obchod Virtual Winery, který s pivovarem velmi blízce spolupracuje. Mají svůj vlastní Ethernet. Tyto dvě společnosti chtějí mít zcela přirozeně z provozních důvodů své sítě propojeny. Nejprve chtějí nastavit bránu, jenž bude doručovat datagramy mezi těmito dvěma podsítěmi. Dále budou chtít být v kontaktu s okolním světem spojením typu UUCP, pomocí nějž budou přijímat konference a poštu. Z dlouhodobého hlediska budou chtít mít nastavené spojení typu SLIP, aby se mohli příležitostně připojit na Internet.

5.4 Nastavení názvu hostitele

Většina, ne-li všechny aplikace, spoléhají na to, že je název místního hostitele nastaven na nějakou rozumnou hodnotu. To se obvykle provede při zavádění pomocí příkazu *hostname*. Chcete-li nastavit název hostitele například na **name**, zadejte:

```
# hostname name
```

U tohoto příkazu je běžné používat nekvalifikované názvy hostitele bez jakéhokoliv názvu domény. Například hostitelé ve Virtual Brewery se mohou nazývat **vale.vbrew.com**, **vlager.vbrew.com** atd. Toto jsou oficiální, plně kvalifikované názvy domén. Názvy jejich místních hostitelů bude tvořit pouze první část z názvu, například **vale**. Protože je však název lokálního hostitele často používán pro vyhledání IP-adresy hostitele, musíte se ujistit, že knihovna resolveru je schopná vyhledat IP-adresu hostitele. To obvykle znamená, že musíte název hostitele specifikovat v souboru `/etc/hosts` (viz níže). Někteří lidé doporučují používat příkaz `domainname`. Tento příkaz se používá pro informování jádra operačního systému o názvu domény zbývající části FQDN. U tohoto způsobu byste měli pro opětovné získání FQDN zkombinovat výstupy příkazů `hostname` a `domainname`. Tento způsob je však jen z poloviny správný. Příkaz `domainname` se všeobecně používá k nastavení NIS-domény hostitele, která může být zcela jiná než DNS-doména, ke které patří váš hostitel. Doména NIS je probírána v kapitole 10.

5.5 Přidělení IP-adres

Chcete-li nakonfigurovat síťový software na svém hostiteli, který není určen pro práci v síti (aby na něm mohl být například spouštěn software síťových konferencí INN), můžete tuto stať s klidným svědomím přeskočit, protože k tomu budete potřebovat pouze IP-adresu svého zpětnovazebného rozhraní, a ta je vždy **127.0.0.1**.

Mírně složitější je tento problém v reálných sítích typu Ethernet. Chcete-li připojit svého hostitele k existující síti, musíte požádat jejího správce, aby vám v této síti přidělil IP-adresu. Nastavujete-li celou síť sami, musíte si také sami přidělit IP-adresy, což probereme později.

Hostitelé, kteří patří do místní sítě, by měli obvykle sdílet adresy ze stejné logické sítě IP. Z toho důvodu musíte přidělit síťovou IP-adresu. Máte-li několik fyzických sítí, musíte jim buď přidělit odlišná síťová čísla, anebo musíte použít podsítě, čímž rozdělíte svůj rozsah IP-adres do několika podsítí.

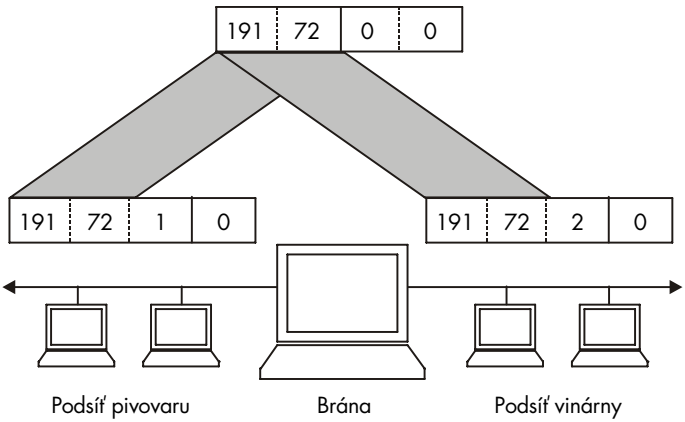
Není-li vaše síť připojena k Internetu, můžete si vybrat libovolnou (platnou) adresu sítě. Pouze se musíte ujistit, že jste vybrali jednu ze tříd A, B nebo C, protože jinak by pravděpodobně vše nefungovalo zcela správně. Plánujete-li v blízké době připojení k Internetu, měli byste si už *nyní* obstarat oficiální IP-adresu. Nejlépe je požádat o pomoc svého poskytovatele síťových služeb. Pokud si chcete obstarat číslo sítě jenom proto, že se někdy v budoucnosti budete chtít připojit na Internet, vyžádejte si na adrese **hostmaster@internic.net** formulář Network Address Application Form.

Budete-li spravovat několik ethernetových sítí (nebo jiných sítí, jakmile pro ně budou dostupné ovladače), musíte rozdělit vaši síť na podsítě. Všimněte si, že vytváření podsítí je požadováno pouze v případě, že vlastníte více než jednu *vysílací síť* (*broadcast network*);

nepočítaje v to spojení pomocí protokolu PPP. Máte-li například jeden Ethernet a jedno nebo více spojení s vnějším světem pomocí protokolu SLIP, nepotřebujete ve své síti vytvářet podsítě. Důvod bude objasněn v kapitole 7.

Síťový správce pivovaru požádal například centrum NIC o přidělení čísla sítě třídy B, obdržel adresu **191.72.0.0**. Aby si správce přizpůsobil své dva Ethernety k obrazu svému, rozhodl se použít osm bitů z části hostitele jako doplňující bity podsítě. Tato úprava poskytla části hostitele dalších osm bitů, což dovoluje až 254 hostitelů v každé podsíti. Dále správce přidělil pivovaru číslo podsítě 1 a vinárně číslo podsítě 2. Tedy jejich příslušné síťové adresy jsou **191.72.1.0** a **191.72.2.0**. Maska podsítě je **255.255.255.0**.

Bráně **vlager**, což je brána mezi dvěma sítěmi, bylo u obou podsítí přiděleno číslo hostitele 1, takže její IP-adresy jsou **191.72.1.1**, resp. **191.72.2.1**. Obrázek 5.1 ukazuje dvě podsítě a bránu.



Obrázek 5.1
Dvě podsítě – Virtual Brewery a Virtual Winery

Všimněte si, že v tomto případě používáme z ilustračních důvodu síť třídy B; síť třídy C by byla mnohem realističtější. U nového síťového kódu není vytváření podsítí limitováno na celé bajty, takže i třída C může být rozdělena do několika podsítí. Dva bity z části hostitele byste mohli například použít na síťovou masku, čímž byste získali čtyři možné podsítě s až 64 hostiteli na každé z nich.[1]

[1] Poslední číslo je v každé podsíti rezervováno pro tzv. vysílací adresu, takže ve skutečnosti může v každé podsíti existovat jen 63 hostitelů.

5.6 Sestavení souborů hosts a networks

Po vytvoření podsítí byste měli za pomoci souboru `/etc/hosts` připravit několik jednoduchých typů rozlišení názvu hostitele. Pokud nehodláte k rozlišení adres používat DNS nebo NIS, musíte do souboru `hosts` vložit všechny hostitele.

Dokonce i v případě, kdy chcete používat DNS nebo NIS při normálních operacích, budete chtít mít v souboru `/etc/hosts` ze všech názvů hostitelů alespoň nějaké skupiny hostitelů. A nějaký druh rozlišení názvů hostitelů budete požadovat i tehdy, nepoběží-li žádné síťové rozhraní (například při zavádění). Tento způsob není vhodný jen z kvůli většímu pohodlí, ale umožní vám také používat ve skriptech `rc.inet` symbolické názvy hostitelů. Takže při změně IP-adres vám postačí pouze zkopírovat aktualizovaný soubor `hosts` na všechny počítače a nemusíte se zatěžovat editací velkého počtu souborů*.rc* Do souboru `hosts` obvykle přidáte všechny místní názvy hostitelů a jejich adresy a jsou-li používány NIS-servery[2] nebo nějaké brány, přidáte i je.

V průběhu úvodního testování byste se měli také ujistit, že váš resolver používá pouze informace ze souboru `hosts`. Software DNS nebo NIS může mít u sebe ukázkové soubory, které, pokud je použijete, mohou produkovat zvláštní výsledky. Chcete-li, aby všechny aplikace při vyhledání IP-adresy hostitele používaly výhradně soubor `/etc/hosts`, musíte upravit soubor `/etc/host.conf`. Všechny řádky, které začínají klíčovým slovem *order,* označte jako komentáře. Provedete to tak, že na první pozici příslušného řádku vložíte znak (#). Dále vložte následující řádek:

```
order hosts
```

Konfigurace knihovny resolveru bude detailně rozebrána v kapitole 6.

Soubor `hosts` obsahuje na každém řádku jednu položku, která se skládá z IP-adresy, názvu hostitele a z nepovinného seznamu přezdívek názvu hostitele. Jednotlivá pole jsou vzájemně oddělena mezerami nebo tabulátory a pole s adresou musí začínat ve sloupci jedna. Cokoliv, co následuje za znakem #, je považováno za komentář a je ignorováno.

Názvy hostitelů mohou být buď plně kvalifikované, nebo relativní vzhledem k místní doméně. U serveru **vale** zadáte obvykle plně kvalifikovaný název **vale.vbrew.com** a do souboru *hosts* napíšete pouze název **vale**. Tím bude definován jak oficiální název, tak i kratší místní název.

[2] Adresu některých serverů NIS budete potřebovat pouze v případě, že budete používat NYS od Petera Erikssona. Ostatní implementace NIS najdou své servery při spuštění pomocí příkazu `ypbind`.

Následující příklad ilustruje, jak by mohl vypadat soubor *hosts* ve společnosti Virtual Brewery. Jsou v něm obsaženy dva speciální názvy, **vlager-if1** a **vlager-if2**, které poskytují adresy pro obě rozhraní používaná bránou **vlager**.

```
#
# Soubor hosts pro společnosti Virtual Brewery/Virtual Winery
#
# IP          lokální jméno       plně kvalifikované doménové jméno
#
127.0.0.1     localhost
#
191.72.1.1    vlager              vlager.vbrew.com
191.72.1.1    vlager-if1
191.72.1.2    vstout              vstout.vbrew.com
191.72.1.3    vale                vale.vbrew.com
#
191.72.2.1    vlager-if2
191.72.2.2    vbeaujolais         vbeaujolais.vbrew.com
191.72.2.3    vbardolino          vbardolino.vbrew.com
191.72.2.4    vchianti            vchianti.vbrew.com
#
```

Stejně jako u IP-adres hostitelů budete také někdy chtít použít symbolický název pro číslo sítě. Proto má soubor `hosts` doprovodný soubor nazvaný `/etc/networks`, který mapuje názvy sítí na čísla sítí a obráceně. Ve společnosti Virtual Brewery můžeme nainstalovat následující soubor `networks`:[3]

```
# soubor /etc/networks pro společnost Virtual Brewery
brew-net       191.72.1.0
wine-net       191.72.2.0
```

[3] Všimněte si, že názvy v souboru `networks` nesmí kolidovat s názvy hostitelů v souboru `hosts`, jinak by mohly některé programy produkovat nesprávné výsledky.

5.7 Konfigurace rozhraní pro protokol IP

Po nastavení hardwaru, které jsme probírali v předchozí kapitole, musíte o těchto zařízeních říci síťovému softwaru jádra. K nastavení síťových rozhraní a inicializaci směrovací tabulky se používá několik příkazů. Tyto úkoly se obvykle provádějí při každém startu počítače ze skriptu `rc.inet1`. Základní konfigurační nástroje se nazývají `ifconfig` (kde „if“ znamená rozhraní - interface) a `route`.

Příkaz `ifconfig` se používá pro zpřístupnění rozhraní síťové vrstvě jádra. Tento proces v sobě zahrnuje přidělení IP-adresy, některých parametrů a aktivaci rozhraní, někdy označovanou jako tzv. „uchopení“. Výraz „aktivní“ znamená, že jádro vysílá a přijímá datagramy pomocí rozhraní IP. Nejjednodušší způsob aktivace rozhraní je následující:

```
ifconfig interface ip-address
```

Výše uvedený příkaz přidělí IP-adresu `ip-address` rozhraní `interface` a aktivuje ho. Všechny ostatní parametry jsou nastaveny na své implicitní hodnoty. Například implicitní maska podsítě je odvozena z IP-adresy podle třídy sítě, například adresa **255.255.0.0** odpovídá adrese třídy B. Příkaz `ifconfig` je detailně popsán na konci této kapitoly.

Příkaz `route` umožňuje přidávat nebo odstraňovat směrování ze směrovací tabulky jádra systému. Může být vyvolán následujícím způsobem

```
route [add|del] target
```

kde argumenty `add` a `del` určují, zda se bude směrování do cíle `target` přidávat nebo se z něj bude odstraňovat.

5.7.1 Zpětnovazebné rozhraní

Jedním z prvních aktivovaných rozhraní je zpětnovazebné rozhraní:

```
# ifconfig lo 127.0.0.1
```

Občas zjistíte, že se místo IP-adresy používá fiktivní název hostitele **localhost**. Příkaz `ifconfig` vyhledá příslušný název v souboru `hosts`, kde by měl být deklarován pro adresu **127.0.0.1**:

```
# Příklad záznamu pro localhost v /etc/hosts
localhost       127.0.0.1
```

Chcete-li si prohlédnout konfiguraci rozhraní, spusťte příkaz `ifconfig` a jako argument zadejte název tohoto rozhraní:

```
$ ifconfig lo
lo       Link encap Local Loopback
         inet addr 127.0.0.1 Bcast [NONE SET] Mask 255.0.0.0
         UP BROADCAST LOOPBACK RUNNING MTU 2000 Metric 1
         RX packets 0 errors 0 dropped 0 overrun 0
         TX packets 0 errors 0 dropped 0 overrun 0
```

Jak vidno, zpětnovazebnému rozhraní byla přidělena síťová maska **255.0.0.0**, protože adresa **127.0.0.1** je adresou třídy A. Dále si můžete všimnout, že rozhraní nemá nastavenu vysílací adresu, protože ta nemá u zpětnovazebného rozhraní žádný význam. Pokud však budete na vašem hostiteli provozovat démona `rwhod`, budete muset nastavit relační adresu zpětnovazebného zařízení, protože jinak by démon `rwhod` nefungoval správně. Nastavení broadcast adresy je vysvětleno dále ve stati „Vše o příkazu `ifconfig`“.

Nyní si můžete začít se svou miniaturní sítí experimentovat. Zatím však ve směrovací tabulce stále chybí položka, která řekne protokolu IP, že může toto rozhraní používat pro směrování k cílové adrese **127.0.0.1**. Tu přidáte následujícím příkazem:

```
# route add 127.0.0.1
```

Opět můžete použít namísto IP-adresy název hostitele **localhost**.

Dále byste měli zkontrolovat, že vše správně funguje, například pomocí použití příkazu `ping`. Příkaz `ping` je síťovým ekvivalentem sonaru[4] a je používán k ověření dostupnosti příslušné adresy a na měření prodlev, které se vyskytují při posílání datagramu tam a zpět. Čas požadovaný na tuto operaci je často uváděn pod označením round trip.

```
# ping localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=32 time=1 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=32 time=1 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=32 time=1 ms
^C

--- localhost ping statistics ---
3 packets transmitter, 3 packets received, 0% packet loss
round-trip min/avg/max = 0/0/1 ms
```

4 Pamatuje si někdo na „Echoes“ od skupiny Pink Floyd?

Spustíte-li příkaz `ping` výše uvedeným způsobem, bude se tento příkaz snažit posílat pakety tak dlouho, dokud ho uživatel nepřeruší. Symbol `^C` označuje místo, kde byla stisknuta kombinace kláves Ctrl C.

Výše uvedený příklad ukazuje, že pakety pro adresu **127.0.0.1** jsou doručovány správně a odpovědi se vracejí příkazu `ping` zpět téměř okamžitě. To naznačuje, že jste při svém prvním nastavení síťového rozhraní uspěli.

Pokud se výstup příkazu `ping` nepodobá výše uvedenému výpisu, pak je tu problém. Zkontrolujte každou chybu, která naznačuje, že některý ze souborů nebyl korektně nainstalován. Zkontrolujte, zda jsou binární soubory příkazů `ifconfig` a `route` kompatibilní s vámi používanou verzí jádra operačního systému a hlavně se podívejte, zda bylo jádro zkompilováno s povolením síťových služeb (to poznáte podle přítomnosti adresáře `/proc/net`). Pokud obdržíte chybovou zprávu „Network unreachable“, pak bude zřejmě chyba v příkazu `route`. Ujistěte se, že používáte stejnou adresu, kterou jste předali příkazu `ifconfig`.

Výše popsané kroky stačí k tomu, abyste mohli na samostatném hostiteli používat síťové aplikace. Jakmile přidáte do souboru `rc.inet1` výše uvedené řádky a ujistíte se, že jsou oba soubory `rc.inet` spouštěny z adresáře `/etc/rc`, můžete počítač restartovat a vyzkoušet různé aplikace. Například příkaz „`telnet localhost`“ by měl s vaším hostitelem navázat spojení typu telnet, které vám nabídne přihlašovací výzvu.

Zpětnovazebné rozhraní je ale užitečné nejen jako příklad vhodný pro knihy o sítích nebo jako testovací prostředek při vývoji. Ve skutečnosti ho používají některé aplikace v průběhu normálních operací.[5] Proto ho musíte nakonfigurovat vždy, bez ohledu na to, zda váš počítač je či není připojen k síti.

5.7.2 Ethernetová rozhraní

Konfigurace ethernetového rozhraní má s nastavením zpětnovazebného rozhraní mnoho společného, pouze ve spojitosti s podsítěmi vyžaduje několik dalších parametrů.

Ve společnosti Virtual Brewery jsme rozdělili síť IP, která byla původně sítí třídy B, na podsítě, které jsou třídy C. Aby se v této změně vaše rozhraní vyznalo, měl by příkaz `ifconfig` vypadat následovně:

```
# ifconfig eth0 vstout netmask 255.255.255.0
```

5 Například všechny aplikace založené na RPC používají při startu zpětnovazebné rozhraní společně s démonem `portmapper` na registraci sebe samých.

Tento příkaz přiřadí rozhraní *eth0* IP-adresu **vstout** (adresa **191.72.1.2**). Pokud bychom vynechali síťovou masku, pak by si ji příkaz `ifconfig` odvodil z třídy sítě IP, ze které vyplyne síťová maska **255.255.0.0**. Rychlé ověření vypíše následující text:

```
# ifconfig eth0
eth 0  Link encap 10 Mps Ethernet Hwaddr 00:00:C0:90:B3:42
       inet addr 191.72.1.2 Bcast 191.72.1.255 Mask 255.255.255.0
       UP BROADCAST RUNNING MTU 1500 Metric 1
       RX packets 0 errors 0 dropped 0 overrun 0
       TX packets 0 errors 0 dropped 0 overrun 0
```

Můžete si všimnout, že příkaz `ifconfig` automaticky nastavil vysílací adresu (výše uvedené pole `Bcast`) na obvyklou hodnotu, což je číslo hostitelovy sítě se všemi nastavenými bity v části hostitele. Také velikost přenosové jednotky (maximální velikost ethernetového paketu, kterou bude generovat jádro pro toto rozhraní) byla nastavena na maximální hodnotu 1500 bajtů. Všechny tyto hodnoty mohou být změněny pomocí speciálních voleb, jež budou popsány dále.

Podobně jako tomu bylo u případu zpětnovazebného rozhraní, musíte nyní nainstalovat směrovací data, která budou jádro informovat o síti, jež je dosažitelná pomocí rozhraní *eth0*. Pro firmu Virtual Brewery byste měli vyvolat příkaz `route` následujícím způsobem:

```
# route add -net 191.72.1.0
```

Na první pohled to vypadá trochu magicky, protože není zcela zřejmé, jak příkaz `route` zjistí, přes které rozhraní má směrovat. Náš trik je ale celkem jednoduchý: jádro operačního systému si ověří všechna rozhraní, která byla v minulosti nakonfigurována a porovná cílovou adresu (v tomto případě **191.72.1.0**) se síťovou částí adresy rozhraní (to znamená, že bude po bitech porovnávat hodnotu získanou z adresy rozhraní a síťové masky). Jediné vyhovující rozhraní bude *eth0*.

K čemu tedy slouží volba `-net`? Používá se z toho důvodu, že příkaz `route` umí obsluhovat jak směrování do sítí, tak i směrování k jednotlivým hostitelům (jak jsme si ukázali u příkladu s názvem **localhost**). Je-li mu předána adresa v tečkové notaci, pokusí se příkaz `route` z prohlédnutých bitů hostitelské části odhadnout, zda se jedná o síť nebo o název hostitele. Je-li hostitelská část adresy rovna nule, bude příkaz `route` předpokládat, že se jedná o síť, v opačném případě ji bude považovat za adresu hostitele. Takto si bude příkaz `route` myslet, že je adresa **191.72.1.0** adresou hostitele a nikoliv číslem sítě. Příkaz `route` nemůže tušit, že používáme podsítě. Tuto informaci mu musíme sdělit explicitně pomocí argumentu `-net`.

Samozřejmě, že vypisování výše uvedeného příkazu `route` je únavné a člověk při něm může udělat chyby. Mnohem pohodlnější je použít názvy sítí, které jsme již nadefinovali v souboru `/etc/networks`. Tento způsob výrazně zlepší čitelnost příkazu `route`; dokonce můžeme vynechat i argument `-net`, protože příkaz `route` již bude vědět, že adresa **191.72.1.0** označuje síť.

```
# route add brew-net
```

Po skončení základních konfiguračních kroků se budete chtít ujistit, že vaše ethernetové rozhraní opravdu funguje správně. Z Ethernetu si vyberte hostitele, například **vlager,** a napište:

```
# ping vlager
PING vlager: 64 byte packets
64 bytes from 191.72.1.1: icmp_seq=0. time=11. ms
64 bytes from 191.72.1.1: icmp_seq=1. time=7. ms
64 bytes from 191.72.1.1: icmp_seq=2. time=12. ms
64 bytes from 191.72.1.1: icmp_seq=3. time=3. ms
^C

----vstout.vbrew.com PING Statistics
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms) min/avg/max = 3/8/12
```

Pokud neuvidíte výstup podobný výše uvedenému, bude zřejmě něco v nepořádku. Dojde-li k neobvyklé ztrátě paketů, bude zřejmě chyba na straně hardwaru, například špatné nebo chybějící terminátory apod. Pokud nepřijmete vůbec žádné pakety, měli byste si zkontrolovat konfiguraci rozhraní pomocí příkazu `netstat`. Statistika paketů zobrazená příkazem `ifconfig` by vám měla říci, zda vůbec byly nějaké pakety poslány rozhraní. Máte-li také přístup ke vzdálenému hostiteli, měli byste zkontrolovat statistiku rozhraní i na tomto počítači. Takto můžete přesně určit, kde se pakety ztratily. Kromě toho byste si měli pomocí příkazu `route` zobrazit směrovací informace a zkontrolovat, zda mají oba hostitelé správnou položku směrování. V případě, že je příkaz `route` vyvolán bez jakýchkoliv dalších argumentů, vytiskne kompletní směrovací tabulku jádra (parametr `-n` pouze zobrazí adresy v tečkové notaci, bez uvedení parametru zobrazí příkaz `route` název hostitele):

```
# route -n
Kernel routing table
Destination  Gateway  Genmask          Flags Metric Ref  Use  Iface
127.0.0.1    *        255.255.255.255  UH    1      0    112  lo
191.72.1.0   *        255.255.255.0    U     1      0    10   eth0
```

Podrobný popis těchto polí najdete dále ve stati Kontrola pomocí příkazu `netstat`. Sloupec `Flag` obsahuje seznam všech symbolů příslušejících každému rozhraní. Aktivní rozhraní mají vždy nastaven symbol `U`, symbol `H` naznačuje, že cílová adresa patří hostiteli. Je-li symbol `H` zobrazen u směrování, míníme tím směrování sítě, pak je nutné společně s příkazem `route` specifikovat i volbu `-net`. To, zda se zadané směrování vůbec používá, zjistíte na základě pole `Use`, které najdete ve druhém sloupci zprava. Jeho hodnota by se mezi dvěma voláními příkazu `ping` měla zvyšovat.

5.7.3 Směrování pomocí brány

V předešlé stati jsme probrali pouze nastavení hostitele v jediné síti Ethernet. Poměrně často se však setkáváme se sítěmi, které jsou vzájemně propojeny pomocí bran. Tyto brány mohou spojovat dva nebo více Ethernetů, ale mohou také poskytovat spojení s okolním světem, a samozřejmě také s Internetem. Abyste mohli využít služeb bran, musíte síťové hladině poskytnout doplňující směrovací informace.

Například Ethernety ve společnostech Virtual Brewery a Virtual Winery jsou propojeny pomocí takovéto brány, tuto funkci konkrétně zastává hostitel **vlager**. Předpokládáme, že hostitel **vlager** již byl nakonfigurován, tudíž nám zbývá přidat pouze další položku do směrovací tabulky hostitele **vstout**, která řekne jádru operačního systému, že všichni hostitelé sítě virtuální vinárny jsou dosažitelní přes bránu **vlager**. Patřičným zaklínadlem příkazu `route` je klíčové slovo `gw`, které sdělí příkazu `route`, že následující argument označuje bránu.

```
# route add wine-net gw vlager
```

Samozřejmě, že každý hostitel v síti vinárny, se kterým chcete hovořit, musí mít odpovídající směrovací položku pro síť pivovaru. V opačném případě byste mohli posílat data pouze z hostitele **vstout** na hostitele **vbardolino**, avšak veškeré odpovědi vrácené hostitelem **vbardolino** by putovaly do koše.

Tento příklad popisuje pouze bránu, která přenáší pakety mezi dvěma izolovanými Ethernety. Nyní budeme předpokládat, že brána **vlager** je také připojena k Internetu (například pomocí doplňujícího spojení s využitím protokolu SLIP). V tom případě budeme po bráně **vlager** požadovat, aby se starala o datagramy putující do *libovolné* cílové sítě, která není identická se sítí pivovaru. To lze provést tak, že označíme bránu **vlager** jako implicitní bránu pro hostitele **vstout**:

```
# route add default gw vlager
```

Název sítě **default** je zkratkou adresy **0.0.0.0**, která označuje implicitní směrování. Tento název nemusíte přidávat do souboru `/etc/networks`, protože je zabudován v příkazu `route`.

Když po aplikaci příkazu `ping` na hostitele, který se nachází za jednou nebo více branami, dojde k velké ztrátě paketů, může to ukazovat na příliš přeplněnou síť. Ztráta paketů není ani tak způsobena odlišnostmi zařízení, jako spíše dočasným špičkovým zatížením předávajících hostitelů, což způsobuje z jejich strany prodlevy případně ztrátu příchozích datagramů.

5.7.4 Konfigurace brány

Konfigurace počítače pro posílání paketů mezi dvěma Ethernety je poměrně přehledná. Budeme předpokládat, že jsme zpět u brány **vlager**, která je vybavena dvěma ethernetovými kartami, z nichž každá je spojena s jednou ze dvou sítí. Pak stačí odděleně nakonfigurovat obě rozhraní, přidělit jim jejich vlastní IP-adresy a to je vše.

Je rozumné přidat informace o obou rozhraních do souboru *hosts*, protože takto získáme pro tato dvě rozhraní šikovné názvy:

```
191.72.1.1       vlager       vlager.vbrew.com
191.72.1.1       vlager-if1
191.72.2.1       vlager-if2
```

Sekvence příkazů `pro nastavení obou rozhraní je následující:`

```
# ifconfig eth0 vlager-if1
# ifconfig eth1 vlager-if2
# route add brew-net
# route add wine-net
```

5.7.5 Rozhraní PLIP

Pokud ke spojení dvou počítačů používáte spojení pomocí protokolu PLIP, budou jednotlivá nastavení mírně odlišná od nastavení použitých při konfiguraci Ethernetu. Spojení pomocí protokolu PLIP se také někdy nazývá spojením typu *point-to-point*, protože se na rozdíl od většiny sítí týká pouze dvou hostitelů („bodů“).

Jako příklad budeme uvažovat počítač v provedení laptop, který mají někteří zaměstnanci společnosti Virtual Brewery. Ten je spojen s bránou **vlager** prostřednictvím protokolu PLIP. Vlastní laptop se nazývá **vlite** a má pouze jediný paralelní port. Při zavádění bude tento port registrován jako rozhraní *plip1*. Abyste spojení aktivovali, musíte nakonfigurovat rozhraní *plip1*[6] za pomoci následujících příkazů:

```
# ifconfig plip1 vlite pointopoint vlager
# route add default gw vlager
```

6 Všimněte si, že pointopoint není překlepem. Skutečně se to takto hláskuje.

První příkaz nastavuje rozhraní a sděluje jádru operačního systému, že se jedná o spojení typu point-to-point, u něhož má vzdálená strana přidělenu adresu **vlager**. Druhý řádek nainstaluje implicitní směrování s využitím hostitele **vlager** jako brány. Na straně brány **vlager** je nutná podobná konfigurace, která spojení zaktivuje (volání příkazu `route` není zapotřebí):

```
ifconfig plip1 vlageg pointopoint vlite
```

Zajímavé je, že rozhraní *plip1* brány **vlager** nemusí mít zvláštní IP-adresu, ale může mu být také přidělena adresa **191.72.1.1**.[7]

Nyní máme vyřešeno směrování z počítače laptop do sítě pivovaru; ale stále ještě nám chybí směrovací cesta z libovolného hostitele pivovaru na počítač **vlite**. Jeden z poměrně nešikovných způsobů spočívá v přidání konkrétního směrování do každé směrovací tabulky hostitele, v níž přiřadíte hostiteli **vlager** funkci brány k hostiteli **vlite**.

```
# route add vlite gw vlager
```

Stojíte-li již tváří v tvář problému s dočasným směrováním, je mnohem výhodnější použít dynamické směrování. Jeden z možných způsobů spočívá v použití směrovacího démona `gated`, který musí být nainstalován na každém hostiteli v síti, aby si mohli dynamicky předávat směrovací informace. Daleko nejjednodušší způsob však počítá s využitím *proxy* ARP. Při nainstalovaném proxy ARP bude brána **vlager** odpovídat na libovolné dotazy ohledně hostitele **vlite** zasláním své vlastní ethernetové adresy. Výsledkem této funkce bude přivolání všech paketů určených pro hostitele **vlite** na bránu **vlager**, která je potom pošle na laptop. K proxy ARP se vrátíme ve stati Kontrola tabulek ARP dále.

Budoucí verze balíku Net-3 bude obsahovat nástroj zvaný `plipconfig`, který bude umožňovat nastavení IRQ používaného paralelního portu. Později by mohl být nahrazen obecnějším příkazem `ifconfig`.

5.7.6 Rozhraní SLIP a PPP

Ačkoliv jsou spojení typu SLIP nebo PPP pouze jednoduchými spojeními typu point-to-point, podobně jako u spojení typu PLIP je vhodné v souvislosti s nimi uvést některé informace. Uskutečnění spojení typu SLIP obvykle vyžaduje zavolání vzdáleného počítače pomocí modemu a nastavení sériové linky do režimu SLIP. Podobně se používá i protokol PPP. Nástroje potřebné pro konfiguraci spojení SLIP a PPP budou popsány v kapitolách 7 a 8.

[7] Avšak z důvodů opatrnosti byste měli konfigurovat spojení PLIP nebo SLIP až poté, co máte kompletně nastaveny ethernetové položky ve směrovací tabulce. V opačném případě by u některých starších jader operačního systému mohlo vaše síťové směrování skončit na spojení point-to-point.

5.7.7 Fiktivní rozhraní

Fiktivní rozhraní je skutečně trošku exotické, nicméně je docela užitečné. Jeho hlavní význam souvisí se samostatnými hostiteli a počítači, jejichž jediné síťové IP-spojení je připojení pomocí modemu. Později se ve skutečnosti začalo mnohem více prosazovat i u samostatných hostitelů.

Dilema u samostatných hostitelů spočívá v tom, že mají aktivní pouze jediné síťové zařízení, a to konkrétně zpětnovazebné zařízení. To má obvykle přidělenu adresu **127.0.0.1**. Nicméně v určitých situacích potřebujete posílat data na oficiální IP-adresu místního hostitele. Například, uvažujme laptop **vlite**, jenž byl během trvání tohoto příkladu odpojen od všech sítí. Aplikace na laptopu **vlite** může chtít poslat nějaká data na vlastního hostitele **vlite**. Prohledá-li na počítači **vlite** soubor `/etc/hosts/`, obdrží IP-adresu **191.72.1.65**, tudíž se aplikace pokusí na tuto adresu poslat data. Protože jediným aktivním rozhraním je pouze zpětnovazebné rozhraní, nemůže jádro operačního systému vědět, že se tato adresa ve skutečnosti vztahuje na něj! Následně jádro operačního systému zničí datagram a vrátí aplikaci chybovou hlášku.

V tomto bodě vstupuje do hry fiktivní rozhraní. Vyřeší dilema tím, že bude sloužit jako pozměněné zpětnovazebné rozhraní. V případě laptopu **vlite** byste mu měli přidělit adresu **191.72.1.65** a přidat směrování hostitele na tuto adresu. Každý datagram pro adresu **191.72.1.65** pak bude doručen lokálně. Správné volání vypadá následovně:

```
# ifconfig dummy vlite
# route add vlite
```

5.8 Vše o příkazu ifconfig

Příkaz `ifconfig` má mnohem více parametrů, než jsme si zatím řekli. Jeho volání v klasické podobě vypadá následovně:

```
ifconfig interface [[-net|-host] address [parameters]]
```

Parametr `interface` představuje název rozhraní, parametr `address` představuje IP-adresu přidělenou danému rozhraní. Může to být buď IP-adresa v tečkové notaci, nebo název, který příkaz `ifconfig` vyhledá v souborech `/etc/hosts` a `/etc/networks`. Volby `-net` a `-host` přinutí příkaz `ifconfig`, aby považoval adresu za číslo sítě, resp. za adresu hostitele.

Je-li příkaz `ifconfig` vyvolán pouze s názvem rozhraní, zobrazí konfiguraci příslušného rozhraní. Je-li spuštěn bez parametrů, zobrazí všechny již dříve nakonfigurovaná rozhraní; volba `-a` donutí příkaz zobrazit i neaktivní rozhraní. Vzorové vyvolání příkazu `ifconfig` pro ethernetové rozhraní *eth0* by mohlo vypadat asi takto:

```
# ifconfig eth0
eth0   Link encap 10Mbps Ehternet  HWaddr 00:00:C0:90:B3:42
       inet addr 191.72.1.2 Bcast 191.72.1.255 Mask 255.255.255.0
       UP BROADCAST RUNNING MTU 1500 Metric 0
       RX packets 3136 errors 217 dropped 7 overrun 26
       TX packets 1752 errors 25 dropped 0 overrun 0
```

Pole `MTU` a `Metric` ukazují aktuální MTU a metrickou hodnotu pro dané rozhraní. Metrickou hodnotu tradičně používají některé operační systémy k výpočtu váhy daného směrování. Linux zatím tuto hodnotu nevyužívá, ale definuje ji z důvodu kompatibility.

Řádky RX a TX ukazují, kolik paketů bylo bezchybně přijato nebo vysláno, ke kolika chybám došlo, kolik paketů bylo zahozeno pravděpodobně z důvodu nedostatku paměti a kolik paketů se ztratilo kvůli přetížení. K přetížení přijímače obvykle dojde v případě, kdy pakety přicházejí rychleji, než stačí jádro operačního systému obsloužit poslední přerušení. Hodnoty příznaků zobrazené příkazem `ifconfig` zhruba odpovídají názvům jeho voleb na příkazovém řádku; budou vysvětleny dále.

Následuje seznam parametrů, které rozeznává příkaz `ifconfig`. Odpovídající názvy symbolů jsou uvedeny v kulatých závorkách. Volby, které zapínají určitou volbu, umožňují i její vypnutí. Toho dosáhnete vložením symbolu mínus (-) před název příslušné volby.

`up` — Označí rozhraní jako „aktivní“, tj. přístupné pro IP-vrstvu. Tato volba předpokládá, že je spolu s ní uvedena na příkazovém řádku i adresa `address`. Lze ji také použít k opětovnému povolení rozhraní, které bylo dočasně zakázáno pomocí volby `down`.

(Této volbě odpovídají symboly UP a RUNNING.)

`down` — Označí rozhraní jako „neaktivní“, tj. nepřístupné pro IP-vrstvu. Tato volba efektivně znemožní jakýkoliv IP-provoz přes dané rozhraní. Všimněte si, že tato volba automaticky nesmaže všechna směrovací data, která používají dané rozhraní. Pokud toto rozhraní trvale znepřístupníte, měli byste tato směrovací data vymazat a je-li to možné, doplnit je o alternativní směrování.

`netmask` *mask* — Tato volba přidělí masku podsítě, kterou bude rozhraní používat. Může být zadána jako 32bitové hexadecimální číslo s předponou `0x` nebo jako čísla desítkové soustavy oddělená tečkou.

`pointopoint` *address*

Tato volba se používá u spojení IP typu point-to-point, které vyžaduje pouze dva hostitele. Tato volba je nutná například při konfiguraci rozhraní SLIP nebo PLIP.

(Byla-li nastavena adresa point-to-point, zobrazí příkaz `ifconfig` symbol `POINTOPOINT`.)

`broadcast` *address*

Vysílací (broadcast) adresa je obvykle vytvořena z čísla sítě nastavením všech bitů části hostitele. Některé implementace protokolu IP používají odlišné schéma; tato volba je zde proto, aby se vysílací adresa přizpůsobila těmto podivným prostředím.

(Je-li nastavena vysílací adresa, zobrazí příkaz `ifconfig` symbol `BROADCAST`.)

`metric` *number* Tato volba slouží k přidělení metrické hodnoty položce směrovací tabulky vytvořené pro dané rozhraní. Tuto metriku používá směrovací informační protokol (RIP) k vytvoření směrovacích tabulek sítí.[8] Implicitní hodnota metriky používaná příkazem `ifconfig` je rovna nule. Pokud nepoužíváte démona RIP, je vám tato volba k ničemu; pokud ano, budete jen zřídka potřebovat tuto hodnotu měnit.

`mtu` *bytes* Tato volba nastavuje tzv. maximální přenosovou jednotku (Maximal Transmission Unit – MTU), která představuje maximální počet oktetů, o něž se může rozhraní postarat při jedné transakci. U sítí typu Ethernet je hodnota MTU implicitně nastavena na 1 500; u rozhraní typu SLIP je MTU nastavena na hodnotu 296.

`arp` Toto je volba typická pro sítě, jako je Ethernet nebo packet radio. Povoluje použití ARP, protokolu pro rozlišení adres, k detekci fyzických adres hostitelů, kteří jsou připojeni k síti. U těchto sítí je tato volba implicitně povolena.

(Je-li protokol ARP zakázán, zobrazí příkaz `ifconfig` symbol `NOARP`.)

`-arp` Na tomto rozhraní zakáže použití protokolu ARP.

[8] Protokol RIP vybírá na základě „délky“ cesty k danému hostiteli optimální směrování. Délku zjistí sečtením metrických hodnot každého spojení mezi hostiteli. Implicitně má skok hodnotu 1, ale ve skutečnosti to může být libovolné celé číslo menší než 16. (Délka směrování 16 odpovídá nekonečné vzdálenosti. Taková směrování jsou považována za nepoužitelná.) Parametr metriky nastavuje váhu skoku, která je potom vysílána směrovacím démonem.

`promisc` Přepne rozhraní do „promiskuitního“ módu. U většiny sítí to bude znamenat, že dané rozhraní bude přijímat všechny pakety, bez ohledu na to, zda byly určeny pro jiného hostitele či nikoliv. Tato volba umožní analyzovat síťový provoz pomocí filtrování paketů (tzv. *sledování Ethernetu*). Je to dobrý způsob, jak vychytat síťové problémy, které by jinak bylo těžké vystopovat.

Na druhou stranu umožní tato volba útočníkům kromě jiných věcí i zjištění potřebných hesel z vašeho síťového provozu. Jednou z ochran proti tomuto typu útoku je všeobecné zakázání pouhého zastrčení svého počítače do vaší ethernetové zásuvky. Další možností je používat bezpečné ověřovací protokoly, jako například Kerberos, nebo přihlašovací balík SRA.[9*]

(Této volbě odpovídá symbol `PROMISC`.)

`-promisc` Tato volba vypne promiskuitní mód.

`allmulti` Multicast-adresy jsou určitým druhem vysílání dat skupině hostitelů, kteří nutně nemusí ležet v jedné podsíti

(Tato volba odpovídá symbolu `ALLMULTI`.)

`-allmulti` Tato volba vypne multicast-adresy.

5.9 Kontrola pomocí příkazu netstat

Nyní se podíváme na užitečný nástroj pro kontrolu konfigurace sítě a síťové aktivity. Nazývá se `netstat` a ve skutečnosti jde spíše o sbírku několika nástrojů shrnutých dohromady. V následujících statích si probereme každou z jeho funkcí.

5.9.1 Zobrazení směrovací tabulky

Spustíte-li příkaz `netstat` s parametrem `-r`, zobrazí se směrovací tabulka jádra operačního systému stejným způsobem, jako tomu bylo u příkazu `route`. Na hostiteli **vstout** se zobrazí:

```
# netstat -nr
Kernel routing table
```

9 Balík SRA můžete získat na serveru **ftp.tamu.edu** z adresáře */pub/sec/TAMU*.

* Poznámka korektora: V poslední době je nejpoužívanějším balíkem pro vzdálené přihlášení program `ssh`. Více informací naleznete na adrese *http://www.ssh.fi*.

```
Destination  Gateway     Genmask          Flags Metric Ref  Use   Iface
127.0.0.     *           255.255.255.255  UH    1      0    50    lo
191.72.1.0   *           255.255.255.0    U     1      0    478   eth0
191.72.2.0   191.72.1.1  255.255.255.0    UGN   1      0    250   eth0
```

Volba `-n` způsobí, že příkaz `netstat` zobrazí adresy jako čísla IP oddělené tečkami, a ne jako symbolické názvy hostitelů a sítí. To je užitečné, chcete-li se vyhnout vyhledávání adres po síti (například na DNS nebo NIS-serveru).

Druhý sloupec výstupu příkazu `netstat` zobrazuje bránu, na kterou ukazují směrovací data. Není-li použita žádná brána, zobrazí se symbol hvězdičky. Sloupec tři ukazuje „všeobecnost" směrování. Je-li zadána IP-adresa, pro kterou se má vyhledat správné směrování, projde jádro systému všechna data směrovací tabulky, na adresu a všeobecnou masku aplikuje bitově orientovanou operaci AND a výsledek porovná s cílem směrování.

Čtvrtý sloupec zobrazuje různé symboly, které popisují dané směrování:

G Směrování používá bránu.

U Používané rozhraní je povolené.

H Daným směrováním může být dosažitelný pouze jediný hostitel. To je například případ dat pro zpětnovazebné rozhraní **127.0.0.1**.

D Tento symbol je zobrazen, pokud byla data vygenerována ICMP-zprávou o přesměrování (viz stať 2.5).

M Tento symbol je zobrazen, pokud byla položka v tabulce změněna ICMP zprávou o přesměrování.

Sloupec `Ref` ve výstupu příkazu `netstat` zobrazuje počet odkazů na dané směrování, to znamená, kolik dalších směrování (například přes brány) spoléhá na přítomnost daného směrování. Poslední dva sloupce zobrazují, kolikrát byla použita směrovací data a dále rozhraní, kterými při doručování procházejí datagramy.

5.9.2 Zobrazování statistik rozhraní

Je-li příkaz `netstat` spuštěn s parametrem `-i`, zobrazí se statistiky pro aktuálně nakonfigurovaná síťová rozhraní. Je-li připojen i parametr `-a`, vypíší se všechna zařízení přítomná v jádru operačního systému, nejenom ta, která již byla nakonfigurována. Na hostiteli **vstout** by výstup příkazu `netstat` vypadal následovně:

```
$ netstat -i
Kernel Interface Table
Iface MTU  Met RX-OK   RX-ERR RX-DRP RX-OVR TX-OK  TX-ERR TX-DRP TX-OVR Flag
lo    0    0   3185    0      0      0      3185    0      0      0      BLRU
eth0  1500 0   972633  17     20     120    628711 217    0      0      BRU
```

Pole `MTU` a `Met` ukazují aktuální hodnoty MTU a metriky daného rozhraní. Sloupce `RX` a `TX` ukazují, kolik paketů bylo bezchybně přijato nebo vysláno (RX-OK/TX-OK), kolik bylo poškozeno (RX-ERR/TX-ERR), kolik bylo zahozeno (RX-DRP/TX-DRP) a kolik se ztratilo z důvodu přetížení (RX-OVR/TX-OVR).

Poslední sloupec zobrazuje symboly nastavené u daného zařízení. Jsou to jednoznakové ekvivalenty dlouhých názvů příznaků, které se vypisují při zobrazení konfigurace rozhraní pomocí příkazu `ifconfig`.

B Byla nastavena vysílací adresa.

L Dané rozhraní je zpětnovazebné rozhraní.

M Jsou přijímány všechny pakety (promiskuitní mód).

O Pro dané rozhraní je vypnut protokol ARP.

P Toto spojení je typu point-to-point.

R Rozhraní právě běží.

U Rozhraní je povoleno.

5.9.3 Zobrazení propojení

Příkaz `netstat` podporuje skupinu voleb pro zobrazení aktivních nebo pasivních socketů. Volby `-t`, `-u`, `-w` a `-x` ukazují aktivní TCP, UDP, RAW nebo unixová socketová spojení. Pokud k nim doplníte i parametr `-a`, budou zobrazeny i sockety čekající na spojení (například při naslouchání). Tato kombinace parametrů vám poskytne úplný výpis všech serverů, které právě běží ve vašem systému.

Při použití příkazu `netstat -ta` se na hostiteli **vlager** zobrazí následující výpis:

```
$ netstat -ta
Active Internet connections
Proto Recv-Q Send-Q Local Address  Foreign Address    (State)
tcp   0      0      *:domain       *:*                LISTEN
tcp   0      0      *:time         *:*                LISTEN
tcp   0      0      *:smtp         *:*                LISTEN
tcp   0      0      vlager:smtp    vstout:1040        ESTABLISHED
tcp   0      0      *:telnet       *:*                LISTEN
tcp   0      0      localhost:1046 vbardolino:telnet  ESTABLISHED
tcp   0      0      *:chargen      *:*                LISTEN
tcp   0      0      *:daytime      *:*                LISTEN
tcp   0      0      *:discard      *:*                LISTEN
tcp   0      0      *:echo         *:*                LISTEN
tcp   0      0      *:shell        *:*                LISTEN
tcp   0      0      *:login        *:*                LISTEN
```

Tento výpis ukazuje, že většina serverů čeká na příchozí spojení. Nicméně čtvrtý řádek ukazuje příchozí spojení typu SMTP z hostitele **vstout** a šestý řádek vám sděluje, že existuje výstupní spojení typu telnet s hostitelem **vbardolino**.[10]

Pokud bychom použili pouze argument `-a`, zobrazí se všechny sockety ze všech rodin.

5.10 Kontrola tabulek ARP

V určitých situacích je vhodné si prohlédnout, případně změnit obsah tabulek ARP jádra systému. Například máte-li podezření, že příčinou občasných síťových problémů je duplicitní internetová adresa. Pro takovéto situace byl vytvořen nástroj `arp`. Jeho volby příkazové řádky jsou následující:

```
arp [-v] [-t hwtype] -a [hostname]
arp [-v] [-t hwtype] -s hostname hwaddr
arp [-v] -d hostname [hostname...]
```

Všechny argumenty názvu hostitele `hostname` jsou buďto symbolické názvy hostitelů, nebo IP-adresy respektující tečkovou notaci.

[10] Zda je dané spojení výstupní, zjistíte podle jeho čísla portu. Číslo portu bude pro *volajícího* hostitele vždy přirozené, zatímco u volaného hostitele se použije známý obslužný port, pro nějž příkaz `netstat` používá symbolický název nacházející se v souboru `/etc/services`.

První typ volání příkazu `arp` zobrazí pro danou IP-adresu nebo pro daného hostitele položku ARP. V případě, že nebyl zadán název hostitele `hostname`, zobrazí se položky ARP pro všechny známé hostitele. Například vyvolání příkazu `arp` na hostiteli **vlager** může vypadat takto:

```
# arp -a
IP address          HW type                 HW address
191.72.1.3          10Mbps Ethernet         00:00:C0:5A:42:C1
191.72.1.2          10Mbps Ethernet         00:00:C0:90:B3:42
191.72.2.4          10Mbps Ethernet         00:00:C0:04:69:AA
```

Zobrazí se ethernetové adresy hostitelů **vlager**, **vstout** a **vale**.

Pomocí volby `-t` můžete omezit výpis pouze na zadaný typ hardwaru. Jím může být *ether*, *ax25* nebo *pronet*, což odpovídá 10Mbps Ethernetu, AMPR AX.25, resp. zařízení IEEE 802.5 Token Ring.

Parametr `-s` slouží k permanentnímu přidání ethernetové adresy názvu hostitele `hostname` do tabulek ARP. Argument `hwaddr` určuje hardwarovou adresu, u níž se implicitně předpokládá, že jde o ethernetovou adresu určenou šesti hexadecimálními bajty oddělenými dvojtečkou. Za pomoci volby `-t` můžete také nastavit hardwarové adresy pro jiné typy hardwaru.

Někdy bude nutné manuální doplnění IP-adresy do tabulky ARP, například když z nějakých příčin selžou na vzdáleném hostiteli dotazy ARP, k čemuž může dojít, je-li na vzdáleném hostiteli chybný ovladač ARP, nebo když v síti existuje další hostitel, který se chybně identifikuje IP-adresou vzdáleného hostitele. Zapsání IP-adresy napevno do tabulky ARP je opatření (velice drastické), kterým se chráníte proti hostitelům z vašeho Ethernetu, jež se vydávají za někoho jiného.

Použijete-li při spuštění příkazu `arp` parametr `-d`, smažou se všechna data ARP, která se vztahují k danému hostiteli. Pomocí tohoto parametru můžete rozhraní přinutit k tomu, aby se znovu pokusilo pomocí dotazu získat ethernetovou adresu odpovídající dané IP-adrese. To je užitečné v případě, kdy špatně nakonfigurovaný systém vysílá špatné informace ARP (samozřejmě předtím musíte chybného hostitele znovu zkonfigurovat).

Volba `-s` slouží k implementaci techniky *proxy* ARP. Je to speciální technika, kdy se hostitel, například **gate**, chová vůči dalšímu hostiteli **fnord** jako brána a předstírá, že se obě adresy vztahují ke stejnému hostiteli, konkrétně k hostiteli **gate**. Provede to tak, že zveřejní ARP položku hostitele **fnord**, která bude ukazovat na své vlastní ethernetové rozhraní. Když nyní nějaký hostitel pošle dotaz ARP na hostitele **fnord**, hostitel **gate** vrátí odpověď, která bude obsahovat jeho vlastní ethernetovou adresu. Potom pošle dotazující se hostitel všechny datagramy na hostitele **gate**, který je následně zašle hostiteli **fnord**.

Tyto záměny jsou nutné například v případě, kdy chcete přistupovat k hostiteli **fnord** z počítače s operačním systémem DOS, který nemá zcela korektní implementaci TCP s dobrým směrováním. Při použití techniky proxy ARP se bude hostitel jevit počítači s operačním systémem DOS jako by byl **fnord** v místní podsíti, takže počítač s operačním systémem DOS nemusí vůbec umět směrovat pomocí brány.

Další velice užitečnou aplikací techniky proxy ARP je případ, kdy se jeden z vašich hostitelů chová jako brána vůči nějakému jinému hostiteli jen dočasně, například při připojení pomocí modemu. V předešlém příkladu jsme se již setkali s laptopem **vlite**, který je občas spojován s bránou **vlager** spojením typu PLIP. Samozřejmě, že tento způsob bude fungovat pouze v případě, kdy je adresa hostitele, na kterém chcete provozovat techniku proxy ARP, ve stejné podsíti IP jako vaše brána. Například hostitel **vstout** by mohl používat techniku proxy ARP pro libovolného hostitele sítě pivovaru (**191.72.1.0**), ale nikdy pro hostitele z podsítě vinárny (**191.72.2.0**).

Správné vyvolání příkazu `arp`, kdy bude hostiteli **fnord** poskytnuta technika proxy ARP, je uvedeno níže; samozřejmě, že předaná ethernetová adresa musí odpovídat hostiteli **gate**.

```
# arp -s fnord 00:00:c0:a1:42:e0 pub
```

Položku proxy ARP můžete opět odstranit následujícím způsobem:

```
# arp -d fnord
```

5.11 Budoucnost

Síťové služby Linuxu se stále vyvíjí. Velké změny v hladině jádra operačního systému s sebou přinesou pružné konfigurační schéma, které vám umožní konfigurovat síťové zařízení i za provozu. Například příkaz `ifconfig` bude akceptovat argumenty nastavující IRQ a kanál DMA.

V blízké budoucnosti se počítá s doplněním příznaku `mtu`, který nastaví pro konkrétní směrování maximální přenosovou jednotku. Tato MTU specifická pro dané směrování potlačí MTU specifikované pro dané rozhraní. Tuto volbu budete obvykle používat u směrování přes bránu, kde bude spojení mezi bránou a cílovým hostitelem vyžadovat velmi malou hodnotu MTU. Dejme tomu, že je třeba hostitel **wonderer** připojen k bráně **vlager** spojením typu SLIP. Když budete posílat data z hostitele **vstout** hostiteli **wonderer**, bude síťová vrstva hostitele **wonderer** používat pakety o velikosti až 1 500 bajtů, protože jsou tyto pakety posílány po Ethernetu. Na druhé straně však pracuje spojení typu SLIP s MTU o velikosti 296 bajtů, takže síťová hladina hostitele **vlager** musí rozdělit IP-pakety na menší části, které by nepřesa-

hovaly 296 bajtů. Kdyby bylo naopak možné nastavit směrování na hostiteli **vstout** tak, aby se od začátku používalo MTU o velikosti 296 bajtů, pak bychom se této poměrně náročné fragmentaci vyhnuli:

```
# route add wanderer gw vlager mtu 296
```

Všimněte si, že volba `mtu` vám umožní selektivně vrátit účinek opatření, které chápe podsítě jako místní (‘Subnets Are Local‘ Policy – SNARL). Toto opatření je konfigurační volbou jádra operačního systému a je popsáno v kapitole 3.

Kapitola

6

Konfigurace resolveru a jmenných služeb

Ve 2. kapitole jsme si řekli, že sítě na bázi protokolu TCP/IP mohou používat různá schémata konverze názvů na adresy. Nejjednodušším způsobem, který však nevyužívá výhody rozdělení jmenného prostoru do jednotlivých zón, je tabulka hostitelů, která je uložena v souboru `/etc/hosts`. Tento způsob je vhodný pouze pro malé lokální sítě, které spravuje jediný správce a které nepoužívají žádnou IP-komunikaci s okolním světem. Formát souboru `hosts` byl podrobně popsán v kapitole 5.

Další možností je použití služby BIND – Berkeley Internet Name Domain Service – ta přiděluje názvy hostitelů jednotlivým IP-adresám. Konfigurace služby BIND je skutečným oříškem, ale jakmile ji jednou zvládnete, bude pro vás začlenění případných změn v síťové topologii velmi snadné. V systému Linux, stejně jako u dalších unixových systémů, poskytuje jmenné služby program `named`. Program `named` načte při startu několik hlavních souborů do své vyrovnávací paměti a dále bude čekat na dotazy od vzdálených nebo místních uživatelských procesů. Existuje několik způsobů, jak nastavit službu BIND a zdaleka ne všechny vyžadují spuštěný jmenný server na každém hostiteli.

V této kapitole by bylo možné uvést mnohem více informací, než jen hrubý náčrt způsobu, jakým lze provozovat jmenný server. Hodláte-li používat službu BIND v prostředí s většími sítěmi než jen LAN a pravděpodobně i s připojením na Internet, pak byste si měli sehnat nějakou kvalitní knihu pojednávající o službě BIND, například knížku „DNS and BIND“ od Cricketa Liua (viz [AlbitzLiu92]). Aktuální informace najdete také v poznámkách vydávaných společně se zdrojovými soubory služby BIND. Otázkám systému DNS se věnuje i konference nazvaná **comp.protocols.tcp-ip.domains**.

6.1 Knihovna resolveru

Hovoříme-li o „resolveru“, nemáme namysli žádnou speciální aplikaci, ale spíše odkazy do *knihovny resolveru*, což je skupina funkcí nacházející se ve standardní knihovně jazyka C. Centrálními rutinami jsou procedury *gethostbyname(2)* a *gethostbyaddr(2)*, které umí vyhledat všechny IP-adresy náležející danému hostiteli, a versa vice. Mohou být nastaveny tak, aby využívaly soubor `hosts`, aby se dotazovaly na počet jmenných serverů nebo aby používaly databázi `hosts` služby NIS (síťové informační služby). Další aplikace, jako je například `smail`, mohou pro tyto účely obsahovat různé ovladače, které však vyžadují speciální péči.

6.1.1 Soubor host.conf

Centrálním souborem, který řídí nastavení resolveru, je soubor `host.conf`. Nachází se v adresáři `/etc` a sděluje resolveru, jakou by měl použít službu a v jakém pořadí.

Jednotlivé volby musí být v souboru `host.conf` uvedeny na samostatných řádcích. Pole mohou být oddělena bílými mezerami (což jsou mezery nebo tabulátory). Symbol # označuje komentář, který končí u dalšího znaku nové řádky.

K dispozici jsou následující volby:

order Tato volba určuje pořadí, ve kterém budou zkoušeny jednotlivé služby resolveru. Přípustné volby jsou *bind* pro použití dotazů na jmenný server, *hosts* pro vyhledávání v souboru `/etc/hosts` a *nis* pro vyhledávání pomocí služby NIS. Může být zadána buď jedna nebo více voleb. Pořadí, ve kterém jsou tyto volby uvedeny, bude určovat pořadí, v kterém budou zkoušeny s nimi související služby.

multi Argumenty této volby jsou *on* nebo *off*. Určuje, zda může mít hostitel v souboru `/etc/hosts` několik IP-adres. Adresy tohoto typu se někdy označují také jako multihomed adresy. U dotazů systému DNS nebo NIS nemá tento přepínač žádný význam.

nospoof V minulé kapitole jsme si řekli, že systém DNS umí najít název hostitele náležející dané IP-adrese. K tomu používá doménu **in-addr.arpa**. Snahy jmenných serverů navrátit chybný název hostitele se označují jako tzv. „*spoofing*“. Obrana proti možným chybám spočívá v konfiguraci resolveru takovým způsobem, aby zjišťoval, zda je původní adresa skutečně spjatá se získaným názvem hostitele. Pokud nikoliv, je název odmítnut a vrácena chybová zpráva. Toto chování se zapíná pomocí volby *nospoof on*.

alert Tato volba má argumenty *on* nebo *off*. Je-li zapnutá, pak veškeré pokusy o spoofing (viz výše) budou zapsány do souboru *syslog*.

trim Tato volba má jako argument název domény, která bude před vyhledáváním z názvů hostitelů odstraněna. Volba je užitečná u těch položek v souboru *hosts*, u kterých jste zadali pouze názvy hostitelů bez místní domény. Při vyhledání lokálního hostitele, jenž má k sobě připojenu místní doménu, bude tato doména odstraněna, takže vyhledání v souboru `/etc/hosts` bude úspěšné.

Volby *trim* lze použít vícekrát, takže lze vašeho hostitele považovat za místního hostitele vůči několika doménám.

Následuje vzorový soubor pro bránu **vlager**:

```
# /etc/host.conf
# Používáme named, NIS (zatím) ne
order    bind hosts
# Jeden hostitel může mít více adres
multi    on
# Ochrana proti spoofingu
nospoof on
# Odstraňuji lokální doménu
trim     vbrew.com.
```

6.1.2 Proměnné pro prostředí resolveru

Nastavení v souboru `host.conf` lze potlačit pomocí několika proměnných prostředí resolveru.

Následuje výpis těchto proměnných:

`RESOLV_HOST_CONF`
Určuje soubor, který bude načten místo souboru `/etc/host.conf`.

`RESOLV_SERV_ORDER`
Potlačí volbu *order* v souboru *host.conf*. Službami mohou být *hosts*, *bind* nebo *nis*. Odděleny mohou být mezerou, čárkou nebo středníkem.

`RESOLV_SPOOF_CHECK`
Určí opatření, která budou použita proti spoofingu. Při hodnotě *off* je kontrola úplně vypnutá. Hodnoty *warn* a *warn off* kontrolují spoofing, avšak je zapnuto respektive vypnuto zapisování chyb. Hodnota * bude kontrolovat spoofing, ale rozsah zaznamenávání chyb ponechá nastavený podle souboru `host.conf`.

`RESOLV_MULTI`
K potlačení voleb *multi* v souboru `host.conf` lze použít hodnoty *on* nebo *off*.

`RESOLV_OVERRIDE_TRIM_DOMAINS`
Tato proměnná určuje seznam domén, který potlačí domény uvedené v souboru *host.conf*.

`RESOLV_ADD_TRIM_DOMAINS`
Tato proměnná specifikuje seznam domén určených k odstranění. Tyto domény budou přidány k doménám, které jsou uvedeny v souboru `host.conf`.

6.1.3 Konfigurace vyhledávání jmenného serveru – resolv.conf

Nakonfigurujete-li knihovnu resolveru tak, aby k vyhledávání hostitelů používala jmennou službu BIND, musíte jí sdělit, jaké má používat jmenné servery. K tomuto účelu se používá speciální soubor s názvem `resolv.conf`. Pokud tento soubor neexistuje nebo je prázdný, bude resolver předpokládat, že se jmenný server nachází na vašem místním hostiteli.

Provozujete-li jmenný server na svém místním hostiteli, musíte ho nastavit samostatně, což si vysvětlíme v dalších statích. Pokud se nacházíte v lokální síti a máte možnost používat existující jmenný server, pak bude tento způsob vždy preferován.

Nejdůležitější volbou v souboru `resolv.conf` je volba *nameserver*, která obsahuje IP-adresu používaného jmenného serveru. Zadáte-li několikanásobným uvedením volby *nameserver* několik jmenných serverů, pak budou tyto jmenné servery zkoušeny v uvedeném pořadí. Z toho důvodu byste měli na prvním místě uvést nejspolehlivější jmenný server. V současné době jsou podporovány až tři jmenné servery.

Není-li uvedena žádna volba *nameserver*, pokusí se resolver spojit se jmenným serverem na místním hostiteli.

Další dvě volby *domain* a *search* ovládají implicitní domény, které budou připojeny k názvu hostitele v případě, že se službě BIND nepodaří při prvním dotazu nalézt příslušný název hostitele. Volba *search* určuje seznam zkoušených názvů domén. Jednotlivé položky v seznamu jsou od sebe odděleny mezerami nebo tabulátory.

Pokud žádnou volbu *search* neuvedete, bude z místního názvu domény sestaven implicitní vyhledávací seznam tím způsobem, že se použije název domény plus všechny nadřazené názvy domén až po kořenovou úroveň. Místní název domény je možné zadat pomocí volby *domain*; není-li tato volba uvedena, pak resolver získá tento název pomocí systémového volání procedury *getdomainname(2)*.

Zdá-li se vám to zmatené, pak se podívejte na následující příklad souboru `resolv.conf`, který používá společnost Virtual Brewery:

```
# /etc/resolv.conf
# Naše doména
domain          vbrew.com
#
# Hostitel vlager je centrálním jmenným serverem:
nameserver      191.72.1.1
```

Při rozkládání názvu **vale** by měl resolver nejprve vyhledat název **vale**. Neuspěje-li, měl by pokračovat vyhledáváním názvů **vale.vbrew.com** a **vale.com**.

6.1.4 Robustnost resolveru

Pokud používáte menší lokální síť v rámci rozsáhlé sítě, měli byste rozhodně používat centrální jmenné servery, jsou-li tyto k dispozici. Výhoda tohoto způsobu spočívá v tom, že centrální jmenné servery si vytvoří velkou vyrovnávací paměť, protože na ně budou směrovány veškeré dotazy. Toto schéma má ale i nevýhody: pokud vám například nedávno zničil oheň kabel na páteřní univerzitní síti, nebude v místním oddělení možné provozovat lokální síť, protože resolver se nebude schopen spojit se žádným jmenným serverem. Nebudete se moci přihlásit k žádnému X-terminálu, nebude fungovat tisk atd.

I když není příliš běžné, aby začala hořet páteř univerzitní sítě, budete asi chtít proti takovýmto případům učinit určitá opatření.

Jednou z možností je nastavit místní jmenný server tak, aby hledal názvy hostitelů z vaší místní domény a všechny ostatní dotazy na další názvy hostitelů předával na hlavní servery. Samozřejmě, že toto schéma bude fungovat pouze v případě, že provozujete svou vlastní doménu.

Máte i druhou možnost – v souboru `/etc/hosts` udržovat záložní tabulku hostitelů vaší domény nebo lokální sítě. Následně byste měli do souboru `/etc/host.conf` přidat volby „*order bind hosts*“, aby se resolver v případě nefunkčního centrálního jmenného serveru vrátil zpět k souboru hostitelů.

6.2 Provozování programu named

Program, který na většině unixových počítačích poskytuje doménové jmenné služby, se obvykle jmenuje `named` ([neimdi:]). Jedná se o serverový program původně vyvinutý pro operační systém BSD, který poskytuje jmenné služby klientům a případně i dalším jmenným serverům. Zdá se, že v současnosti se na většině instalacích Linuxu používá verze BIND-4.8.3. Novější verze BIND-4.9.3 existuje momentálně ve verzi beta a již brzy by se měla stát součástí Linuxu.

Tato stať vyžaduje určité znalosti způsobu, jakým funguje doménový jmenný systém. Budou-li pro vás následující diskuse tak trochu španělskou vesnicí, mě-li byste si znovu přečíst 2. kapitolu, kde najdete více informací o základech systému DNS.

Program `named` je obvykle spuštěn při zavádění systému a běží tak dlouho, dokud počítač nevypnete. Informace získává z konfiguračního souboru `/etc/named.boot` a z mnoha dalších souborů, které obsahují data týkající se mapování názvů domén na IP-adresy atp. Posledně zmiňované soubory se nazývají *soubory zón*. Jejich sémantika a formát bude vysvětlen v následující stati.

Program `named` spustíte v příkazové řádce zadáním:

```
# /usr/sbin/named
```

Program `named` načte konfigurační soubor `named.boot` a všechny další v něm uvedené soubory zón. Své identifikační číslo procesu zapíše ve formátu ASCII do souboru `/var/run/named.pid` a je-li to nutné, načte soubory zón z primárních serverů a spustí na portu číslo 53 odposlouchávání DNS dotazů.[1]

6.2.1 Soubor named.boot

Soubor `named.boot` je zpravidla velmi malý a obsahuje pouze ukazatele na hlavní soubory s informacemi o zónách a ukazatele na další jmenné servery. V tomto zaváděcím souboru začínají komentáře středníkem a končí u dalšího znaku nové řádky. Dříve, než si probereme formát souboru `named.boot` podrobněji, podíváme se na vzorový soubor pro bránu **vlager**, který vidíte na obrázku 6.1.[2]

1 Na FTP serverech se nachází několik druhů binárních souborů programu `named`, z nichž každý se nastavuje nepatrně odlišným způsobem. Některé z nich mají svůj soubor pid uložen v adresáři `/etc`, jiné ho ukládají do adresářů `/tmp` nebo `/var/tmp`.

2 Všimněte si, že názvy domén v tomto příkladu nejsou uváděny *včetně* postfixové tečky. Starší verze programu `named` chápaly postfixové tečky jako chybu a příslušnou řádku v tichosti ignorovaly. Verze programu BIND-4.9.3 by snad měla mít tuto chybu opravenu.

```
;
; /etc/named.boot pro vlager.vbrew.com
;
directory      /var/named
;
;             doména                        soubor
;-------------------------------------------------
cache         .                             named.ca
primary       vbrew.com                     named.hosts
primary       0.0.127.in-addr.arpa          named.local
primary       72.191.in-addr.arpa           named.rev
```

Obrázek 6.1
Soubor *named.boot* pro hostitele **vlager**

Příkazy `cache` a `primary`, použité v tomto příkladu, načítají do programu `named` informace. Tyto informace jsou převzaty z hlavních souborů zadaných v druhém argumentu. Obsahují textové verze položek zdrojových záznamů systému DNS, které si popíšeme dále.

V tomto příkladu jsme nakonfigurovali program `named` jako primární jmenný server pro tři domény, což naznačují tři příkazy `primary` na konci tohoto souboru. První z těchto tří řádek například říká programu `named`, aby se choval jako primární server pro adresu **vbrew.com** a aby načetl data ze souboru `named.hosts`. Klíčové slovo `directory` udává, že všechny soubory zón jsou umístěny v adresáři `/var/named`.

Volba `cache` je speciální a rozhodně by měla být přítomna na všech počítačích provozujících jmenný server. Má dvojí funkci: přikazuje programu `named`, aby povolil vyrovnávací paměť a načítá ze zadaného souboru vyrovnávací paměti (v našem příkladu je to soubor `named.ca`) kořenové jmenné servery. Ke kořenovým jmenným serverům se vrátíme později.

Následuje seznam nejdůležitějších voleb, které můžete použít v souboru `named.boot`:

`directory` Tato volba určuje adresář, ve kterém budou umístěny soubory zón. Názvy souborů mohou být zadávány relativně vůči tomuto adresáři. Několikanásobným použitím volby `directory` lze zadat i více adresářů. Podle standardů souborového systému Linuxu by tímto adresářem měl být adresář `/var/named`.

`primary` Tato volba používá jako argumenty **název domény** a **název souboru** a deklaruje autoritativní jmenný server pro danou doménu. Protože jde o primární server, načte program `named` informace o zónách z příslušného hlavního souboru.

Obecně bude v souboru `named.boot` vždy minimálně jedna položka s volbou `primary`, a tou bude zpětné mapování sítě **127.0.0.0**, což je místní zpětnovazebná síť.

`secondary` Tato volba používá jako argumenty **název domény**, **seznam adres** a **název souboru**. Pro danou doménu určuje místní server, který se stane sekundárním jmenným serverem.

Sekundární server také uchovává důležité údaje o doméně. Nezískává je však ze souborů, ale snaží se je stáhnout z primárního serveru. Proto musí být v seznamu adres příkazu `named` uveden minimálně jeden primární server. Místní server bude kontaktovat každý server z daného seznamu, dokud se mu nepodaří úspěšně přenést databázi zóny, která bude potom uložena jako záložní soubor pod názvem, který byl uveden jako třetí argument. Neodpovídá-li ani jeden z primárních serverů, použijí se data získaná ze záložních souborů.

Příkaz `named` se potom pokusí v pravidelných intervalech obnovovat data zóny. Tato problematika je vysvětlena dále společně se zdrojovými záznamy typu SOA.

`cache` Tato volba má jako argumenty **doménu** a **název souboru**. Tento soubor obsahuje seznam záznamů ukazujících na kořenové jmenné servery. Jsou rozlišovány pouze záznamy typů NS a A. Argumentem **doména** je obvykle název kořenové domény „.“.

Tyto informace jsou pro program `named` rozhodující: Pokud v souboru `named.boot` neexistuje volba `cache`, nebude program `named` vytvářet místní vyrovnávací paměť. To způsobí výrazné snížení výkonu a zvýšení zatížení sítě v případě, že se dotazovaný server nenachází v místní síti. Kromě toho se nebude moci program `named` spojit s žádným kořenovým jmenným serverem, a tak nebude moci rozložit žádné adresy kromě těch, pro které je správcem. Výjimkou z tohoto pravidla je použití tzv. forwardujících serverů (srov. s níže uvedenou volbou `forwarders`).

`forwarders` Tato volba má jako argument **seznam adres**. IP-adresy v seznamu určují seznam jmenných serverů, kterých se může program `named` dotazovat v případě, že se mu nepodaří vyřešit dotaz pomocí své místní vyrovnávací paměti. Jmenné servery jsou v příslušném pořadí neustále dotazovány, dokud některý z nich neodpoví na dotaz.

`slave` Tento příkaz označí jmenný server jako *řízený* server. To znamená, že nikdy nebude sám provádět rekurzivní dotazy, ale bude je posílat na servery uvedené ve volbě `forwarders`.

Ještě jsme neuvedli dvě další volby, `sortlist` a `domain`. Kromě toho existují další dvě direktivy, které lze použít uvnitř databázových souborů zón. Jedná se o příkazy *$INCLUDE* a *$ORIGIN*. Protože jsou používány jen velmi zřídka, nebudeme se jimi dále zabývat.

6.2.2 Databázové soubory systému DNS

Hlavní soubory, které využívá program `named`, například soubor `named.hosts`, obsahují vždy nějakou doménu, se kterou jsou sdruženy. Tato doména se nazývá *počátek (origin)*. Doménu tvoří název domény zadaný pomocí příkazů `cache` a `primary`. V rámci hlavního souboru můžete zadávat názvy domén a hostitelů relativně vůči této doméně. Název uvedený v konfiguračním souboru je považován za *absolutní*, pokud končí jednou tečkou, v opačném případě je považován za relativní vůči počátku. Vlastní počátek se může odkazovat sám na sebe za pomoci symbolu „@“.

Všechna data obsažená v hlavním souboru jsou rozdělena do tzv. *zdrojových záznamů (resource record)*, zkráceně záznamy RR. Vytvářejí nejmenší jednotky informace dostupné pomocí systému DNS. Každý zdrojový záznam je určitého typu. Například záznamy typu A mapují název hostitele na IP-adresu a záznam typu CNAME přiřazuje přezdívky hostitele oficiálnímu názvu hostitele. Máte-li zájem o nějaký příklad, podívejte se na obrázek 6.3, který ukazuje hlavní soubor `named.hosts` ve společnosti Virtual Brewery.

Položky zdrojových záznamů v hlavních souborech sdílejí následující společný formát:

```
[domain] [ttl] [class] type rdata
```

Pole jsou navzájem oddělena pomocí mezer nebo tabulátorů. Položka může pokračovat na několika řádcích, pokud před první řádek vložíte levou kulatou závorku a za poslední pole vložíte pravou kulatou závorku. Vše mezi středníkem a novým řádkem bude ignorováno.

`domain` Toto je název domény, ke které se vztahuje daná položka. Není-li uveden žádný název domény, bude se předpokládat, že se záznam RR vztahuje k doméně uvedené v předchozím záznamu RR.

`ttl` Aby bylo po uplynutí určité doby možno donutit resolver k vyřazení určitých informací, je ke každému záznamu RR přirazen tzv. „*čas přežití*“, zkráceně `ttl`. Pole `ttl` udává čas v sekundách, po který budou ještě informace po stažení ze serveru platné. Čas `ttl` je desítkové číslo s maximálně osmi číslicemi.

Nezadáte-li žádný časový údaj `ttl`, bude jeho implicitní hodnota rovna hodnotě pole *minimum* předcházejícího záznamu typu SOA.

`class` Tato volba určuje třídu adresy, například třídu IN pro IP-adresy nebo HS pro objekty z třídy Hesiod. U sítí na bázi protokolu TCP/IP by měla být tato volba rovna IN.

Není-li pole `class` zadáno, použije se třída z předchozího záznamu typu RR.

`type` Tato volba popisuje typ záznamu RR. Nejběžnějšími typy jsou záznamy A, SOA, PTR a NS. V následující stati budeme popisovat různé typy záznamů RR.

`rdata` Tato volba se stará o data sdružená se záznamem RR. Formát tohoto pole závisí na typu záznamu RR. Dále bude tato volba popisována odděleně pro každý typ záznamu RR.

Následuje nekompletní seznam typů záznamů RR, které lze použít v hlavních souborech systému DNS. Existuje sice ještě více typů záznamů, ale zde je nebudeme popisovat. Jsou experimentální, a proto mají jen malé obecné použití.

SOA Tento typ záznamu popisuje správní zónu (SOA znamená „počátek správy – Start of Authority"). Tento záznam signalizuje, že záznamy následující po záznamu RR typu SOA budou obsahovat správní informace o doméně. Každý hlavní soubor obsažený v příkazu *primary* musí pro tuto zónu obsahovat záznam typu SOA. Zdrojová data obsahují následující pole:

origin To je kanonický název hostitele primárního jmenného serveru této domény. Obvykle je zadán jako absolutní název.

contact Toto je e-mailová adresa osoby odpovědné za správu domény, u které je znak ‚@' nahrazen tečkou. Je-li například ve společnosti Virtual Brewery odpovědnou osobou **janet**, potom by toto pole mělo obsahovat adresu *janet.vbrew.com*

serial Toto je číslo verze souboru zóny vyjádřené jednou desítkovou číslicí. Kdykoliv se v souboru zóny změní data, mělo by se toto číslo zvýšit.

Sériová čísla používají sekundární jmenné servery k rozpoznávání změn v informacích o zónách. Aby byly tyto informace aktuální, požadují sekundární servery v určitých intervalech po primárních serverech záznam typu SOA a porovnávají jeho sériové číslo s číslem,

které je obsaženo v záznamu typu SOA uloženém ve vyrovnávací paměti. Změnilo-li se toto číslo, potom sekundární servery přenesou z primárního serveru celou databázi zóny.

`refresh` Tato volba udává interval v sekundách, po který mají sekundární servery čekat, než provedou opětovné zkontrolování záznamu typu SOA s primárním serverem. Opět se jedná o desítkové číslo s maximálně osmi číslicemi.

Síťová topologie se obecně příliš často nemění, takže by toto číslo mělo v rozsáhlejších sítích odpovídat zhruba dnům a v menších sítí by tento interval měl být ještě delší.

`retry` Toto číslo určuje interval, po jehož uplynutí by se měl sekundární server znovu spojit s primárním serverem, když se nepodaří nějaký požadavek nebo aktualizace zónových informací. Tento interval by neměl být příliš malý, jinak by dočasný výpadek serveru nebo nějaký síťový problém mohl způsobit obrovské plýtvání se síťovými zdroji ze strany sekundárních serverů. Vhodnou hodnotou pro tento interval je jedna hodina nebo půl hodiny.

`expire` Tato volba udává čas v sekundách, po jehož uplynutí by měl server skartovat všechna data o zónách, pokud se mu během tohoto intervalu nepodařilo spojit s primárním serverem. Normálně by měla být tato hodnota hodně velká. Craig Hunt ([Hunt92]) doporučuje 42 dnů.

`minimum` Toto je implicitní hodnota času `ttl` pro zdrojové záznamy, které ji nemají explicitně zadanou. Tato volba přikazuje ostatním jmenným serverům, aby po určité předem zadané době zrušily záznamy RR. Nemá však nic společného s časem, po kterém se budou snažit sekundární servery o aktualizaci svých informací.

Hodnota *minimum* by měla být dostatečně vysoká, zejména u sítí typu LAN, u nichž se prakticky nikdy nemění síťová topologie. Vhodné je použít hodnotu týden nebo dokonce měsíc. V případech, kde se jednotlivé záznamy RR často mění, můžete použít vlastní dobu `ttl`.

A — Tento typ záznamu RR přiřazuje IP-adresu k názvu hostitele. Pole zdrojových dat obsahuje adresu respektující tečkovou notaci.

Každý hostitel musí mít pouze jeden záznam typu A. Název hostitele použitý v tomto záznamu typu A je považován za oficiální název hostitele neboli za *kanonický* název hostitele. Všechny další názvy hostitelů jsou přezdívky a pomocí záznamu typu CNAME musí být mapovány na kanonický název hostitele.

NS — Tento typ záznamu RR ukazuje na hlavní jmenný server podřazené zóny. Vysvětlení, proč někdo potřebuje záznamy typu NS, najdete ve stati 2.6. Pole zdrojových dat obsahuje název hostitele jmenného serveru. K nalezení názvu hostitele potřebujeme ještě jeden záznam typu A (někdy se mu také říká *tmelící záznam*), který poskytuje IP-adresu jmenného serveru.

CNAME — Danému hostiteli přiřadí přezdívku, která bude propojena s jeho *kanonickým názvem*. Kanonický název hostitele je takový název, pro který existuje v hlavním souboru záznam typu A; přezdívky jsou s tímto názvem jednoduše spojeny pomocí záznamu typu CNAME, ale jinak nemají žádné další vlastní záznamy.

PTR — Tento typ záznamu slouží ke sdružení názvů v doméně **in-addr.arpa** s názvy hostitelů. Tento záznam se používá ke zpětnému mapování IP-adres na názvy hostitelů. Zadaný název hostitele musí být v kanonickém tvaru.

MX — Tento záznam RR definuje *poštovní server* (mail exchanger) pro danou doménu. Důvody použití poštovních serverů jsou probírány ve stati Směrování pošty v Internetu, kterou najdete ve 13. kapitole. Syntaxe záznamu typu MX je tato:

```
[domain] [ttl] [class] MX preference host
```

Argument `host` přiděluje poštovnímu serveru pro doménu `domain` název hostitele `host`. Každý poštovní server má přiřazen celočíselný argument `preference`. Zprostředkovatel přenosu pošty, který chce doručit poštu do domény `domain`, se bude tak dlouho snažit spojit se všemi hostiteli, kteří mají pro danou doménu uvedený záznam typu MX, dokud neuspěje. Nejprve je kontaktován hostitel s nejnižší hodnotou argumentu `preference`, následně ostatní hostitelé v pořadí, které určuje zvyšující se hodnota argumentu `preference`.

HINFO — Tento typ záznamu poskytuje informace o hardwaru a softwaru daného systému. Jeho syntaxe je:

```
[domain] [ttl] [class] HINFO hardware software
```

Pole `hardware` určuje typ hardwaru, který používá daný hostitel. Pro jeho specifikaci existují určité zvyklosti. Seznam korektních názvů obsahuje specifikace „Assigned Numbers“ (RFC 1340). Pokud toto pole obsahuje nějaké mezery, pak musí být tyto mezery uzavřeny do uvozovek. Pole `software` obsahuje používaný operační systém. Opět by měl být vybrán korektní název ze seznamu „Assigned Numbers“ RFC.*

6.2.3 Sestavení hlavních souborů

Obrázky 6.2, 6.3, 6.4 a 6.5 obsahují vzorové příklady souborů pro jmenný server pivovaru, který je umístěn na hostiteli **vlager**. Vzhledem k povaze diskutované sítě (jednoduchá síť typu LAN) je příklad poměrně zřejmý. Máte-li složitější požadavky a nedaří-li se vám rozchodit program `named`, pak se poohlédněte po knize „DNS and BIND“ od Cricketa Liua a Paula Albitze ([AlbitzLiu92]).

Soubor `named.ca`, který vidíte na obrázku 6.2, ukazuje vzorové záznamy pro kořenové jmenné servery. Typický soubor vyrovnávací paměti obvykle popisuje zhruba deset jmenných serverů. Aktuální seznam jmenných serverů kořenové domény můžete získat pomocí nástroje `nslookup`, který bude popsán na konci této kapitoly.[3]

```
;
; /var/named/named.ca
;                   Nejsme na Internetu a proto nepotřebujeme
;                   kořenové servery. Pro aktivaci těchto záznamů
;                   stačí odstranit středníky.
;
; .                 99999999    IN     NS   NS.NIC.DDN.MIL
; NS.NIC.DDN.MIL    99999999    IN     A    26.3.0.103
; .                 99999999    IN     NS   NS.NASA.GOV
; NS.NASA.GOV       99999999    IN     A    128.102.16.10
```

Obrázek 6.2
Soubor `named.ca`

* Poznámka korektora: Z bezpečnostních důvodů není vhodné používat zdrojové záznamy HINFO.

3 Všimněte si, že vašemu jmennému serveru nemůžete předat dotaz na kořenové servery, dokud nemáte nějaké nainstalovány: Hlava XXII! Abyste z toho vybruslili, můžete buď nařídit nástroji `nslookup`, aby používal odlišný jmenný server, nebo můžete použít jako vstupní bod zdrojový soubor uvedený na obrázku 6.2 a potom získat úplný seznam korektních serverů.

6.2.4 Kontrola nastavení jmenného serveru

Pro kontrolu správné funkce nastavení vašeho jmenného serveru existuje speciální nástroj. Nazývá se `nslookup` a lze ho používat jak interaktivně, tak i z příkazové řádky. V druhém případě ho spustíte následovně:

```
nslookup hostname
```

Nástroj `nslookup` se bude dotazovat jmenného serveru zadaného v souboru `resolv.conf` na název hostitele `hostname`. (Je-li v tomto souboru uveden více než jeden server, pak `nslookup` náhodně zvolí jeden z těchto serverů.)

Interaktivní režim je však mnohem zajímavější. Kromě vyhledávání jednotlivých hostitelů se můžete dotazovat na libovolný typ záznamu systému DNS a dále můžete v rámci příslušné domény přenášet veškeré informace o zónách.

Je-li nástroj `nslookup` vyvolán bez argumentů, zobrazí používaný jmenný server a přepne se do interaktivního režimu. Na příkazové řádce „>“ můžete zadat libovolný název domény, na který by se měl nástroj `nslookup` dotazovat. Implicitně se ptá na záznamy typu A, což jsou ty, které obsahují IP-adresy vztahující se k danému názvu domény.

```
;
; /var/named/named.hosts          Místní hostitelé v pivovaru
;                                 Počátek je vbrew.com
;
@                   IN     SOA    vlager.vbrew.com. (
                                  janet.vbrew.com.
                                  16          ; sériové číslo
                                  86400       ; refresh: denně
                                  3600        ; retry:   1 hodina
                                  3600000     ; expire:  42 dní
                                  604800      ; minimum: 1 týden
                                  )
                    IN     NS     vlager.vbrew.com.
;
; lokální pošta je doručována na vlager
                    IN     MX     10 vlager
;
; zpětnovazebné rozhraní
localhost.          IN     A      127.0.0.1
```

```
; Ethernet pivovaru
vlager              IN     A      191.72.1.1
vlager-if1          IN     CNAME  vlager
; vlager je také news serverem
news                IN     CNAME  vlager
vstout              IN     A      191.72.1.2
vale                IN     A      191.72.1.3
; Ethernet vinárny
vlager-if2          IN     A      191.72.2.1
vbardolino          IN     A      191.72.2.2
vchianti            IN     A      191.72.2.3
vbeaujolais         IN     A      191.72.2.4
```

Obrázek 6.3
Soubor `named.hosts`

```
;
; /var/named/named.local          Reverzní mapování sítě 127.0.0
;                                 Počátek je 0.0.127.in-addr.arpa.
;
@                   IN     SOA    vlager.vbrew.com. (
                                  joe.vbrew.com.
                                  1           ; sériové číslo
                                  360000      ; refresh: 100 hodin
                                  3600        ; retry:   1 hodina
                                  3600000     ; expire:  42 dní
                                  360000      ; minimum: 100 hodin
                                  )
                    IN     NS     vlager.vbrew.com.
1                   IN     PTR    localhost.
```

Obrázek 6.4
Soubor `named.local`

```
;
; /var/named/named.rev            Reverzní mapování našich IP-adres
;                                 Počátek je 72.191.in-addr.arpa.
;
@                   IN     SOA    vlager.vbrew.com. (
```

```
                                joe.vbrew.com.
                                16              ; sériové číslo
                                86400           ; refresh: denně
                                3600            ; retry:   1 hodina
                                3600000         ; expire:  42 dní
                                604800          ; minimum: 1 týden
                                )
                    IN      NS      vlager.vbrew.com.
; pivovar
1.1                 IN      PTR     vlager.vbrew.com.
2.1                 IN      PTR     vstout.vbrew.com.
3.1                 IN      PTR     vale.vbrew.com.
; vinárna
1.2                 IN      PTR     vlager-if1.vbrew.com.
2.2                 IN      PTR     vbardolino.vbrew.com.
3.2                 IN      PTR     vchianti.vbrew.com.
4.2                 IN      PTR     vbeaujolais.vbrew.com.
```

Obrázek 6.5
Soubor `named.rev`

Tento typ je možné změnit příkazem „**`set type`**=`type`“, kde parametr `type` tvoří buď jeden z názvů zdrojových záznamů, které jsme popisovali ve stati 6.2, nebo za něj může být dosazeno klíčové slovo ANY.

S nástrojem `nslookup` můžete vést například následující dialog:

```
$ nslookup
Default Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

> sunsite.unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
Name:    sunsite.unc.edu
Address:  152.2.22.81
```

Pokusíte-li se dotazovat na název, který nemá přidělenu žádnou IP-adresu, ale v databázi systému DNS byly nalezeny jiné záznamy, vrátí příkaz `nslookup` chybovou zprávu „`No type A records found`“ (nebyly nalezeny žádné záznamy typu A). S pomocí příkazu `nslookup` je možné se dotazovat i na jiné záznamy než typu A. To lze provést příkazem „**`set type`**“. Chcete-li například získat záznam typu SOA na adrese **unc.edu**, měli byste spustit následující sekvenci příkazů:

```
> unc.edu
*** No address (A) records available for unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

> set type=SOA
> unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
unc.edu
        origin = ns.unc.edu
        mail addr = shava.ns.unc.edu
        serial = 930408
        refresh = 28800 (8 hours)
        retry   = 3600 (1 hour)
        expire  = 1209600 (14 days)
        minimum ttl = 86400 (1 day)
Authoritative answers can be found from:
UNC.EDU nameserver = SAMBA.ACS.UNC.EDU
SAMBA.ACS.UNC.EDU        internet address = 128.109.157.30
```

Podobným způsobem se můžete dotazovat na záznamy typu MX atd. Pokud použijete typ záznamu ANY, vrátí příkaz `nslookup` všechny zdrojové záznamy, které jsou sdruženy s daným názvem.

```
> set type=MX
> unc.edu
Non-authoritative answer:
unc.edu preference = 10, mail exchanger = lambada.oit.unc.edu
lambada.oit.unc.edu      internet address = 152.2.22.80
```

```
Authoritative answers can be found from:
UNC.EDU nameserver = SAMBA.ACS.UNC.EDU
SAMBA.ACS.UNC.EDU        internet address = 128.109.157.30
```

Praktickou aplikací příkazu `nslookup` je kromě ladění jmenného serveru i získávání aktuálního seznamu kořenových jmenných serverů pro soubor `named.ca`. Lze to provést tak, že se budete dotazovat na všechny typy záznamů typu NS, které jsou spojeny s kořenovou doménou:

```
> set type=NS
> .
Name Server:  fb0430.mathematik.th-darmstadt.de
Address:  130.83.2.30

Non-authoritative answer:
(root)  nameserver = NS.INTERNIC.NET
(root)  nameserver = AOS.ARL.ARMY.MIL
(root)  nameserver = C.NYSER.NET
(root)  nameserver = TERP.UMD.EDU
(root)  nameserver = NS.NASA.GOV
(root)  nameserver = NIC.NORDU.NET
(root)  nameserver = NS.NIC.DDN.MIL

Authoritative answers can be found from:
(root)  nameserver = NS.INTERNIC.NET
(root)  nameserver = AOS.ARL.ARMY.MIL
(root)  nameserver = C.NYSER.NET
(root)  nameserver = TERP.UMD.EDU
(root)  nameserver = NS.NASA.GOV
(root)  nameserver = NIC.NORDU.NET
(root)  nameserver = NS.NIC.DDN.MIL

NS.INTERNIC.NET internet address = 198.41.0.4
AOS.ARL.ARMY.MIL         internet address = 128.63.4.82
AOS.ARL.ARMY.MIL         internet address = 192.5.25.82
AOS.ARL.ARMY.MIL         internet address = 26.3.0.29
C.NYSER.NET      internet address = 192.33.4.12
TERP.UMD.EDU     internet address = 128.8.10.90
NS.NASA.GOV      internet address = 128.102.16.10
```

```
NS.NASA.GOV      internet address = 192.52.195.10
NS.NASA.GOV      internet address = 45.13.10.121
NIC.NORDU.NET    internet address = 192.36.148.17
NS.NIC.DDN.MIL   internet address = 192.112.36.4
```

Kompletní seznam příkazů, které lze použít ve spojitosti s nástrojem `nslookup`, získáte příkazem `help`, který musí být zadán během interaktivní práce s nástrojem `nslookup`.

6.2.5 Další užitečné nástroje

Existuje několik nástrojů, které vám mohou pomoci při úkolech, s nimiž se setkáte jako správce služby BIND. Zde si popíšeme dva z nich. Chcete-li získat informace o jejich použití, nahlédněte do dokumentace, která je dodávána spolu s nimi.

Nástroj `hostcvt` pomáhá při prvotní konfiguraci služby BIND. Provádí konverzi souboru `/etc/hosts` do hlavních souborů pro program `named`. Vygeneruje data jak pro přímé mapování (typ záznamu A), tak i pro zpětné mapování (typ záznamu PTR) a postará se i o přezdívky. Samozřejmě, že nemůže udělat vše. Stále se budete muset postarat například o hodnoty časových intervalů v záznamu typu SOA, o přidání záznamů typu MX atp. Přesto vám však může ušetřit několik aspirinů. Nástroj `hostcvt` je částí zdrojového kódu služby BIND, ale je možné ho nalézt i jako samostatný balík na některých linuxových FTP serverech.

Jakmile nakonfigurujete vlastní jmenný server, budete si ho zřejmě chtít otestovat. Ideálním (a podle mých vědomostí) zřejmě i jediným nástrojem, který byl vytvořen za tímto účelem, je `dnswalk`. Jedná se o balík napsaný v jazyce Perl, který projde vaši databázi systému DNS, vyhledá běžné chyby a ověří konzistenci informací. Nástroj `dnswalk` byl teprve nedávno uvolněn skupině **comp.sources.misc** a měl by být dostupný na všech serverech, které tento server zrcadlí (neznáte-li ve svém okolí žádný takový systém, pak zmiňovaný balík bezpečně najdete na serveru **ftp.uu.net**).

Kapitola

7

IP po sériové lince

Protokoly pro sériové linky, SLIP a PPP, umožňují připojení k Internetu i chudším vrstvám. Kromě modemu a sériového portu vybaveného vyrovnávací pamětí typu FIFO již není zapotřebí žádný další hardware. Jeho použití není o nic složitější, než používání poštovní schránky, a přitom připojení pomocí modemu nabízí za rozumnou cenu stále více soukromých firem.

V systému Linux je dostupný jak ovladač pro protokol SLIP, tak i ovladač pro protokol PPP. Protokol SLIP je v něm zabudován už poměrně dlouhou dobu a tudíž pracuje poměrně spolehlivě. Ovladač PPP vytvořili teprve nedávno Michael Callahan a Al Longyear. Tento ovladač bude popsán v další kapitole.

7.1 Obecné požadavky

Abyste mohli používat protokoly SLIP nebo PPP, bude třeba nastavit několik základních síťových vlastností, které byly popsány v předchozích kapitolách. Přinejmenším musíte nastavit zpětnovazebné rozhraní a povolit službu pro rozlišení názvů. Když se budete připojovat k Internetu, budete samozřejmě chtít používat systém DNS. Nejjednodušší způsob, jak toho docílit, spočívá ve vložení adresy nějakého jmenného serveru do souboru `resolv.conf`. Jakmile bude aktivováno spojení pomocí protokolu SLIP, bude poslán dotaz tomuto serveru. Čím blíže bude název tohoto serveru od místa, ze kterého voláte, tím lépe.

Toto řešení však není optimální, protože všechna vyhledávání názvů budou využívat spojení SLIP/PPP. Pokud vám dělá starosti šířka pásma, která je k tomuto účelu využívána, můžete nastavit tzv. `caching-only` jmenný server. Ten ve skutečnosti neobsluhuje doménu, ale působí pouze jako prostředník pro všechny dotazy systému DNS, které pochází z vašeho hos-

titele. Výhodou tohoto schématu je vytvoření vyrovnávací paměti, kdy je většina dotazů posílána po sériové lince pouze jednou. Soubor `named.boot` pro tento server vypadá podobně jako následující výpis:

```
; soubor named.boot pro caching-only server
directory                          /var/named
primary      0.0.127.in-addr.arpa  db.127.0.0 ; zpětnovazebná síť
cache         .                    db.cache   ; kořenové servery
```

K tomuto souboru `named.boot` musíte v souboru `db.cache` nastavit korektní sezname kořenových jmenných serverů. To je popsáno na konci kapitoly Konfigurace resolveru.

7.2 Provozování protokolu SLIP

Dial-up IP-servery často nabízejí službu SLIP pomocí speciálních uživatelských účtů. Po přihlášení k takovému účtu nejste vpuštěni do nějakého obecného uživatelského rozhraní; namísto toho je spuštěn program nebo skript rozhraní, který povolí na serveru pro danou sériovou linku ovladač SLIP a nakonfiguruje patřičné síťové rozhraní. Totéž se musí provést na vaší straně spojení.

V některých operačních systémech je ovladač SLIP speciálním uživatelským programem; v operačním systému Linux je součástí jádra systému, takže je výrazně rychlejší. Je však nutné, aby byla sériová linka explicitně převedena do režimu SLIP. To se provede pomocí speciálního *tty* režimu linky, tzv. SLIPDISC. Je-li zařízení *tty* v normálním režimu linky (DISC0), bude si vyměňovat data pouze s uživatelskými procesy pomocí standardních volání *read(2)* a *write(2)* a ovladač SLIP nebude schopen zapisovat nebo číst ze zařízení *tty*. V režimu SLIPDISC jsou role obráceny: nyní nebude moci zapisovat nebo číst ze zařízení žádný uživatelský proces, ale všechna data přicházející na sériový port budou přímo předána ovladači SLIP.

Vlastní ovladač protokolu SLIP rozumí množství variací protokolu SLIP. Kromě běžného protokolu SLIP ovládá také protokol CSLIP, který u odcházejících IP-paketů provádí tzv. Van Jacobsonovu kompresi hlaviček.[1] To výrazně zvyšuje propustnost dat při interaktivní práci. Mimoto existují i šestibitové verze obou těchto protokolů.

Jednoduchý způsob, jak zkonvertovat sériovou linku do režimu SLIP, spočívá ve využití nástroje `slattach`. Předpokládejme, že máte modem nastaven na `/dev/cua3` a že jste se úspěšně přihlásili k serveru SLIP. Potom spustíte následující příkaz:

1 Van Jacobsonova komprese hlavičky je popsána v RFC 1441.

```
# slattach /dev/cua3 &
```

Tento příkaz přepne linku `cua3` do režimu SLIPDISC a připojí ji k jednomu ze síťových rozhraní SLIP. Je-li to vaše první aktivní spojení pomocí protokolu SLIP, bude linka připojena k rozhraní *sl0*; další bude připojena k rozhraní *sl1* atd. Aktuální verze jader operačního systému podporují až osm současných spojení pomocí protokolu SLIP.

Implicitní zapouzdření, které vybere příkaz `slattach` je protokol CSLIP. K volbě některého jiného režimu lze použít argument `-p`. Chcete-li používat standardní protokol SLIP (bez komprese), pak byste měli použít následující příkaz:

```
# slattach -p slip /dev/cua3 &
```

Další volitelné režimy jsou: `cslip`, `slip6`, `cslip6` (pro šestibitové verze protokolu SLIP) a `adaptive` pro adaptivní protokol SLIP. Poslední volba nechává nalezení typu zapouzdření protokolu SLIP, který používá vzdálený počítač, na jádru operačního systému.

Pamatujte, že musíte používat stejný typ zapouzdření, jaký používá váš protějšek. Pokud například hostitel **cowslip** používá protokol CSLIP, musíte ho použít také. Neshody jednotlivých verzí protokolu SLIP mohou například způsobit, že příkaz `ping` použitý na vzdáleného hostitele neobdrží zpátky žádné pakety. Pokud nějaký jiný hostitel spustí příkaz `ping` s vaším jménem, může se na vaší konzole objevit zpráva „`Can't build ICMP header`." Jeden ze způsobů, jak se vyhnout takovýmto komplikacím, spočívá v použití adaptivního protokolu SLIP.

Příkaz `slattach` ale nepovoluje jen použití protokolu SLIP, ale také jiné druhy protokolů, jako jsou například protokoly PPP nebo KISS (další protokol používaný v síti ham radio). Chcete-li více detailů, nahlédněte prosím na manuálovou stránku k příkazu `slattach(8)`.

Jakmile lince přiřadíte ovladač protokolu SLIP, musíte nakonfigurovat síťové rozhraní. Opět k tomu využijeme standardní příkazy `ifconfig` a `route`. Předpokládejme, že jsme se z brány **vlager** připojili k serveru jménem **cowslip**. Pak je nutné spustit následující sekvenci příkazů:

```
# ifconfig sl0 vlager pointopoint cowslip
# route add cowslip
# route add default gw cowslip
```

První příkaz nastaví rozhraní na spojení s hostitelem **cowslip** na typ point-to-point, druhý a třetí příkaz přidá směrování na hostitele **cowslip** a implicitní směr, přičemž bude hostitele **cowslip** využívat jako bránu.

Při rušení spojení pomocí protokolu SLIP musíte nejprve odstranit všechna směrování přes hostitele **cowslip**. K tomu slouží příkaz `route` s volbou `del`. Potom je nutné odpojit rozhraní a poslat nástroji `slattach` signál zavěšení. Následně byste měli znovu zavěsit modem pomocí svého terminálového programu:

```
# route del default
# route del cowslip
# ifconfig sl0 down
# kill -HUP 516
```

7.3 Použití nástroje dip

Až sem to bylo poměrně jednoduché. Přesto však možná budete chtít výše uvedené kroky zautomatizovat natolik, aby bylo možné celý postup vyvolat pouze jediným příkazem. K tomuto účelu slouží nástroj `dip`.[2] Aktuální číslo verze tohoto nástroje je 3.3.7. Ta však byla velmi často upravována a pozměňována, takže vlastně nemůže být o nějakém nástroji `dip` ani řeč. Doufejme, že tyto odlišnosti budou začleněny do budoucí verze.

Nástroj `dip` je interpretem jednoduchého skriptového jazyka, který se může starat o modem, nastavovat linku do režimu SLIP a konfigurovat různá rozhraní. Je to poměrně primitivní a omezující, ale ve většině případů to stačí. Doufejme, že některá nová verze nástroje `dip` bude obsahovat všestrannější jazyk.

Aby bylo možné konfigurovat rozhraní SLIP, musí být nástroji `dip` přidělena práva **superuživatele**. Teď to možná vypadá, že pokud nastavíte nástroji `dip` práva **superuživatele**, budou se moci všichni uživatelé spojit s libovolným serverem SLIP, aniž by měli přidělena přístupová práva **superuživatele**. To je velmi nebezpečné, protože nastavení falešných rozhraní a implicitních směrů pomocí nástroje `dip` může výrazně poškodit směrování ve vaší síti. A co je ještě horší, vaši uživatelé tím získají možnost spojení s libovolným serverem SLIP a budou moci provádět nebezpečné útoky na vaši síť. Pokud tedy chcete umožnit svým uživatelům provozovat spojení pomocí protokolu SLIP, napište pro každý plánovaný server SLIP malé obslužné programy a teprve z těchto programů volejte nástroj `dip` s konkrétním skriptem, který založí spojení SLIP. Až potom lze těmto programům bezpečně nastavit práva **superuživatele**.[3]

2 `dip` znamená *Dialup IP*. Tento nástroj napsal Fred van Kempen.

3 Práva lze nastavit i pro příkaz `diplogin`. Podrobnosti najdete na konci této kapitoly.

7.3.1 Vzorový skript

Na obrázku 7.1 vidíte vzorový skript. Lze ho použít ke spojení s hostitelem **cowslip**. Nástroj `dip` je možné spustit s argumentem, který bude obsahovat název skriptu:

```
# dip cowslip.dip
DIP: Dialup IP Protocol Driver version 3.3.7 (12/13/93)
Written by Fred N. van Kempen, MicroWalt Corporation

connected to cowslip.moo.com with addr 193.174.7.129
#
```

Po spojení s hostitelem *cowslip* a povolení protokolu SLIP se proces `dip` odpojí od terminálu a dále bude spuštěn na pozadí. Potom můžete začít používat normální síťové služby po lince SLIP. Chcete-li spojení ukončit, vyvolejte nástroj *dip* s parametrem `-k`. Tento příkaz pošle procesu `dip` signál zavěšení, přičemž použije záznamy identifikačního čísla procesu `dip`, které se nacházejí v souboru `/etc/dip.pid`:[4]

```
# Jednoduchý dip script pro volání hostitele cowslip
# Nastav místní a vzdálené jméno a adresu
get $local vlager
get $remote cowslip

port cua3                  # volba sériového portu
speed 38400                # maximální rychlost
modem HAYES                # typ modemu
reset                      # reset modemu a tty
flush

# Příprava na vytočení
send ATQ0V1E1X1\r
wait OK 2
if $errlvl != 0 goto error
dial 41988
if $errlvl != 0 goto error
wait CONNECT 60
if $errlvl != 0 goto error
```

4 Další zábavu s třípísmennými akronymy přináší konference **alt.tla**.

```
# Nyní jsme spojeni
sleep 3
send \r\n\r\n
wait ogin: 10
if $errlvl != 0 goto error
send Svlager\r
wait ssword: 5
if $errlvl != 0 goto error
send hey-jude\n
wait running 30
if $errlvl != 0 goto error

# Nyní jsme přihlášeni a vzdálený hostitel spustil SLIP.
print Connected to $remote with address $rmtip
default                    # Nastavení implicitní směrové cesty
mode CSLIP                 # Spuštění protokolu SLIP
# v případě chyby pokračujte zde

error:
print SLIP to $remote failed.
```

Obrázek 7.1
Vzorový skript k nástroji `dip`

```
# kill -k
```

Ve skriptovém jazyku nástroje `dip` znamenají klíčová slova, před nimiž je uveden symbol dolaru, názvy proměnných. Nástroj `dip` má předdefinovanou skupinu proměnných, která bude uvedena níže. Například proměnné *$remote* a *$local* obsahují názvy místního a vzdáleného hostitele, kteří se účastní spojení pomocí protokolu SLIP.

První dva příkazy ve vzorovém skriptu jsou příkazy `get`, které reprezentují způsob, jakým nástroj `dip` nastavuje proměnné. Zde je nastaven název místního a vzdáleného hostitele **vlager**, resp. **cowslip**.

Dalších pět řádek nastavuje terminálovou linku a modem. Příkaz `reset` pošle modemu inicializační řetězec; u modemů kompatibilních se standardem Hayes odpovídá tento řetězec příkazu *ATZ*. Další příkaz nastaví odpovědi modemu tak, aby přihlašovací sekvence, která následuje na dalších řádcích, pracovala správně. Tato přihlašovací sekvence je poměrně přehledná: nejdříve se vytočí číslo 41 988, což je telefonní číslo hostitele *cowslip* a hostitel se přihlásí na

účet *Svlager* s heslem *hey-jude*. Příkaz `wait` způsobí prodlevu nástroje `dip`, protože tento bude čekat na vstup odpovídající řetězci, který je uveden v prvním argumentu; číslo uvedené v druhém argumentu odpovídá době, po kterou se bude čekat v případě, že se nepodaří získat hodnotu odpovídající prvnímu argumentu. Příkaz `if` průběžně uváděný v přihlašovací proceduře kontroluje, zda při spouštění příkazu nedošlo k chybě.

Posledními příkazy spouštěnými po přihlášení jsou `default`, jenž nastaví u spojení SLIP implicitní směr na všechny hostitele, a `mode`, jenž povolí na dané lince režim SLIP a nakonfiguruje rozhraní a směrovací tabulku.

7.3.2 Manuál k nástroji dip

I když je nástroj `dip` velmi rozšířen, není zatím příliš dobře zdokumentován. Proto v této stati uvádíme popis většiny příkazů nástroje `dip`. Přehled všech dostupných příkazů získáte, když nástroj `dip` spustíte v testovacím režimu zadáním příkazu `help`. Chcete-li zjistit syntaxi příkazu, stačí ho zadat bez argumentů; samozřejmě, že výše uvedené nefunguje u příkazů, které nemají žádné argumenty.

```
$dip -t
DIP: Dialup IP Protocol Driver version 3.3.7 (12/13/93)
Written by Fred N. van Kempen, MicroWalt Corporation

DIP> help
DIP knows about the following commands:

databits      default       dial          echo          flush
get           goto          help          if            init
mode          modem         parity        print         port
reset         send          sleep         speed         stopbits
term          wait

DIP> echo
Usage: echo on|off
DIP>
```

V průběhu následujícího výkladu budou řádky začínající řetězcem `DIP>` ukazovat, jak se zadává příslušný příkaz v testovacím režimu a následující řádky pak budou zobrazovat jeho výstup. Příklady, které příkazovou řádku neobsahují, by měly být chápány jako výpisy skriptu.

Příkazy pro modem

Nástroj `dip` poskytuje značný počet příkazů pro konfiguraci sériové linky a modemu. Význam některých z nich je zřejmý, například příkaz `port`, který vybírá sériový port, nebo příkazy `speed, databits, stopbits a parity`, které nastavují obecné parametry linky.

Příkaz `modem` volí typ modemu. V současné době je jediným podporovaným typem modemu typ *HAYES* (velká písmena jsou vyžadována). Nástroji `dip` musíte poskytnout typ modemu, v opačném případě by totiž odmítl spustit příkazy `dial` a `reset`. Příkaz `reset` posílá modemu vstupní inicializační řetězec; použitý typ řetězce závisí na vybraném typu modemu. U modemů kompatibilních se standardem HAYES je tímto řetězcem příkaz *ATZ*.

Příkaz `flush` slouží ke smazání všech odpovědí, které modem v minulosti poslal. V opačném případě by mohl být přihlašovací skript, který následuje po příkazu `reset`, špatný, protože modem by mohl číst odpovědi OK z předchozích příkazů.

Příkaz `init` vybírá inicializační řetězec, který bude předán modemu před začátkem vytáčení. Implicitním řetězcem pro modemy kompatibilní se standardem HAYES je „*ATE0 Q0 V1 X1*“, který zapne zobrazování příkazů, dlouhé kódy výsledků a nastaví volání naslepo (bez detekce vytáčecího tónu).

Nakonec pošle příkaz `dial` modemu inicializační řetězec a vytočí číslo vzdáleného systému. Implicitním vytáčecím příkazem pro modemy kompatibilní se standardem HAYES je příkaz *ATD*.

Příkazy echo a term

Příkaz `echo` slouží jako ladící prostředek, protože použití příkazu `echo on` způsobí, že nástroj `dip` bude na konzole zobrazovat vše, co je posíláno na sériové zařízení. Tuto vlastnost lze opět zrušit příkazem `echo off`.

Nástroj `dip` také umožňuje dočasně opustit skriptový režim a vstoupit do terminálového režimu. V tomto režimu lze nástroj `dip` používat jako jakýkoliv jiný běžný terminálový program. Můžete v něm zapisovat na sériovou linku nebo z ní číst. Chcete-li tento režim opustit, stiskněte kombinaci kláves **Ctrl +**].

Příkaz get

Příkaz `get` používá nástroj `dip` k nastavování proměnných. Nejjednodušším způsobem je nastavit proměnnou jako konstantu, což je používáno ve výše uvedeném příkladu. Můžete ale také požádat uživatele, aby hodnotu zadal. K tomu slouží klíčové slovo `ask`, které je nutno uvést na místě hodnoty proměnné:

```
DIP> get $local ask
Enter the value for $local: _
```

Třetí metoda spočívá v pokusu o získání této hodnoty ze vzdáleného hostitele. Na první pohled to vypadá zvláštně, ale v některých případech je tento způsob velmi užitečný: Některé servery SLIP vám nedovolí při spojení pomocí protokolu SLIP používat vlastní IP-adresu. Místo toho vám při každém přihlášení přidělí nějakou adresu ze seznamu adres a zobrazí zprávu, která vás bude o přidělené adrese informovat. Pokud tato zpráva vypadá podobně jako „`Your address:  193.174.7.202`“, umožní vám následující část kódu nástroje `dip` zvolit vlastní adresu:

```
... login chat ....
wait address: 10
get $locip remote
```

Příkaz print

Tento příkaz umožňuje vypsat text na konzolu, ze které byl spuštěn nástroj `dip`. V rámci příkazu `print` lze použít libovolné proměnné definované v nástroji `dip`, například:

```
DIP> print Using port $port at speed $speed
Using port cua3 at speed 38400
```

Názvy proměnných

Nástroj `dip` rozumí pouze předdefinované skupině proměnných. Název proměnné vždy začíná symbolem dolaru a musí být psán malými písmeny.

Proměnné *$local* a *$locip* obsahují název místního hostitele a IP-adresu. Po nastavení názvu hostitele uloží nástroj `dip` do proměnné *$local* kanonický název hostitele a zároveň přiřadí proměnné *$locip* odpovídající IP-adresu. Analogický proces probíhá, když nastavujete proměnnou *$locip*.

Proměnné *$remote* a *$rmtip* provádí totéž s názvem vzdáleného hostitele a jeho IP-adresou. Proměnná *$mtu* obsahuje hodnotu MTU pro dané spojení.

Tyto proměnné jsou jedinými proměnnými, kterým mohou být přímo přidělovány hodnoty pomocí příkazu `get`. Dalších proměnné lze nastavovat pouze za pomoci odpovídajících příkazů, nicméně i tyto proměnné lze používat v rámci příkazu `print`; konrétně se jedná o proměnné *$modem*, *$port* a *$speed*.

Proměnná *$errlvl* umožňuje přistupovat k výsledku naposledy spuštěného příkazu. Návratová hodnota rovná nula značí úspěšnou operaci, zatímco nenulová hodnota naznačuje chybnou operaci.

Příkazy if a goto

Příkaz `if` představuje podmíněné větvení. Jeho syntaxe je následující:

```
if var op number goto label
```

Výraz musí být jednoduchým porovnáním jedné z proměnných *$errlvl*, *$locip* a *$rmtip* a celočíselné hodnoty; operátorem `op` může být jeden z následujících operátorů: `==`, `!=`, `<`, `>`, `<=` a `>=`.

Příkaz `goto` umožní, aby provádění skriptu pokračovalo na řádce následující po daném návěští. Návěští musí být uvedeno na začátku řádku a bezprostředně za ním musí následovat dvojtečka.

Příkazy send, wait a sleep

Tyto příkazy pomáhají implementovat do nástroje `dip` jednoduché skripty s rozhovorem. Příkaz `send` zapíše své argumenty na sériovou linku. Nepodporuje žádné proměnné, avšak rozumí všem kombinacím znaků s převráceným lomítkem, které pocházejí z jazyka C, jako je například *\n* a *\b*. Znak vlnovky (~) je používán jako zkratka pro návrat vozíku/nová řádka (`CR/LF`).

Argumentem příkazu `wait` je slovo a jeho funkce spočívá v monitorování všech vstupů po sériové lince, dokud se dané slovo neobjeví. Vlastní slovo nemůže obsahovat žádné mezery. K příkazu `wait` můžete přidat i druhý nepovinný argument, který bude reprezentovat délku trvání příkazu; pokud nebude očekávané slovo obdrženo v daném časovém limitu, příkaz skončí a nastaví proměnnou *$errlvl* na hodnotu 1.

Příkaz `sleep` slouží k nastavení časové prodlevy, například aby nástroj `dip` trpělivě vyčkal na dokončení přihlašovací procedury. Tento interval je opět zadáván v sekundách.

Příkazy mode a default

Tyto příkazy se používají k přepnutí sériové linky do režimu SLIP a ke konfiguraci rozhraní. Příkaz `mode` je posledním příkazem spuštěným v nástroji `dip` předtím, než se nástroj `dip` přepne do režimu démona.

Příkaz `mode` přijímá jako argument název protokolu. Nástroj `dip` v současné době akceptuje jako korektní názvy protokolů *SLIP* a *CSLIP*. Aktuální verze nástroje `dip` bohužel neakceptuje adaptivní protokol SLIP.

Jakmile na sériové lince povolíte režim SLIP, spustí nástroj `dip` příkaz `ifconfig`, aby nastavil rozhraní jako spojení typu point-to-point, a dále zavolá příkaz `route`, který nastaví směrování ke vzdálenému hostiteli.

Pokud skript navíc spustí před příkazem `mode` i příkaz `default`, nastaví nástroj `dip` také implicitní směr pro spojení pomocí protokolu SLIP.

7.4 Spouštění v režimu server

Nastavení klienta s protokolem SLIP bylo poměrně obtížné. Nastavení protějšku, konkrétně takové konfigurace vašeho hostitele, aby se choval jako server SLIP, je mnohem jednodušší.

Jedním ze způsobů, jak to provést, je použít nástroj `dip` v režimu server. Toho dosáhnete za pomoci nástroje `diplogin`. Jeho hlavním konfiguračním souborem je soubor `/etc/diphosts`, který sdružuje přihlašovací jména s adresou, která je danému hostiteli přidělena. Alternativně můžete používat i nástroj `sliplogin`, který je odvozen z balíku BSD a umožňuje mnohem pružnější konfigurační schéma, na jehož základě lze spouštět skripty daného rozhraní kdykoliv se hostitel připojí nebo odpojí. V současné době je tento nástroj ve fázi beta.

Oba programy vyžadují, abyste pro každého klienta SLIP nastavili jeden přihlašovací účet. Předpokládejme, že poskytujete službu SLIP panu Arthuru Dentovi, který má adresu **dent.beta.com**. Účet s názvem **dent** vytvoříte tak, že do souboru `passwd` přidáte následující řádku:

```
dent:*:501:60:Arthur Dent's SLIP account:/tmp:/usr/sbin/diplogin
```

Potom byste měli pomocí utility `passwd` nastavit heslo pro účet **dent**.

Když se nyní uživatel **dent** přihlásí, spustí se nástroj `dip` v režimu server. Aby nástroj `dip` zjistil, zda je tento uživatel oprávněn používat službu SLIP, vyhledá jeho jméno v souboru `/etc/diphosts`. Tento soubor popisuje přístupová práva a parametry spojení každého uživatele služby SLIP. Vzorová položka pro uživatele **dent** vypadá takto:

```
dent::dent.beta.com:Arthur Dent:SLIP,296
```

První pole oddělené dvojtečkou reprezentuje název účtu, pod kterým se musí uživatel přihlásit. Druhé pole může obsahovat dodatečné heslo (viz níže). Třetí pole obsahuje název hostitele nebo IP-adresu volajícího hostitele. Dále následuje informační pole, které nemá žádný speciální význam (prozatím). Poslední pole popisuje parametry spojení. Obsahuje seznam oddělený čárkou, který určuje používaný protokol (momentálně buď *SLIP,* nebo *CSLIP*), a za ním následuje hodnota MTU.

Když se uživatel **dent** přihlásí, nástroj `diplogin` si o něm vytáhne informace ze souboru `diphosts`, a pokud není druhé pole prázdné, vyzve ho k zadání „externího bezpečnostního hesla“. Řetězec zadaný uživatelem pak porovná s (nezašifrovaným) heslem v souboru `diphosts`. Pokud nesouhlasí, bude pokus o přihlášení zamítnut.

V opačném případě začne nástroj `diplogin` s přepnutím sériové linky do režimu CSLIP nebo SLIP a nastaví také rozhraní a směrování. Toto spojení trvá až do té doby, kdy se uživatel odpojí a modem zavěsí linku. Poté nástroj `diplogin` přepne linku zpět do normálního režimu a skončí.

Nástroj `diplogin` vyžaduje speciální uživatelská práva. Pokud nástroji `dip` nepřidělíte práva **superuživatel**, měli byste nástroje `diplogin` a `dip` od sebe oddělit (například vytvořením samostatné kopie nástroje `diplogin`) a nepoužívat je při jednoduchém spojení společně. V takovém případě mohou být nástroji `diplogin` přidělena přístupová práva, která neovlivní práva nástroje `dip`.

Kapitola

8

Point-to-Point Protokol

8.1 Vysvětlení protokolu PPP

Protokol PPP slouží, stejně jako protokol SLIP, k posílání datagramů po sériové lince, avšak odstraňuje spoustu nedostatků, kterými trpí protokol SLIP. Umožňuje, aby si komunikující strany na počátku spojení dohodly parametry spojení, kterými mohou být například IP-adresa nebo maximální velikost datagramu. Dále poskytuje protokol PPP možnost ověření totožnosti klienta. Pro každou z těchto vlastností má protokol PPP samostatný protokol. V této kapitole si tyto základní stavební prvky protokolu PPP popíšeme. Kapitola však zdaleka neposkytuje veškeré informace; chcete-li se toho dozvědět o protokolu PPP více, měli byste si přečíst jeho specifikaci v RFC 1548 a dále přibližně tucet doprovodných RFC.[1]

Na nejnižší hladině protokolu PPP se nachází tzv. *Vysokoúrovňový protokol pro řízení datového spojení* (*High-Level Data Link Control Protocol*), zkráceně HDLC[2], který definuje pole jednotlivých rámců protokolu PPP a poskytuje 16bitový kontrolní součet. Na rozdíl od primitivnějšího zapouzdření v protokolu SLIP může rámec v protokolu PPP obsahovat pakety i jiných protokolů, než IP, například protokolu IPX sítě Novell nebo protokolu Appletalk. Této vlastnosti je v protokolu PPP dosaženo tak, že k základnímu rámci protokolu HDLC je přidáno speciální protokolové pole, které identifikuje typ paketu přenášeného daným rámcem.

Protokol LCP, neboli protokol pro řízení spojení (Link Control Protocol), se používá nad protokolem HDLC a používá se ke sjednávání parametrů týkajících se datového spojení, jako je například maximální příjmová jednotka (Maximum Receive Unit – MRU), jež uvádí maximální velikost datagramu, kterou je jedna ze stran ochotná přijímat.

---

1 Významné RFC jsou uvedeny na konci této knihy v bibliografii s poznámkami.

2 Ve skutečnosti je protokol HDLC mnohem obecnější protokol, který navrhla organizace International Standards Organization (ISO).

Důležitým krokem ve fázi konfigurace spojení pomocí protokolu PPP je ověření totožnosti klienta. Ačkoliv není povinné, je u většiny linek nabízejících připojení pomocí modemu de facto nezbytné. Volaný hostitel (server) obvykle vyzve klienta k ověření své totožnosti za pomocí nějakého tajného klíče. Není-li volající schopen poskytnout správný tajný klíč, bude spojení ukončeno. U protokolu PPP funguje ověřování totožnosti oběma směry; to znamená, že i klient může požádat server, aby prokázal svou totožnost. Obě tyto ověřovací procedury jsou na sobě vzájemně nezávislé. Pro účely těchto dvou různých způsobů ověřování jsou k dispozici dva protokoly, o kterých se ještě zmíníme. Nazývají se protokol pro ověření hesla (Password Authentication Protocol), zkráceně PAP, a protokol na ověření inicializační výzvy (Challenge Handshake Authentication Protocol – CHAP).

Každý síťový protokol, který je směrován po datové lince, například protokoly IP, AppleTalk atd., je konfigurován dynamicky za pomoci odpovídajícího protokolu pro řízení sítě (Network Control Protocol – NCP). Například při posílání IP-datagramu po lince si musí nejprve oba hostitelé s protokolem PPP dohodnout, jakou IP-adresu bude každý z nich používat. Řídicím protokolem, který se k tomuto účelu používá, je protokol IPCP, neboli protokol na řízení internetového protokolu (Internet Protocol Control Protocol).

Kromě vlastního posílání IP-datagramů po lince podporuje protokol PPP také Van Jacobsonovu kompresi hlaviček IP-datagramů. To je technika používaná ke zmenšování velikosti hlaviček TCP-paketů až na tři bajty. Tato technika se používá také v protokolu CSLIP a obecně je známá pod názvem VJ komprese hlaviček. I použití komprese může být sjednáno při spuštění pomocí protokolu IPCP.

8.2 Protokol PPP v Linuxu

V Linuxu je funkce protokolu rozdělena na dvě části, na vysokoúrovňový ovladač HDLC, který se nachází v jádru operačního systému, a na démona uživatelského prostoru `pppd`, který se stará o různé řídicí protokoly.

Ovladač protokolu PPP v jádru operačního systému napsal Michael Callahan. Démon `pppd` byl odvozen z volně šiřitelné implementace protokolu PPP v počítačích Sun a 386BSD, jehož autorem je Drew Perkins a kol. a nyní jej spravuje Paul Mackerras. Na platformu Linuxu ho převedl Al Longyear.[3] Program `chat` napsal Karl Fox.[4]

3 Oba autoři říkají, že jsou čas od času velmi zaneprázdnění. Máte-li nějaké obecné dotazy týkající se protokolu PPP, uděláte nejlépe, když se na ně zeptáte lidí v kanálu NET, kteří jsou přihlášeni v poštovní konferenci fandů Linuxu.

4 **karl@morningstar.com**.

Protokol PPP je, stejně jako protokol SLIP, implementován pomocí speciálního režimu linky. Chcete-li používat nějakou sériovou linku jako linku s protokolem PPP, musíte nejprve obvyklým způsobem vytvořit spojení pomocí modemu a následně převést linku do režimu protokolu PPP. V tomto režimu budou všechna příchozí data podstoupena ovladači protokolu PPP, který ověří platnost rámců protokolu HDLC (každý rámec HDLC je doplněn 16bitovým kontrolním součtem), rozbalí je a odešle. V současné době je schopen spravovat IP-datagramy, volitelně i použití Van Jacobsonovy komprese hlaviček. Jakmile bude Linux podporovat i protokol IPX, dojde i k rozšíření ovladače PPP o správu IPX paketů.* Ovladači jádra operačního systému pomáhá démon `pppd`, což je démon protokolu PPP, který provádí veškerou inicializační fázi a fázi ověřování totožnosti, což je nutné před zahájením provozu po příslušné lince. Chování démona `pppd` je možné doladit pomocí mnoha parametrů. Jelikož je ovladač protokolu PPP poměrně složitý, není možné popsat všechny tyto parametry v jediné kapitole. Tato kniha tak nepokrývá všechny aspekty démona `pppd`, poskytuje jen stručný úvod. Chcete-li se o tomto problému dozvědět více, nahlédněte do manuálu a do souborů `README`, které jsou součástí distribuce zdrojového kódu démona `pppd`, a tam byste měli nalézt většinu dotazů, jež nejsou součástí této kapitoly. Pokud budete mít problémy i po přečtení těchto materiálů, obraťte se na konferenci **comp.protocols.ppp**, která sdružuje většinu lidí zainteresovaných na vývoji démona `pppd`.

8.3 Spuštění démona pppd

Když se chcete připojit na Internet pomocí protokolu PPP, musíte nejdříve nastavit základní síťovou podporu, jako je zpětnovazebné zařízení a resolver. Oba druhy síťové podpory byly probírány v předchozích kapitolách. S použitím systému DNS souvisí ještě některé věci, o kterých je třeba se zmínit; najdete je v kapitole věnované protokolu SLIP.

V úvodním příkladu, který se bude zabývat navázáním spojení PPP pomocí démona `pppd`, předpokládáme, že jste opět na hostiteli **vlager**. Již jste vytočili server PPP s názvem **c3po** a přihlásili jste se na účet **ppp**. Server **c3po** již aktivoval svůj ovladač protokolu PPP. Po ukončení komunikačního programu, který jste použili k vytočení čísla serveru, spustíte následující příkaz:

```
# pppd /dev/cua3 38400 crtscts defaultroute
```

Tento příkaz přepne sériovou linku `cua3` do režimu PPP a naváže IP-spojení se serverem **c3po**. Přenosová rychlost použitá sériovým portem bude 38 400 bps. Parametr `crtscts` zapíná na daném portu hardwarové řízení toku dat, které u rychlostí nad 9 600 bps musí být rozhodně zapnuto.

* Poznámka korektora: Linux v současné době protokol IPX plně podporuje.

Démon pppd si ihned po spuštění domluví některé charakteristiky se svým protějšekem. K tomu použije protokol LCP. Při sjednávání charakteristik budou obvykle fungovat implicitní parametry, takže se jimi zde nemusíme dále zabývat. V další kapitole se na protokol LCP podíváme detailněji.

Budeme také předpokládat, že server **c3po** od nás nebude vyžadovat žádný typ ověřování totožnosti, čímž máme konfigurační fázi úspěšně za sebou.

Následně domluví démon pppd se svým protějškem parametry IP, k čemuž použije protokol IPCP, což je protokol řídící IP. Protože jsme démonu pppd nepředali žádnou konkrétní IP-adresu, pokusí se ji získat tak, že nechá resolver vyhledat místní název hostitele. Potom si oba hostitelé sdělí své IP-adresy.

Tato implicitní nastavení jsou zpravidla dostačující. Dokonce i v případě, že je váš počítač připojen do sítě Ethernet, můžete použít stejnou IP-adresu jak pro Ethernet, tak i pro rozhraní PPP. Přesto vám démon pppd dovolí použít odlišnou adresu, případně může požádat váš protějšek, aby použil nějakou konkrétní adresu. Tyto volby jsou rozebírány dále.

Po úspěšném dokončení fáze nastavení protokolu IPCP připraví démon pppd síťovou vrstvu, která se bude používat při spojení pomocí protokolu PPP. Nejprve nastaví síťové rozhraní PPP jako spojení typu point-to-point, a to tak, že pro první aktivní spojení pomocí protokolu PPP použije rozhraní ppp0, pro druhé aktivní spojení ppp1 atd. Dále nastaví položku ve směrovací tabulce, která bude odkazovat na hostitele na druhém konci spojení. V dříve zmíněném příkladu nastaví démon pppd implicitní směr na server **c3po**, protože mu byla přiřazena volba defaultroute.[5] Tato volba způsobí, že všechny datagramy poslané hostitelům, kteří se nenacházejí ve vaší místní síti, budou poslány na server **c3po**. Démon pppd podporuje velké množství směrovacích schémat. Ty budou detailněji probrány dále v této kapitole.

8.4 Použití souborů options

Dříve, než démon pppd analyzuje argumenty příkazové řádky, projde několik souborů, zda neobsahují implicitní volby. Tyto soubory mohou obsahovat libovolné argumenty příkazové řádky, jež mohou být uvedeny na několika řádkách. Komentáře jsou uvozeny symbolem křížku.

Prvním souborem voleb je soubor s názvem /etc/ppp/options, který se při spuštění démona pppd vždy prohlíží. Je dobré využít tohoto souboru ke specifikaci některých globálních implicitních nastavení, čímž zabráníte vašim uživatelům v nechtěném poškození bezpečnosti celého systému. Chcete-li například, aby démon pppd vyžadoval od svého protějšku ně-

[5] Implicitní směr je instalován pouze v případě, že ještě není přítomen žádný implicitní směr.

jaký typ ověření totožnosti (buď pomocí protokolu PAP, nebo pomocí protokolu CHAP), uveďte v tomto souboru volbu `auth`. Tuto volbu nemůže uživatel potlačit, takže nemůže navázat spojení pomocí protokolu PPP s žádným systémem, který není v ověřovací databázi.

Druhým souborem voleb, který se čte po souboru `/etc/ppp/options`, je soubor `.ppprc`. Nachází se v domovském adresáři příslušného uživatele. Každý uživatel tak může specifikovat svoji vlastní množinu implicitních voleb.

Vzorový soubor `/etc/ppp/options` by mohl vypadat asi takto:

```
# Globální volby pro pppd běžící na vlager.vbrew.com
auth                    # požaduj autorizaci
usehostname             # pro CHAP použij lokální jméno
lock                    # používej zamykání UUCP
domain vbrew.com        # naše doména
```

První dvě volby se vztahují k procesu ověřování totožnosti a budou vysvětleny dále. Klíčové slovo `lock` nastaví démona `pppd` tak, aby vyhovoval standardní metodě UUCP, která definuje blokování zařízení. Podle tohoto standardu vytvoří každý proces, který přistupuje k sériovému zařízení, konkrétně k zařízení `/dev/cua3`, v dočasném adresáři UUCP zamykající soubor s názvem `LCK..cua3`, jehož přítomnost bude signalizovat, že je dané zařízení momentálně využíváno. To proto, aby jiné programy, například `minicom` nebo `uucico`, nemohly otevřít sériové zařízení v okamžiku, kdy ho používá protokol PPP.

Důvodem začlenění těchto voleb do globálního konfiguračního souboru je skutečnost, že uživatel nemůže výše uvedené volby potlačit a ty tím pádem poskytují solidní úroveň zabezpečení. Všimněte si však, že některé volby potlačit lze; příkladem budiž řetězec *connect*.

8.5 Vytáčení za pomoci programu chat

Jednou z věcí, která se vám mohla zdát v předchozím příkladu nepohodlná, je nutnost manuálně navázat spojení dříve, než se spustí démon `pppd`. Na rozdíl od nástroje `dip` nemá démon `pppd` svůj vlastní skriptový jazyk, který by umožňoval vytočení a přihlášení ke vzdálenému systému. Místo toho spoléhá na nějaký externí program nebo skript příkazového interpretu, který za něj tento úkol provede. Příkaz, který se má provést, lze předat démonu `pppd` pomocí volby příkazové řádky *connect*. Démon `pppd` přesměruje standardní vstup a výstup příkazu na sériovou linku. Pro tento účel existuje jeden užitečný program s názvem `expect`, jehož autorem je Don Libes. Obsahuje výkonný jazyk vycházející z jazyka Tcl, který byl navržen přesně pro tento druh aplikace.

Balík pppd obsahuje podobný program s názvem chat, který umožňuje zadat komunikační skript ve stylu UUCP. Komunikační skript je v podstatě složen ze střídající se sekvence očekávaných řetězců, které přijdou ze vzdáleného systému a z odpovědí, jež posíláme. Následuje typický výpis z komunikačního skriptu:

```
ogin: b1ff ssword: s3kr3t
```

Tento příkaz říká programu chat, aby vyčkal, dokud vzdálený systém nepošle výzvu k přihlášení, a potom mu poslal přihlašovací jméno **b1ff**. Čekáme pouze na řetězec ogin:, takže nebude vadit, když bude přihlašovací výzva začínat malým nebo velkým písmenem l, nebo když dorazí ve zkomolené formě. Následující řetězec je opět očekávaný řetězec, který pozdrží program chat, dokud nepřijde výzva k zadání hesla, potom odešle naše heslo jako odpověď.

To je v podstatě hlavní účel komunikačních skriptů. Kompletní skript sloužící k vytáčení serveru PPP by musel samozřejmě obsahovat patřičné příkazy pro modem. Předpokládejme, že váš modem rozumí množině příkazů standardu Hayes a že telefonní číslo vytáčeného serveru je 318 714. Kompletní vyvolání programu chat, které by provedlo spojení se serverem **c3po,** by potom vypadalo následovně:

```
$ chat -v '' ATZ OK ATDT318714 CONNECT '' ogin: ppp word: GaGariN
```

Podle definice musí být první řetězec očekávaný řetězec, ale protože nám modem nic neodpoví, dokud ho nepobídneme, musíme sdělit programu chat, aby první očekávaný řetězec přeskočil. Provedeme to tak, že tento řetězec zadáme jako prázdný řetězec. Pak pokračujeme odesláním příkazu ATZ, což je inicializační řetězec pro modemy kompatibilní se standardem Hayes a následně budeme čekat na odpověď (OK). Následující řetězec pošle vytáčecí program programu chat společně s telefonním číslem a jako odpověď očekává zprávu CONNECT. Za ní opět následuje prázdný řetězec. Protože v tuto chvíli nechceme nic posílat, čekáme spíše na výzvu k přihlášení. Zbytek komunikačního skriptu funguje naprosto shodně s výše popsaným postupem.

Parametr -v zajistí, že program chat bude veškeré aktivity zapisovat do souboru.[6]

Zadávání komunikačního skriptu na příkazové řádce s sebou nese jistá rizika, protože si uživatelé mohou prohlédnout příkazovou řádku daného procesu pomocí příkazu ps. Tomuto problému se vyhnete, když umístíte komunikační skript do souboru, řekněmě *dial-c3po*. Pomocí parametru -f, za kterým následuje název souboru, sdělíte programu chat, aby četl skript ze souboru místo z příkazové řádky. Kompletní spuštění démona pppd by mohlo nyní vypadat následovně:

6 Upravíte-li soubor syslog.conf z důvodu přesměrování těchto kontrolních zpráv do souboru, pak se ujistěte, že k tomuto souboru nemůže přistupovat každý, protože program chat implicitně zapisuje celý komunikační skript, tedy úplně vše včetně hesel.

```
# pppd connect "chat -f dial-c3po" /dev/cua3/ 38400 -detach \
        crtscts modem defaultroute
```

Kromě volby *connect* specifikující skript pro vytáčení jsme do příkazové řádky přidali ještě další dvě volby: `-detach` sdělí démonu `pppd`, aby se odpojil od konzoly a jako proces se přepnul na pozadí. Klíčové slovo `modem` sděluje démonu `pppd`, aby na sériovém zařízení provedl některé akce specifické pro modem, jako je zavěšení linky před a po volání. Pokud toto klíčové slovo nepoužijete, nebude démon `pppd` monitorovat linku DCD na sériovém portu, a proto nebude schopen detekovat neočekávané zavěšení na vzdálené straně.

Výše uvedené příklady byly poměrně jednoduché; program `chat` však umožňuje psát mnohem složitější komunikační skripty. Jednou z užitečných vlastností je určení řetězce, při kterém se komunikační skript zastaví s chybou. Typickými řetězci pro zastavení běhu skriptu jsou zprávy jako `BUSY` nebo `NO CARRIER`, které generuje váš modem v případě, že je volané číslo obsazené nebo vzdálená strana nezvedá telefon. Aby program `chat` rozpoznal tyto zprávy okamžitě, ne až po uplynutí určité doby, můžete je zadat na začátku skriptu pomocí klíčového slova `ABORT`:

```
$ chat -v ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ...
```

Podobným způsobem můžete ve skriptu změnit hodnotu čekací doby pomocí volby `TIMEOUT`. Podrobnosti získáte na stránkách manuálu programu `chat(8)`.

Někdy budete také potřebovat určitý druh podmíněného spouštění jednotlivých částí komunikačního skriptu. Když například neobdržíte od vzdáleného konce výzvu pro přihlášení, budete chtít poslat příkaz BREAK nebo znak návrat vozíku (CR). Toho docílíte přidáním podřetězce k očekávanému řetězci. Ten se bude skládat ze sekvence posílaných a očekávaných řetězců, stejně jako tomu bylo v celém skriptu. Tyto řetězce jsou odděleny znakem pomlčka. Podřetězec je spuštěn vždy, když očekávaný řetězec, ke kterému se daný podřetězec vztahuje, není přijat v zadaném čase. V níže uvedeném příkladu upravíme komunikační řetězec následujícím způsobem:

```
ogin:-BREAK-ogin: ppp ssword: GaGariN
```

Když program `chat` neuvidí výzvu pro přihlášení, která by měla přijít ze vzdáleného konce, spustí podřetězec, který nejdříve pošle příkaz BREAK a potom bude čekat na opětovnou výzvu pro přihlášení. Pokud se nyní tato výzva objeví, bude skript pokračovat běžným způsobem, v opačném případě skončí s chybou.

8.6 Ladění nastavení protokolu PPP

Démon `pppd` bude implicitně zapisovat veškeré varování a chybové zprávy do souboru *syslog démona*. Do souboru `syslog.conf` musíte přidat položku, která toto zapisování přesměruje do souboru, případně na konzolu, v opačném případě prostředky syslog tyto zprávy zruší. Následující položka způsobí, že budou všechny zprávy posílány do souboru `/var/log/ppp-log`:

```
daemon.*                    /var/log/ppp-log
```

Jestliže vaše nastavení nefunguje ihned, můžete na základě tohoto log-souboru poskytnout první indicii o tom, co je v nepořádku. Pokud vám ani to nepomůže, můžete také pomocí volby `debug` zapnout speciální ladicí výstup. Tato volba nařídí démonu `pppd`, aby zapisoval do prostředků `syslog` obsahy všech poslaných nebo přijatých řídicích paketů.

Konečně nejdrastičtějším způsobem je povolení ladicích informací na úrovni jádra operačního systému. To je možné provést spuštěním démona `pppd` s volbou `kdebug`. Za touto volbou následuje číselný argument, který je bitově orientovanou operací OR nad následujícími hodnotami: 1 pro obecné ladicí informace, 2 pro vytištění obsahu všech příchozích rámců protokolu HDLC a při hodnotě 4 vypíše ovladač veškeré odcházející rámce protokolu HDLC. Abyste mohli zachytávat ladicí zprávy jádra operačního systému, musíte buď spustit démona `syslogd`, který čte soubor `/proc/kmsg` nebo démona `klogd`. Oba tito démoni směrují ladící informace jádra do souboru `syslog`.

8.7 Volby pro konfiguraci protokolu IP

Protokol IPCP slouží k dohodnutí několika parametrů IP v době konfigurace spojení. Každá z komunikujících stran může poslat paket s požadavky na konfiguraci protokolu IPCP (IPCP Configuration Reguest), který obsahuje informace o implicitních hodnotách, jež se mají změnit a jaké mají být jejich hodnoty. Po přijetí tohoto paketu prozkoumá vzdálený konec střídavě každou z těchto voleb a buď ji přijme, nebo ji odmítne.

Démon `pppd` vám dává k dispozici velký prostor pro řízení parametrů protokolu IPCP, které se bude snažit dohodnout. Tyto parametry můžete nastavovat pomocí různých voleb příkazové řádky, které probereme později.

8.7.1 Výběr IP-adres

V předchozím příkladu nám démon `pppd` vytočil server **c3po** a navázal spojení IP. Nebyla provedeny žádná opatření, která by některému z obou konců přidělila konkrétní IP-adresu. Místo toho jsme použili adresu hostitele **vlager** jako místní IP-adresu a serveru **c3po** jsme po-

nechali jeho vlastní IP-adresu. Někdy je však užitečné mít kontrolu nad tím, která adresa bude použita na jednom nebo druhého konci spojení. Démon `pppd` poskytuje několik typů takovéto kontroly.

Chcete-li si vyžádat konkrétní adresy, stačí démonu `pppd` předat následující volbu:

```
local_addr:remote_addr
```

Parametry `local_addr` a `remote_addr` mohou být zapsány buď ve formě tečkové notace, nebo jako názvy hostitelů.[7] Tento příkaz způsobí, že se démon `pppd` pokusí použít první adresu jako svou vlastní IP-adresu a druhou adresu jako IP-adresu protějšku. Pokud protějšek v průběhu sjednávání parametrů pomocí protokolu IPCP některou z těchto adres odmítne, nedojde k řádnému spojení IP.[8]

Pokud chcete nastavit pouze místní adresu a přitom hodláte akceptovat libovolnou adresu protějšku, ponechte část `remote_addr` volnou. Chcete-li například, aby server **vlager** používal místo své vlastní IP-adresy adresu **130.83.4.27**, měli byste na příkazové řádce zadat `130.83.4.27:`. Podobně chcete-li nastavit pouze adresu vzdáleného počítače, ponechte volné pole `local_addr`. Démon `pppd` pak použije adresu sdruženou s názvem vašeho hostitele.

Některé servery PPP, které obsluhují velké množství klientů, přidělují IP-adresy dynamicky: adresy jsou systému přiděleny pouze v okamžiku volání a ihned po jeho odhlášení jsou opět uvolněny. Spojujete-li se s takovýmto typem serveru, musíte se ujistit, že démon `pppd` nevyžaduje od serveru žádnou konkrétní IP-adresu, ale přijme adresu, kterou po vás server požaduje. To znamená, že nemůžete použít argument `local_addr`. Kromě toho musíte použít volbu `noipdefault`, která sdělí démonu `pppd`, aby nepoužil místní adresu hostitele, ale počkal na přidělení IP-adresy od protějšího počítače.

8.7.2 Směrování přes spojení pomocí protokolu PPP

Jakmile je nastaveno síťové rozhraní, nastaví obvykle démon pouze hostitelské směrování ke svému protějšku. Je-li vzdálený hostitel v síti LAN, budete se jistě chtít spojit také s hostiteli, kteří jsou „za hranicemi“ vašeho protějšku; to znamená, že musí být nastaveno směrování sítí.

Už jsme si ukázali, že démona `pppd` je možné za pomoci volby `defaultroute` požádat, aby nastavil implicitní směr. Tato volba je velmi užitečná v případě, že se vámi vytočený server PPP bude chovat jako internetová brána.

---

7 Použijete-li v této volbě názvy hostitelů, bude to mít nepříznivé důsledky na ověření totožnosti pomocí protokolu CHAP. Podrobnosti najdete v níže uvedené stati o protokolu CHAP.

8 Protějšku s protokolem PPP můžete povolit potlačení vašich návrhů IP-adres tak, že démonu *pppd* předáte volby `ipcp-accept-local` a `ipcp-accept-remote`. Detaily získáte na stránkách manuálu.

Realizace obráceného případu, kdy se váš systém chová pro konkrétního hostitele jako brána, je také poměrně jednoduchá. Vezměte si například několik zaměstnanců ze společnosti Virtual Brewery, jejichž domovský počítač má název **loner**. Když se takový zaměstnanec spojí s hostitelem **vlager** pomocí protokolu PPP, použije ke spojení adresu podsítě virtuálního pivovaru. Na bráně **vlager** můžeme nyní použít volbu `proxyarp`, která nainstaluje pro hostitele **loner** proxy ARP. Takto bude hostitel **loner** automaticky dostupný ze všech hostitelů, kteří se nacházejí v síti pivovaru nebo vinárny.

Ne vždy to ale jde tak jednoduše, jako v tomto případě, příkladem budiž spojení dvou lokálních sítí. To obvykle vyžaduje přidání konkrétního síťového směrování, protože tyto sítě již mohou mít své vlastní implicitní směry. Kromě toho, když necháte u obou protějšků nastaven implicitní směr na spojení pomocí protokolu PPP, vznikne smyčka, ve které budou mezi oběma protějšky obíhat pakety určené pro neznámé lokace tak dlouho, dokud nevyprší jejich doba přežití (TTL).

Jako příklad předpokládejme, že si společnost Virtual Brewery otevře pobočku v nějakém jiném městě. Tato pobočka bude mít svůj vlastní Ethernet, který bude používat IP-adresu **191.72.3.0**, což je podsíť 3 v síti pivovaru. Síť pivovaru je třídy B. Lidé z pobočky se chtějí spojit s hlavním Ethernetem pivovaru pomocí spojení PPP, aby si mohli aktualizovat databáze zákazníků atd. Hostitel **vlager** se opět chová jako brána; jeho protějšek se nazývá **sub-etha** a má přidělenu IP-adresu **191.72.3.1**.

Když se hostitel **sub-etha** spojí s hostitelem **vlager**, nastaví jako obvykle implicitní směr na hostitele **vlager**. Nicméně na bráně **vlager** musíme nainstalovat síťové směrování pro podsíť 3, které bude ukazovat na hostitele **sub-etha**. K tomuto účelu použijeme ještě neprobranou vlastnost démona `pppd` – příkaz `ip-up`. Jedná se o skript příkazového interpretu nebo program umístěný v adresáři `/etc/ppp`, který je spouštěn po konfiguraci rozhraní PPP. Je-li tento program přítomný, spustí se s následujícími parametry:

```
ip-up iface device speed local_addr remote_addr
```

Argument `iface` označuje používané síťové rozhraní, parametr `device` představuje cestu k používanému sériovému zařízení (jsou-li použity parametry stdin/stdout, bude tato cesta ukazovat do souboru `/dev/tty`) a parametr `speed` představuje rychlost daného zařízení. Parametry `local_addr` a `remote_addr` předávají IP-adresy, které použijí oba počítače účastnící se spojení. Tyto adresy jsou uvedeny v tečkové notaci. V našem příkladu by měl skript `ip-up` obsahovat následující část kódu:

```
#!/bin/sh
case $5 in
191.72.3.1)             # to je sub-etha
```

```
        route add -net 191.72.3.0 gw 191.72.3.1;;
esac
exit 0
```

Jakmile je spojení pomocí protokolu PPP ukončeno, je podobným způsobem použit skript `/etc/ppp/ip-down`, který vrátí zpět všechny akce provedené skriptem `ip-up`.

Nicméně směrovací schéma ještě není kompletní. Na obou hostitelích s protokolem PPP jsme nastavili položky směrovací tabulky, avšak žádný z hostitelů obou sítí dosud neví nic o existujícím spojení pomocí protokolu PPP. To nebude problém v případě, že mají všichni hostitelé v pobočce vinárny nastaveno svůj implicitní směr na hostitele **sub-etha** a všichni hostitelé v síti pivovaru mají nastaven implicitní směrování na bránu **vlager**. Není-li to váš případ, budete zřejmě nuceni použít nějakého směrovacího démona, například démona *gated*. Jakmile na bráně **vlager** vytvoříte síťové směrování, bude směrovací démon vysílat nové směrování všem hostitelům, kteří jsou připojeni k podsítím.

8.8 Řídicí parametry spojení

V předcházejících statích jsme se setkali s protokolem LCP (Protokol na řízení spojení), který je používán pro sjednávání charakteristik spojení a k testování spojení.

Dvě nejdůležitější volby, jež mohou být sjednávány pomocí protokolu LCP, jsou maximální příjmová jednotka (MRU) a asynchronní řídicí mapa znaků (Asynchronous Control Character Map). Existuje i spousta dalších voleb, které je možné pomocí tohoto protokolu nastavovat, ty jsou však příliš specializované na to, abychom se zde jimi mohli zabývat. Jejich případný popis najdete v RFC 1548.

Asynchronní řídicí mapa znaků, které se hovorově říká asynchronní mapa, slouží u asynchronních spojení, jako jsou telefonní linky, k identifikaci řídicích znaků, které musí být vynechány (musí být nahrazeny konkrétní sekvencí dvou znaků). Měli byste se například vyvarovat použití znaků XON a XOFF, které jsou používány k softwarovému řízení toku dat a špatně nakonfigurovaný modem by se mohl při přijetí takového znaku zablokovat. Dalším kandidátem je kombinace kláves `Ctrl+]` (znak pro ukončení `telnetu`). Protokol PPP umožňuje vynechat libovolné znaky s ASCII kódy od 0 do 31, pokud jsou uvedeny v asynchronní mapě.

Asynchronní mapa je bitová mapa o šířce 32 bitů, kde nejnižší platný bit odpovídá ASCII znaku NUL a nejvyšší platný bit odpovídá ASCII znaku s hodnotou 31. Je-li příslušný bit nastaven, znamená to, že odpovídající znak musí být před odesláním po lince vynechán. Implicitně je asynchronní mapa nastavena na hodnotu *0xffffffff*, to znamená, že budou všechny řídicí znaky vynechány.

Chcete-li sdělit vašemu protějšku, že nemusí přeskakovat všechny řídicí znaky, ale jen některé z nich, můžete zadat novou asynchronní mapu, kterou předáte démonu `pppd` pomocí volby `asyncmap`. Mají-li se přeskočit pouze znaky `^S` a `^Q` (znaky s ASCII hodnotou 17 a 19, které se obecně používají pro řízení XON a XOFF), použijte následující volbu:

```
asyncmap 0x000A0000
```

Maximální příjmová jednotka neboli MRU signalizuje protějšku, jakou maximální velikost rámců protokolu HDLC chceme používat. I když vám to možná připomíná hodnotu MTU (maximální přenosová jednotka), nemají spolu tyto dvě hodnoty téměř nic společného. Hodnota MTU je parametrem pro síťové zařízení jádra operačního systému a popisuje maximální velikost rámce, o kterou se může příslušné rozhraní starat. Hodnota MRU říká vzdálenému konci, že nemá generovat rámce větší než je hodnota MRU; rozhraní však musí být bez ohledu na tuto skutečnost schopno přijmout rámečky až do velikosti 1 500 bajtů.

Volba hodnoty MRU proto není ani tak otázkou toho, co je daná linka schopna přenést, jako spíše otázkou nastavení, se kterým dosáhnete nejvyššího výkonu. Pokud hodláte po příslušném spojení provozovat interaktivní aplikace, pak je dobré nastavit hodnotu MRU přinejmenším na hodnotu 296 bajtů, aby nějaký větší paket (konkrétně například paket z relace FTP) nezpůsobil „skok“ vašeho kurzoru. Chcete-li démonu `pppd` sdělit, že hodláte používat MRU o velikosti 296 bajtů, měli byste mu tuto hodnotu předat pomocí parametru `mru 296`. Nicméně malé hodnoty MRU mají smysl pouze v případě, že nemáte zakázánu VJ kompresi hlaviček (implicitně je povolena).

Démon `pppd` rozumí také některým volbám protokolu LCP, které nastavují celkové chování procesu sjednávání parametrů, jako je maximální počet konfiguračních požadavků, které si mohou obě strany vyměnit, než se spojení ukončí. Dokud si nebudete absolutně jisti tím, co děláte, neměli byste tyto parametry měnit.

Kromě toho existují ještě dvě volby týkající se opakování echo zpráv protokolu LCP. Protokol PPP definuje dvě zprávy, opakování požadavku (Echo Request) a opakování odpovědi (Echo Response). Démon `pppd` používá tuto vlastnost ke kontrole funkčnosti spojení. Tuto vlastnost můžete povolit za pomoci volby `lcp-echo-interval`, za kterou následuje časový údaj v sekundách. Pokud nebudou v tomto intervalu ze vzdáleného hostitele přijaty žádné rámečky, vygeneruje démon `pppd` zprávu Echo Request a bude očekávat, že mu protějšek vrátí hlášku Echo Response. Jestliže se tak nestane, pak se spojení po určitém počtu odeslaných požadavků ukončí. Tento počet je možné nastavit pomocí volby `lcp-echo-failure`. Implicitně je tato volba zakázána.

8.9 Obecné úvahy nad bezpečností

Špatně nakonfigurovaný démon protokolu PPP může mít z hlediska bezpečnosti zhoubný vliv na celý systém. V nejhorším případě to vypadá tak, že má každý uživatel možnost připojení do vaší sítě Ethernet (a to je velice špatné). V této stati si povíme o několika málo opatřeních, které by měly zabezpečit vaši konfiguraci protokolu PPP.

Jedním z problémů démona `pppd` je to, že pro konfiguraci síťových zařízení a směrovací tabulky vyžaduje přístupová práva **superuživatele**. To obvykle vyřešíte tak, že ho necháte běžet setuid **root**. Nicméně démon `pppd` dovoluje uživatelům nastavovat z hlediska bezpečnosti různě významné volby. Abyste se chránili před útoky, které může uživatel spustit při manipulaci s těmito volbami, doporučuje se nastavit několik implicitních hodnot do souboru `/etc/ppp/options`, například ty, které jsme použili v ukázkovém souboru v sekci Použití souboru `options`. Některé z nich, například volby pro ověření totožnosti, nemůže uživatel potlačit, a proto poskytují dostatečnou ochranu před zneužitím.

Samozřejmě, že se musíte chránit i ze strany systémů, se kterými provozujete spojení na bázi protokolu PPP. Abyste se chránili před hostiteli, kteří se vydávají za někoho jiného, měli byste při komunikaci s protějškem vždy používat nějaký druh ověření totožnosti. Kromě toho byste měli cizím hostitelům zakázat používání IP-adres dle vlastního výběru a omezit jejich výběr jen na několik málo IP-adres. V následující stati se budeme zabývat právě těmito tématy.

8.10 Ověřování totožnosti pomocí protokolu PPP

8.10.1 Protokol CHAP versus protokol PAP

Pokud systém používá protokol PPP, může požadovat po svém protějšku, aby dokázal svoji totožnost pomocí jednoho ze dvou protokolů sloužících k ověřování totožnosti. Jsou to protokol pro ověření hesla (Password Authentication Protocol - PAP) a protokol pro ověření inicializační výzvy (Challenge Handshake Authentication Protocol – CHAP). Po navázání spojení může každá strana žádat po svém protějšku, aby dokázal svoji totožnost, bez ohledu na to, zda jde o volajícího nebo o volaného. V dalším výkladu budu v případech, kdy bude nutné, rozlišovat mezi ověřovaným systémem a ověřovatelem neurčité pojmy ‚klient' a ‚server'. Démon protokolu PPP může požádat svůj protějšek, aby dokázal svoji totožnost. To provede tak, že pomocí protokolu LCP pošle další konfigurační požadavek, v němž bude uveden požadovaný ověřovací protokol.

Protokol PAP pracuje v podstatě stejně jako klasická přihlašovací procedura. Klient ověří svoji totožnost tak, že serveru pošle jméno uživatele a (volitelně zašifrované) heslo, tato data server porovná se svou vlastní tajnou databází. Tuto techniku mohou obejít tzv. naslouchači, kteří se snaží získat potřebná hesla tak, že odposlouchávají sériovou linku a potom provádějí útoky systémem pokus omyl.

Protokol CHAP takové nedostatky nemá. Ověřovatel (tj. server) pošle klientovi náhodně vygenerovaný řetězec s „výzvou“ a spolu s ním i svůj název hostitele. Klient na základě názvu hostitele vyhledá příslušné tajné informace, zkombinuje je s přijatou výzvou a zašifruje tento řetězec pomocí jednosměrné šifrovací funkce. Výsledek pak společně s názvem hostitele klienta pošle zpět serveru. Server pak provede stejné výpočty a dojde-li k témuž výsledku, povolí klientovi přístup.

Další vlastností protokolu CHAP je, že nevyžaduje po klientovi ověření jeho totožnosti jen při spuštění, ale posílá výzvy v pravidelných intervalech, aby se ujistil, že klienta nenahradil nějaký vetřelec, například přepnutím telefonních linek.

Démon `pppd` implicitně nevyžaduje po svém protějšku ověření totožnosti, ale souhlasí se svým vlastním ověřením totožnosti v případě, že je o to vzdáleným počítačem požádán. Protože je protokol CHAP mnohem výkonnější než protokol PAP, snaží se ho démon `pppd` používat, kdykoliv jen je to možné. Pokud ho protějšek nepodporuje nebo pokud démon `pppd` nemůže pro vzdálený systém nalézt v souboru `chap-secrets` tajné informace protokolu CHAP, pak démon přepne na protokol PAP. Nenajde-li tajné informace pro svůj protějšek ani v protokolu PAP, odmítne prokázat svoji totožnost. V důsledku toho dojde k ukončení spojení.

Toto chování se může upravit několika způsoby. Použijete-li například klíčové slovo `auth`, bude démon `pppd` požadovat po svém protějšku, aby prokázal svou totožnost. Pro tento účel bude démon souhlasit s použitím protokolu CHAP nebo protokolu PAP, má-li ve své databázi CHAP, resp. PAP uloženy tajné informace pro svůj protějšek. Existují i další volby, kterými je možno zapnout nebo vypnout konkrétní protokol pro ověření totožnosti, avšak ty zde nebudeme rozebírat. Chcete-li o nich více podrobností, nahlédněte prosím do manuálové stránky démona `pppd(8)`.

Budou-li všechny systémy, se kterými máte spojení pomocí protokolu PPP, souhlasit s ověřením své totožnosti, měli byste vložit volbu `auth` do globálního souboru `/etc/ppp/options` a pro každý systém definovat v souboru `chap-secrets` příslušná hesla. Pokud daný systém protokol CHAP nepodporuje, vložte záznam o tomto systému do souboru `pap-secrets`. Pak si můžete být jisti, že se žádný neověřený systém k vašemu hostiteli nepřipojí.

Následující dvě stati budou probírat tajné soubory protokolu PPP, konkrétně soubory `pap-secrets` a `chap-secrets`. Najdete je v adresáři `/etc/ppp` a obsahují trojici klientů, serverů a hesel, volitelně následované seznamem IP-adres. Interpretace polí klienta a serveru je u protokolu CHAP jiná, než u protokolu PAP a závisí také na tom, zda dokazujeme svoji totožnost našemu protějšku, nebo zda požadujeme po serveru, aby nám svoji totožnost prokázal on.

8.10.2 Soubor Secrets protokolu CHAP

Když musíte nějakému serveru dokázat svoji totožnost pomocí protokolu CHAP, vyhledá démon `pppd` v souboru `chap-secrets` položku, která má pole klienta totožné s polem názvu místního hostitele a pole serveru bude mít totožné s názvem vzdáleného hostitele, který byl doručen pomocí výzvy protokolu CHAP. Když požadujete po protějšku, aby dokázal svoji totožnost, budou role obrácené: démon `pppd` vyhledá položku, u které se pole klienta shoduje s názvem vzdáleného hostitele (který je zaslán v odpovědi protokolu CHAP) a pole serveru je shodné s názvem místního hostitele.

Následuje vzorový soubor `chap-secrets` pro bránu **vlager**:[9]

```
# Soubor CHAP secrets pro vlager.vbrew.com
#
# klient          server           tajemství            adresa
#-------------------------------------------------------------------
vlager.vbrew.com  c3po.lucas.com   "Use The Source" vlager.vbrew.com
c3po.lucas.com    vlager.vbrew.com "riverrun, pasteve"  c3po.lucas.com
*                 vlager.vbrew.com "VeryStupidPassword" pub.vbrew.com
```

Při spojení pomocí protokolu PPP s hostitelem **c3po** požádá hostitel **c3po** bránu **vlager**, aby mu dokázala svoji totožnost pomocí protokolu CHAP. Brána to provede tak, že pošle výzvu protokolu CHAP. Poté démon `pppd` vyhledá v souboru `chap-secrets` položku, která má pole klienta shodné s adresou **vlager.vbrew.com** a její pole serveru se shoduje s adresou **c3po.lucas.com**;[10] v našem příkladu to bude položka na prvním řádku. Pak vytvoří démon `pppd` z řetězce obsahujícího výzvu a z tajných informací (`Use The Source`) odpověď pro protokol CHAP a pošle ji hostiteli **c3po**.

Ve stejném okamžiku sestaví démon `pppd` výzvu protokolu CHAP určenou pro hostitele **c3po**, která bude obsahovat jedinečný řetězec s výzvou společně s plně kvalifikovaným názvem hostitele **vlager.vbrew.com**. Hostitel **c3po** vytvoří výše popsaným způsobem odpověď pro protokol CHAP a tuto odpověď vrátí zpět bráně **vlager**. Nyní démon `pppd` vyjme z od-

[9] Uvozovky nejsou součástí hesla, slouží pouze k ochraně bílých mezer uvnitř hesla.

[10] Toto jméno je získáno z výzvy protokolu CHAP

povědi hostitelský název klienta (**c3po.vbrew.com**) a vyhledá v souboru `chap-secrets` řádek, v němž klientovi odpovídá hostitel **c3po** a serveru pak hostitel **vlager**. Této podmínce vyhovuje druhý řádek, takže démon `pppd` zkombinuje výzvu protokolu CHAP s tajnou informací `riverrun, pasteve`, zašifruje je a výsledek porovná s odpovědí protokolu CHAP, která přišla od hostitele **c3po**.

Volitelné čtvrté pole obsahuje seznam IP-adres, které jsou přijatelné pro klienty uvedené v prvním poli. Adresy mohou být zadány v tečkové notaci nebo jako názvy hostitelů, které vyhledá resolver. Požaduje-li například hostitel **c3po** během sjednávání spojení pomocí protokolu IPCP IP-adresu, která není uvedena v tomto seznamu, bude požadavek zamítnut a protokol IPCP bude ukončen. Proto je ve výše uvedeném vzorovém souboru hostitel **c3po** omezen pouze na použití své vlastní IP-adresy. Je-li pole s adresami prázdné, budou povoleny libovolné adresy; pokud je zde znak „–", nemůže daný klient použít žádnou IP-adresu.

Třetí řádek ve vzorovém souboru `chap-secrets` povoluje libovolnému hostiteli navázat spojení pomocí protokolu PPP s bránou **vlager**, protože hodnota * u pole klienta nebo serveru odpovídá libovolnému názvu hostitele. Jediným požadavkem na spojení je, že klient musí znát tajné informace a musí používat adresu **pub.vbrew.com**. Položky se zástupnými znaky představující názvy hostitelů se mohou v souboru s tajnými informacemi objevit kdekoliv, protože démon `pppd` bude vždy používat nejkonkrétnější položku, která se vztahuje k danému páru polí server/klient.

Dále bychom měli říci něco o způsobu, jakým démon `pppd` dospěje k názvům hostitelů v souboru tajných informací. Jak jsme si již dříve vysvětlili, název vzdáleného hostitele dodá vždy protějšek v paketu výzvy protokolu CHAP nebo v paketu odpovědi protokolu CHAP. Místní název hostitele bude implicitně odvozen z volání funkce *gethostname(2)*. Pokud jste nastavili název systému jako nekvalifikovaný název hostitele, pak musíte démonu `pppd` poskytnout název domény pomocí volby `domain`:

```
# pppd ...domain vbrew.com
```

Tato volba připojí k hostiteli **vlager** název domény pivovaru u všech aktivit, které se vztahují k ověřování totožnosti. Dalšími volbami, které upravují představu démona `pppd` o názvu místního hostitele, jsou volby `usehostname` a `name`. Když na příkazovou řádku zadáte místní IP-adresu pomocí volby „`local` : varremote", kde parametr `local` je název hostitele místo adresy respektující tečkovou notaci, použije démon `pppd` tento zápis jako název místního hostitele. Více podrobností najdete na manuálové stránce `pppd(8)`.

8.10.3 Soubor Secrets protokolu PAP

Soubor s tajnými informacemi protokolu PAP je velmi podobný souboru, který využívá protokol CHAP. První dvě pole vždy obsahují jméno uživatele a název serveru; třetí pole obsahuje tajné informace protokolu PAP. Když vzdálený počítač pošle požadavek na ověření totožnosti, použije démon `pppd` položku, která má pole server shodné s názvem místního hostitele a pole se jménem uživatele má shodné se jménem uživatele, které je posláno v žádosti. V době, kdy démon `pppd` dokazuje svému protějšku svoji totožnost, vybere démon `pppd` ze souboru tajných informací tu položku, u níž se pole se jménem uživatele shoduje s místním uživatelským jménem a pole serveru se shoduje s názvem vzdáleného hostitele. Potom tyto tajné informace odešle.

Vzorový soubor tajných informací protokolu PAP může vypadat asi takto:

```
# /etc/ppp/pap-secrets
#
# uživatel    server        heslo               adresa
vlager-pap    c3po          cresspahl           vlager.vbrew.com
c3po          vlager        DonaldGNUth         c3po.lucas.com
```

První řádek používáme při rozhovoru s hostitelem **c3po**. Druhý řádek popisuje, jak nám může uživatel se jménem **c3po** dokázat svoji totožnost.

Název **vlager-pap** ve sloupci jedna představuje jméno uživatele, které pošleme hostiteli **c3po**. Démon `pppd` vezme implicitně název místního hostitele jako uživatelské jméno, ale pomocí volby `user` můžete také zadat jiné jméno uživatele.

Při výběru položky ze souboru `pap-secrets` musí démon `pppd` znát název vzdáleného hostitele. Protože neexistuje žádný způsob, jak by ho mohl zjistit, musíte mu ho předat na příkazové rádce pomocí volby `remotename`, za níž následuje název hostitele vašeho protějšku. Například, abychom mohli používat výše uvedenou položku pro dokázání naší totožnosti hostiteli **c3po**, musíme na příkazovou řádku démona `pppd` doplnit následující volbu:

```
# pppd ... remotename c3po user vlager-pap
```

Ve čtvrtém poli (a ve všech následujících polích) můžete zadat, jaké adresy může daný hostitel používat, je to stejné jako u souboru s tajnými informacemi protokolu CHAP. Protějšek pak může požadovat adresy pouze z tohoto seznamu. Ve vzorovém příkladu požadujeme, aby hostitel **c3po** používal svou skutečnou IP-adresu.

Všimněte si, že protokol PAP představuje poměrně slabou metodu na ověření totožnosti a proto se doporučuje, kdykoliv je to možné, používat místo něj protokol CHAP. Z toho důvodu zde nebudeme detailněji rozebírat protokol PAP. Popis některých dalších vlastností protokolu PAP najdete na manuálové stránce `pppd(8)`.

8.11 Konfigurace serveru PPP

Spuštění démona `pppd` jako serveru je pouze otázkou doplnění příslušných voleb do příkazové řádky. Teoreticky je nutné vytvořit speciální účet, konkrétně účet **ppp**, přidělit mu nějaký skript nebo program, který bude fungovat jako přihlašovací příkazový interpret, jenž spustí démona `pppd` s těmito volbami. Do souboru `/etc/passwd` byste mohli například přidat následující řádek:

```
ppp:*:500:200:Public PPP Account:/tmp:/etc/ppp/ppplogin
```

Je možné, že budete chtít používat jiná než výše uvedená uid a gid. Dále budete muset nastavit pro výše uvedený účet nějaké heslo pomocí příkazu `passwd`.

Pak by mohl skript `ppplogin` vypadat následovně:

```
#!/bin/sh
#  ppplogin - skript pro spuštění pppd při přihlášení
mesg n
stty -echo
exec pppd -detach silent modem crtscts
```

Příkaz `msg` zabrání tomu, aby mohli ostatní uživatelé zapisovat do tty, například pomocí příkazu `write`. Příkaz `stty` vypíná opakování znaků. To je nutné z toho důvodu, že jinak by se na protějším počítači opakovaly všechny znaky, které pošle. Nejdůležitější z výše uvedených voleb démona `pppd` je volba `-detach`, protože zabraňuje démonu `pppd` odpojení od tty. Pokud tuto volbu neuvedeme, přejde démon `pppd` do pozadí a tím pádem se ukončí i skript příkazového interpretu.. To by ve svém důsledku mohlo znamenat zavěšení sériové linky a ukončení spojení. Volba `silent` sděluje démonu `pppd`, aby před začátkem posílání paketů vyčkal na příchozí paket z volaného systému. Tato volba zamezí vypršení přenosové doby v případě, kdy je volající systém příliš pomalý při spouštění svého klienta PPP. Volba `modem` sděluje démonu `pppd`, aby kontroloval linku DTR, pomocí níž lze zjistit, zda protějšek neukončil spojení. Volba `crtscts` zapíná hardwarové řízení toku dat.

Kromě těchto voleb můžete vyžadovat určitý druh ověření totožnosti, například pomocí volby `auth` v příkazové řádce nebo v globálním souboru s volbami. Manuálové stránky se také zmiňují o speciálních volbách, které slouží k zapnutí nebo vypnutí protokolů na ověření totožnosti.

Kapitola

9

Různé síťové aplikace

Poté, co úspěšně nastavíte IP a resolver, musíte začít věnovat pozornost službám, které hodláte po síti poskytovat. Tato kapitola se zabývá konfiguracemi několika malých síťových aplikací, včetně serveru `inetd` a programů z rodiny `rlogin`. Krátce se také zmíníme o rozhraní RPC (Remote Procedure Call), na kterém jsou založeny služby NFS (Network File System) a NIS (Network Information System). Bohužel konfigurace systémů NFS a NIS vyžaduje trochu více prostoru, proto budou tyto systémy probrány v samostatných kapitolách. To platí i pro elektronickou poštu a pro síťové news.

Samozřejmě v této knize nemůžeme probrat všechny síťové aplikace. Budete-li si chtít nainstalovat nějakou z aplikací, která zde není uvedena, jako například programy `talk`, `gopher` nebo `Xmosaic`, nahlédněte prosím na stránky jejich manuálů, kde najdete více informací.

9.1 Super-server inetd

Služby jsou často poskytovány pomocí tzv. *démonů*. Démon je program, který otevře určitý port a čeká na příchozí spojení. Pokud k takovému spojení dojde, vytvoří proces potomka, jenž toto spojení přijme, zatímco rodičovský proces bude stále pokračovat v odposlouchávání dalších požadavků. Tento koncept má nevýhodu v tom, že každá poskytovaná služba musí mít spuštěného démona, který odposlouchává port z důvodu případného výskytu spojení, což obecně znamená, že se plýtvá systémovými zdroji, například odkládacím prostorem.

Proto většina unixových instalací spouští tzv. „super-server“, který vytvoří sockety pro většinu služeb, které současně odposlouchává pomocí systémového volání *select(2)*. Když vzdálený hostitel požádá o některou z nabízených služeb, super-server to zaznamená a pro daný port vytvoří speciální server.

Obecně používaný super-server se nazývá `inetd`, internetový démon (Internet Daemon). Je spouštěn při procesu zavádění systému a seznam služeb, které má spravovat, získává ze spouštěcího souboru s názvem `/etc/inetd.conf`. Kromě výše zmíněných volaných serverů existuje i spousta triviálních služeb, které provádí démon `inetd` sám. Tyto služby se nazývají *vnitřní služby*. Jednou z nich jsou služba `chargen`, která generuje řetězec znaků a služba `daytime`, která vrací systémový čas.

V tomto souboru se položka skládá z jednotlivých řádek, které jsou tvořeny následujícími poli:

```
service type protocol wait user server cmdline
```

Jednotlivá pole mají následující význam:

`service` Určuje název služby. Název služby musí být převeden na číslo portu pomocí vyhledání názvu služby v souboru `/etc/services`. Tento soubor bude popsán ve stati Soubory `services` a `protocols`, která následuje níže.

`type` Určuje typ socketu. Nabývá buď hodnoty `stream` (pro protokoly využívající vlastností spojení), nebo hodnoty `dgram` (pro protokoly založené na datagramech). Proto by měly služby založené na protokolu TCP používat vždy hodnotu `stream`, zatímco služby založené na protokolu UDP by měly vždy používat hodnotu `dgram`.

`protocol` Určuje název přenosového protokolu, který bude daná služba používat. Musí to být korektní název protokolu, který se nachází v souboru `protocols`. Bude vysvětleno dále.

`wait` Tato volba se vztahuje pouze na typ socketu `dgram`. Nabývá hodnoty `wait` nebo `nowait`. Je-li zadána volba `wait`, spustí démon `inetd` pro daný port vždy pouze jeden server. V opačném případě bude po spuštění serveru okamžitě pokračovat v odposlouchávání portu.

To je užitečné u „jednocestných“ (single-threaded) serverů, které budou číst všechny přicházející datagramy tak dlouho, dokud nepřestanou přicházet a potom se ukončí. Většina serverů RPC vyhovuje tomuto typu a tudíž by u nich měla být uvedena volba `wait`. Druhý typ serverů, tzv. „vícecestné servery“ (multi-threaded), umožňují současně spouštět neomezený počet instancí; to je používáno jen velmi zřídka. U těchto serverů by měla být volba `nowait`.

Sockety typu `stream` by měly vždy používat volbu `nowait`.

`user` Tato volba určuje přihlašovací uživatelské id, pod kterým bude daný proces spouštěn. Tímto uživatelem je často uživatel **root**, avšak některé služby mohou používat i jiné účty. Zde je velmi vhodné uplatňovat princip nejnižších práv, což znamená, že byste neměli příkaz spouštět na účtu s vyššími právy, pokud je spouštěný program pro svou správnou funkci nevyžaduje. Například server news NNTP běží s právy **news**, zatímco služby, které by mohly představovat bezpečnostní rizika (jako je služba `tftp` nebo `finger`) jsou často spouštěny s právy **nobody**.

`server` Určuje název plné cesty ke spouštěnému programu serveru. Vnitřní služby jsou označeny klíčovým slovem `internal`.

`cmdline` Tato volba určuje příkazovou řádku, která bude předána danému serveru. Tato volba obsahuje argument 0, který odpovídá názvu příkazu. Pokud se program nechová jinak při vyvolání pod jiným názvem, je tímto názvem příkazu obvykle vlastní název programu.

Toto pole je u interních služeb prázdné.

```
#
# inetd služby
ftp        stream tcp nowait root    /usr/sbin/ftpd     in.ftpd -l
telnet     stream tcp nowait root    /usr/sbin/telnetd in.telnetd -
       b/etc/issue
#finger     stream tcp nowait bin     /usr/sbin/fingerd in.fingerd
#tftp       dgram  udp wait    nobody /usr/sbin/tftpd    in.tftpd
#tftp       dgram  udp wait    nobody /usr/sbin/tftpd    in.tftpd /bo-
ot/diskless
login      stream tcp nowait root    /usr/sbin/rlogind in.rlogind
shell      stream tcp nowait root    /usr/sbin/rshd     in.rshd
exec       stream tcp nowait root    /usr/sbin/rexecd   in.rexecd
#
#        vnitřní služby inetd
#
daytime    stream tcp nowait root internal
daytime    dgram  udp nowait root internal
time       stream tcp nowait root internal
time       dgram  udp nowait root internal
echo       stream tcp nowait root internal
echo       dgram  udp nowait root internal
```

```
discard    stream tcp nowait root internal
discard    dgram  udp nowait root internal
chargen    stream tcp nowait root internal
chargen    dgram  udp nowait root internal
```

Obrázek 9.1
Vzorový soubor `/etc/inetd.conf`

Na obrázku 9.1 vidíte vzorový soubor `inetd.conf`. Služba `finger` je uvedená jako komentář, což znamená, že není dostupná. To je často z důvodů bezpečnostních, protože může být útočníky zneužita k získání jmen uživatelů ve vašem systému.

Také služba `tftp` je okomentována. Služba `tftp` implementuje tzv. *primitivní protokol pro přenos souborů* (*Primitive File Transfer Protocol*), který umožňuje z vašeho systému přenést libovolný okolnímu světu dostupný soubor, aniž by byla prováděna jakákoliv kontrola pomocí hesla apod. To je nepříjemné zejména u souboru `/etc/passwd` a ještě horší je to v případě, kdy nepoužíváte stínová hesla.

Protokol TFTP obecně používají bezdiskoví klienti a X-terminály ke stahování startovacího kódu ze zaváděcích serverů. Pokud potřebujete spouštět službu `tftpd` z tohoto důvodu, ujistěte se, že jste omezili její rozsah pouze na ty adresáře, ze kterých budou klienti získávat soubory. To provedete tak, že příkazové řádce služby `tftpd` přidáte názvy těchto adresářů. Výše uvedené demonstruje druhý řádek výpisu.

9.2 Prostředky na řízení přístupu

Protože přístup k počítačům prostřednictvím sítě přináší mnoho bezpečnostních rizik, byly navrženy aplikace, které chrání před několika typy útoků. Některé z těchto aplikací mohou obsahovat chyby (nejdrastičtěji to demonstruje červ RTM Internet) nebo nemusí umět rozlišovat mezi bezpečnými hostiteli, od kterých budou přijímány požadavky na konkrétní službu, a nebezpečnými hostiteli, jejichž požadavky by měly zamítnout. Již jsme se krátce zmínili o službách `finger` a `tftp`. Tyto služby budete asi chtít povolit pouze „důvěryhodným hostitelům“, což ale nelze pomocí standardního nastavení, při kterém super-server `inetd` poskytuje tuto službu buď všem klientům, anebo žádnému klientovi.

Pro tento účel je vhodný nástroj `tcpd`,[1] tzv. zástupce démonů. U služeb protokolu TCP, které chcete monitorovat nebo chránit, je tento démon volán místo programu serveru. Nástroj `tcpd` zapisuje požadavky do démona `syslog`, kontroluje, zda je vzdálený hostitel oprávněn

[1] Napsal ho Wietse Venema, jehož e-mailová adresa je **wietse@wzv.win.tue.nl**

používat danou službu a pouze v tom případě spustí skutečný program serveru. Všimněte si, že tento postup nefunguje u služeb založených na protokolu UDP.

Chcete-li například zastoupit démona `finger`, musíte v souboru `inetd.conf` změnit odpovídající řádku následujícím způsobem:

```
# zastoupení démona finger
finger   stream   tcp      nowait   root     /usr/sbin/tcpd    in.fingerd
```

Pokud nepřidáte žádné řízení přístupu, bude se tato úprava klientovi jevit stejně, jako obvyklé nastavení služby `finger`, s tou výjimkou, že veškeré požadavky budou zapisovány s prioritou `auth` do souboru `syslog`.

Řízení přístupu je implementováno za pomoci dvou souborů, které se jmenují `/etc/hosts.allow` a `/etc/hosts.deny`. Tyto soubory obsahují položky, které některým hostitelům povolují, resp. zakazují přístup k určitým službám. Když nástroj `tcpd` vyřizuje od klienta s názvem hostitele **biff.foobar.com** požadavek na použití určité služby, jako je služba `finger`, vyhledá v souborech `hosts.allow` a `hosts.deny` (v uvedeném pořadí) položky odpovídající jak požadované službě, tak i hostiteli klienta. Pokud je odpovídající položka nalezena v souboru `hosts.allow`, bude přístup povolen bez ohledu na položky v souboru `hosts.deny`. Pokud ale bude odpovídající položka nalezena v souboru `hosts.deny`, bude požadavek odmítnut a spojení se ukončí. Jestliže se ani v jednom souboru odpovídající položka nenajde, bude požadavek přijat.

Položky v souboru s přístupy vypadají následovně:

```
servicelist: hostlist [:shellcmd]
```

Pole `servicelist` obsahuje seznam názvů služeb ze souboru `/etc/services` nebo klíčové slovo *ALL*. Chcete-li zadat všechny služby kromě služeb `finger` a `tftp`, použijte řetězec „*ALL EXCEPT finger, tftp*“.

Pole `hostlist` obsahuje seznam názvů hostitelů nebo IP-adres, případně klíčová slova *ALL*, *LOCAL* nebo *UNKNOWN*. Klíčovému slovu *ALL* vyhovují všichni hostitelé, zatímco klíčovému slovu *LOCAL* vyhovují pouze názvy hostitelů, kteří neobsahují tečku.[2] Klíčovému slovu *UNKNOWN* odpovídají všichni hostitelé, u nichž selhalo vyhledávání jejich názvu nebo adresy. Název začínající tečkou vyhovuje všem hostitelům, jejichž doména je shodná s poskytnutým názvem. Například název domény **.foobar.com** vyhovuje hostitelům na adrese **biff.foobar.com**. Podobná opatření existují i pro síťové IP-adresy a pro čísla podsítí. Chcete-li více detailů, nahlédněte prosím do manuálové stránky *hosts_access(5)*.

2 Tečku obvykle neobsahují jména hostitelů ze souboru `/etc/hosts`.

Chcete-li zakázat přístup ke službám `finger` a `tftp` všem hostitelům s výjimkou místních hostitelů, vložte do souboru `/etc/hosts.deny` následující řádku a soubor `/etc/hosts.allow` ponechte prázdný:

```
in.tftpd, in.fingerd:  ALL EXCEPT LOCAL, .your.domain
```

Volitelné pole `shellcmd` může obsahovat příkaz rozhraní, jenž bude vykonán při splnění dané položky. To je užitečné při nastavování pastiček, které mohou odhalit potenciální útočníky:

```
in.ftpd: ALL EXCEPT LOCAL, .vbrew.com : \
       echo "request from %d@%h" >> /var/log/finger.log; \
       if [ %h != "vlager.vbrew.com" ]; then \
            finger -l @%h >> /var/log/finger.log \
fi
```

Nástroj `tcpd` nahradí argumenty *%h* a *%d* skutečným názvem hostitele klienta, resp. skutečným názvem služby. Chcete-li více detailů, nahlédněte prosím do manuálových stránek *hosts_access(5)*.

9.3 Soubory services a protocols

Čísla portů, na kterých jsou nabízeny jisté „standardní“ služby, jsou definovány v RFC „Assigned Numbers“. Aby mohly servery nebo klienti převádět názvy služeb na tato čísla, musí být alespoň část z tohoto seznamu uložena na každém hostiteli; tato část je uložena v souboru s názvem `/etc/services`. V tomto souboru má každá položka následující syntaxi:

```
service port/protocol [aliases]
```

Zde pole `service` definuje název služby, pole `port` definuje port, na kterém je daná služba nabízena, a pole `protokol` definuje typ používaného transportního protokolu. Obecně toto pole nabývá buď hodnoty `udp`, nebo hodnoty `tcp`. Službu je možné nabízet i pro více protokolů, stejně tak lze nabízet různé služby na stejném portu, pokud používají odlišné protokoly. Pole `aliases` umožňuje zadat alternativní názvy stejné služby.

Soubor `services`, který je dodáván společně se síťovým softwarem, nebudete muset obvykle měnit. Přesto zde uvádíme malý výpis z tohoto souboru:

```
# Soubor services:
#
# známé služby
echo          7/tcp                  # Echo
echo          7/udp                  #
discard       9/tcp  sink null       # Discard
discard       9/udp  sink null       #
daytime       13/tcp                 # Daytime
daytime       13/udp                 #
chargen       19/tcp  ttytst source  # Character Generator
chargen       19/udp  ttytst source  #
ftp-data      20/tcp                 # File Transfer Protocol (Data)
ftp           21/tcp                 # File Transfer Protocol (Contr
telnet        23/tcp                 # Virtual Terminal Protocol
smtp          25/tcp                 # Simple Mail Transfer Protocol
nntp         119/tcp  readnews       # Network News Transfer Protoco
#
# unixové služby
exec         512/tcp                 # BSD rexecd
biff         512/udp  comsat         # nová pošta
login        513/tcp                 # vzdálené přihlášení
who          513/udp  whod           # vzdálené who a update
shell        514/tcp  cmd            # vzdálené příkazy
syslog       514/udp                 # vzdálené protokolování
printer      515/tcp  spooler        # vzdálený tisk
route        520/udp  router routed  # směrovací protokol
```

Všimněte si, že například služba `echo` je poskytována na portu 7 jak pro protokol TCP, tak i pro protokol UDP a port 512 používají dvě odlišné služby, konkrétně démon COMSAT (který upozorňuje uživatele na nově došlou poštu, viz *xbiff(1x)*), jenž používá protokol UDP, a služba vzdáleného spouštění (*rexec(1)*), která používá protokol TCP.

Podobně jako soubor se službami potřebuje i síťová knihovna nějaký způsob, jakým by mohla přeložit názvy protokolů – například těch protokolů, které jsou použity v souboru `services` – na čísla protokolů, kterým by rozuměly IP-vrstvy ostatních hostitelů. To se provede vyhledáním daného názvu v souboru `/etc/protocols`. Ten obsahuje na každé řádce jednu položku. Každá položka obsahuje název protokolu a číslo sdružené s daným protokolem. Provádění změn v tomto souboru je ještě méně pravděpodobné, než zasahování do souboru `/etc/services`. Nyní následuje vzorový soubor `protocols`:

```
#
# Internetové (IP) protokoly
#
ip      0       IP      # internet protocol
icmp    1       ICMP    # internet control message protocol
igmp    2       IGMP    # internet group multicast protocol
tcp     6       TCP     # transmission control protocol
udp     17      UDP     # user datagram protocol
raw     255     RAW     # RAW IP interface
```

9.4 Vzdálené volání procedur – RPC

Balík RPC neboli *Vzdálené volání procedur* (*Remote Procedure Call*) poskytuje velmi obecný mechanismus pro aplikace typu klient-server. Balík RPC vyvinula firma Sun Microsystems a v podstatě se jedná o sbírku nástrojů a knihoven funkcí. Důležitou aplikací postavenou na balíku RPC je systém NFS, síťový souborový systém, a systém NIS, síťový informační systém. Oba tyto systémy budou probírány v následujících kapitolách.

Server RPC se skládá ze sady procedur, které si může klient volat tím způsobem, že pošle serveru požadavek RPC společně s parametry procedury. Server spustí jménem klienta požadovanou proceduru a existuje-li návratová hodnota, pošle ji zpět klientovi. Aby mohl být balík RPC nezávislý na typu hardwaru, jsou všechna data předávaná mezi klientem a serverem převedena vysílajícím počítačem do tzv. formátu *externího vyjádření dat* (*External Data Representation* – XDR) a přijímající počítač je převede zpět do místního vyjádření dat.

Někdy způsobí zdokonalení RPC aplikace nekompatibilní změny v rozhraní volání procedur. Samozřejmě, že prostá výměna serveru může způsobit spadnutí všech aplikací, které stále očekávají původní chování. Proto mají programy RPC přiděleny číslo své verze, obvykle se začíná hodnotou 1 a při každé nové verzi rozhraní RPC je tato hodnota zvýšena. Server může často současně nabízet několik verzí RPC; v takovém případě označí klient ve svém požadavku číslo verze, kterou chce ve své implementaci dané služby používat.

Vlastní síťová komunikace probíhající mezi servery RPC a jejich klienty je zvláštní. Server RPC nabízí jednu nebo více skupin procedur; každá množina procedur se nazývá *program* a je jednoznačně určena tzv. *číslem programu*. Seznam definující přiřazení názvů služeb k číslům programů je obvykle uložen v souboru `/etc/rpc`. Na obrázku 9.2 vidíte výpis z tohoto souboru.

```
#
# /etc/rpc - různé služby založené na RPC
#
portmapper          100000  portmap sunrpc
rstatd              100001  rstat rstat_svc rup perfmeter
rusersd             100002  rusers
nfs                 100003  nfsprog
ypserv              100004  ypprog
mountd              100005  mount showmount
ypbind              100007
walld               100008  rwall shutdown
ypasswss            100009  ypasswd
bootparam           100026
ypupdated           100028  ypupdate
```

Obrázek 9.2
Vzorový soubor `/etc/rpc`

U sítí na bázi protokolu TCP/IP čelili autoři balíku RPC problému, jak sdružit čísla programů s obecnými síťovými službami. Zvolili takovou variantu, kdy každý server poskytuje pro každý program a pro každou verzi programu jak port pro protokol TCP, tak i port pro protokol UDP. Obecně budou aplikace při posílání dat používat protokol UDP a protokol TCP budou používat pouze v případě, že se posílaná data nevejdou do jediného datagramu protokolu UDP.

Samozřejmě, že klienti musí mít možnost zjistit port, na který je namapováno dané číslo programu. V tomto případě by bylo použití konfiguračního souboru příliš nepružné; jelikož aplikace RPC nepoužívají vyhrazené porty, nemůžeme mít žádnou jistotu, že port původně určený pro použití naší databázovou aplikací nebude obsazen nějakým jiným procesem. Proto aplikace RPC vyberou některý z dostupných portů a zaregistrují ho pomocí démona, tzv. *mapovače portů* (*portmapper deamon*). Tento démon se chová jako zprostředkovatel mezi všemi servery RPC, které jsou spuštěny na daném počítači: klient, který chce kontaktovat službu s daným číslem programu, se nejdříve bude dotazovat mapovače portů umístěné na hostiteli serveru, ten mu pak vrátí čísla portů TCP a UDP, na kterých může být daná služba dosažitelná.

Tato metoda má konkrétní nevýhodu v tom, že zavádí princip jediného selhání, podobně jako to činí démon `inetd` u standardních služeb. Avšak tento případ je ještě horší, protože když selže démon mapující porty, ztratí se veškeré informace o portech RPC; to obvykle způsobí, že budete muset manuálně znovu nastartovat všechny servery RPC nebo budete muset znovu nastartovat celý počítač.

V Linuxu se program na mapování portů nazývá `rpc.portmap` a je umístěn v adresáři `/usr/sbin`. Kromě toho, že je třeba se ujistit, zda je démon mapující porty skutečně spouštěn ze souboru `rc.inet2`, není třeba žádná další konfigurace.

9.5 Konfigurace příkazů r

Ke spouštění příkazů na vzdálených hostitelích existuje spousta možností. Jsou to například příkazy `rlogin`, `rsh`, `rcp` a `rcmd`. Všechny tyto příkazy vyvolají na vzdáleném hostiteli příkazový interpret a umožní uživateli spouštět programy. Samozřejmě, že klient musí mít na hostiteli, na kterém hodlá spouštět příkazy, založený účet. Tedy všechny tyto příkazy provádějí ověření totožnosti. Klient obvykle sdělí serveru své přihlašovací uživatelské jméno a server okamžitě požádá o zadání hesla, které je ověřeno běžným způsobem.

Avšak u konkrétních uživatelů je někdy vhodné nevyžadovat ověření totožnosti. Když se například často přihlašujete k ostatním počítačům v lokální síti LAN, měli byste být do ní vpuštěni, aniž byste museli pokaždé zadávat své heslo.

Vypnutí ověřování totožnosti je vhodné pouze u malého počtu hostitelů, jejichž databáze hesel jsou synchronizovány, nebo u malého počtu privilegovaných uživatelů, kteří musí z administrativních důvodů přistupovat k mnoha počítačům. Kdykoliv chcete lidem povolit přihlášení k vašemu hostiteli bez zadání přihlašovacího id nebo hesla, ujistěte se, že náhodně nepovolíte přístup někomu jinému.

Vypnutí ověřování totožnosti u příkazů z rodiny *r* lze provést dvěma způsoby. První z nich je určen pro superuživatele, a umožňuje povolit přihlašování několika nebo všech uživatelů na několik nebo všechny hostitele (později uvedené případy jsou velmi nešťastné), aniž by byli dotazováni na heslo. Tento přístup řídí soubor, který se jmenuje `/etc/hosts.equiv`. Soubor obsahuje seznam hostitelů a jmen uživatelů, kteří jsou považováni za rovnocenné s uživateli místního hostitele. Druhý způsob je pro uživatele, kteří povolí dalším uživatelům z určitých hostitelů přístup ke svému účtu. Tito uživatelé mohou být vypsáni v souboru `.rhosts`, který se nachází v domovském adresáři daného uživatele. Z bezpečnostních důvodů musí být tento soubor vlastněn uživatelem nebo superuživatelem a tento soubor nesmí být symbolickým odkazem, v opačném případě bude ignorován.[3]

Když klient požádá o službu z rodiny *r*, vyhledá se nejprve jeho hostitel a jeho uživatelské jméno v souboru `/etc/hosts.equiv` a potom v souboru `.rhosts` uživatele, pod jehož účtem se chce daný uživatel přihlásit. Jako příklad předpokládejme, že uživatel **janet** pracuje

[3] V prostředí systému NFS ho můžete chránit pomocí hodnoty 444, protože superuživatel je často velice omezován v přístupu k souborům na discích, které jsou připojeny pomocí systému NFS.

na hostiteli **gauss** a pokouší se přihlásit k účtu uživatele **joe** na hostiteli **euler**. V následujícím výkladu budeme považovat uživatele Janet za uživatele *klienta* a uživatele Joe za *místního* uživatele. Nyní napíše uživatel Janet na hostiteli **gauss** příkaz:

```
$ rlogin -l joe euler
```

Server nejprve zkontroluje v souboru *hosts.equiv*[4], zda má být uživateli Janet povolen volný přístup, a pokud neuspěje, pokusí se tohoto uživatele vyhledat v souboru `.rhosts`, který se nachází v domovském adresáři uživatele **joe**.

Soubor `hosts.equiv` na hostiteli **euler** vypadá asi takto:

```
gauss
euler
-public
quark.physics.groucho.edu      andres
```

Každý záznam se skládá z názvu hostitele, za nímž může volitelně následovat jméno uživatele. Jestliže se na řádce objeví pouze samotný název hostitele, bude všem uživatelům daného hostitele umožněn přístup ke svým místním účtům bez jakékoliv kontroly. Ve výše uvedeném příkladu bude uživateli Janet povoleno přihlášení ke svému účtu s názvem **janet**, pokud se bude přihlašovat z hostitele **gauss,** a totéž bude platit i pro kteréhokoliv dalšího uživatele s výjimkou uživatele **root**. Pokud se ale bude chtít uživatel Janet přihlásit jako uživatel **joe**, bude obvyklým způsobem požádán o zadání hesla.

Je-li za názvem hostitele uvedeno jméno uživatele, jako je tomu na poslední řádce výše uvedeného příkladu, bude tomuto uživateli umožněn přístup ke *všem* účtům kromě účtu **root** bez hesla.

Před názvem hostitele může být uveden symbol minus, viz položka „**-public**“. V tomto případě požadujeme, aby hostitel **public** ověřoval totožnost všech účtů, bez ohledu na to, jaká práva mají uživatelé přidělena ve svých souborech `.rhosts`.

Formát souboru `.rhosts` je identický s formátem souboru `hosts.equiv`, ale jeho význam je poněkud odlišný. Podívejte se na soubor `.rhosts` uživatele Joe na hostiteli **euler**:

```
chomp.cs.groucho.edu
gauss       janet
```

[4] Poznamenejte si, že pokud se někdo pokusí přihlásit jako uživatel **root**, není prohledáván soubor `hosts.equiv`

První položka povolí uživateli **joe** volný přístup v případě, že se přihlašuje z hostitele **chomp.cs.groucho.edu**. Tato položka neovlivní práva žádného dalšího účtu na hostiteli **euler** nebo **chomp**. Druhá položka je variací výše uvedeného, která spočívá v tom, že uživateli **janet** bude povolen volný přístup k účtu uživatele Joe, pokud se přihlásí z hostitele **gauss**.

Pamatujte, že název hostitele klienta je získán pomocí reverzního mapování adresy volajícího hostitele na název hostitele, takže tento postup nebude fungovat u hostitelů, které resolver nezná. Název hostitele klienta odpovídá názvu uvedenému v souborech *hosts*, pokud je splněn jeden z následujících případů:

- Kanonický název hostitele klienta (nikoliv jeho přezdívka) přesně odpovídá názvu hostitele, který je uveden v tomto souboru.
- Je-li název hostitele klienta plně kvalifikovaným doménovým jménem (jako například jménem, které vrátí resolver při spuštěném systému DNS) a neodpovídá-li přesně názvu hostitele, který je uveden v souboru *hosts*, bude porovnán s názvem hostitele rozšířeným o název místní domény.

Kapitola

10

Síťový informační systém (NIS)

Provozujete-li lokální síť, budete chtít poskytovat uživatelům takové prostředí, které by jim připadalo transparentní. Jedním z důležitých kroků, které vedou k tomuto cíli, je zajištění synchronizace důležitých dat, jako jsou informace o účtech uživatelů, mezi všemi hostiteli. Ukázali jsme si, že pro rozlišení názvů hostitelů existuje mocná a propracovaná služba, konkrétně systém DNS. Pro ostatní úkoly žádná taková speciální služba neexistuje. Kromě toho, pokud spravujete pouze malou lokální síť, která nemá žádné spojení s Internetem, nebude se mnoho administrátorů namáhat s nastavováním systému DNS.

Tato situace vedla k tomu, že firma Sun vyvinula systém NIS, *síťový informační systém*. Systém NIS poskytuje prostředky pro obecný přístup k databázím, které lze využít k šíření informací všem hostitelům ve vaší síti. Těmito informacemi mohou být například informace obsažené v souborech `passwd` a `group`. To umožňuje, aby síť vypadala jako jediný systém se stejnými účty na všech hostitelích. Podobným způsobem můžete používat systém NIS k šíření informací o hostitelích, které jsou uloženy v souboru `/etc/hosts`, na všechny počítače v síti.

Systém NIS je založen na balíku RPC a skládá se ze serveru, z knihovny na straně klienta a z několika administrativních nástrojů. Původně se systém NIS nazýval *Yellow pages* (*Žluté stránky*), neboli YP. Toto označení se ještě stále používá jako informativní odkaz na tuto službu. Na druhou stranu jsou Yellow pages ochrannou známkou společnosti British Telecom, která požadovala po firmě Sun, aby tento název přestala používat. I s odstupem času však zůstaly některé názvy zakořeněny a zkratka YP se proto stále používá jako prefix názvů většiny příkazů, které se vztahují k sytému NIS, například `ypserv`, `ypbind` apod.

Dnes je systém NIS dostupný snad ve všech verzích Unixu a existují dokonce i jeho volné implementace. Jednu z těchto implementací obsahuje balík BSD verze Net-2. Je odvozena z veřejně dostupné implementace, kterou poskytla firma Sun. Již dlouhou dobu je kód knihovny klienta z této verze umístěn v knihovně GNU *libc*, zatímco administrativní nástroje teprve ne-

dávno portoval na platformu Linuxu pan Swen Thümmler.[1] Server systému NIS nemá žádnou referenční implementaci. Tobias Reber napsal další balík systému NIS, který obsahuje všechny nástroje i server; tento balík se nazývá `yps`.[2]

V současné době dokončil Peter Eriksson[3] kompletní přepis kódu systému NIS pod názvem NYS, který podporuje jak holý systém NIS, tak i mnohem dokonalejší systém NIS+ od firmy Sun. Balík NYS neposkytuje pouze množinu nástrojů a server, ale přidává do knihovny také celou novou sadu funkcí, které se časem zřejmě stanou součástí standardní knihovny *libc*. Tyto funkce obsahují nové konfigurační schéma pro rozlišování názvů hostitelů, které nahrazuje současné schéma používající soubor `host.conf`. Přínosy těchto funkcí budou rozebrány níže.

Tato kapitola se bude více soustřeďovat na balík NYS, než na další dva konkurenční balíky, které budeme označovat jako tzv. „tradiční“ kód systému NIS. Chcete-li pracovat s některým z těchto balíků, pak vám možná nebudou informace obsažené v této kapitole stačit. Chcete-li o nich získat více informací, sežeňte si prosím knihu o systému NIS, například publikaci *NFS and NIS* od Hala Sterna (viz [Stern92]).

Balík NYS se v současné době stále ještě vyvíjí, a proto standardní linuxové utility, jako jsou síťové programy nebo program `login`, zatím neovládají konfigurační schéma balíku NYS. Než se balík NYS stane součástí hlavní knihovny *libc*, budete si muset všechny tyto binární soubory aplikací sami překompilovat, aby byly schopny podporovat balík NYS. Při kompilaci libovolné z těchto aplikací pomocí programu *Makefile* zadejte sestavovacímu programu jako poslední parametr před knihovnou *libc* parametr `-lnsl`. Tento parametr vloží na místo standardních funkcí z knihovny C odpovídající funkce z knihovny *libnsl*, knihovny NYS.

10.1 Seznámení se systémem NIS

Systém NIS uchovává databázové informace v tzv. *mapách*, které obsahují páry klíčových hodnot. Mapy jsou uloženy na centrálním hostiteli, na němž běží systém NIS a z něhož mohou klienti získávat informace pomocí různých volání procedur RPC. Poměrně často bývají mapy uloženy v souborech DBM.[4]

---

1 Tento pán je k zastižení na adrese **swen@uni-paderborn.de**. Klienti systému NIS jsou dostupní v souboru s názvem `yp-linux.tar.gz` na adrese **sunsite.unc.edu** v adresáři `system/Network`.

2 Aktuální verze (v době psaní této knihy) byla `yps-0.21` a získáte ji na adrese **ftp.lysator.liu.se** v adresáři `/pub/NYS`.

3 Je k zastižení na adrese **pen@lysator.liu.se**.

4 DBM je jednoduchá knihovna pro správu databáze, která ke zrychlení vyhledávacích operací využívá transformačních technik. Z projektu GNU je k dispozici volně šiřitelná implementace DBM s názvem `gdbm`, která je součástí většiny distribucí Linuxu.

Vlastní mapy jsou obvykle generovány z hlavních textových souborů, jako jsou například soubory `/etc/hosts` nebo `/etc/passwd`. Pro některé soubory se vytvoří několik map, pro každý typ vyhledávacího klíče se vždy vytvoří jedna mapa. Například v souboru `hosts` můžete hledat jak název hostitele, tak i IP-adresu. Z tohoto souboru budou takto odvozeny dvě mapy, které se budou nazývat `hosts.byname`, resp. `hosts.byaddr`. Tabulka 10.1 uvádí běžně se vyskytující mapy společně se soubory, ze kterých byly vygenerovány.

Hlavní soubor	Mapa (Mapy)	
`/etc/hosts`	`hosts.byname`	`hosts.byaddr`
`/etc/networks`	`networks.byname`	`networks.byaddr`
`/etc/passwd`	`passwd.byname`	`passwd.byuid`
`/etc/group`	`group.byname`	`group.bygid`
`/etc/services`	`services.byname`	`services.bynumber`
`/etc/rpc`	`rpc.byname`	`rpc.bynumber`
`/etc/protocols`	`protocols.byname`	`protocols.bynumber`
`/usr/lib/aliases`	`mail.aliases`	

Tabulka 10.1
Některé standardní mapy systému NIS a jim odpovídající soubory

Možná se setkáte s tím, že některé balíky systému NIS nebo některé další programy podporují i jiné soubory a mapy. Tyto soubory a mapy mohou obsahovat informace o aplikacích, které nejsou v této knize probírány (například mapa `bootparams`) a mohou využívat některé servery BOOTP, anebo tyto soubory a mapy nemají v současné době v Linuxu žádnou funkci (příkladem jsou mapy `ethers.byname` nebo `ethers.byaddr`).

Pro některé mapy lidé obecně používají *přezdívky*, které jsou kratší a z tohoto důvodu se lépe píší. Chcete-li získat kompletní seznam přezdívek, kterým vaše nástroje systému NIS rozumí, spusťte následující příkaz:

```
$ ypcat -x
NIS map nickname translation table:
       "passwd" -> "passwd.byname"
       "group" -> "group.byname"
       "networks" -> "networks.byname"
       "hosts" -> "hosts.byname"
       "protocols" -> "protocols.byname"
       "services" -> "services.byname"
```

```
"aliases" -> "aliases.byname"
"ethers" -> "ethers.byname"
"rpc" -> "rpc.byname"
"netmasks" -> "netmasks.byname"
"publickey" -> "publickey.byname"
"netid" -> "netid.byname"
"passwd.adjunct" -> "passwd.adjunct.byname"
"group.adjunct" -> "group.adjunct.byname"
"timezone" -> "timezone.byname"
```

Server NIS se tradičně nazývá `ypserv`. Pro protřeby průměrné sítě zpravidla postačuje jediný server; rozsáhlé sítě možná dají přednost provozování několika těchto serverů na různých počítačích, čímž různé segmenty sítě odlehčí celkové zatížení serverových počítačů a směrovačů. Tyto servery jsou vzájemně synchronizovány tak, že některý z těchto serverů je nastaven jako *řídící server* a ostatní servery jsou nastaveny jako *řízené servery*. Mapy budou vytvořeny pouze na hostiteli s řídícím serverem. Odtud jsou distribuovány na všechny řízené servery.

Možná jste si všimli, že po celou dobu jsme o „sítích“ hovořili značně neurčitě; samozřejmě, že v systému NIS existuje přesný koncept týkající se dané sítě, kterou tvoří skupina všech hostitelů sdílejících pomocí systému NIS část jejích systémových konfiguračních dat: tímto konceptem je *doména* systému NIS. Bohužel domény systému NIS nemají nic společného s doménami systému DNS, s nimiž jsme se již setkali. Abychom se v této kapitole vyhnuli dvojsmyslnostem, bude vždy uváděn typ příslušné domény.

Domény systému NIS nabízí jen velmi slabé administrativní funkce. Ty jsou, kromě sdílení hesel mezi všemi počítači příslušné domény, většinou pro uživatele neviditelné. Proto má název poskytnutý doméně systému NIS význam pouze pro správce sítě. Zpravidla bude fungovat jakýkoliv název, který bude odlišný od všech názvů domén systému NIS vyskytujících se ve vaší lokální síti. Například správci ve společnosti Virtual Brewery mohou vytvořit dvě domény systému NIS, jednu z nich pro vlastní společnost Virtual Brewery a jednu pro společnost Virtual Winery, kterým přiřadí názvy **brewery**, resp. **winery**. Dalším poměrně běžným schématem je použití stejného názvu jak pro doménu systému DNS, tak i pro doménu systému NIS. K nastavení a zobrazení názvu domény systému NIS na vašem hostiteli můžete použít příkaz `domainname`. Když tento příkaz vyvoláte bez argumentů, zobrazí název aktuální domény systému NIS; budete-li chtít nastavit název domény, přihlašte se jako superuživatel a napište:

```
# domainname brewery
```

Domény NIS určují, na který server se budou aplikace dotazovat. Například program *login* se na hostiteli ve Virtual Winery může dotazovat na informace o uživatelských heslech pouze serveru NIS v této společnosti (nebo jednoho ze serverů v této společnosti, pokud je jich více); zatímco aplikace na hostiteli ve Virtual Brewaery může komunikovat pouze se serverem v této společnosti.

Teď ještě zbývá vyřešit jednu záhadu, konkrétně jak klient zjistí, se kterým serverem se má spojit. Nejjednodušší by bylo použít konfigurační soubor, v němž bude uveden hostitel, na kterém se server má hledat. Avšak tento přístup je poměrně nepružný, protože neumožňuje klientům používat různé servery (samozřejmě z téže domény) v závislosti na jejich dostupnosti. Proto spoléhají tradiční implementace systému NIS na speciálního démona nazývaného `ypbind`, který detekuje vhodný server NIS v jejich doméně systému NIS. Dříve, než může nějaká aplikace předat systému NIS jakýkoliv dotaz, si musí nejprve od démona `ypbind` zjistit, jaký server má k tomuto účelu použít.

Démon `ypbind` prozkoumává servery za pomoci vysílání do místní sítě IP; první, kdo odpoví, je považován za potenciálně nejrychlejší server, a proto bude použit pro všechny následující dotazy systému NIS. Po uplynutí určité doby nebo dojde-li k přerušení spojení se serverem, začne démon `ypbind` znovu prozkoumávat aktivní servery.

A nyní se dostáváme ke spornému bodu použití dynamické vazby, který spočívá v tom, že ji budete potřebovat pouze zřídka a její použití s sebou navíc přináší jisté bezpečnostní problémy: Démon `ypbind` slepě uvěří každému, kdo mu odpoví, což může být jak prostý server NIS, tak i zlomyslný vetřelec. Netřeba ani říkat, že tato vlastnost je zvláště nepříjemná, spravujete-li pomocí systému NIS své databáze s hesly. Abyste tomuto předešli, nepoužívá balík NYS implicitně démona `ypbind`, ale název hostitele serveru přebírá z konfiguračního souboru.

10.2 Systém NIS versus systém NIS+

Systémy NIS a NIS+ toho mají společného více, než jen svůj název a cíl. Systém NIS+ je strukturován zcela odlišným způsobem. Místo přímého jmenného prostoru s oddělenými doménami systému NIS používá hierarchický prostor s názvy, který je podobný jmennému prostoru systému DNS. Místo map jsou používány tzv. *tabulky*, které jsou složeny z řádků a sloupců, kde každý řádek reprezentuje objekt databáze systému NIS+, zatímco sloupce obsahují vlastnosti objektů, které systém NIS+ zná a o něž se stará. Každá tabulka příslušné domény systému NIS+ v sobě zahrnuje i tabulky svých rodičovských domén. Mimoto může položka v tabulce obsahovat spojení na další tabulku. Tyto vlastnosti umožňují strukturovat informace mnoha způsoby.

Tradiční systém NIS má číslo verze balíku RPC rovno 2, zatímco systém NIS+ používá verzi 3.

Zatím to vypadá tak, že systém NIS+ není příliš používán a ani já toho vlastně o něm příliš nevím. (Ve skutečnosti o něm nevím skoro nic.) Proto ho zde nebudeme rozebírat. Pokud se o tento systém zajímáte hlouběji, nahlédněte prosím do administrativního manuálu systému NIS+ od firmy Sun ([NISPlus]).

10.3 Systém NIS na straně klienta

Ovládáte-li psaní a portování síťových aplikací, pak si jistě všimnete, že většina výše uvedených map systému NIS odpovídá funkcím, které jsou obsaženy v knihovně C. Chcete-li například získat informace ze souboru `passwd`, budete k tomuto účelu běžně používat funkce *getpwnam(3)* a *getpwuid(3)*, které vrací informace o účtu sdružené s příslušným jménem uživatele, resp. s číselným id uživatele. Za normálních okolností provedou tyto funkce požadované vyhledání ve standardním souboru, což je soubor `/etc/passwd`.

Avšak implementace těchto funkcí v systému NIS toto chování upraví a vloží takové volání procedury RPC, aby server NIS, vyhledal příslušné uživatelské jméno nebo jeho id. Toto chování bude pro aplikaci zcela transparentní. Funkce může buď k danému souboru „připojit“ mapu systému NIS nebo může touto mapou původní soubor „nahradit“. Neznamená to samozřejmě skutečnou úpravu daného souboru, pouze to znamená, že se aplikaci bude takový soubor jevit, jako by k němu byla daná mapa připojena nebo jakoby byl touto mapou nahrazen.

V tradičních implementacích systému NIS existovala jistá pravidla týkající se map, které mají nahrazovat původní informace a které se mají připojovat k původním informacím. Některé z těchto map, například mapy ze skupiny `passwd`, vyžadovaly úpravy souboru `passwd`, které mohly při nesprávném postupu dát vzniknout bezpečnostním dírám. Aby se tomu zabránilo, používá balík NYS obecné konfigurační schéma, které určuje, zda bude konkrétní množina funkcí klienta používat původní soubory systému NIS nebo systému NIS+ a v jakém pořadí. Toto konfigurační schéma bude náplní dalších statí této kapitoly.

10.4 Provozování serveru NIS

Po přehršli teoretických technických informacích přišel čas věnovat se skutečné konfigurační praci. V této stati si probereme konfiguraci serveru NIS. Pokud už ve vaší síti běží nějaký server NIS, nebudete zřejmě chtít nakonfigurovat další vlastní server; v tom případě můžete tuto stať přeskočit.

Při experimentování se serverem se nezapomeňte ujistit, že jste mu nepřiřadili název domény systému NIS, který je již ve vaší síti používán. To by totiž mohlo narušit všechny síťové služby a rozzlobit spoustu lidí.

V současné době jsou v Linuxu volně dostupné dva servery NIS, jeden je obsažen v balíku `yps` od Tobiase Rebera a druhý v balíku `ypserv` od Petera Erikssona. Nemělo by záležet na tom, který z těchto balíků si pro provozování zvolíte, ani na tom, zda budete používat balík NYS nebo standardní kód klienta NIS, který je v současnosti součástí knihovny *libc*. Při psaní této knihy se zdálo, že kód pro správu řízených serverů NIS je dokonalejší v balíku `yps`. Tedy pokud budete muset používat řízené servery, bude možná vhodnější použít balík `yps`.

Po nainstalování programu serveru (`ypserv`) do adresáře `/usr/sbin` byste měli vytvořit adresář, v němž budou uloženy soubory map, které bude váš server distribuovat. Bude-li nastavena doména systému NIS na název domény **brewery,** měly by být mapy umístěny v adresáři `/var/yp/brewery`. Server zjistí, zda spravuje konkrétní doménu systému NIS tak, že ověří přítomnost adresáře s mapami. Pokud zakážete službu některé domény NIS, ujistěte se, že jste odstranili také odpovídající adresář.

Mapy jsou zpravidla kvůli rychlejšímu vyhledávání uloženy v souborech DBM. Tyto soubory jsou vytvořeny z hlavních souborů za pomoci speciálního programu s názvem `makedbm` (pro server od Tobiase) nebo `dbmload` (pro server od Petera). Tyto programy se nesmí zaměňovat. Převod hlavního souboru do formy analyzovatelné programem `dbmload` obvykle vyžaduje jistý trik typu `awk` nebo `sed`, který je poměrně obtížné popsat a navíc je hůře zapamatovatelný. Proto obsahuje balík `ypserv` od Petera Erikssona program pro vytvoření tohoto souboru (nazvaný `ypMakefile`), který všechnu potřebnou práci udělá za vás. Měli byste ho nainstalovat jako soubor `Makefile` do svého adresáře s mapami a upravit ho tak, aby obsahoval vámi distribuované mapy. Směrem k začátku souboru naleznete položku *all*, ve které jsou uvedeny služby nabízené programem `ypserv`. Tento řádek vypadá zhruba následovně:

```
all: ethers hosts networks protocols rpc services passwd group netid
```

Pokud například nechcete vytvořit mapy `ethers.byname` a `ethers.byaddr`, odstraňte z výše uvedené položky volbu `ethers`. Chcete-li si své nastavení otestovat, budou vám stačit pouze jedna nebo dvě mapy, například mapy `services.*`.

Po úpravě souboru `Makefile` napište v adresáři s mapami příkaz `make`. Tento příkaz požadované mapy automaticky vygeneruje a nainstaluje. Je třeba zajistit, aby se při každé změně hlavních souborů aktualizovaly i soubory s mapami, v opačném případě by síť provedené změny nezjistila.

Další stať popisuje, jak nakonfigurovat kód klienta systému NIS. Pokud vaše nastavení nefunguje, pak je třeba zjistit, zda na váš server přichází nějaké požadavky. Pokud serveru z balíku NYS zadáte jako příznak příkazové řádky `-d`, budou se na konzole vypisovat všechny příchozí dotazy systému NIS včetně vrácených odpovědí. Takto možná zjistíte v čem je kámen úrazu. Server od Tobiase žádnou takovou volbou nedisponuje.

10.5 Nastavení klienta systému NIS pomocí balíku NYS

Až do konce této kapitoly se budeme zabývat konfigurací klienta systému NIS.

Nejprve byste měli sdělit balíku NYS, který server se bude pro službu NIS používat, což provedete za pomoci konfiguračního souboru `/etc/yp.conf`. Velice jednoduchý vzorový soubor pro hostitele v síti Virtual Winery by mohl vypadat asi takto:

```
# yp.conf - konfigurační soubor pro systém NIS
#
domainname winery
server vbardolino
```

První příkaz sdělí všem klientům systému NIS, že jsou přiřazeni k doméně systému NIS s názvem **winery**. Pokud na tento řádek zapomenete, použije balík NYS název domény, který jste vašemu systému přidělili za pomoci příkazu `domainname`. Příkaz `server` označuje server NIS, který se bude používat. Samozřejmě, že v souboru `hosts` musí být nastavena IP-adresa odpovídající názvu hostitele **vbardolino**; popřípadě můžete v příkazu `server` použít vlastní IP-adresu.

Ve výše uvedeném příkladu sděluje příkaz `server` balíku NYS, aby použil označený server pokaždé, když je dostupná aktuální doména systému NIS. Pokud však svůj počítač často přemísťujete v rámci různých domén systému NIS, budete možná chtít mít v souboru `yp.conf` informace o několika doménách. V souboru `yp.conf` můžete mít informace o serverech pro různé domény systému NIS. Stačí pouze přidat k příkazu `server` název požadované domény systému NIS. Například pro laptop můžete výše uvedený vzor změnit následujícím způsobem:

```
# yp.conf - konfigurační soubor pro systém NIS
#
server vbardolino winery
server vstout     brewery
```

Takto upravený soubor `yp.conf` vám umožní přenášet laptop mezi libovolnými dvěma doménami. Při procesu zavádění systému pouze stačí příkazem `domainname` nastavit požadovanou doménu systému NIS.

Po vytvoření tohoto základního konfiguračního souboru a ujištění se, že je tento soubor všem dostupný, byste měli spustit svůj první test a zjistit, zda se můžete spojit se svým serverem. Vyberte si nějakou serverem distribuovanou mapu, například `hosts.byname`, a pokuste se ji získat pomocí utility `ypcat`. Nástroj `ypcat` by se měl, stejně jako všechny ostatní administrativní nástroje, nacházet v adresáři `/usr/sbin`.

```
# ypcat hosts.byname
191.72.2.2          vbeaujolais     vbeaujolais.linus.lxnet.org
191.72.2.3          vbardolino      vbardolino.linus.lxnet.org
191.72.1.1          vlager          vlager.linus.lxnet.org
191.72.2.1          vlager          vlager.linus.lxnet.org
191.72.1.2          vstout          vstout.linus.lxnet.org
191.72.1.3          vale            vale.linus.lxnet.org
191.72.2.4          vchianti        vchianti.linus.lxnet.org
```

Získaný výstup by měl vypadat podobně jako výše uvedený výpis. Pokud místo tohoto obdržíte chybovou zprávu „`Can't bind to server which serves domain`“ nebo nějakou podobnou zprávu, pak buď vámi zadaný název domény systému NIS nemá v souboru `yp.conf` definovaný odpovídající server, nebo je server z nějakého důvodu nedostupný. V druhém případě se ujistěte, že příkaz `ping` směrovaný na příslušného hostitele vrátí přijatelné výsledky, a že daný hostitel má skutečně spuštěn server NIS. Přítomnost serveru NIS na daném hostiteli můžete ověřit pomocí příkazu `rpcinfo`, který by měl vrátit následující výstup:

```
# rpcinfo -u serverhost ypserv
program 100004 version 2 ready and waiting
```

10.6 Výběr vhodných map

Abyste zajistili dosažitelnost daného serveru NIS, musíte rozhodnout, které konfigurační soubory nahradíte mapami systému NIS a ke kterým konfiguračním souborům mapy systému NIS připojíte. Mapy systému NIS budete zpravidla používat u funkcí, které vyhledávají hostitele a hesla. Funkce vyhledávající hostitele mají význam tehdy, pokud nemáte spuštěnu službu BIND. Funkce vyhledávající hesla povolují všem uživatelům přihlášení ke svému účtu z libovolného systému v příslušné doméně systému NIS; tato funkce obvykle vyžaduje sdílení centrálního adresáře `/home` pomocí systému NFS mezi všema hostiteli. Tato funkce je podrobně probrána v následující stati. Ostatní mapy, jako například mapa `service.byname`, nejsou příliš užitečné, ale mohou vám ušetřit část konfiguračních prací v případě, kdy instalujete nějaké síťové aplikace, které používají název služby, jenž není uveden ve standardním souboru `services`.

V případech, kdy vyhledávací funkce používá místní soubory nebo kdy se dotazuje serveru NIS, si budete zpravidla chtít zachovat určitou možnost volby. Balík NYS dovoluje nastavit pořadí, ve kterém daná funkce k těmto službám přistupuje. Toto pořadí je řízeno konfiguračním souborem `/etc/nsswitch.conf`, který nahrazuje *přepínač názvových služeb* (*Name*

Service Switch). Tento konfigurační soubor se samozřejmě neomezuje pouze na jmenné služby. Pro každou funkci, která je podporována balíkem NYS a vyhledává určitá data, obsahuje konfigurační soubor řádek s výčtem služeb, které se budou používat.

Správné pořadí služeb závisí na typu dat. Je málo pravděpodobné, že by mapa `services.byname` obsahovala položky, které se liší od položek uvedených v místním souboru `services`; tato mapa jich jen může obsahovat o něco více. Bylo by tedy dobré, kdyby se dotazování odehrávalo nejdříve v místních souborech a teprve v případě, že by název požadované služby nebyl nalezen, pokračovalo pomocí služby NIS. Na druhou stranu se informace o názvu hostitele mohou velmi často měnit, takže by server DNS nebo server NIS měl mít vždy nejaktuálnější informace, zatímco místní soubor `hosts` je uchováván pouze jako záložní kopie, kdyby systém DNS nebo systém NIS selhal. V tomto případě budete chtít prozkoumat místní soubor až jako poslední.

Následující příklad ukazuje způsob nastavení funkcí *gethostbyname(2)*, *gethostbyaddr(2)* a *getservbyaddr(2)* tak, aby se chovaly výše popsaným způsobem. Tyto funkce postupně zkouší uvedené služby; je-li vyhledávání úspěšné, vrátí výsledek, v opačném případě se bude zkoušet další služba.

```
# jednoduchý příklad /etc/nsswitch.conf
#
hosts:      nis dns files
services:   files nis
```

Dále následuje kompletní seznam služeb, které mohou být použity jako položky konfiguračního souboru `nsswitch.conf`. Jaké mapy, soubory a servery budou ve skutečnosti dotazovány závisí na názvu položky.

nisplus nebo *nis+* — Pro danou doménu se použije server NIS+. Umístění příslušného serveru se získá ze souboru `/etc/nis.conf`.

nis — Pro danou doménu se použije aktuální server NIS. Umístění dotazovaného serveru je nastaveno v souboru `yp.conf`, který jsme si popsali v předchozí stati. Pro položku `hosts` budou dotazovány mapy `hosts.byname` a `hosts.byaddr`.

dns — Použije se jmenný server systému DNS. Tento typ služby je užitečný pouze ve spojitosti s položkou `hosts`. Dotazované jmenné servery jsou opět převzaty ze standardního souboru `resolv.conf`.

files — Použije se místní soubor, pro položku `hosts` je to například soubor `/etc/hosts`.

dbm Informace se vyhledají v souborech typu DBM, které jsou umístěny v adresáři `/var/dbm`. Název použitého souboru vyplývá z odpovídající mapy systému NIS.

Balík NYS v současné době akceptuje v souboru `nsswitch.conf` následující položky: `hosts`, `networks`, `passwd`, `group`, `shadow`, `gshadow`, `services`, `protocols`, `rpc` a `ethers`. Časem budou pravděpodobně přidány další.

Obrázek 10.1 ukazuje o něco složitější příklad, který ukazuje novou vlastnost konfiguračního souboru `nsswitch.conf`: Klíčové slovo *[NOTFOUND=return]*, které je uvedeno v položce `hosts`, sděluje balíku NYS, aby se v případě, že nelze požadovanou položku nalézt v databázi systému NIS nebo systému DNS, vrátil zpět k dříve uvedeným volbám. To znamená, že balík NYS bude pokračovat v prohledávání místních souborů *pouze v případě*, kdy volání serveru NIS nebo serveru DNS ztroskotá z nějakého jiného důvodu. Potom budou místní soubory použity pouze při procesu zavádění nebo jako záložní kopie v případě, že server NIS není spuštěn.

```
# /etc/nsswitch.conf
#
hosts:          nis dns [NOTFOUND=return] files
networks:       nis [NOTFOUND=return] files

services:       files nis
protocols:      files nis
rpc:            files nis
```

Obrázek 10.1
Vzorový soubor `nsswitch.conf`

10.7 Použití map passwd a group

Jednou z hlavních aplikací systému NIS je synchronizace informací o uživatelích a účtech mezi všemi hostiteli v rámci příslušné domény systému NIS. V důsledku toho se používá pouze malý soubor `/etc/passwd`, ke kterému se připojují informace z map systému NIS, které obsahují data od ostatních hostitelů. Obvykle ale nestačí pouze povolit v souboru `nsswitch.conf` vyhledávání dané služby pomocí systému NIS.

Spoléháte-li se na informace s hesly šířené pomocí systému NIS, musíte se nejprve ujistit, že číselné uživatelské id libovolného uživatele, které máte umístěno ve svém lokálním souboru `passwd`, odpovídá představě serveru NIS o uživatelském id daného uživatele. Totéž budete požadovat i pro jiné účely. Příkladem může být připojení k vlastní síti svazků systému NFS ostatními hostiteli.

Pokud se některé z číselných id v souboru `/etc/passwd` nebo v souboru `/etc/group` liší od číselných id, které jsou uvedeny v mapách, musíte upravit vlastnictví všech souborů, které se vztahují k danému uživateli. Nejprve byste měli přiřadit nové hodnoty všem uid a gid v souborech `passwd` a `group`; potom byste měli vyhledat všechny soubory, které patří změněným uživatelům, a nakonec byste měli změnit vlastnictví těchto souborů. Předpokládejme, že účet **news** měl uživatelské id rovno 9 a účet **okir** měl uživatelské id rovno 103. Tato uživatelská id se změnila na jinou hodnotu; pak byste měli spustit následující příkazy:

```
# find / -uid   9 -print >/tmp/uid.9
# find / -uid 103 -print >/tmp/uid.103
# cat /tmp/uid.9   | xargs chown news
# cat /tmp/uid.103 | xargs chown okir
```

Je důležité, abyste tyto příkazy spustili s *nově* nainstalovaným souborem `passwd` a abyste si zjistili názvy všech souborů dříve, než začnete měnit vlastnictví kteréhokoliv z nich. K aktualizaci vlastnictví souborů skupiny použijte obdobnou sekvenci příkazů.

Jakmile provedete tyto změny, budou ve vašem systému souhlasit číselná uid a gid s těmi, které jsou uvedeny na všech ostatních hostitelích ve vaší doméně systému NIS. V dalším kroku přidáme do souboru `nsswitch.conf` další konfigurační řádky, které povolí vyhledávání informací o uživatelích a skupinách pomocí systému NIS:

```
# /etc/nsswitch.conf - passwd a group
passwd: nis files
group:  nis files
```

Tyto příkazy zajistí, že program `login` a všechny jeho příbuzné programy se budou při pokusu o přihlášení uživatele nejprve dotazovat map systému NIS a pokud vyhledávání pomocí systému NIS neuspěje, bude vyhledávání pokračovat v místních souborech. Ze svých místních souborů většinou odstraníte záznamy většiny uživatelů a ponecháte v nich pouze záznamy pro uživatele **root** a obecné účty, jako například účet **mail**. To proto, že některé důležité systémové úkoly mohou vyžadovat mapování uživatelských id na jména uživatelů a naopak. Například administrativní úkoly `cron` mohou spustit příkaz `su`, aby dočasně přešly na účet

news nebo může podsystém protokolu UUCP poslat zprávu o svém současném stavu na účet **uucp**. Pokud místní soubor `passwd` neobsahuje položky pro účty **news** a **uucp**, pak tyto úkoly skončí výpadem při použití systému NIS.

Zde se vynoří dvě závažné námitky: na jednu stranu bude dosud popsané nastavení fungovat pouze s přihlašovacími příkazy, které neobsahují stínová hesla. Jako příklad těchto přihlašovacích příkazů můžeme uvést programy z balíku `util-linux`. Nesnáze spojené s použitím stínových hesel společně se systémem NIS budou probrány dále. Na druhou stranu nepřistupují k souboru `passwd` pouze přihlašovací příkazy – podívejte se na příkaz `ls`, který používá většina lidí takřka permanentně. Kdykoliv provádíte nějaké dlouhé výpisy, zobrazí příkaz `ls` symbolické názvy vlastníků souboru (uživatelů a skupin); to znamená, že kdykoliv příkaz `ls` narazí na nějaké uid nebo gid, bude se muset dotazovat serveru NIS. To velmi podstatně zpomalí chod programu v případě, že je vaše místní síť ucpaná nebo, v horším případě, není server NIS ve stejné fyzické síti, takže musí datagramy procházet přes směrovač.

Zde však celé povídání zdaleka nekončí. Představte si, co se stane, když si chce uživatel změnit své heslo. Obvykle spustí příkaz `passwd`, který přijme nové heslo a aktualizuje místní soubor `passwd`. Tento postup není možný s využitím systému NIS, protože tento soubor již není lokálně dostupný. Chce-li si uživatel změnit heslo, nepomůže mu ani přihlášení k serveru NIS. Proto systém NIS poskytuje náhradu k příkazu `passwd`, a to příkaz `yppasswd`, který provádí analogickou změnu hesla, avšak při spuštěném systému NIS. Chcete-li změnit heslo na hostiteli serveru, spojí se příkaz `yppasswd` za pomoci volání RPC s démonem `yppasswdd`, který již na tomto hostiteli běží, a poskytne mu aktualizované informace o heslu. Příkaz `yppasswd` se obvykle nainstaluje místo klasického příkazu `passwd` za pomoci následující sekvence příkazů:

```
# cd /bin
# mv passwd passwd.old
# ln yppasswd passwd
```

Současně musíte na serveru nainstalovat démona `rpc.yppasswdd` a spustit ho ze skriptu `rc.inet2`. Tyto úpravy efektivně skryjí před vašimi uživateli všechny změny, které vyžaduje systém NIS.

10.8 Použití systému NIS s podporou stínových hesel

Zatím neexistuje žádná podpora systému NIS pro systémy, které používají balík pro stínová hesla. John F. Haugh, autor stínového balíku, teprve nedávno uvolnil na adrese **comp.sources.misc** verzi knihovny se stínovými funkcemi, kterou vložil do knihovny GPL, jež je sou-

částí projektu GNU. Tato knihovna již obsahovala určitou podporu pro systém NIS, ale ta stále ještě není kompletní a její soubory zatím nebyly přidány do standardní knihovny C. Na druhou stranu může uveřejnění informací ze souboru `/etc/shadow` pomocí systému NIS určitým způsobem zmařit účel použití stínového balíku.

I když funkce pro vyhledávání hesel v balíku NYS nepoužívají mapu `shadow.byname` nebo nějakou podobnou mapu, podporuje balík NYS použití souboru `/etc/shadow`. Když je zavolána příslušná implementace příkazu `getpwnam` z balíku NYS, který má vyhledat informace vztahující se k danému přihlašovacímu jménu, budou dotazovány ty prostředky, které jsou uvedeny v záznamu `passwd` v souboru `nsswitch.conf`. Služba *nis* jednoduše vyhledá název na serveru NIS v mapě `passwd.byname`. Avšak služba *files* zkontroluje, zda je přítomen soubor `/etc/shadow`, a pokud ano, pokusí se ho otevřít. Jestliže takový soubor neexistuje nebo pokud nemá uživatel práva **root**, vrátí se služba *files* k tradičnímu chování a informace o uživateli se budou vyhledávat v souboru `/etc/passwd`. Pokud však soubor `shadow` existuje a jde-li otevřít, vytáhne si z něho balík NYS uživatelovo heslo. Funkce *getpwuid* je implementována obdobným způsobem. Tímto způsobem se binární soubory zkompilované pomocí balíku NYS zřetelně vypořádají s místní instalací stínového balíku.

10.9 Použití tradičního kódu systému NIS

Používáte-li kód klienta, který je v současné době součástí standardní knihovny *libc*, bude konfigurace klienta systému NIS nepatrně odlišná. K vysílání směrem k aktivním serverům se používá démon `ypbind`, takže se informace nezískávají z konfiguračního souboru. Z toho důvodu je třeba se ujistit, že při zavádění systému dochází ke spouštění démona `ypbind`. Tento démon musí být spuštěn po nastavení domény systému NIS a po spuštění mapovače portů RPC. Použití příkazu `ypcat` pro otestování serveru by pak mělo fungovat výše popsaným způsobem.

Nedávno byl hlášen velký počet chyb, při kterých systém NIS selhal a vypsal chybovou zpsávu „`clntudp_create: RPC: portmapper failure – RPC: unable to receive`“. Tato zpráva se objevovala z důvodu nekompatibility ve způsobu, jakým démon `ypbind` sděluje funkcím v knihovně informace o vazbách. Obstarání posledních zdrojových kódů utilit systému NIS a jejich následná rekompilace by měly tento problém vyřešit.[5]

Také způsob, jakým se tradiční systém NIS rozhoduje, zda a jak má připojit informace systému NIS k informacím z místních souborů, se liší od způsobu, který používá balík NYS. Chcete-li například, aby systém NIS používal mapy s hesly, je třeba do souboru své mapy `/etc/passwd` přidat následující řádek:

5 Zdrojový kód pro balík `yp-linux` může být získán na adrese **ftp.uni-paderborn.de** v adresáři `/pub/Linux/LOCAL`.

```
+:*:0:0:::
```

Tento výraz označuje místo, kam mají funkce pro vyhledávání hesla „vložit“ mapy systému NIS. Vložíte-li podobnou řádku (bez závěrečných dvojteček) do souboru `/etc/group`, zjistíte stejnou funkci pro mapy `group.*`. Chcete-li používat mapy `hosts.*` distribuované pomocí systému NIS, změňte řádek `order` v souboru `host.conf`. Když chcete používat například systémy NIS, DNS a soubor `/etc/hosts` (v tomto pořadí), musíte změnit řádek `order` následovně:

```
order yp bind hosts
```

V současné době nepodporuje tradiční implementace systému NIS žádné další mapy.

Kapitola

11

Síťový souborový systém (NFS)

Síťový souborový systém neboli NFS je pravděpodobně nejvýznamnější síťovou službou, která využívá balík RPC. Umožňuje přístup k souborům na vzdálených hostitelích, který je uskutečňován zcela stejným způsobem jako přístup uživatele k místním souborům. Toto chování je možné díky kombinaci funkce jádra operačního systému na straně klienta (který používá vzdálený souborový systém) a serveru NFS na straně serveru (který poskytuje souborová data). Tento přístup k souborům je pro klienta zcela transparentní a funguje na nejrůznějších serverových a klientských architekturách.

Systém NFS nabízí několik výhod:

- Data, k nimž přistupují všichni uživatelé, mohou být uchovávána na centrálním hostiteli a klienti si tento adresář připojí při zavádění systému. Například můžete uchovávat všechny účty uživatelů na jediném hostiteli a všichni hostitelé ve vaší síti si z tohoto hostitele připojí adresář `home`. Je-li tento systém nainstalován společně se systémem NIS, potom se uživatelé mohou přihlásit k libovolnému systému a přitom mohou stále pracovat s jednou sadou souborů.

- Data, která zabírají velké množství diskového prostoru, mohou být uchovávána na jediném hostiteli. Například všechny soubory a programy vztahující se k balíkům LaTeX a METAFONT mohou být uloženy a spravovány na jednom místě.

- Administrativní data mohou být uchovávána na jediném hostiteli. Již není třeba používat příkaz `rcp` k nainstalování stejného souboru na 20 různých počítačích.

Na systému NFS má v Linuxu zásluhu zejména Rick Sladkey,[1] který napsal kód systému NFS pro jádro operačního systému a značnou část serveru NFS. Server NFS byl odvozen ze serveru *unfsd* (uživatelský server NFS), který původně vytvořil Mark Shand, a ze serveru *hnfs* (Harrisův server NFS), jehož autorem je Donald Becker.

Podívejme se nyní, jak systém NFS funguje: Klient může požádat o připojení adresáře ze vzdáleného hostitele ke svému místnímu adresáři naprosto stejným způsobem, jako při připojování fyzického zařízení. Avšak syntaxe používaná k zadání vzdáleného adresáře je odlišná. Chcete-li třeba připojit adresář `/home` z hostitele **vlager** k adresáři `/users` na hostiteli **vale**, měl by správce na hostiteli **vale** spustit následující příkaz:[2]

```
# mount -t nfs vlager:/home /users
```

Příkaz `mount` se pokusí spojit pomocí procedur RPC s připojovacím démonem `mountd`, který běží na hostiteli **vlager**. Server zkontroluje, zda má hostitel **vale** povoleno příslušné připojení, a pokud ano, vrátí mu souborový klíč. Tento souborový klíč bude použit ve všech následujících souborových požadavcích pod adresářem `/users`.

Při přístupu k souboru pomocí systému NFS spustí jádro systému volání procedury RPC směrem na démona `nfsd` (démon systému NFS), který se nachází na počítači serveru. Toto volání získá souborový klíč, název souboru, k němuž se má přistupovat, a jako parametry i id uživatele a skupiny, které se vztahují k danému uživateli. Tyto hodnoty slouží k určení přístupových práv k zadanému souboru. Aby se zabránilo neautorizovaným uživatelům ve čtení nebo úpravě souborů, musí být id uživatele a skupiny na obou hostitelích totožná.

Ve většině unixových implementací je systém NFS jak pro klienta, tak i pro server implementován pomocí démonů, kteří běží na úrovni jádra operačního systému a jsou spouštěni z uživatelského prostoru při zavádění systému. Na hostiteli serveru je spuštěn démon systému NFS (`nfsd`) a na hostiteli klienta je spuštěn *blokový V/V démon* (*Block I/O daemon* – `biod`). Aby se zvýšila propustnost, provádí démon `biod` asynchronní V/V komunikaci s využitím dopředného čtení a opožděného zápisu; současně je obvykle spuštěno několik démonů `nfsd`.

Implementace systému NFS v Linuxu se nepatrně liší tím, že kód klienta je pevně integrován do virtuálního systému souborů (virtual file system – VFS) v jádru operačního systému a nepotřebuje dodatečné řízení pomocí démona `biod`. Na druhou stranu běží kód serveru kompletně v uživatelském prostoru, takže je v zásadě nemožné současně spustit více kopií serveru, protože by mohlo dojít k problémům se synchronizací.

---

1 Ricka můžete zastihnout na adrese **jrs@world.std.com**.

2 Všimněte si, že argument `-t nfs` je možné vynechat, protože příkaz `mount` podle dvojtečky poznal, že jde o svazek systému NTFS.

Největší problém linuxového kódu systému NFS spočívá v tom, že jádro operačního systému Linux není ve verzi 1.0 schopno alokovat více paměti, než 4 KB; v důsledku toho neumí síťový kód spravovat větší datagramy, než 3500 bajtů, což je velikost datagramu po odečtení velikosti hlavičky atd. To znamená, že přenosy z a na démony systému NFS, které implicitně používají velké UDP datagramy (například datagramy o velikosti 8 KB, které používá SunOS), musí být uměle upraveny na podporovanou velikost datagramu. To může mít za určitých okolností tvrdý dopad na výkon systému.[3] Tento limit se podařilo odstranit teprve nedávno uvedením jádra operačního systému Linux verze 1.1. Kód klienta byl navíc upraven tak, aby této výhody uměl využít.

11.1 Příprava systému NFS

Před použitím systému NFS, ať už jako klienta nebo jako serveru, je třeba se ujistit, že jádro vašeho operačního systému má zkompilovanou podporu pro systém NFS. Novější jádra mají pro tento účel jednoduché rozhraní, které poskytuje informace o souborových systémech. Veškeré informace najdete v souboru `/proc/filesystems`, který zobrazíte příkazem `cat`:

```
$ cat /proc/filesystems
minix
ext2
msdos
nodev   proc
nodev   nfs
```

Pokud v tomto seznamu chybí souborový systém `nfs`, pak je nutné zkompilovat vlastní jádro operačního systému s podporou systému NFS. Konfigurace síťových voleb jádra operačního systému je probrána ve stati „Konfigurace jádra operačního systému“, která je ve 3. kapitole.

Ve starších verzích jádra operačního systému, před verzí Linuxu 1.1, je nejjednodušším způsobem, jak zjistit, zda vaše jádro má povolenou podporu systému NFS, vyzkoušet připojit nějaký souborový systém NFS. Za tím účelem můžete třeba v adresáři `/tmp` vytvořit podadresář a zkusit k němu připojit nějaký místní adresář:

```
# mkdir /tmp/test
# mount localhost:/etc /tmp/test
```

3 Takto mi to vysvětlil pan Alan Cox: Specifikace systému NFS vyžaduje, aby server předtím, než vrátí potvrzení, uložil všechny zápisy na disk. Protože jádra balíku BSD jsou schopna zapisovat pouze data o velikosti stránky (což jsou 4 KB), budou k zápisu 4 bloků o velikosti 1 KB dat na server NFS, který používá balík BSD, použity čtyři operace zápisu, z nichž každá bude pracovat se 4 KB.

Pokud tento pokus o připojení místního adresáře selže a objeví se zpráva „`fs type nfs no supported by kernel`“, pak bude třeba vytvořit nové jádro operačního systému s povolenou podporou systému NFS. Jakékoliv další chybové zprávy jsou naprosto neškodné, protože jste zatím na svém hostiteli nenakonfigurovali démony systému NFS.

11.2 Připojení svazku systému NFS

Svazky systému NFS[4] se připojují podobně, jako běžné souborové systémy. Spustíte příkaz `mount`, který bude mít následující syntaxi:

```
# mount -t nfs nfs_volume local_dir options
```

Svazek `nfs_volume` je předán ve tvaru `remote_host:remote_dir` (`vzdálený_hostitel:vzdálený_adresář`). Protože je tento zápis jedinečný pro systémy NFS, můžete vynechat volbu `-t nfs`.

Existuje také spousta doplňkových voleb, které můžete při připojování svazku systému NFS použít v příkazu `mount`. Tyto volby mohou buď následovat na příkazové řádce za přepínačem `-o`, nebo je lze pro daný svazek uvedést v příslušné položce voleb v souboru `/etc/fstab`. V obou případech se více voleb navzájem odděluje čárkami. Volby zadané na příkazové řádce vždy potlačí volby, které jsou uvedeny v souboru `fstab`.

Vzorová položka v souboru *fstab* může vypadat takto:

```
# svazek                připojovací bod    typ    volby
news:/usr/spool/news   /usr/spool/news    nfs    timeo=14,intr
```

Pak lze tento svazek připojit za pomoci příkazu

```
# mount news:/usr/spool/news
```

Pokud by v souboru `fstab` nebyly uvedeny žádné volby, bylo by použití příkazu `mount` na systém NFS složitější. Předpokládejme například, že si chcete připojit domovské adresáře svých uživatelů z počítače **moonshot**, který používá pro operace čtení/zápis implicitní velikost bloku 4 KB. Velikost bloku můžete snížit pomocí následujícího příkazu na 2 KB, aby vyhovovala omezení vztahujícímu se na velikosti datagramu v operačním systému Linux:

```
# mount moonshot:/home /home -o rsize=2048,wsize=2048
```

4 Nepoužíváme zde pojem souborový systém, protože systém NFS není pravým souborovým systémem.

Kompletní seznam všech korektních voleb je popsán na stránkách manuálu *nfs(5)*, jenž je součástí nástroje `mount` od Ricka Sladkeyho. Tento nástroj spolupracuje se systémem NFS (najdete ho v balíku `util-linux` od Rika Faitha). Následuje neúplný seznam voleb, které se vám mohou hodit:

rsize=n a *wsize=n* — Tyto volby určují velikosti datagramu, které budou používat klienti systému NFS u požadavků na čtení, resp. zápis. Jejich současná implicitní hodnota je, z důvodu výše popsaného omezení velikosti datagramu protokolu UDP, 1 024 bajtů.

timeo=n — Tato volba nastavuje čas (v desetinách sekundy), po který bude klient čekat na splnění jeho požadavku. Implicitní hodnota je 0,7 sekundy.

hard — Explicitně označí tento svazek jako pevně připojený svazek. Tato volba je implicitně zapnutá.

soft — Ovladač připojí svazek volně (to je opak pevného připojení).

intr — Tato volba povolí signálům přerušit volání systému NFS. Tato volba je užitečná, když potřebujete zrušit volání, například v situaci, kdy server neodpovídá.

Kromě voleb *rsize* a *wsize* všechny výše uvedené volby nastavují chování klienta pro případ, kdy server není dočasně dostupný. Tyto volby spolu souvisí následovně: kdykoliv klient pošle požadavek na server NFS, očekává, že bude operace dokončena v daném časovém limitu (ten udává volba *timeout*). Pokud v tomto intervalu nepřijde žádné potvrzení, nastane tzv. *vedlejší překročení časového limitu* (*minor timeout*) a operace se spustí znovu s dvojnásobnou hodnotou časového intervalu. Po dosažení maximálního časového intervalu 60 sekund dojde k tzv. *hlavnímu překročení časového limitu* (*major timeout*).

Hlavní překročení časového limitu implicitně způsobí, že klient vypíše na konzolu zprávu a celý proces se bude znovu opakovat, tentokrát ovšem s časovým intervalem rovným dvojnásobku časového intervalu z předchozího pokusu. Teoreticky by tento postup mohl trvat nekonečně dlouhou dobu. Svazky, které se tvrdohlavě pokoušejí opakovat operaci tak dlouho, dokud nebude server opět dostupný, se nazývají *pevně připojené* svazky. Opačný typ svazků představují tzv. *volně připojené* svazky, které při překročení hlavního časového limitu vygenerují pro daný volaný proces V/V chybu. Protože se používá opožděný zápis, který jsme si představili společně s vyrovnávací pamětí cache, nebude tato chybová zpráva předána vlastnímu procesu dříve, než bude volána funkce *write(2)*, takže program nemá nikdy jistotu, zda operace zápisu u volně připojeného svazku proběhla správně.

Zda se má daný svazek připojit pevně anebo volně není jen pouhou otázkou vkusu, ale tato volba musí záviset na typu informací, ke kterým chcete na tomto svazku přistupovat. Pokud si například pomocí systému NFS připojíte programy X, určitě nebudete chtít, aby se vaše relace chovala divně jen proto, že někdo zablokoval síť například současným spuštěním sedmi kopií programu *xv* nebo krátkodobým vytáhnutím ethernetové zástrčky. Když tento svazek připojíte pevně, budete mít jistotu, že váš počítač bude čekat tak dlouho, dokud se znovu neobnoví spojení s vaším server NFS. Na druhou stranu mohou existovat volně připojené svazky, které neobsahují kritická data, příkladem může být připojení části news pomocí systému NFS nebo připojení archívů protokolu FTP. Pokud nebude vzdálený počítač dočasně dosažitelný, nebo pokud bude vypnutý, nezablokuje volné připojení svazků vaši relaci. Je-li vaše spojení se serverem nestálé nebo prochází přes vytížený směrovač, můžete buď zvýšit počáteční hodnotu časového intervalu (pomocí volby *timeo*), nebo můžete dané svazky připojit pevně, avšak zároveň musíte povolit signály přerušující volání systému NFS, protože jinak nebudete moci přerušit zablokovaný přístup k souboru.

Démon `mountd` bude obvykle nějakým způsobem informován o tom, které adresáře jsou připojeny kterým hostitelem. Tyto informace lze zobrazit za pomoci programu `showmount`, který je taktéž součástí balíku serveru systému NFS.

11.3 Démoni systému NFS

Pokud chcete službu NFS poskytovat i ostatním hostitelům, musíte mít na svém počítači spuštěny démony `nfsd` a `mountd`. Protože se jedná o programy založené na balíku RPC, nejsou spravovány super serverem `inetd`, ale jsou spouštěny při procesu zavádění systému a sami se registrují pomocí mapovače portů. Proto se musíte ujistit, že je spouštíte až po spuštění démona `rpc.portmap`. Obvykle se do souboru `rc.inet2` přidají následující dvě řádky:

```
if [ -x /usr/sbin/rpc.mountd ]; then
        /usr/sbin/rpc.mountd; echo -n " mountd"
fi
if [ -x /usr/sbin/rpc.nfsd ]; then
        /usr/sbin/rpc.nfsd; echo -n " nfsd"
fi
```

Informace o vlastnictví, které poskytuje démon systému NFS svým klientům, obvykle obsahují pouze číselná id uživatele a skupiny. Pokud klient i server sdruží s těmito číselnými id shodné názvy uživatelů a skupin, potom říkáme, že sdílejí stejný prostor uid/gid. Příkladem tohoto sdílení může být použití systému NIS k distribuci informací ze souboru `passwd` všem hostitelům ve vaší lokální síti.

V některých případech se však tato čísla neshodují. Potom spíše aktualizujte čísla uid a gid na straně klienta tak, aby odpovídala číslům na straně serveru. K tomuto účelu můžete použít mapovacího démona `ugidd`. Použijete-li níže popsanou volbu *map_daemon*, řeknete tím démonu `nfsd`, že má prostor uid/gid nacházející se na serveru přiřadit k prostoru uid/gid, který se nachází na straně klienta. K této operaci se využije démona `ugidd`, který je spuštěný na straně klienta.

Démon `ugidd` je server založený na balíku RPC a spouští se ze souboru `rc.inet2`, stejně jako démoni `nfsd` a `mountd`.

```
if [ -x /usr/sbin/rpc.ugidd ]; then
         /usr/sbin/rpc.ugidd; echo -n " ugidd"
fi
```

11.4 Soubor exports

Zatímco výše uvedené volby se týkají konfigurace systému NFS na straně klienta, na straně serveru existuje odlišná skupina voleb, které konfigurují jeho chování vůči klientovi. Tyto volby musí být nastaveny v souboru `/etc/exports`.

Démon `mountd` implicitně nepovolí žádnému uživateli připojení adresářů z místního hostitele, což je poměrně rozumný přístup. Chcete-li jednomu nebo více hostitelům povolit připojení adresáře pomocí systému NFS, musí být tento adresář *exportován*, což znamená, že musí být uveden v souboru `exports`. Vzorový soubor `exports` by mohl vypadat následovně:

```
# soubor exports pro vlager
/home               vale(rw) vstout(rw) vlight(rw)
/usr/X386           vale(ro) vstout(ro) vlight(ro)
/usr/TeX            vale(ro) vstout(ro) vlight(ro)
/                   vale(rw,no root squash)
/home/ftp           (ro)
```

Každá řádka definuje adresář a hostitele, kteří si ho mohou připojit. Název hostitele je obvykle plně kvalifikovaným doménovým jménem, ale kromě toho může obsahovat i zástupné znaky * a *?*, které fungují stejným způsobem, jako v příkazovém interpretu `bash`. Například adrese **lab*.foo.com** vyhovuje jak adresa **lab01.foo.com**, tak i adresa **laber.foo.com**. Pokud není zadán žádný název hostitele, jako tomu bylo ve výše uvedeném příkladu s adresářem `/home/ftp`, potom si budou moci tento adresář připojit všichni uživatelé.

Při ověřování klienta pomocí souboru `exports` vyhledá démon `mountd` název hostitele klienta pomocí volání procedury *gethostbyaddr(2)*. V systému DNS vrátí tato procedura kanonický název hostitele klienta, takže je třeba se ujistit, že v souboru `exports` nepoužíváte žádné přezdívky. Bez použití systému DNS bude vrácený název odpovídat prvnímu nalezenému názvu v souboru `hosts`, který odpovídá adrese klienta.

Za názvem hostitele může následovat volitelný seznam příznaků oddělených čárkami a uzavřený do kulatých závorek. Tyto příznaky mohou nabývat následujících hodnot:

insecure Povoluje neautorizovaný přístup z tohoto počítače.

unix-rpc Vyžaduje ověření unixové domény pomocí balíku RPC z tohoto počítače. Tento příznak vyžaduje, aby požadavky přicházely z rezervovaného internetového portu (tj. číslo portu musí být nižší než hodnota 1 024). Tato volba je implicitně zapnutá.

secure-rpc Vyžaduje bezpečné ověření totožnosti pomocí balíku RPC z tohoto počítače. Tato funkce zatím nebyla implementována. Viz dokumentace Secure RPC od firmy Sun.

kerberos Aby byl umožněn přístup z tohoto počítače, je vyžadováno ověření totožnosti pomocí balíku Kerberos. Tato funkce nebyla zatím implementována. Viz dokumentace MIT, kde najdete informace o ověřovacím systému balíku Kerberos.

root_squash Toto je bezpečnostní vlastnost, která na zadaných hostitelích zablokuje superuživatelům speciální přístupová práva. Toho se docílí přesměrováním požadavku z čísla uid 0 na straně klienta na číslo uid 65 534 (-2) na straně serveru. Toto číslo uid by mělo být přiřazeno uživateli jménem **nobody**.

no_root_squash Nebudou se mapovat požadavky od uživatele, který má uid rovno 0. Tato volba je implicitně zapnutá.

ro Hierarchie souborového systému se připojí v režimu pouze pro čtení. Tato volba je implicitně zapnutá.

rw Hierarchie souborového systému se připojí v režimu pro zápis i čtení.

link_relative Tato volba převede absolutní symbolické odkazy (to jsou takové odkazy, které začínají symbolem lomítka) na relativní odkazy. Toho se docílí připojením potřebného počtu posloupnosti znaků „../“ před vlastní název symbolického odkazu, tím se z adresáře obsahujícího daný odkaz

stane kořenový adresář na serveru. Tato volba má smysl pouze v případě, kdy je připojen celý souborový systém hostitele, v opačném případě by se mohlo stát, že některá spojení nebudou ukazovat nikam, anebo v horším případě na soubory, které by rozhodně neměly být vidět.

Tato volba je implicitně zapnutá.

link_absolute Zanechá všechny symbolické odkazy v původním stavu (toto je normální chování serverů NFS, které jsou dodávány firmou Sun).

map_identity Volba *map_identity* říká serveru, aby předpokládal, že klient používá stejná čísla uid a gid jako server. Tato volba je implicitně zapnutá.

map_daemon Tato volba sdělí serveru NFS, aby předpokládal, že klient a server nesdílí společný prostor s čísly uid/gid. Potom démon `nfsd` vytvoří seznam mapující čísla id mezi klientem a serverem. Ty získá tak, že se bude na straně klienta dotazovat démona `ugidd`.

Chyba zjištěná při analýze souboru `exports` bude zapsána do souboru `syslog` s úrovní *notice* (*oznámení*). Tato chyba se zapíše vždy, když je spuštěn démon `nfsd` nebo `mountd`.

Pamatujte, že názvy hostitelů jsou získány z IP-adresy klienta pomocí zpětného mapování, takže je nutné mít správně nakonfigurovaný resolver. Pokud používáte službu BIND a zakládáte si na bezpečnosti, měli byste v souboru `host.conf` povolit kontrolu spoofingu.

11.5 Automatické připojování (automounter) v operačním systému Linux

V některých případech není hospodárné připojovat všechny svazky systému NFS, ke kterým by mohli chtít uživatelé přistupovat; buď z důvodu pouhého množství připojovaných svazků, nebo z důvodu časové prodlevy, kterou by si tento proces vyžádal při zavádění systému. Schůdnou alternativou řešení tohoto problému je tzv. *démon automatického připojování*. Jedná se o démona, který podle potřeby automaticky a zcela transparentně připojuje libovolné svazky systému NFS a taktéž je automaticky odpojuje, pokud nebyly po určitou dobu používány. Další šikovnou vlastností, kterou disponuje démon automatického připojování, je možnost připojení určitého svazku z alternativních míst. Například kopie programů X a podpůrných souborů je možné uchovávat pouze na dvou nebo třech hostitelích a všichni ostatní hostitelé si tyto programy a podpůrné soubory připojí pomocí systému NFS. Když k tomuto účelu použijete démona automatického připojování, můžete všechny tři hostitele namapovat do adresáře `/usr/X386`; démon automatického připojování se potom bude zkoušet připojit ke všem třem hostitelům, dokud se k některému nepřipojí.

Běžně používaný démon automatického připojování se v Linuxu nazývá `amd`. Původně ho napsal Jan-Simonem Pendry a na platformu Linuxu ho portoval Rick Sladkey. Aktuální verze tohoto démona je `amd-5.3`.

Vysvětlení démona `amd` přesahuje rozsah této kapitoly; jako dobrý manuál by vám však mohl posloužit jeho zdrojový kód; tento zdrojový kód obsahuje soubor s textovými informacemi, které jsou velmi podrobné.

Kapitola

12

Správa protokolu Taylor UUCP

12.1 Historie

Protokol UUCP byl navržen koncem sedmdesátých let panem Mikem Leskem ze společnosti AT&T Bell Laboratories. Tento protokol měl poskytovat jednoduché připojení k síti pomocí veřejných telefonních linek. Protože většina lidí, kteří chtějí mít na svém domácím počítači elektronickou poštu a usenetové news, stále používá ke komunikaci modemy, zůstává protokol UUCP i nadále velice populární. I přes vysoký počet implementací tohoto protokolu, které lze provozovat na širokém spektru hardwarových platforem a operačních systémů, jsou tyto implementace navzájem vysoce kompatibilní.

Nicméně jako u většiny softwaru, u něhož se až v průběhu let vyvinul určitý „standard“, neexistuje ani u protokolu UUCP taková implementace, která by se nazývala jednoduše UUCP. Protokol UUCP prošel od své první verze, jež byla uvedena v roce 1976, neustálým evolučním vývojem. V současné době existují dva hlavní druhy tohoto protokolu, které se navzájem značně odlišují jak v podporovaném hardwaru, tak i ve své vlastní konfiguraci. Z těchto dvou druhů protokolu UUCP se vyvinula spousta různých implementací, které se od svých předků liší jen minimálně.

Jeden druh se nazývá „protokol UUCP verze 2“ a jeho vznik je datován do roku 1977, kdy Mike Lesk, David A. Novitz a Greg Chesson vytvořili novou implementaci protokolu UUCP. I když je tato implementace poměrně stará, stále se hojně používá. Novější implementace verze 2 poskytuje spoustu vymožeností obsažených v novějších druzích protokolu UUCP.

Druhý typ protokolu byl vytvořen v roce 1983 a běžně se označuje názvy BNU (Basic Networking Utilities – základní síťové utility) nebo HoneyDanBer UUCP, zkráceně HDB. Tento název je odvozen ze jmen autorů, P. Honeymana, D. A. Novitze a B. E. Redmana. Protokol HDB byl vymyšlen proto, aby odstranil některé nedostatky protokolu UUCP verze 2. Byly

například přidány nové transportní protokoly a dočasný odkládací adresář byl rozdělen tak, že v současnosti existuje jeden adresář pro každý systém, se kterým komunikujete na bázi protokolu UUCP.

V současnosti se společně s operačním systémem Linux dodává implementace protokolu UUCP pod názvem Taylor UUCP verze 1.04.[1] Z této verze vychází i tato kapitola. Protokol Taylor UUCP verze 1.04 byl dán do oběhu v únoru roku 1993 a bývá zkompilován tak, aby kromě tradičních konfiguračních souborů ovládal i konfigurační soubory nové generace – tzv. „Taylorovy“ konfigurační soubory.

Nedávno byla uvolněna verze 1.05 a již brzy se tato verze stane součástí většiny distribucí Linuxu. Odlišnosti mezi těmito verzemi se většinou týkají rysů, které nikdy nebudete používat, takže na základě informací uvedených v této knize budete moci nakonfigurovat i protokol Taylor UUCP verze 1.05.

Ve většině distribucí Linuxu je obvykle protokol UUCP zkompilován tak, aby byl buď kompatibilní s utilitami BNU nebo kompatibilní s Taylorovým konfiguračním schématem, případně s oběma verzemi. Protože je Taylorovo konfigurační schéma mnohem flexibilnější a pravděpodobně i snáze pochopitelné, než často nesrozumitelné konfigurační soubory utilit BNU, budu v této kapitole popisovat Taylorovo konfigurační schéma.

Účelem této kapitoly není poskytnout vyčerpávající popis všech voleb příkazové řádky pro všechny existující příkazy protokolu UUCP a popis funkce všech těchto příkazů, nýbrž úvod do problematiky nastavení funkčního uzlu UUCP. Doufejme, že první stať vám poskytne zevrubný úvod do problematiky implementace vzdáleného spouštění a přenosu souborů pomocí protokolu UUCP. Nejste-li úplným nováčkem v oblasti práce s protokolem UUCP, můžete tuto kapitolu přeskočit a přejít na stať zabývající se konfiguračními soubory protokolu UUCP, která vysvětluje použití různých souborů pro nastavení protokolu UUCP.

Budeme ale předpokládat, že znáte programy z balíku UUCP. Konkrétně jsou to programy *uucp* a *uux*. Popis těchto příkazů naleznete na on-line manuálových stránkách.

Kromě veřejně dostupných programů *uux* a *uucp* obsahuje balík UUCP také množství příkazů, které se používají pouze k administrativním účelům. Používají se k monitorování UUCP dopravy ve vašem uzlu, k odstraňování starých souborů se záznamy nebo k sestavování statistik. Ani jeden z těchto příkazů zde nebudeme rozebírat, protože nejsou pro hlavní činnost protokolu UUCP důležité. Kromě toho je jejich dokumentace na velmi slušné úrovni a člověk jim snadno porozumí. Existuje však ještě třetí kategorie, která představuje skutečného „zá-

[1] Tuto implementaci napsal v roce 1993 Ian Taylor, který vlastní i autorská práva.

vodního koně“ protokolu UUCP. Tyto programy se nazývají `uucico` (kde zkratka cico znamená copy-in copy-out – kopie dovnitř kopie ven) a `uuxqt`, který spouští úkoly přijaté ze vzdálených systémů.

12.1.1 Další informace o protokolu UUCP

Ti, co nenajdou vše potřebné v této kapitole, by si měli přečíst dokumentaci, která je součástí balíku UUCP. Tato dokumentace se skládá z několika textových souborů popisujících nastavení při použití Taylorova konfiguračního schématu. Textové informace mohou být pomocí příkazů `tex` a `makeinfo` převedeny do informačních souborů formátu DVI, resp. GNUinfo.

Pokud chcete používat konfigurační soubory kompatibilní s utilitami BNU nebo dokonce (a z toho mě mrazí v zádech) konfigurační soubory verze 2, mohla by vám pomoci kvalitní kniha „Managing UUCP and Usenet“ ([Oreilly89]). Považuji ji za velmi užitečnou. Dalším velmi dobrým zdrojem informací o linuxovém protokolu UUCP je informační dokument UUCP-HOWTO od Vince Skahana.

Dále existuje diskusní skupina zabývající se protokolem UUCP, která se nazývá **comp.mail.uucp**. Máte-li specifické dotazy ohledně Taylorova konfiguračního schématu, bude lepší, když se na ně zeptáte přímo v této diskusní skupině, než ve skupině **comp.os.linux**.

12.2 Úvod

12.2.1 Přehled protokolu UUCP - přenos souborů a vzdálené spouštění

Pro pochopení protokolu UUCP je důležité porozumět konceptu *úkolů*. Každý přenos iniciovaný uživatelem pomocí příkazů `uucp` nebo `uux` se nazývá *úkol*. Ten se skládá z *příkazu*, který má být spuštěn na vzdáleném systému, a ze skupiny *souborů*, které mají být mezi propojenými systémy přeneseny. Jednu z těchto částí lze vynechat.

Jako příklad budeme předpokládat, že jste na svém hostiteli spustili následující příkaz, který sdělí protokolu UUCP, aby překopíroval soubor `netguide.ps` na hostitele **pablo** a aby spustil příkaz `lpr`, který tento soubor vytiskne.

```
$ uux -r pablo!lpr !netguide.ps
```

Protokol UUCP se většinou hned nespojí se vzdáleným systémem, aby provedl patřičný úkol (okamžitě to lze provést pomocí programu `kermit`). Místo toho dočasně uloží popis daného úkolu. Tento proces se nazývá *dočasné odkládání*. Strom s adresáři, ve kterém jsou uloženy jednotlivé úkoly, se nazývá *dočasný adresář* a je zpravidla umístěn v adresáři `/var/spo-`

ol/uucp. V našem příkladu by měly informace o úkolu obsahovat vzdálený příkaz (lpr), který se má spustit, jméno uživatele, který o toto spuštění žádá, a několik dalších položek. Kromě popisů jednotlivých úkolů musí protokol UUCP ukládat vstupní soubor, konkrétně soubor netguide.ps.

Přesné umístění a názvy dočasných odkládacích souborů se mohou lišit v závislosti na nastavení některých voleb při sestavování. Protokol UUCP kompatibilní s protokolem HDB zpravidla dočasně ukládá soubory v adresáři s názvem /var/spool/uucp/*site*, kde *site* je název vzdáleného systému. Když je protokol UUCP zkompilován s podporou Taylorova konfiguračního schématu, vytvoří protokol UUCP pro různé typy dočasně odkládaných souborů podadresáře v dočasném adresáři konkrétního systému.

Protokol UUCP se pak v pravidelných intervalech spojuje se vzdáleným systémem. Když se uskuteční spojení se vzdáleným systémem, přenese protokol UUCP soubory popisující daný úkol a všechny vstupní soubory. Příchozí úkoly nebudou spuštěny okamžitě, ale až po skončení spojení. To provede příkaz uuxqt, který se rovněž stará o doručení úkolů, které jsou určeny pro jiný systém.

Aby protokol UUCP rozlišil důležité a méně důležité úkoly, přiřazuje každému úkolu *prioritu*. Priorita je definována jediným znakem v rozmezí od 0 do 9, od A do Z a od a do z. Hodnota 0 odpovídá nejvyšší prioritě, hodnota z odpovídá prioritě nejnižší. Pošta je obvykle dočasně ukládána s prioritou B nebo C, zatímco news jsou ukládány s prioritou N. Úkoly s vyšší prioritou jsou přenášeny dříve. Prioritu můžete přiřadit příkazům uucp a uux pomocí parametru -g.

V určitých časových intervalech můžete také zabránit přenášení úkolů, které mají nižší prioritu, než zadáte. Tato vlastnost se také označuje jako tzv. *maximální odkládací priorita* povolená při komunikaci a implicitně je nastavena na hodnotu z. Všimněte si této terminologické dvojsmyslnosti: soubor je přenesen pouze v případě, že má prioritu *vyšší* nebo *shodnou* s maximální odkládací prioritou.

12.2.2 Vnitřní funkce programu uucico

Abyste pochopili, proč potřebuje program uucico znát určitá data, bude vhodné uvést rychlý popis toho, jakým způsobem se tento program ve skutečnosti spojuje se vzdáleným systémem.

Když na příkazové řádce spustíte příkaz uucico -s system, musí se příkaz uucico nejprve fyzicky připojit. Provedené akce budou záviset na typu navazovaného spojení – například při použití telefonní linky musí příkaz uucico nejprve najít modem a vytočit číslo. U spojení pomocí protokolu TCP musí nejprve zavolat funkci *gethostbyname(3)*, aby převedl název hostitele na síťovou adresu, poté musí zjistit port, který se má otevřít, a přiřadit adresu odpovídajícímu socketu.

Po navázání spojení je třeba provést ověření totožnosti. To se zpravidla skládá z výzvy vzdálenému systému, aby zadal přihlašovací jméno a volitelně i heslo. Tento proces se obecně nazývá *přihlašovací komunikace*. Procedura ověření totožnosti se buď provede pomocí běžného balíku `getty/login`, nebo – u socketů protokolu TCP – vlastním programem `uucico`. Pakliže je ověření totožnosti úspěšné, spustí se na vzdáleném konci program `uucico`. Místní kopie programu `uucico`, která byla původcem spojení, se označuje jako *řídicí* a vzdálená kopie se označuje jako *řízená*.

Dále následuje *inicializační fáze*: řídicí program pošle svůj název hostitele společně s několika příznaky. Řízený program ověří povolení k přihlášení pro zadaný název hostitele, pošle a přijme soubory atd. Příznaky popisují (kromě jiných vlastností) maximální prioritu dočasně ukládaných souborů, které se budou přenášet. Pokud je povolena sekvenční kontrola hovoru, uskuteční se v této fázi i ověření počtu hovorů neboli *sekvenčního počtu hovorů*. Tato volba způsobí, že si budou oba systémy hlídat počet úspěšných spojení a ty pak porovnají. Pokud si počty úspěšných spojení neodpovídají, pak inicializační fáze neuspěje. Tato vlastnost je užitečná při ochraně vašeho systému před potenciálními podvodníky.

A nakonec se oba programy `uucico` pokusí dohodnout na společném *přenosovém protokolu*. Tento protokol řídí způsob, jakým jsou přenášena data, ověřuje soudržnost dat a dojde-li k chybě, pošle data znovu. Více přenosových protokolů je zapotřebí z důvodu odlišných podporovaných typů spojení. Například telefonní linky vyžadují „bezpečný“ protokol, který je z hlediska výskytu chyb značně pesimistický, zatímco přenos pomocí protokolu TCP je již ze své podstaty spolehlivý, a proto může používat efektivnější protokol, který vynechává většinu dodatečných detekcí chyb.

Po skončení inicializační fáze začne skutečná přenosová fáze. Oba konce zapnou ovladač vybraného protokolu. Ovladače mohou eventuálně provést inicializační sekvenci, která závisí na typu vybraného protokolu.

Nejprve pošle řídicí program vzdálenému systému všechny soubory, které čekají ve frontě a jejichž priorita dočasného uložení je dostatečně veliká. Když tento přenos dokončí, bude řízený program informován o ukončení přenosu a o tom, že eventuálně může zavěsit. Řízený program nyní může buď souhlasit se zavěšením, nebo může převzít řízení komunikace. Takže de facto dojde k výměně rolí: nyní se program na vzdáleném systému stane řídicím a program na místním hostiteli se stane řízeným. Nový řídicí program nyní pošle své soubory. Po skončení tohoto přenosu souborů si oba programy `uucico` vymění zavěšující řetězce a spojení se přeruší.

Při popisu celého procesu nebudeme zabíhat do větších detailů: chcete-li více informací, nahlédněte buď do zdrojového kódu programu, anebo do nějaké kvalitní knihy, která se zabývá problematikou protokolu UUCP. Kromě toho se někde po Internetu pohybuje poměrně starý

článek od Davida A. Novitze, ve kterém najdete detailní popis protokolu UUCP. Informační bulletin FAQ k protokolu Taylor UUCP také obsahuje některé podrobnosti ohledně způsobu implementace protokolu UUCP. Pravidelně je posílán na adresu **comp.mail.uucp**.

12.2.3 Volby příkazové řádky programu uucico

Tato stať popisuje nejdůležitější volby příkazové řádky programu *uucico*. Kompletní seznam těchto voleb získáte na manuálových stránkách *uucico(1)*.

`-s system` — Zavolá uvedený systém, pokud to není v době, kdy je to zakázáno. V konfiguračních souborech lze určit časové rozmezí, ve kterém je možné se s daným systémem spojit.

`-S system` — Bez jakýchkoliv podmínek zavolá uvedený systém `system`.

`-r1` — Spustí program `uucico` v řídícím režimu. Tento režim je implicitní, použijete-li parametry `-s` nebo `-S`. Samostatná volba `-r1` způsobí, že se program `uucico` pokusí zavolat všechny systémy uvedené v souboru *sys*, pokud toto volání není zakázáno časovým rozmezím pro volání nebo dobou pro opětovný pokus.

`-r0` — Spustí program `uucico` v řízeném režimu. Tento režim je implicitní, pokud nejsou zadány parametry `-s` nebo `-S`. V řízeném režimu se předpokládá, že standardní vstup/výstup bude připojen buď na sériový port, nebo se použije port pro protokol TCP zadaný pomocí volby `-p`.

`-x type`, `-X type` — Zapne ladění zadaného typu. Pomocí seznamu odděleného čárkami může být zadáno několik typů ladění. Platné jsou následující typy: *abnormal*, *chat*, *handshake*, *uucp-proto*, *proto*, *port*, *config*, *spooldir*, *execute*, *incoming*, *outgoing*. Použijete-li klíčové slovo *all*, zapnou se všechny volby. Z důvodu kompatibility s ostatními implementacemi protokolu UUCP může být zadáno místo názvu číslo, které zapíná ladění prvních `n` položek z výše uvedeného seznamu.

Ladící informace budou zapsány do souboru `Debug`, který se nachází v podadresáři `/var/spool/uucp`.

12.3 Konfigurační soubory protokolu UUCP

Na rozdíl od jednoduchých programů pro přenos souborů byl protokol UUCP navržen tak, aby se uměl automaticky postarat o všechny přenosy. Když se vám povede tento protokol správně nastavit, nebude nutný každodenní zásah ze strany správců systému. Požadované konfigurační informace jsou uchovávány v několika *konfiguračních souborech*, které jsou umístěny v adresáři `/usr/lib/uucp`. Většina z těchto souborů je používána pouze při volání směrem z vašeho systému.

12.3.1 Jemný úvod do protokolu Taylor UUCP

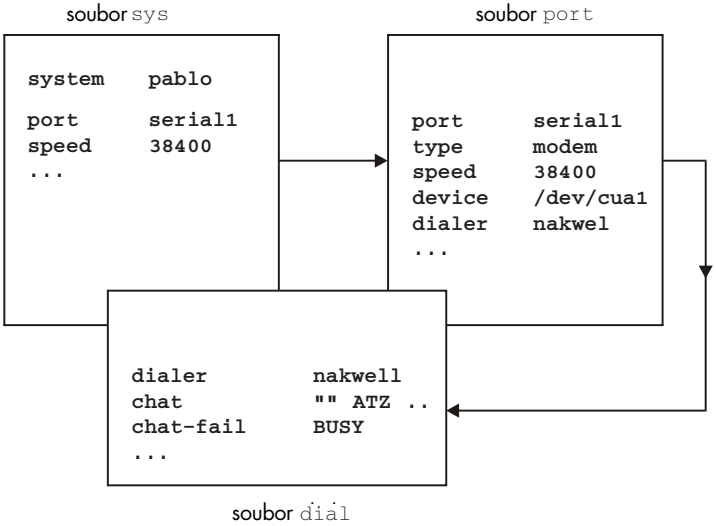
Lze říci, že konfigurace protokolu UUCP je poměrně těžká na pochopení. Jde skutečně o nepříjemnou záležitost a stručný formát konfiguračních souborů vám to rozhodně neulehčí (i když Taylorův formát je v porovnání se staršími formáty protokolů HDB nebo verze 2 poměrně čitelný).

Abyste poznali, jak spolu všechny tyto konfigurační soubory souvisí, představíme vám alespoň nejdůležitější z nich a podíváme se na vzorové položky těchto souborů. Teď se nebudeme zabývat detaily; přesnější vysvětlení najdete v následujících samostatných statích. Chcete-li, aby váš počítač podporoval protokol UUCP, bude nejlepší začít s několika vzorovými soubory, které budete postupně upravovat. Jako vzorové soubory můžete použít níže uvedené příklady nebo soubory, které jsou součástí vaší oblíbené distribuce operačního systému Linux.

Všechny soubory popisované v této stati jsou uloženy v adresáři `/usr/lib/uucp` nebo v některém z jeho podadresářů. Některé distribuce Linuxu obsahují binární soubory protokolu UUCP, které mají zkompilovanou podporu jak pro konfiguraci podle standardu HDB, tak i pro Taylorovu konfiguraci, a pro každý druh konfiguračních souborů používají jiné podadresáře. V adresáři `/usr/lib/uucp` se obvykle nachází soubor `README`.

Aby protokol UUCP správně fungoval, musí být vlastníkem jeho souborů uživatel **uucp**. Některé z těchto souborů obsahují hesla a telefonní čísla, a proto by měly mít nastavena přístupová práva 600.[2]

[2] Všimněte si, ačkoliv většina příkazů protokolu UUCP musí mít nastaveno uid na hodnotu **uucp**, musíte se ujistit, že program `uuchk` nemá přiděleno toto uid. V opačném případě by si mohli uživatelé prohlížet hesla, i když by tato hesla měla nastavena přístupová práva 600.



Obrázek 12.1
Vztahy mezi jednotlivými konfiguračními soubory protokolu Taylor UUC.

Ústředním souborem protokolu UUCP je soubor `/usr/lib/uucp/config`, který se používá pro nastavení obecných parametrů. Nejdůležitějším z těchto parametrů (a momentálně i jediným parametrem) je název vašeho hostitele protokolu UUCP. Ve společnosti Virtual Brewery používají jako bránu protokolu UUCP hostitele **vstout**:

```
# /usr/lib/uucp/config - hlavní konfigurační soubor UUCP
hostname          vstout
```

Dalším důležitým konfiguračním souborem je soubor `sys`. V něm jsou obsaženy veškeré informace týkající se systémů, s nimiž jste propojeni. Tyto informace obsahují název systému a informace o vlastním spojení, například telefonní číslo používané při modemovém spojení. Typický záznam v souboru *sys* pro systém s názvem **pablo** připojený pomocí modemu vypadá asi takto:

```
# /usr/lib/uucp/sys - sousední počítače UUCP
#
system                pablo
time                  Any
phone                 123-456
port                  serial1
speed                 38400
chat                  ogin: vstout ssword: lorca
```

Pole *port* označuje používaný port a pole *time* určuje časy, ve kterých se lze s daným systémem spojit. Pole *chat* popisuje přihlašovací komunikační skripty – souslednost řetězců, které si musí systémy navzájem vyměnit dříve, než se může program `uucico` přihlásit k hostiteli **pablo**. Ke komunikačním skriptům se vrátíme později. Příkaz *port* neoznačuje speciální soubor zařízení, jako je například `/dev/cua1`, ale označuje záznam v souboru *port*. Název portu může mít libovolný název, ale musí odkazovat na korektní záznam v souboru *port*.

Soubor `port` obsahuje informace týkající se vlastního spojení. U modemových spojení tento soubor popisuje speciální soubor používaného zařízení, rozsah podporovaných rychlostí a typ volacího zařízení, které je připojeno k danému portu. Níže uvedený záznam popisuje soubor zařízení `/dev/cua1` (což odpovídá sériovému portu COM 2), ke kterému je připojen modem NakWell, jenž je schopný pracovat s rychlostmi až do 38 400 bps. Název záznamu byl zvolen tak, aby odpovídal názvu portu, který je uveden v souboru `sys`.

```
# /usr/lib/uucp/port - porty UUCP
# /dev/cua1 (COM2)
port                serial1
type                modem
device              /dev/cua1
speed               38400
dialer              nakwell
```

Informace vztahující se k vytáčecímu zařízení jsou uchovávány v dalším speciálním souboru, jehož název jistě uhodnete – `dial`. Pro každý typ vytáčeného zařízení obsahuje tento soubor sekvenci příkazů, které musí být spuštěny, aby se pomocí zadaného čísla vytočil vzdálený systém. Této sekvenci příkazů se také říká komunikační skript. Například záznam pro výše uvedený modem NakWell by vypadal nějak takto:

```
# /usr/lib/uucp/dial - informace o vytáčecích zařízeních
# modemy NakWell
dialer          nakwell
chat            "" ATZ OK ATDT\T CONNECT
```

Řádek začínající polem *chat* definuje komunikační řetězec modemu skládající se z posloupnosti příkazů poslaných na modem a přijatých z modemu. Tyto příkazy modem inicializují a vytáčejí požadované číslo. Sekvenci znaků „\T“ nahradí program `uucico` zadaným telefonním číslem.

Abychom vám mohli poskytnout hrubé schéma způsobu práce programu `uucico` s konfiguračními soubory, budeme předpokládat, že jste spustili na příkazovém řádku následující příkaz:

```
$ uucico -s pablo
```

Program uucico nejprve vyhledá hostitele **pablo** v souboru sys. Ze záznamu hostitele **pablo**, který je umístěn v souboru sys, program uucico zjistí, že ke spojení by měl použít port serial1. Soubor port sdělí programu uucico, že požadovaný port je modemový port a že je k tomuto portu připojen modem NakWell.

Nyní vyhledá program uucico v souboru dial záznam, který popisuje modem NakWell, a když takový záznam nalezne, otevře sériový port /dev/cua1 a spustí vytáčecí komunikační skript. To znamená, že pošle modemu příkaz „ATZ“ a bude čekat na odpověď „OK“ atd. Jakmile najde řetězec „\T“, nahradí ho skutečným telefonním číslem (123-456), které získá ze souboru sys.

Poté, co modem vrátí odpověď „CONNECT“, se uskuteční vlastní spojení a komunikační skript modemu skončí. Nyní se program uucico vrátí k souboru sys a spustí přihlašovací komunikační skript. V našem příkladu bude čekat na výzvu „login:“, potom pošle jméno uživatele (vstout), vyčká na výzvu „password:“ a následně pošle heslo „lorca“.

Po skončení ověřování totožnosti se předpokládá, že vzdálený konec spustí svůj vlastní program uucico. Potom oba tyto programy vstoupí do inicializační fáze, která byla popsána v předcházející stati.

Způsob, jakým na sobě závisí jednotlivé konfigurační soubory, je ukázán na obrázku 12.1.

12.3.2 Co potřebuje znát protokol UUCP

Dříve, než začnete sestavovat konfigurační soubory protokolu UUCP, musíte získat některé informace, které potřebuje protokol UUCP vědět.

Nejprve musíte zjistit, ke kterému sériovému zařízení je připojen váš modem. Porty COM 1 až COM 4 (v DOSu) se obvykle mapují na speciální soubory zařízení /dev/cua0 až /dev/cua3. Většina distribucí, příkladem budiž distribuce Slackware, vytvoří soubor /dev/modem, který je symbolickým odkazem na patřičný soubor zařízení cua* . Tyto distribuce pak nakonfigurují programy kermit, seyon atd. tak, aby používaly obecný soubor modem. V tomto případě byste měli ve své konfiguraci protokolu UUCP také používat soubor /dev/modem.

Soubor modem byste měli používat z toho důvodu, že všechny programy obsluhující modem používají k signalizaci používání daného sériového portu tzv. zamykací soubory. Názvy těchto zamykacích souborů se tvoří kombinací řetězce LCK.. s názvem souboru příslušného zařízení, například LCK..cua1. Pokud by programy používaly pro stejné zařízení odlišné názvy, nebudou schopny zjistit vzájemné zamykací soubory. V důsledku toho dojde k ukončení obou relací, pokud jsou tyto relace spuštěny současně. To však není příliš pravděpodobné, plánujete-li své hovory UUCP s využitím tabulky crontab.

Podrobnosti týkající se nastavení sériových portů získáte v kapitole 4.

Nyní se podíváme, jakou rychlostí komunikuje váš modem s operačním systémem Linux. Tuto rychlost musíte nastavit na maximální očekávanou efektivní přenosovou rychlost. Efektivní přenosová rychlost může být mnohem vyšší, než skutečná fyzická rychlost podporovaná vaším modemem. Například mnoho modemů posílá a přijímá data rychlostí 2 400 bps (bitů za sekundu). Při použití kompresních protokolů, jako je V.42bis, se může skutečná přenosová rychlost vyšplhat až na 9 600 bps.[5]

Má-li být protokol UUCP k něčemu užitečný, budete samozřejmě potřebovat telefonní číslo volaného systému. Taktéž budete potřebovat korektní přihlašovací číslo id a volitelně i heslo.[3]

Dále musíte *přesně* vědět, jak se lze do daného systému přihlásit. Musíte třeba předtím, než se objeví výzva k přihlášení, stisknout klávesu BREAK? Zobrazuje výzva řetězec `login:` nebo `user:`? Tyto údaje jsou důležité pro sestavení *komunikačního skriptu*, který si můžeme představit jako recept, jenž sděluje programu `uucico`, jak se má přihlásit. Pokud tyto údaje neznáte nebo pokud selže i obvyklý komunikační skript, pokuste se dovolat na daný systém pomocí terminálového programu, jako je `kermit` nebo `minicom`, a zapište si přesně ty úkony, které musíte provést.

12.3.3 Pojmenování systému

Stejně jako u sítí na bázi protokolu TCP/IP musí mít váš hostitel i v sítích na bázi protokolu UUCP nějaký název. Pokud hodláte používat protokol UUCP pouze na přenos souborů do nebo ze systémů, se kterými se přímo spojujete, nebo hodláte-li používat tento protokol pouze pro přenos souborů v místní síti, nemusí tento název splňovat žádné standardy.[4]

Používáte-li však protokol UUCP pro příjem pošty nebo news, měli byste přemýšlet o tom, zda by nebylo lepší nechat si zaregistrovat váš název pomocí projektu mapování názvů pro protokol UUCP (UUCP Mapping project). Projekt mapování názvů pro protokol UUCP je popsán v kapitole 13. Dokonce i když jste členem domény, měli byste zvážit použití oficiálního názvu pro protokol UUCP.

Lidé si často zvolí svůj název pro protokol UUCP tak, aby odpovídal první části plně kvalifikovaného názvu domény. Předpokládejme, že adresa domény vašeho systému je **swim.twobirds.com**, potom by název vašeho hostitele pro protokol UUCP měl být **swim**. Pamatujte

3 Pokud si pouze chcete vyzkoušet protokol UUCP, sežeňte si číslo archivního systému, který se nachází v blízkosti vašeho počítače. Poznamenejte si přihlášení do systému a heslo – vzhledem k tomu, že se jedná o veřejné systémy, je možné provádět i anonymní stahování souborů. Jméno uživatele a heslo představuje obvykle kombinace podobných řetězců, jako je **uucp/uucp** nebo **nuucp/uucp**.

4 Jediným omezením je maximální délka názvu 7 znaků. Nesmíme si to plést s hostiteli, kteří mají souborové systémy podporující pouze velmi malý počet znaků v názvu souboru.

5 Pozn. V současnosti jsou rychlosti podstatně vyšší.

na to, že systémy UUCP se navzájem nepoznávají na základě svého názvu. Samozřejmě, že můžete používat název, který nemá nic společného s plně kvalifikovaným názvem domény.

Ujistěte se však, že v poštovních adresách nepoužíváte nekvalifikovaný název systému, pokud ho nemáte zaregistrován jako oficiální název pro protokol UUCP.[6] Pošta poslaná na neregistrovaného hostitele protokolu UUCP by v lepším případě zmizela v nějaké černé díře. Použijete-li název, který již používá nějaký jiný systém, bude pošta směrována na tento systém a způsobí poštmistrům tohoto systému značné problémy.

Balík UUCP implicitně používá název systému nastavený pomocí příkazu `hostname`. Tento název je obecně nastaven ve skriptu `/etc/rc.local`. Pokud se váš název pro protokol UUCP liší od názvu, který jste přiřadili vašemu hostiteli, musíte použít v souboru `config` volbu *hostname*, která sdělí programu `uucico` název pro protokol UUCP. Tento postup bude popsán dále.

12.3.4 Taylorovy konfigurační soubory

Nyní se vrátíme zpět ke konfiguračním souborům. Protokol Taylor UUCP získává potřebné informace z následujících souborů:

`config` Toto je hlavní konfigurační soubor. Zde můžete zadat název vašeho systému pro protokol UUCP.

`sys` Tento soubor popisuje všechny systémy, které znáte. Pro každý systém je v tomto souboru uveden název systému, časy, ve kterých se lze s tímto systémem spojit, volané číslo (pokud nějaké existuje), typ používaného zařízení a způsob, jakým se lze k tomuto systému přihlásit.

`port` Obsahuje položky popisující každý dostupný port, u kterého jsou uvedeny podporovaná rychlost linky a typ používaného vytáčecího zařízení.

`dial` Popisuje vytáčecí zařízení, které se používá pro spojení po telefonní lince.

`dialcode` Obsahuje expanze pro jednotlivé symbolické vytáčecí kódy.

`call` Obsahuje přihlašovací jméno a heslo, které se použije při volání daného systému. Používá se jen zřídka.

`passwd` Obsahuje přihlašovací jména a hesla, která mohou systémy používat při přihlašování. Tento soubor se používá pouze v případě, že si program `uucico` sám provádí ověření hesla.

6 Projekt mapování názvů pro protokol UUCP registruje po celém světě všechny názvy hostitelů pro protokol UUCP a zajišťuje jejich jedinečnost. Chcete-li si zaregistrovat svůj název pro protokol UUCP, zeptejte se správce systému, který vám poskytuje poštu.

Taylorovy konfigurační soubory se zpravidla skládají z řádků obsahujících páry hodnot klíčových slov. Symbol (#) značí komentář, který sahá až do konce příslušného řádku. Chcete-li použít vlastní symbol (#), zadejte ho společně s nahrazujícím symbolem zpětného lomítka.

Pomocí těchto konfiguračních souborů můžete ovládat poměrně velké množství voleb. Nebudeme zde probírat všechny parametry, proto si probereme pouze nejdůležitější z nich. Tyto volby by vám měly pomoci nastavit modemové spojení pomocí protokolu UUCP. Další stati budou popisovat nutné úpravy, které je třeba učinit, budete-li chtít používat protokol UUCP v síti na bázi protokolu TCP/IP nebo společně s přímou sériovou linkou. Kompletní popis je uveden v Texinfo dokumentech, které doprovázejí zdrojový kód protokolu Taylor UUCP.

Pokud si myslíte, že máte svůj systém s protokolem UUCP kompletně nakonfigurován, můžete si svoji konfiguraci zkontrolovat pomocí nástroje `uuchk` (ten najdete v adresáři `/usr/lib/uucp`). Tento nástroj přečte vaše konfigurační soubory a vytiskne podrobnou zprávu obsahující konfigurační hodnoty, které jste použili u každého systému.

12.3.5 Všeobecné konfigurační volby – soubor config

Do tohoto souboru obvykle nebudete zapisovat příliš mnoho vlastností. Většinou ho použijete jen k definici názvu hostitele pro protokol UUCP. Ten bude implicitně používat název, který nastavíte pomocí příkazu `hostname`, ale je užitečné explicitně nastavit vlastní název pro protokol UUCP. Následuje vzorový soubor `config`:

```
# /usr/lib/uucp/config - hlavní konfigurační soubor UUCP
hostname        vstout
```

Samozřejmě existuje mnoho rozmanitých parametrů, které lze nastavit, příkladem může být název dočasného adresáře nebo přístupová práva k anonymní službě UUCP. Definice přístupových práv k anonymní službě UUCP bude popsána v některé z dalších statí.

12.3.6 Jak sdělit protokolu UUCP informace o jiných systémech – soubor sys

Soubor *sys* popisuje systémy, které zná váš počítač. Záznam je uveden klíčovým slovem *system*; řádky mezi dvěma klíčovými slovy *system* definují parametry, které se vztahují k danému systému. Obecně definuje záznam systému takové parametry, jako je například telefonní číslo a přihlašovací komunikační skript.

Parametry uvedené před prvním výskytem klíčového slova *system* definují implicitní hodnoty, které se budou používat pro všechny systémy. V sekci implicitních hodnot budou obvykle uvedeny parametry protokolu a podobné vlastnosti.

Následuje poměrně podrobný popis nejvýznamnějších polí.

Název systému

Příkaz *system* definuje název vzdáleného systému. Musíte zadat korektní název vzdáleného systému, ne pouze přezdívku, kterou jste si vymysleli, protože program `uucico` porovná tento název s názvem, který pošle vzdálený systém v okamžiku vašeho přihlášení.[7]

Každý název systému se může objevit pouze jedenkrát. Pokud chcete pro daný systém používat několik skupin konfigurací (například různá telefonní čísla, která by měl program `uucico` postupně zkoušet vytáčet), můžete zadat tzv. *zástupce*. Zástupci jsou probíráni dále.

Telefonní číslo

Má-li být vzdálený systém dosažitelný pomocí telefonní linky, určuje pole *phone* číslo, které by měl modem vytočit. Toto pole může obsahovat několik symbolů, které posléze zpracuje vytáčecí procedura programu `uucico`. Symbol rovnítka říká programu `uucico`, aby vyčkal na sekundární vytáčecí tón a symbol pomlčky generuje pauzu dlouhou jednu sekundu. Některé telefonní rozvody se totiž ucpou, když neuděláte pauzu mezi číslem, které musíte vytočit, abyste se dostali z vnitřní sítě, a mezi vlastním telefonním číslem.

Nevím, zda existuje nějaký vhodný pojem, který by toto číslo jednoznačně definoval – ale z vlastní zkušenosti víte, že u některých soukromých telefonních rozvodů uvnitř firmy musíte vytočit číslo 0 nebo 9, a teprve potom se dostanete na vnější telefonní linku.

K utajení informací týkajících se daného systému, jako je telefonní číslo jednotlivých uzlů, můžete použít libovolný řetězec. Každý takovýto řetězec bude pomocí souboru `dialcode` převeden na vytáčecí kód. Předpokládejme, že máte následující soubor `dialcode`:

```
# /usr/lib/uucp/dialcode - převod vytáčecích kódů
Bogoham          024881
Coxton           035119
```

Pomocí těchto definic můžete v souboru *sys* používat například telefonní číslo *Bogoham773*, které snad vypadá přehledněji.

7 Starší verze 2 protokolu UUCP nevysílá svůj název při volání vzdáleného systému; avšak novější implementace protokolu UUCP tak činí a stejně tak i protokol Taylor UUCP.

Port a rychlost

Volby *port* a *speed* se používají k výběru zařízení, jenž bude používáno k vytáčení vzdáleného systému a k nastavení maximální rychlosti, na kterou může být dané zařízení nastaveno.[8] Záznam *system* může používat buď jednu z těchto voleb, nebo kombinaci obou voleb. Při vyhledávání vhodného zařízení v souboru *port* vybere systém pouze ty porty, které odpovídají názvu portu a/nebo zvolenému rozsahu rychlostí.

Obvykle by měla stačit pouze volbu *speed*. Máte-li v souboru `port` definováno pouze jedno sériové zařízení, zvolí program `uucico` vždy právě toto zařízení, takže pro ně musíte pouze nastavit požadovanou rychlost. Máte-li ke svým systémům připojeno několik modemů, nebudete muset často určovat konkrétní port, protože když program `uucico` zjistí, že danému požadavku vyhovuje více zařízení, bude postupně zkoušet každé z těchto zařízení, dokud nenajde nějaké nepoužívané zařízení.

Přihlašovací komunikační skript

Ve výše uvedeném výkladu jsme se již setkali s přihlašovacím komunikačním skriptem, který sděluje programu `uucico`, jakým způsobem se má přihlásit ke vzdálenému systému. Přihlašovací komunikační skript se skládá ze seznamu symbolů, které definují očekávané řetězce a řetězce posílané místním procesem `uucico`. Cílem je, aby program `uucico` počkal, dokud vzdálený počítač nepošle výzvu k přihlášení. Místní proces pak vrátí přihlašovací jméno, počká až vzdálený systém pošle výzvu k zadání hesla a nakonec pošle vlastní heslo. Očekávané a posílané řetězce se vzájemně střídají. Program `uucico` automaticky přidá ke všem posílaným řetězcům znak návrat vozíku (\r). Jednoduchý komunikační skript by mohl vypadat asi takto:

```
ogin: vstout ssword:  catch22
```

Všimněte si, že pole očekávaných řetězců neobsahují celé výzvy. To proto, aby přihlášení uspělo i v případě, kdy vzdálený systém vyšle místo řetězce `login:` řetězec `Login:`.

Program `uucico` umožňuje i určitý druh podmíněného spouštění, například když je potřeba program `getty` na vzdáleném počítači znovu inicializovat. V tomto případě můžete k očekávanému řetězci připojit komunikační podřetězce, které jsou vzájemně odděleny pomocí pomlček. Komunikační podřetězce se spustí pouze v případě, že selže hlavní očekávaný řetězec, například při překročení časového limitu. Jedním z případů využití této vlastnosti je obdržení kódu BREAK v situaci, kdy vzdálený systém nepošle výzvu k přihlášení. Následující příklad uvádí všeobecný komunikační skript, který bude fungovat i v případě, že před zobrazením výzvy k přihlášení stisknete klávesu Enter. Řetězec „“ sděluje UUCP, aby na nic nečekal a okamžitě pokračoval v posílání řetězce.

[8] Přenosová rychlost režimu tty linky musí být nastavena přinejmenším na hodnotu maximální přenosové rychlosti.

```
"" \n\r\d\r\n\c ogin:-BREAK-ogin: vstout ssword: catch22
```

V komunikačním skriptu lze použít spoustu speciálních řetězců a únikových znaků. Následuje neúplný seznam korektních symbolů, které mohou být uvedeny v očekávaných řetězcích:

„“ Prázdný řetězec. Tento řetězec sděluje programu `uucico`, aby na nic nečekal a okamžitě pokračoval v posílání následujícího řetězce.

\t Znak tabulátoru.

\r Znak návrat vozíku (CR).

\s Znak mezera. Tento znak potřebujete k vložení mezer do komunikačního řetězce.

\n Znak nový řádek.

\\ Znak zpětné lomítko.

V posílaných řetězcích se mohou kromě výše uvedených symbolů vyskytnout následující únikové znaky a řetězce:

EOT Znak konce přenosu (`^D`).

BREAK Znak přerušení (Break).

\c Potlačí na konci řetězce poslání znaku návrat vozíku.

\d Přeruší posílání na jednu sekundu.

\E Povolí kontrolu opakování. Tato vlastnost vyžaduje, aby program `uucico` počkal s odezvou tak dlouho, dokud nebudou všechny zapsané informace znovu přečteny z příslušného zařízení. Teprve pak se bude pokračovat ve vykonávání komunikačního skriptu. Tato volba je užitečná zejména při použití modemových komunikačních skriptů (se kterými se setkáme v dalším výkladu). Implicitně je kontrola opakování vypnutá.

\e Vypne kontrolu opakování.

\K Totéž jako symbol *BREAK*.

\p Zastaví chod na zlomek sekundy.

Zástupci

Někdy je vhodné mít k dispozici několik záznamů daného systému, například tehdy, je-li systém dosažitelný pomocí několika modemových linek. U protokolu Taylor UUCP je možné pro jeden systém definovat několik záznamů pomocí tzv. *zástupců*.

Záznam zástupce přebírá veškerá nastavení z hlavního záznamu systému a jsou v něm zadány pouze ty hodnoty, které by měly potlačit implicitní hodnoty záznamu systému nebo které by měly přidat hodnoty k tomuto záznamu. Zástupce je oddělen od hlavního záznamu systému řádkem, který obsahuje klíčové slovo *alternate*.

Chcete-li u hostitele **pablo** používat dvě telefonní čísla, měli byste následujícím způsobem upravit záznam v souboru *sys*:

```
system        pablo
phone         123-456
... další záznamy...
alternate
phone         123-455
```

Když nyní budete volat hostitele **pablo**, pokusí se nejprve program *uucico* vytočit číslo 123-456 a pokud toto volání neuspěje, pokusí se vytočit číslo zástupce. Záznam zástupce přebírá veškerá nastavení z hlavního záznamu systému a potlačuje pouze nastavení telefonního čísla.

Omezení času volání

Protokol Taylor UUCP nabízí množství způsobů pro omezení časů, ve kterých je možné uskutečnit hovory se vzdáleným systémem. Tato omezení můžete zavést buď kvůli omezením, která klade vzdálený hostitel na své služby během pracovní doby, nebo proto, abyste se vyhnuli dobám s vysokými tarifními poplatky. Poznamenejte si, že pomocí volby `-S` nebo `-f` je možné v programu `uucico` vždy potlačit omezení času volání.

Protokol Taylor UUCP implicitně zakazuje spojení v jakémkoliv čase, takže *musíte* v souboru *sys* použít volbu, která povolí nějaký časový rozsah volání. Pokud vás nezajímá omezení času volání, můžete zadat do souboru `sys` volbu *time*, které přiřadíte hodnotu *Any*.

Nejjednodušší způsob omezení času nabízí volba *time*, za níž následuje řetězec tvořený poli den a čas. Pole den může nabývat libovolné kombinace hodnot *Mo*, *Tu*, *We*, *Th*, *Fr*, *Sa*, *Su* (Pondělí až Neděle) nebo hodnot *Any* (kdykoliv), *Never* (nikdy) nebo *Wk* (pracovní dny). Pole čas se skládá ze dvou 24hodinových časových hodnot, které jsou odděleny pomlčkou. Tyto hodnoty udávají časový rozsah, ve kterém se mohou hovory uskutečňovat. Kombinace těchto polí musí být zapsána bez bílého místa. Pomocí čárek může být seskupen libovolný počet denních a časových údajů, například:

```
time                MoWe0300-0730,Fr1805-2000
```

Tento příkaz povoluje volání v pondělí a ve středu v době od 03:00 do 07:30 a v pátek v době od 18:05 do 20:00. Když časové pole překračuje půlnoc, konkrétně *Mo1830-0600*, znamená to, že hovory budou povoleny v pondělí v době od půlnoci do 06:00 a v době od 18:30 do půlnoci.

Speciální řetězce *Any* a *Never* fungují přesně tak, jak naznačují: hovory se mohou uskutečnit kdykoliv, resp. nikdy.

Příkaz `time` může obsahovat volitelný druhý argument, který definuje čas v minutách, po jehož uplynutí se má uskutečnit nový pokus. Když selže pokus o spojení, nepovolí program `uucico` po určitou předem zadanou dobu další pokus o spojení se vzdáleným hostitelem. Program `uucico` implicitně používá exponenciální zpětné schéma, při kterém se s každým opakovaným selháním zvyšuje čas do nového pokusu. Zadáte-li například čas opakování pokusu 5 minut, odmítne program `uucico` zavolat vzdálený systém dříve, než 5 minut po posledním neúspěšném volání.

Příkaz `timegrade` vám umožňuje přidělit plánu maximální priority. Předpokládejme například, že v záznamu *system* máte následující příkazy `timegrade`:

```
timegrade           N Wk1900-0700,SaSu
timegrade           C Any
```

Tyto příkazy umožní přenos úkolů s prioritou C nebo vyšší (pošta se obvykle uchovává s prioritou B nebo C) při každém uskutečněném spojení, zatímco news (které se obvykle uchovávají s prioritou N) budou přenášeny pouze v noci a o víkendech.

Stejně jako u příkazu `time` lze zadat i u příkazu `timegrade` volitelný třetí argument, který definuje čas (v minutách) do nového pokusu.

Zde se však může objevit následující námitka ohledně priorit: Zaprvé se volba *timegrade* vztahuje pouze na soubory, které posílají *vaše* systémy; vzdálený systém může i přesto přenášet cokoliv ho napadne. Chcete-li explicitně definovat požadavek na vzdálený systém, aby posílal pouze úkoly, které mají vyšší než zadanou prioritu, použijte k tomu volbu *call-timegrade*; nemáte však žádnou záruku, že se bude vzdálený systém tímto požadavkem řídit.[9]

Podobně nebude ani pole *timegrade* ověřeno v případě, kdy si vzdálený systém sám zavolá. V takovém případě mu budou poslány všechny připravené úkoly. Vzdálený systém může explicitně požádat váš program `uucico`, aby se omezil pouze na určitou prioritu.

9 Jestliže používá vzdálený systém protokol Taylor UUCP, bude se tímto požadavkem řídit.

12.3.7 Jaká zařízení existují – soubor port

Soubor `port` informuje program `uucico` o dostupných portech. Těmito porty mohou být jak modemové porty, tak i ostatní podporované typy portů, jako jsou přímé sériové linky nebo sockety protokolu TCP.

Stejně jako soubor `sys`, je i soubor `port` složen ze samostatných záznamů, které začínají klíčovým slovem *port*, za kterým následuje název daného portu. Tento název můžete použít v souboru `sys` v příkazu *port*. Název nemusí být jedinečný; existuje-li více portů se shodným názvem, bude program `uucico` zkoušet postupně všechny porty, dokud nenajde port, který se v daném okamžiku nepoužívá.

Za příkazem *port* by měl následovat příkaz *type*, který definuje typ popisovaného portu. Korektní typy jsou *modem*, typ *direct* pro přímá spojení a typ *tcp* pro sockety protokolu TCP. Pokud příkaz *port* chybí, bude implicitní typ portu nastaven na *modem*.

V této stati budeme probírat pouze modemové porty; porty pro protokol TCP a přímé linky budou probrány v dalších statích.

U modemových a přímých portů musíte pomocí příkazu *device* zadat volací zařízení. Obvykle je to název speciálního souboru zařízení, který se nachází v adresáři `/dev`, například `/dev/cua1`.[10]

V případě modemového zařízení určuje záznam portu také typ modemu, který je k danému portu připojen. Různé typy modemů musí být nakonfigurovány odlišným způsobem. Dokonce i modemy, které tvrdí, že jsou kompatibilní se standardem Hayes, nemusí být ve skutečnosti navzájem kompatibilní.

Z tohoto důvodu musíte programu `uucico` sdělit, jak má inicializovat daný modem a jak má vytočit požadované telefonní číslo. Protokol Taylor UUCP uchovává popisy všech vytáčecích zařízení v souboru `dial`. Chcete-li použít některé z těchto zařízení, musíte pomocí příkazu *dialer* zadat název požadovaného vytáčecího zařízení.

Někdy budete chtít používat modem odlišným způsobem podle typu volaného systému. Například některé starší modemy nerozumí tomu, když se nějaký vysokorychlostní modem pokouší spojit rychlostí 14 400 bps; tyto modemy linku prostě zavěsí, místo aby se snažily spojit například rychlostí 9 600 bps. Víte-li, že systém **drop** používá právě takovýto typ hloupého modemu, musíte nastavit svůj modem odlišným způsobem. K tomu potřebujete další záz-

[10] Někteří lidé namísto těchto zařízení používají zařízení `ttyS*`, která jsou určena pouze na volání směrem do vašeho systému.

nam v souboru `port`, který bude definovat odlišný typ vytáčecího zařízení. Nyní můžete přiřadit novému portu odlišný název, například *serial1-slow,* a v souboru `sys` použít v záznamu pro systém **drop** příkaz *port*.

Výhodnější je ale rozdělit porty podle podporovaných rychlostí. Dva záznamy portů mohou pro výše popsanou situaci vypadat následovně:

```
# NakWell modem; spojení na vysoké rychlosti
port          serial1                  # jméno portu
type          modem                    # modem
device        /dev/cua1                # COM2
speed         38400                    # podporovaná rychlost
dialer        nakwell                  # vytáčecí zařízení

# NakWell modem; spojení na nízké rychlosti
port          seriál1                  # jméno portu
type          modem                    # modem
device        /dev/cua1                # COM2
speed         9600                     # podporovaná rychlost
dialer        nakwell-slow             # nezkoušej vysokou rychlost
                                         připojení
```

Záznam systému pro systém **drop** přiřadí portu název `serial1`, ale bude požadovat, aby se používala pouze rychlost 9 600 bps. Potom program `uucico` automaticky vybere druhý záznam portu. Všechny ostatní systémy, které mají v záznamu systému uvedenu rychlost 38 400 bps, budou volány s použitím prvního záznamu portu.

12.3.8 Jak vytočit číslo – soubor dial

Soubor `dial` popisuje způsob, jakým se používají různá vytáčecí zařízení. Protokol UUCP většinou raději komunikuje s vytáčecími zařízeními, než s fyzickými modemy, protože dříve bývalo obvyklé jedno (drahé) automatické vytáčecí zařízení, které obsluhovalo sadu modemů. Dnes má většina modemů zabudovanou podporu pro vytáčení čísel, takže rozdíl je poměrně nevýrazný.

Přesto však mohou odlišná vytáčecí zařízení nebo modemy vyžadovat odlišné konfigurace. Každou z těchto konfigurací popíšete v souboru *dial*. Záznamy v tomto souboru začínají příkazem *dialer*, který přiřazuje název vytáčecímu zařízení.

Kromě této položky je nejdůležitější komunikační skript, který se zadává pomocí příkazu `chat`. Stejně jako tomu bylo u přihlašovacího komunikačního skriptu, skládá se i tento skript ze sekvence řetězců, které program `uucico` posílá na vytáčecí zařízení, a z odpovědí, které

očekává z vytáčecího zařízení. Obecně se tento skript používá k inicializaci modemu do nějakého známého stavu a k vytočení požadovaného čísla. Následující příklad záznamu vytáčecího zařízení ukazuje typický komunikační skript pro modemy kompatibilní s modemovým standardem Hayes:

```
# NakWell modem; spojení na vysoké rychlosti
dialer                nakwell  # jméno vytáčecího zařízení
chat                  "" ATZ OK\r ATH1E0Q0 OK\r ATDT\T CONNECT
chat-fail             BUSY
chat-fail             ERROR
chat-fail             NO\sCARRIER
dtr-toggle            true
```

Komunikační skript modemu začíná řetězcem „“, což je prázdný očekávaný řetězec. Program `uucico` proto okamžitě pošle první příkaz (`ATZ`). Příkaz `ATZ` inicializuje modemy, které jsou kompatibilní se standardem *Hayes*. Pak bude `uucico` čekat, dokud modem nepošle odpověď `OK` a následně další příkaz, který vypne místní opakování znaků. Jakmile modem opět pošle odpověď `OK`, pošle program `uucico` příkaz pro vytáčení (`ATDT`). Úniková sekvence `\T` v řetězci bude nahrazena skutečným telefonním číslem, které systém získá z příslušného záznamu v souboru `sys`. Potom bude program `uucico` čekat, dokud modem nepošle odpověď `CONNECT`, která bude signalizovat, že bylo spojení se vzdáleným systémem úspěšné.

Často se stává, že se modemu nepodaří spojení se vzdáleným systémem, například když už vzdálený systém komunikuje s nějakým jiným systémem a linka je obsazená. V takovém případě vrátí modem nějakou chybovou zprávu, která označuje příčinu neúspěchu. Komunikační skripty modemu nejsou schopny detekovat takovéto zprávy, takže program `uucico` bude čekat na očekávaný řetězec tak dlouho, dokud nedojde k překročení časového limitu. Proto se v log-souboru protokolu UUCP objeví místo pravého důvodu pouze neurčitá zpráva „`timed out in chat script`“ (v komunikačním skriptu došlo k překročení časového limitu).

Avšak protokol Taylor UUCP umožní sdělit tyto hlášky programu `uucico`. Lze to provést pomocí příkazu *chat-fail*, který je uveden ve výše uvedeném výpisu. Když program `uucico` detekuje při spuštění komunikačního skriptu modemu řetězec, který odpovídá neúspěšnému provedení komunikačního skriptu, zruší aktuální volání a zapíše do log-souboru protokolu UUCP chybovou zpráva.

Poslední příkaz v příkladu sděluje protokolu UUCP, aby přepnul linku DTR dříve, než se spustí komunikační skript modemu. Většina modemů může být nakonfigurována tak, aby zavěsili telefonní linku a přešli do příkazového režimu v případě, že zjistí změnu linky DTR.[11]

[11] Některé modemy můžete také nakonfigurovat tak, aby provedly reset, pokud zjistí změnu na lince DTR. Některé z těchto modemů však tuto vlastnost nepodporují a mohou občas zavěsit.

12.3.9 Použití protokolu UUCP v sítích na bázi protokolu TCP

Na první pohled se může zdát použití protokolu UUCP pro přenos dat po sítích na bázi protokolu TCP absurdní, ale v zásadě to není špatný nápad, zvláště při přenosu velkého množství dat typu usenetových news. U spojení na bázi protokolu TCP se zpravidla vyměňují news pomocí protokolu NNTP, s jehož pomocí se o články žádá a jsou odesílány individuálně, bez komprese a bez jakékoliv další optimalizace. Ačkoliv tato technika vyhovuje rozsáhlým systémům s několika souběžnými news, není příliš vhodná pro malé systémy, které získávají news přes pomalá spojení, jako je ISDN. Takové systémy budou obvykle chtít kombinovat výhody protokolu TCP s výhodami, které přináší posílání news ve velkých skupinách, které lze zkomprimovat a pak je přenést s menší režií. Standardní způsob přenášení takových skupin spočívá v použití protokolu UUCP po sítích na bázi protokolu TCP.

V souboru `sys` musíte následujícím způsobem definovat systém, který má být volán pomocí protokolu TCP:

```
system          gmu
address         news.groucho.edu
time            Any
port            tcp-conn
chat            ogin: vstout word: clouseau
```

Příkaz *address* udává IP-adresu hostitele nebo jeho plně kvalifikované doménové jméno. Odpovídající záznam v souboru `port` by měl vypadat asi takto:

```
port            tcp-conn
type            tcp
service         540
```

Záznam uvádí, že spojení pomocí protokolu TCP by mělo být použito tehdy, pokud záznam v souboru `sys` obsahuje hodnotu *tcp-conn,* a že program `uucico` by se měl pokusit spojit se vzdáleným hostitelem pomocí sítě TCP na portu 540. Toto je implicitní číslo portu pro službu UUCP. Místo čísla portu můžete v příkazu `service` uvést symbolický název portu. Číslo portu, které odpovídá tomuto názvu portu, bude vyhledáno v souboru `/etc/services`. Běžný název pro službu UUCP je `uucpd`.

12.3.10 Použití přímého spojení

Předpokládejme, že pro spojení vašeho systému **vstout** s hostitelem **tiny** používáte přímou linku. Stejně jako v případě použití modemu budete muset pro daný systém zapsat do souboru `sys` záznam. Příkaz `port` označuje sériový port, ke kterému je hostitel *tiny* připojen.

```
system              tiny
time                Any
port                direct1
speed               38400
chat                ogin: cathcart word: catch22
```

V souboru `port` musíte popsat sériový port, který se používá pro přímé spojení. Příkaz *dialer* není nutný, protože u přímé linky není třeba vytáčet žádná čísla.

```
port                direct1
type                direct
speed               38400
```

12.4 Co umí a neumí protokol UUCP – nastavování přístupových práv

12.4.1 Spouštění příkazů

Úkolem protokolu UUCP je kopírovat soubory z jednoho systému do druhého a žádat spouštění určitých příkazů na vzdálených hostitelích. Samozřejmě, že jako správce systému budete chtít řídit práva, která budete přidělovat ostatním systémům – rozhodně není dobré povolit na svém systému spouštění libovolných příkazů.

Jedinými příkazy, které na vašem počítači povolí protokol Taylor UUCP spouštět ostatním systémům, jsou implicitně příkazy `rmail` a `rnews`, které se obecně používají pro spojení pomocí protokolu UUCP k výměně elektronické pošty a usenetových news. Implicitní vyhledávací cesta, kterou používá program `uuxqt`, se definuje v době sestavování tohoto programu, ale obvykle obsahuje adresáře `/bin`, `/usr/bin` a */usr/local/bin*. Chcete-li pro konkrétní systém změnit množinu příkazů, použijte k tomu klíčové slovo *commands*, které přidáte do souboru *sys*. Podobně může být pomocí příkazu *command-path* změněna i vyhledávací cesta. Chcete-li například, aby mohl systém **pablo** spouštět společně s příkazy `rmail` a `rnews` i příkaz `rsmtp`, upravte soubor `sys` následovně:[12]

```
system              pablo
...
commands            rmail rnews rsmtp
```

[12] Příkaz `rsmtp` se používá k doručování pošty pomocí hromadného protokolu SMTP. To je popsáno v kapitolách o poště.

12.4.2 Přenosy souborů

Protokol Taylor UUCP také dovoluje velice podrobné nastavení přenosu souborů. Na jedné straně můžete zcela zakázat přenosy souborů z konkrétního systému nebo do konkrétního systému. Stačí jen nastavit volbu *request* na hodnotu *no* a vzdálený systém nebude moci ani přijímat soubory z vašeho systému, ani žádné soubory vašemu systému posílat. Podobně můžete pomocí volby *transfer*, kterou nastavíte na hodnotu *no*, zakázat svým uživatelům přenos souborů do vašeho systému nebo z vašeho systému. Implicitně je jak místním uživatelům, tak i uživatelům ze vzdáleného systému povoleno posílat a přijímat soubory.

Kromě toho můžete nastavit adresáře, do kterých nebo ze kterých mohou být soubory kopírovány. Obvykle budete chtít omezit přístup ze vzdálených systémů pouze jen na jednu hierarchii adresářů a zároveň budete chtít umožnit vašim uživatelům posílání souborů ze svého domovského adresáře. Uživatelé mohou obecně přijímat ze vzdáleného systému pouze soubory z veřejného adresáře protokolu UUCP, což je adresář `/var/spool/uucppublic`. Toto je tradiční místo, kde je možné soubory zpřístupnit veřejnosti; je to velmi podobné serverům FTP na Internetu. Na tento adresář se běžně odkazuje pomocí znaku vlnka.

Z toho důvodu poskytuje protokol Taylor UUCP čtyři odlišné příkazy, které se používají ke konfiguraci adresářů pro přijímání a posílání souborů. Jsou to konkrétně příkazy *local-send*, který specifikuje seznam adresářů, z nichž může uživatel stahovat soubory pomocí protokolu UUCP, dále příkaz *local-receive*, který uvádí seznam adresářů, do nichž může uživatel přijímat soubory pomocí protokolu UUCP, a příkazy *remote-send* a *remote-receive*, které se chovají analogicky při požadavku z cizího systému. Uvažte následující příklad:

```
system              pablo
...
local-send          /home
local-receive       /home ~/receive
remote-send         ~ !~/incoming !~/receive
remote-receive      ~/incoming
```

Příkaz *local-send* povoluje uživatelům posílat z vašeho hostitele na hostitele **pablo** libovolné soubory z adresářové struktury pod adresářem `home` a z veřejného adresáře protokolu UUCP. Příkaz *local-receive* povoluje těmto uživatelům přijímat soubory buď do volně dostupného adresáře `receive`, do něhož lze zapisovat a který se nachází pod adresářem `uucppublic`, nebo do libovolného adresáře s možností zápisu, který se nachází v adresářové struktuře pod adresářem `/home`. Příkaz *remote-send* povoluje uživatelům z hostitele **pablo** žádat o soubory z adresáře `/var/spool/uucppublic`, vyjma souborů z adresářových struktur pod ad-

resáři `incoming` nebo `receive`. Tyto vyjmuté adresáře jsou signalizovány programu `uucico` tak, že před jejich názvy je uveden vykřičník. A konečně poslední řádek povoluje uživatelům z hostitele **pablo** přenášet libovolné soubory do adresáře `incoming`.

Jedním z největších problémů souvisejících s přenosem souborů prostřednictvím protokolu UUCP je skutečnost, že lze přijímat soubory pouze do adresářů, do kterých lze zapisovat a které jsou volně dostupné. To může svádět některé uživatele k tomu, aby chystali pastičky na ostatní uživatele. Tomuto ale nelze nijak zabránit, ledaže byste zablokovali veškeré přenosy souborů pomocí protokolu UUCP.

12.4.3 Doručování

Protokol UUCP nabízí mechanismus, pomocí něhož můžete na ostatních systémem spustit přenosy souborů pod svým jménem. Například následující mechanismus umožňuje, aby pro vás hostitel **seci** přijal soubor z hostitele **uchile** a poslal ho do vašeho systému.

```
$ uucp -r seci!uchile!~/find-ls.gz ~/uchile.filez.gz
```

Tato technika, pomocí níž lze předávat úkol přes několik systémů, se nazývá *doručování*. Ve výše uvedeném příkladu může být důvodem použití doručovací techniky například to, že hostitel **seci** má přístup pomocí protokolu UUCP k hostiteli **uchile**, zatímco váš hostitel nemá přístup k tomuto hostiteli. Avšak pokud provozujete systém s protokolem UUCP, budete chtít omezit doručovací službu pouze na několik důvěryhodných hostitelů, kteří vám nevyženou telefonní účet do neúnosných mezí například tím, že by chtěli po vašem hostiteli stáhnout poslední verzi zdrojového kódu balíku X11R6.

Implicitně má protokol Taylor UUCP zablokované veškeré doručování. Chcete-li povolit doručování nějakému konkrétnímu systému, použijte k tomu příkaz *forward*. Tento příkaz udává seznam systémů, ze kterých a na které je možné doručovat úkoly. O doručování úkolů vás může požádat nějaký další systém. Například správce protokolu UUCP na hostiteli **seci** bude chtít přidat do souboru *sys* následující řádky, které povolí hostiteli **pablo** požadovat soubory po hostiteli **uchile**:

```
####################
# pablo
system                pablo
...
forward               uchile
####################
# uchile
```

```
system                  uchile
...
forward-to              pablo
```

Položka *forward-to* je pro hostitele **uchile** nutná, aby jakékoliv soubory poslané z tohoto hostitele byly předány hostiteli **pablo**. V opačném případě by je protokol UUCP zahodil. Tato položka je obměnou příkazu *forward* a hostiteli **uchile** povoluje posílat soubory pouze na hostitele **pablo** přes hostitele **seci**; jakákoliv jiná cesta je nepřípustná.

Chcete-li povolit doručování do libovolného systému, použijte k tomu speciální klíčové slovo *ANY* (vyžadují se velká písmena).

12.5 Nastavení systému pro příchozí volání

Chcete-li nastavit svůj systém pro příchozí volání, musíte u svých sériových portů povolit přihlášení a přizpůsobit některé ze systémových souborů tak, aby poskytovaly účty pro protokol UUCP. To vše bude náplní této stati.

12.5.1 Nastavení programu getty

Pokud chcete používat sériovou linku jako port pro příchozí volání, musíte na tomto portu povolit proces `getty`. Bohužel některé implementace programu `getty` nejsou pro tento účel vhodné, protože budete většinou chtít používat sériový port jak pro volání ze systému, tak pro volání směrem do systému. Proto se musíte ujistit, že používáte takový program `getty`, který je schopen sdílet linku s dalšími programy, jako jsou například programy `uucico` nebo `minicom`. Jeden z programů, který tyto požadavky splňuje, je program `uugetty` z balíku `getty_ps`. Většina linuxových distribucí tento program obsahuje; zkontrolujte, zda máte program `uugetty` ve svém adresáři `/sbin`. Pak je tu ještě jeden program od Gerta Doeringa s názvem `mgetty`, který navíc podporuje příjem faxů. Nejnovější verze těchto programů můžete získat buď v podobě binárních programů, nebo jako zdrojové kódy na adrese **sunsite.unc.edu**.

Vysvětlení odlišností ve způsobu, jakým se tyto programy starají o přihlášení, přesahuje rámec této malé stati; chcete-li více informací, nahlédněte prosím do dokumentu Serial HOWTO od Gregha Hankinse a také do dokumentace, která přichází společně s balíky `getty_ps` a `mgetty`.

12.5.2 Poskytování účtů protokolu UUCP

Dále budete muset nastavit uživatelské účty, které umožní vzdáleným systémům přihlášení do vašeho systému a uskutečnění spojení pomocí protokolu UUCP. Zpravidla poskytnete každému systému, s nímž jste v kontaktu, samostatné přihlašovací jméno. Když budete nastavovat účet pro hostitele **pablo**, přiřadíte mu asi jméno uživatele **Upablo**.

Účty systémů, které k vám volají po sériovém portu, musíte obvykle přidat do systémového souboru hesel `/etc/passwd`. Osvědčilo se vložit všechna přihlášení pomocí protokolu UUCP do speciální skupiny, například s názvem **uuguest**. Domovský adresář daného účtu by měl být nastaven do veřejného dočasného adresáře `/var/spool/uucppublic`; jeho přihlašovací rozhraní musí tvořit program `uucico`.

Jestliže máte nainstalovaný balík se stínovými hesly, můžete k tomuto nastavení použít příkaz `useradd`:

```
# useradd -d /var/spool/uucppublic -G uuguest
  -s /usr/lib/uucp/uucico Upablo
```

Jestliže nepoužíváte balík se stínovými hesly, budete pravděpodobně muset ručně upravit soubor `/etc/passwd`, přidat do něj níže uvedenou řádku, kde hodnoty 5 000 a 150 jsou číselná uid a gid, která jsou přidělena uživateli **Upablo**, resp. skupině **uuguest**.

```
Upablo:x:5000:150:UUCP Account:/var/spool/uucppublic:
  /usr/lib/uucp/uucico
```

Po nainstalování účtu musíte tento účet zaktivovat, to lze provést nastavením hesla pomocí příkazu *passwd*.

Abyste mohli obsluhovat systémy UUCP, které se připojují k vašemu systému pomocí sítě na bázi protokolu TCP, musíte nastavit super-server `inetd`, jenž se bude starat o přicházející spojení na portu *uucp*. To lze provést přidáním následující řádky do souboru `/etc/inetd.conf`:[13]

```
uucp    stream  tcp    nowait  root
  /usr/sbin/tcpd  /usr/lib/uucp/uucico -l
```

Parametr `-l` sděluje programu `uucico`, aby prováděl při přihlášení vlastní proceduru ověření totožnosti. Program `uucico` vyzve, stejně jako standardní program `login`, k zadání přihlašovacího jména a hesla, avšak bude se spoléhat pouze na svoji vlastní soukromou databázi, a nikoliv na soubor `/etc/passwd`. Tento soukromý soubor s hesly se nazývá `/usr/lib/uucp/passwd` a obsahuje páry tvořící přihlašovací jména a hesla:

[13] Pamatujte, že démon `tcpd` běží obvykle v režimu 700, takže ho musíte vyvolat jako uživatel **root** a nikoliv jako uživatel **uucp**, což byste za normálních okolností asi udělali.

```
Upablo  IslaNegra
Ulorca  co'rdoba
```

Samozřejmě tento soubor musí vlastnit uživatel **uucp** a musí mu být přidělena přístupová práva 600.

Jestliže vám připadá tato databáze jako velmi dobrá myšlenka, kterou byste chtěli uplatnit i u standardních sériových přihlašovacích procedur, budete asi zklamáni, když vám řeknu, že to momentálně nepřipadá v úvahu, aniž byste provedli zásadní úpravy. Zaprvé k tomu budete potřebovat protokol Taylor UUCP verze 1.05, protože ten umožňuje, aby program `getty` pomocí volby `-u` podstoupil jméno volajícího uživatele programu `uucico`.[14] Potom musíte obelstít vámi používaný program `getty`, aby namísto běžného vyvolání programu `/bin/login` vyvolal program `uucico`. U balíku `getty_ps` to lze provést přidáním volby *LOGIN* do konfiguračního souboru. Avšak tato volba úplně zakáže interaktivní přihlašování. Na druhou stranu má balík `mgetty` krásnou vlastnost, která vám umožní na základě poskytnutého jména uživatele vyvolat odlišné přihlašovací příkazy. Například můžete programu `mgetty` sdělit, aby používal program `uucico` u všech uživatelů, jejichž přihlašovací jméno začíná písmenem velké U a u všech ostatních jmen uživatelů bude program `mgetty` používat standardní příkaz `login`.

Abyste chránili své uživatele protokolu UUCP od volajících, kteří udávají falešný název systému a kteří sledují veškerou poštu vašich uživatelů, měli byste ke každému záznamu systému v souboru `sys` přidat příkaz *called-login*. Ten je popsán v následující stati Jak se chráníme před podvodníky.

12.5.3 Jak se chráníme před podvodníky

Jeden z největších problémů s protokolem UUCP spočívá v tom, že volající systém může lhát o svém názvu; on sice po přihlášení ohlásí svůj název volanému systému, ale server nemá žádnou možnost, jak by ho mohl ověřit. Takhle se může útočník přihlásit k jeho nebo k jejímu účtu protokolu UUCP a předstírat, že je někdo jiný a může si z tohoto systému vyzvednout poštu. To je zejména problematické v situaci, kdy nabízíte přihlášení pomocí anonymní služby UUCP, jejíž heslo je veřejně dostupné.

Dokud si nejste jisti, že můžete věřit všem systémům volajícím na váš systém, že jsou poctivé, *musíte* se jistit před tímto druhem podvodníků. Lék na tuto nemoc spočívá v tom, že budete po každém systému vyžadovat, aby používal konkrétní přihlašovací jméno, což lze provést uvedením volby *called-login* v souboru `sys`. Vzorový záznam pro daný systém by mohl vypadat asi takto:

[14] Parametr `-u` je také dostupný ve verzi 1.04, avšak nemá přiřazenu žádnou funkci.

```
system           pablo
... obvyklé volby ...
called-login     Upablo
```

Výsledkem je, že kdykoliv se systém přihlásí a bude předstírat, že je hostitel **pablo**, zkontroluje program `uucico`, zda se tento systém přihlásil jako uživatel **Upablo**. Jestliže se volající systém nepřihlásil jako tento uživatel, bude odmítnut a spojení bude zavěšeno. Měli byste si zvyknout přidat volbu *called-login* ke každému záznamu systému, který přidáte do souboru `sys`. Je důležité, abyste toto opatření provedli u *všech* systémů, bez ohledu na to, zda se již s vaším systémem spojily, či nikoliv. U těch systémů, jež vás ještě nikdy nevolaly, byste měli nastavit volbu *called-login* na nějaké úplně umělé jméno uživatele, například na jméno **neverlogsin**.

12.5.4 Buďte paranoidní – sekvenční kontrola hovorů

Další způsob, který může detekovat a odrazit podvodníky, představuje použití sekvenčních kontrol hovorů. Sekvenční kontrola hovorů vám může pomoci s ochranou před vetřelci, kteří si nějakým způsobem opatřili přihlašovací heslo vašeho systému s protokolem UUCP.

Při použití sekvenční kontroly hovorů si oba počítače uchovávají záznamy o počtu v minulosti uskutečněných hovorů. S každým spojením se tento počet zvyšuje. Po procesu přihlášení vyšle volající svůj sekvenční počet hovorů a volaný si tento počet zkontroluje se svým vlastním počtem uskutečněných hovorů. Jestliže si počty neodpovídají, bude pokus o uskutečnění spojení odmítnut. Jestliže je základní počet zvolen jako náhodné číslo, stráví útočníci dlouhou dobu, než se jim povede uhodnout správný sekvenční počet hovorů.

Avšak sekvenční kontrola hovorů toho dokáže ještě více: Dokonce i když se nějaké velmi chytré osobě povede detekovat váš sekvenční počet hovorů společně s vaším heslem, stejně na to přijdete. Když nějaký útočník zavolá na systém UUCP vašeho zásobitele a ukradne vám poštu, zvýší se u vašeho zásobitele sekvenční číslo hovoru o jedničku. Když se v budoucnu budete chtít spojit s vaším zásobitelem a pokusíte-li se přihlásit, vzdálený program `uucico` vás odmítne, protože čísla se v žádném případě nebudou shodovat!

Jestliže máte povolenou sekvenční kontrolu hovorů, měli byste pravidelně sledovat soubory log, zdali se v nich nevyskytují chybové hlášky, jež by mohly naznačovat možné útoky. Jestliže váš systém odmítne sekvenční počet hovorů, který poskytne volající systém, vloží program `uucico` hlášku do log-souboru, která říká něco ve smyslu „`Out of sequence call rejected`“ (odmítnuto z důvodu chybného sekvenčního počtu hovorů). Jestliže je váš systém odmítnut svým zásobitelem, protože sekvenční čísla nejsou synchronní, vloží do souboru log hlášku „`Handshake failed (RBADSEQ)`“ (neuspěla inicializační fáze (RBADSEQ)).

Chcete-li povolit sekvenční kontrolu, musíte přidat do záznamu systému následující řádku:

```
# povolení sekvenční kontroly hovorů
sequence         true
```

Kromě toho musíte vytvořit soubor obsahující vlastní sekvenční čísla. Protokol Taylor UUCP uchovává sekvenční čísla v souboru s názvem `.Sequence`, jenž se nachází v dočasném adresáři vzdáleného hostitele. Vlastníkem tohoto souboru *musí* být uživatel **uucp** a musí mu být přiřazen režim 600 (to znamená, že do něj může zapisovat a z něho může číst pouze uživatel **uucp**). Nejlepší je tento soubor založit s libovolnou, předem dohodnutou počáteční hodnotou. V opačném případě se může útočníkovi povést uhodnout skutečný počet hovorů například tak, že vyzkouší všechny hodnoty menší než číslo 60.

```
# cd /var/spool/uucp/pablo
# echo 94316 > .Sequence
# chmod 600 .Sequence
# chown uucp.uucp .Sequence
```

Samozřejmě, že vzdálený systém musí zároveň povolit sekvenční kontrolu hovorů a ta musí mít nastaven sekvenční počet hovorů na přesně stejnou hodnotu, na jakou ho máte nastaven vy.

12.5.5 Anonymní přístup pomocí protokolu UUCP

Chcete-li poskytovat anonymní přístup pomocí protokolu UUCP, musíte nejprve dříve opsaným způsobem nastavit speciální účet pro tuto službu. Běžně se tomuto účtu přiřazuje stejné přihlašovací jméno a heslo, například **uucp**.

Kromě toho musíte ještě pro neznámé systémy nastavit několik bezpečnostních voleb. Můžete jim například zakázat spouštění jakýchkoliv příkazů na vašem systému. Avšak tyto parametry nemůžete nastavit v záznamu v souboru `sys`, protože příkaz *system* vyžaduje název systému, který vy zatím neznáte. Protokol Taylor UUCP řeší tento problém pomocí dalšího příkazu *unknown*. Příkaz *unknown* lze použít v souboru `config` ke specifikaci libovolného příkazu, který by za normálních okolností byl uveden v záznamu systému:

```
unknown          remote-receive ~/incoming
unknown          remote-send ~/pub
unknown          max-remote-debug none
unknown          command-path /usr/lib/uucp/anon-bin
unknown          commands rmail
```

První dva řádky povolí neznámým systémům stahování souborů pouze z adresářové struktury, která se nachází pod adresářem `pub`, a posílání souborů povolí pouze do adresáře `incoming`, který se nachází pod adresářem `/var/spool/uucppublic`. Další řádek sdělí pro-

gramu `uucico`, aby ignoroval veškeré požadavky ze vzdáleného systému, které by požadovaly zapnutí místního ladění. Poslední dva řádky povolují neznámým systémům spouštět příkaz `rmail`; avšak cesta uvedená u tohoto příkazu sděluje programu `uucico`, aby hledal příkaz `rmail` pouze v soukromém adresáři s názvem `anon-bin`. To vám umožní poskytovat speciální příkaz `rmail`, který bude například umět doručit superuživateli veškerou poštu k prozkoumání. Tento příkaz současně umožní anonymním uživatelům zastihnout správce systému, ale zabrání jim vložit jakoukoliv poštu, která je určena pro jiné systémy.

Abyste povolili anonymní službu UUCP, musíte zadat do souboru `config` minimálně jeden příkaz *unknown*. V opačném případě by program `uucico` odmítl všechny anonymní systémy.

12.6 Nízkoúrovňové protokoly protokolu UUCP

Ke sjednání řízení relace a přenosu souborů se vzdáleným hostitelem používá program `uucico` skupinu standardizovaných zpráv. Tato skupina zpráv se často označuje jako tzv. vysokoúrovňový protokol. Během inicializační fáze a během fáze zavěšování se tyto zprávy posílají jako řetězce. Avšak během přenosové fáze se používá doplňkový nízkoúrovňový protokol, který je většinou pro vyšší úrovně transparentní. To proto, aby se například při použití nespolehlivých linek mohla provádět detekce chyb.

12.6.1 Celkový pohled na protokol

Protože se protokol UUCP používá pro rozdílné typy spojení, jako jsou sériové linky nebo sítě na bázi protokolu TCP nebo dokonce X.25, jsou zapotřebí nízkoúrovňové protokoly. Kromě toho různé implementace protokolu UUCP obsahují odlišné protokoly, které v zásadě provádějí stejné služby.

Protokoly se mohou rozčlenit do dvou kategorií: na protokoly interaktivní a na protokoly používající pakety. Protokoly interaktivní přenášejí soubor jako celek, eventuálně mohou u tohoto souboru vypočítat kontrolní součet. Tento typ nemá žádnou režii, avšak vyžaduje spolehlivé spojení, protože jakákoliv chyba způsobí, že soubor musí být celý poslán znovu. Tyto protokoly se zpravidla používají u spojení na bázi protokolu TCP, avšak jejich použití není vhodné u telefonních linek. I když moderní modemy odvádějí poměrně dobrou práci, co se týká korekce chyb, přesto nejsou perfektní a navíc ani neexistuje žádná detekce chyb mezi vaším počítačem a modemem.

Na druhou stranu protokoly používající pakety rozdělují soubor na několik stejně velikých kousků. Každý paket je posílán a přijímán odděleně, je vypočítáván kontrolní součet a zasílateli je podáváno potvrzení o přijetí. Aby se doprava ještě více zefektivnila, byly vynalezeny protokoly s proměnlivými okny, které připouštějí v libovolném okamžiku omezený počet (ok-

no) nevyřízených potvrzení. To značně sníží množství času, které stráví program `uucico` během přenosu vyčkáváním. Navíc jsou tyto protokoly neefektivní u sítí na bázi protokolu TCP, protože mají poměrně velkou režii v porovnání s interaktivními protokoly.

Rozdíl je i v šířce přenosové cesty. Někdy je po sériovém spojení nemožné poslat osmibitové znaky, například když spojení prochází přes hloupý terminálový server. V takovém případě musí být při posílání znaky z osmibitové sady uvozeny. Když posíláte osmibitové znaky po sedmibitovém spojení, musí tyto znaky projít konverzí, která zdvojnásobí množství přenášených dat, i když toto množství dat může být nakonec určitým způsobem kompenzováno interní hardwarovou kompresí. Linky, jež jsou schopny přenášet libovolné osmibitové znaky, se zpravidla nazývají přímé osmibitové linky. To je případ všech spojení na bázi protokolu TCP, stejně tak je tomu i u většiny modemových spojení.

V protokolu Taylor UUCP verze 1.04 jsou dostupné následující nízkoúrovňové protokoly:

g Toto je nejběžnější protokol a měly by mu rozumět snad všechny programy `uucico`. Protokol *g* umí důkladnou detekci chyb, a proto je vhodný zejména pro nekvalitní telefonní linky. Protokol *g* vyžaduje přímé osmibitové spojení. Je to paketově orientovaný protokol používající techniku proměnlivých oken.

i Toto je obousměrný protokol používající pakety, protokol, který může současně posílat a přijímat soubory. Tento protokol vyžaduje spojení s obousměrným přenosem dat a přímou osmibitovou datovou cestu. Tento protokol zvládá momentálně pouze protokol Taylor UUCP.

t Tento protokol je navržen pro použití u spojení na bázi protokolu TCP nebo u jiných skutečně bezchybných sítí. Používá pakety o velikosti 1 024 bajtů a vyžaduje přímé osmibitové spojení.

e Tento protokol dělá v zásadě totéž, co protokol *t*. Hlavní odlišnost spočívá v tom, že protokol *e* je interaktivní protokol.

f Tento protokol byl navržen pro použití u spolehlivých spojení X.25. Je to interaktivní protokol a očekává sedmibitovou datovou cestu. Osmibitové znaky jsou uvozeny, což může být značně neefektivní.

G Toto je verze protokolu *g* z balíku System V verze 4. Tento protokol také zvládají některé další verze protokolu UUCP.

a Tento protokol je podobný protokolu *ZMODEM*. Vyžaduje osmibitové spojení, avšak uvozuje určité kontrolní znaky, jako je XON a XOFF.

12.6.2 Nastavení přenosového protokolu

Všechny protokoly umožňují určitou změnu velikosti paketů, změnu hodnot překročení časového limitu atp. Ve standardních podmínkách obvykle fungují implicitní hodnoty dobře, avšak ve vaší situaci nemusí být optimální. Například protokol *g* může používat velikosti oken v rozmezí od 1 do 7 a velikosti paketů, jako násobky čísla 2, v rozsahu od 64 do 4 096.[15] Jestliže je vaše telefonní linka většinou rušená tak, že se po ní ztrácí více než 5 procent všech paketů, měli byste asi snížit velikost paketu a zmenšit velikost okna. Na druhou stranu, u velmi kvalitních telefonních linek vám může připadat režie protokolu příliš nehospodárná, protože protokol bude posílat potvrzení ACK u každých 128 bajtů. V tomto případě můžete chtít zvýšit velikost paketu na hodnotu 512 bajtů nebo dokonce na 1 024 bajtů.

Protokol Taylor UUCP poskytuje mechanismus, který může vyhovět vašim potřebám. Všechny tyto parametry lze nastavit pomocí příkazu *protocol-parameter*, jenž se umísťuje do souboru *sys*. Například chcete-li pro komunikaci s hostitelem **pablo** nastavit u protokolu *g* velikost paketu na 512 bajtů, přidejte do souboru *sys* následující řádku:

```
system           pablo
...
protocol-parameter g  packet-size  512
```

Nastavitelné parametry a jejich názvy se liší protokol od protokolu. Chcete-li kompletní seznam těchto parametrů, nahlédněte prosím do dokumentace, která je součástí zdrojového kódu protokolu Taylor UUCP.

12.6.3 Výběr konkrétních protokolů

Ne každá implementace programu `uucico` komunikuje a rozumí každému protokolu, takže během inicializační fáze se musí oba procesy dohodnout na použití společného protokolu. Řídicí program `uucico` nabídne řízenému programu `uucico` seznam podporovaných protokolů (pošle řetězec `Pprotlist`), ze kterého si může řízený program vybrat nějaký protokol.

Na základě typu používaného portu (modem, protokol TCP nebo přímá linka) sestaví program `uucico` implicitní seznam protokolů. U modemových a přímých spojení bude tento seznam zpravidla obsahovat protokoly *i*, *a*, *g*, *G* a *j*. U spojení na bázi protokolu TCP bude seznam obsahovat protokoly *t*, *e*, *i*, *a*, *g*, *G*, *j* a *f*. Tento seznam můžete potlačit pomocí příkazu *protocols*, jenž může být uveden buď v záznamu systému, nebo v záznamu portu. Například v souboru *port* můžete upravit záznam vašeho modemového portu následujícím způsobem:

15 Většina binárních souborů, které jsou součástí standardních distribucí Linuxu, mají nastavenou implicitní velikost okna na hodnotu 7 a velikost paketů na 128 bajtů.

```
port              serial1
...
protocols         igG
```

Tento příkaz bude vyžadovat, aby veškerá přicházející a odcházející spojení na tomto portu používala protokoly *i*, *g* nebo *G*. Jestliže vzdálený systém nepodporuje ani jeden z těchto protokolů, pak bude rozhovor neúspěšný.

12.7 Hledání a odstraňování problémů

Tato stať popisuje to, co by mohlo zlobit u vašeho spojení pomocí protokolu UUCP a dává vám doporučení, kde byste měli hledat chyby. Avšak otázky byly sestaveny přímo z hlavy. Existuje mnohem více věcí, které by mohly dělat problémy.

V každém případě povolte ladění pomocí volby `-xall` a podívejte se na výpis v adresáři `Debug`, který se nachází ve vašem dočasném adresáři. Tento výpis by vám měl rychle pomoci rozpoznat, v čem problém vězí. Taktéž se mi osvědčilo zapnout reproduktor u mého modemu v případě, že jsem se nemohl spojit se vzdáleným systémem. U modemů kompatibilních se standardem Hayes to lze provést přidáním řetězce „`ATL1M1 OK`“ do komunikačního skriptu modemu, jenž se nachází v souboru `dial`.

Nejprve byste měli vždy zkontrolovat, zda jsou správně nastavena všechna práva u souborů. Program `uucico` by měl mít nastaveno uid na **uucp** a všechny soubory v adresářích `/usr/lib/uucp`, `/var/spool/uucp` a `/var/spool/uucppublic` by měly být vlastněny uživatelem **uucp**. V dočasném adresáři existuje také několik skrytých souborů[16], které musí vlastnit uživatel **uucp**.

Program `uucico` vypíše „`Wrong time to call`“: Tato hláška obvykle znamená, že jste buď v záznamu systému v souboru `sys` nezadali příkaz *time*, který určuje, v jakých časech se můžete spojit se vzdáleným systémem, nebo jste v něm zadali takový příkaz `time`, jenž v současné době zakazuje volání. Jestliže není poskytnut žádný časový plán volání, bude program `uucico` předpokládat, že se nedá se systémem nikdy spojit.

Program `uucico` si stěžuje, že daný systém je již zablokován: To znamená, že program `uucico` detekoval pro daný vzdálený systém zamykací soubor v adresáři `/var/spool/uucp`. Zamykací soubor může zůstat z dřívějšího spojení se systémem, které zhavarovalo nebo které bylo zrušeno. Ale je také pravděpodobné, že by mohl existovat ještě jeden proces `uucico`, který se snažil dovolat na vzdálený systém a zůstal viset někde v komunikač-

15 To znamená soubory, jejichž názvy začínají tečkou. Takové soubory nejsou příkazem *ls* standardně zobrazeny.

ním skriptu apod. Jestliže se procesu `uucico` nepodaří úspěšně spojit se vzdáleným systémem, ukončete tento proces vysláním zavěšujícího signálu a odstraňte všechny zamykací soubory, které po tomto procesu zůstaly.

Mohu se spojit se vzdáleným systémem, ale komunikační skript nefunguje správně: Podívejte se na text, jenž obdržíte ze vzdáleného systému. Jestliže je zkomolený, může to naznačovat, že jde o problém s rychlostí. V opačném případě si ověřte, že obdržený text skutečně souhlasí s textem, který očekává váš komunikační skript. Pamatujte si, že komunikační skript vždy začíná očekávaným řetězcem. Jestliže obdržíte výzvu k přihlášení a pošlete svoje uživatelské jméno, ale už nikdy neobdržíte výzvu k zadání hesla, vložte mezi tyto řetězce nějakou prodlevu, nebo dokonce můžete vložit prodlevu mezi jednotlivé znaky. V tomto případě byste mohli být rychlejší, než je váš modem.

Můj modem nevytáčí: Jestliže váš modem neindikuje vzestup linky DTR, když se váš program `uucico` snaží volat směrem ze systému, pravděpodobně jste programu `uucico` nesdělili správné zařízení. Jestliže váš modem rozeznává linku DTR, ověřte si pomocí terminálového programu, že na toto zařízení můžete posílat znaky. Jestliže posílání znaků funguje, zapněte na začátku komunikačního skriptu modemu opakování znaků pomocí direktivy *\E*. Jestliže se ani po zapnutí opakování znaků nebudou během komunikačního skriptu modemu zobrazovat vaše příkazy, zkontrolujte, zda není na váš modem rychlost linky příliš vysoká nebo nízká. Jestliže vidíte opakování znaků, zkontrolujte, zda jste nevypnuli odpovědi modemu nebo zda jste je nenastavili na číselné kódy. Zkontrolujte, zda je vlastní komunikační skript správný. Pamatujte si, že pokud chcete poslat na modem zpětné lomítko, musíte za sebou napsat dvě zpětná lomítka.

Můj modem se pokouší volat, ale nemůže se dostat na vnější telefonní síť: Do telefonního čísla vložte prodlevu. To je zejména užitečné v případě, kdy se snažíte volat z vnitřní telefonní sítě společnosti. Lidé z Evropy, kteří vytáčejí čísla pomocí pulsní volby, by měli zkusit tónovou volbu. V některých zemích teprve nedávno zlepšili telekomunikační společnosti své sítě. V těchto případech by mohla pomoci tónová volba.

V log-souboru je uvedeno, že mám extrémně vysokou ztrátu paketů: To vypadá na problém s rychlostí komunikace. Že by spojení mezi počítačem a modemem bylo příliš pomalé (pamatujte si, že byste toto spojení měli nastavit na nejvyšší možnou efektivní rychlost)? Nebo, že by váš hardware byl tak pomalý a nedokázal by obsloužit včas přerušení? Říká se, že sériový port s čipovou sadou NSC 16550A pracuje docela dobře při rychlosti 38 400 bps; avšak bez vyrovnávací paměti FIFO (například u čipů 16 450) je nejvyšší možná rychlost 9 600 bps. Také byste se měli ujistit, že máte u vaší sériové linky povoleno hardwarové řízení toku dat.

Další pravděpodobnou příčinou by mohlo být to, že nemáte na portu povoleno hardwarové řízení toku dat. Protokol Taylor UUCP verze 1.04 neobsahuje žádné prostředky na nastavení řízení toku dat na RTS/CTS. Toto řízení musíte explicitně povolit v souboru *rc.serial* pomocí následující řádky:

```
$ stty crtscts < /dev/cua3
```

Mohu se přihlásit, avšak inicializační fáze neuspěje: Zde se může vyskytovat spousta problémů. Hodně by vám měly pomoci záznamy uvedené v souboru `log`. Podívejte se na seznam protokolů, které nabízí vzdálený systém (vzdálený systém pošle během inicializační fáze řetězec `Pprotlist`). Možná ani jeden z protokolů nemají oba systémy společný (zvolili jste vůbec nějaký protokol v souboru `sys` nebo `port`?).

Jestliže vzdálený systém pošle řetězec `RLCK`, v tom případě na vzdáleném systému existuje pro váš systém starý zamykací soubor. Jestliže to není z důvodu, že jste již spojeni se vzdáleným systémem na nějaké jiné lince, požádejte o jeho odstranění.

Jestliže vzdálený systém pošle řetězec `RBADSEQ`, v tom případě je na vzdáleném systému zapnuta sekvenční kontrola hovorů, avšak sekvenční počty hovorů si neodpovídají. Jestliže vzdálený systém pošle řetězec `RLOGIN`, nebylo vám povoleno přihlášení pod tímto číslem id.

12.8 Log-soubory

Pokud byl váš balík UUCP sestaven se zapisováním do log-souborů kompatibilních s protokolem Taylor UUCP, budete mít pouze tři globální log-soubory, které budou umístěny v dočasném adresáři. Hlavní log-soubor se nazývá `Log` a obsahuje všechny informace o uskutečněných spojení a o přenesených souborech. Typický výpis z tohoto souboru vypadá asi následovně (po malém přeformátování, aby se vešel na šířku stránky):

```
uucico pablo - (1994-05-28 17:15:01.66 539)
  Calling system pablo (port cua3)
uucico pablo - (1994-05-28 17:15:39.25 539) Login successful
uucico pablo - (1994-05-28 17:15:39.90 539) Handshake successfull
                (protocol 'g' packet size 1024 window 7)
uucico pablo postmaster (1994-05-28 17:15:43.65 539)
  Receiving D.pabloB04aj
uucico pablo postmaster (1994-05-28 17:15:46.51 539)
  Receiving X.pabloX04ai
uucico pablo postmaster (1994-05-28 17:15:48.91 539)
  Receiving D.pabloB04at
```

```
uucico pablo postmaster (1994-05-28 17:15:51.52 539)
  Receiving X.pabloX04as
uucico pablo postmaster (1994-05-28 17:15:54.01 539)
  Receiving D.pabloB04c2
uucico pablo postmaster (1994-05-28 17:15:57.17 539)
  Receiving X.pabloX04c1
uucico pablo - (1994-05-28 17:15:59.05 539)
  Protocol 'g' packets: sent 15,
                  resent 0, received 32
uucico pablo - (1994-05-28 17:15:16.02 539) Call complete
uuxq pablo postmaster (1994-05-28 17:16:11.11 546)
  Executing X.pabloX04ai
                (rmail okir)
uuxq pablo postmaster (1994-05-28 17:16:13.30 546)
  Executing X.pabloX04as
                (rmail okir)
uuxq pablo postmaster (1994-05-28 17:16:13.51 546)
  Executing X.pabloX04c1
                (rmail okir)
```

Další důležitý log-soubor se nazývá `Stats` a jsou v něm vypsány statistiky přenesených souborů. Část ze souboru `Stats`, která odpovídá výše uvedenému přenosu, vypadá asi takto:

```
postmaster pablo (1994-05-28 17:15:44.78)
              received 1714 bytes in 1.802 seconds (851 bytes/sec)
postmaster pablo (1994-05-28 17:15:46.66)
              received 57 bytes in 0.634 seconds (89 bytes/sec)
postmaster pablo (1994-05-28 17:15:49.91)
              received 1898 bytes in 1.599 seconds (1186 bytes/sec)
postmaster pablo (1994-05-28 17:15:51.67)
              received 65 bytes in 0.555 seconds (117 bytes/sec)
postmaster pablo (1994-05-28 17:15:55.71)
              received 3217 bytes in 2.254 seconds (1427 bytes/sec)
postmaster pablo (1994-05-28 17:15:57.31)
              received 65 bytes in 0.590 seconds (110 bytes/sec)
```

Opět byly řádky rozděleny tak, aby se vešly na šířku stránky.

Třetím souborem je soubor `Debug`. Do tohoto souboru se zapisují ladicí informace. Pokud používáte ladění, měli byste se ujistit, že má tento soubor nastavena přístupová práva 600. V závislosti na vámi vybraném ladicím režimu může tento soubor obsahovat přihlašovací jméno a heslo, které používáte při spojení se vzdáleným systémem.

Některé binární soubory protokolu UUCP, které jsou součástí distribucí operačního systému Linux, byly sestaveny se zapisováním do log-souborů, které jsou kompatibilní se standardem HDB. Protokol HDB UUCP používá velké množství log-souborů, které jsou uloženy v adresářové struktuře pod adresářem `/var/spool/uucp/.Log`. Tento adresář obsahuje další tři podadresáře, konkrétně `uucico`, `uuxqt` a `uux`. V nich jsou uloženy záznamy ve formě log-souborů, které generuje každý z uvedených programů. Názvy log-souborů se pro každý systém liší. Tedy výstup z programu *uucico* půjde při spojení se systémem **pablo** do souboru `.Log/uucico/pablo`, zatímco výpis z následně spuštěného příkazu `uuxqt` půjde do souboru `.Log/uuxqt/pablo`. Nicméně řádky uvedené v různých log-souborech jsou totožné s řádky, které jsou uvedeny v log-souborech při použití zapisování, které je kompatibilní s protokolem Taylor UUCP.

Když u zapisování do log-souborů, které vyhovují standardu HDB, povolíte ladicí informace, pak tyto ladicí informace půjdou do adresáře `.Admin`, který se nachází pod adresářem `/var/spool/uucp`. Při hovorech směrem ze systému půjdou tyto informace do souboru `.Admin/audit.local`, zatímco v případě, když někdo volá váš systém, půjde výstup z programu `uucico` do souboru `.Admin/audit`.

Kapitola

13

Elektronická pošta

Elektronická pošta představuje jedno z nejvýznamnějších využití síťových služeb od doby, kdy byla síť vynalezena. Původně to byla jednoduchá služba, která kopírovala soubor z jednoho počítače do druhého, kde ho připojila k souboru *poštovní schránky* příjemce. V zásadě tento proces představuje podstatu elektronické pošty, i když stále rostoucí síť vedla se svými složitými směrovacími požadavky a se svým stále se zvyšujícím počtem zpráv k nutnosti vynalezení propracovanějšího schématu.

Byly vynalezeny různé standardy výměny elektronické pošty. Systémy v síti Internet se drží standardu, který byl uveden v dokumentu RFC 822 a který byl doplněn několika dalšími dokumenty RFC. Tyto dokumenty RFC popisovaly způsob přenášení speciálních znaků apod., který není závislý na počítačové platformě. Mnoho nových nápadů bylo doplněno teprve nedávno a vznikla tzv. „multimediální pošta“, kdy mohou být v poštovních zprávách obsaženy i obrázky a zvuky. Další standard definovala společnost CCITT a nazývá se X.400.

Na platformu Unixu byl přenesen poměrně velký počet programů pro přenos pošty. Jedním z nejznámějších programů je `sendmail`. Byl vyvinut na univerzitě v Berkeley a nyní se používá na velkém počtu platforem. Původním autorem tohoto programu je Eric Allman, který v současné době opět aktivně spolupracuje s týmem, který program `sendmail` vytvořil. V Linuxu jsou dostupné dvě implementace programu `sendmail`, jedna z těchto implementací bude popsána v 15. kapitole. V současnosti se vyvíjí verze 8.9.

Nejvíce používaným poštovním agentem v Linuxu je program `smail-3.1.28`, který napsali Curt Landon Noll a Ronald S. Karr. Tento program je součástí většiny distribucí Linuxu. V následujícím výkladu ho budeme zjednodušeně označovat jako `smail`, i když existují i jiné verze tohoto programu, které jsou naprosto odlišné, ale jimi se zde nebudeme zabývat.

Ve srovnání s programem `sendmail` je program `smail` poměrně mladý. Mají-li se oba programy starat o poštu v malém systému, kde neexistují složité směrovací požadavky, jsou jejich možnosti přibližně stejné. Ovšem u velkých systémů vždy vyhrává program `sendmail`, protože má mnohem pružnější konfigurační schéma.

Oba programy `sendmail` a `smail` podporují skupinu konfiguračních souborů, které si musíte přizpůsobit. Kromě informací, které je nutné zadat, aby vůbec mohl poštovní subsystém běžet (například název místního hostitele), lze nastavovat i množství dalších parametrů. Konfiguračnímu souboru programu `sendmail` je zprvu velmi těžké porozumět. Vypadá podobně, jako kdyby si vaše kočka zdřímla na klávesnici, a packu měla položenou na klávese Shift. Konfigurační soubory programu `smail` jsou strukturovanější a lze jim lépe porozumět než konfiguračnímu souboru programu `sendmail`, ale na druhou stranu nabízí uživateli menší volnost při nastavování chování maileru. U malých systémů UUCP nebo u malých systémů v síti Internet je však práce strávená nastavováním obou programů zhruba rovnocenná.

V této kapitole se budeme zabývat pojmem elektronické pošty a jaké problémy budete muset jako správce řešit. Kapitoly 14 a 15 vám poskytnou instrukce, jak poprvé nastavit programy `smail` a `sendmail`. Informace získané v těchto kapitolách by vám měly stačit ke zprovoznění malých systémů, avšak tyto programy nabízejí mnohem více voleb, takže při konfiguraci nejrůznějších vlastností možná strávíte spoustu hodin.

Na konci této kapitoly si ve zkratce probereme nastavení programu `elm`, což je na mnoha unixových systémem, včetně Linuxu, nejčastěji používaný poštovní agent.

Chcete-li získat více informací o problémech, které se týkají elektronické pošty v Linuxu, nahlédněte prosím do dokumentu Vince Skahana Electronic Mail HOWTO, který je pravidelně posílán na adresu **comp.os.linux.announce**.* Zdrojové distribuce programů `elm`, `sendmail` a `smail` také obsahují velmi rozsáhlou dokumentaci, kde byste měli najít odpovědi na většinu otázek týkajících se jejich nastavení. Hledáte-li obecné informace o elektronické poště, existuje spousta dokumentů RFC, které se tímto tématem zabývají. Dokumenty RFC jsou uvedeny v seznamu literatury na konci této knihy.

13.1 Co je to poštovní zpráva?

Poštovní zpráva se zpravidla skládá z textu zprávy, což je text napsaný odesílatelem, a z dat, které označují příjemce, transportní médium atd. a podobají se adrese na dopisní obálce.

Tato administrativní data spadají do dvou kategorií; v první kategorii jsou zastoupena všechna data vztahující se k transportnímu médiu, jako je adresa zasilatele a adresa příjemce. Z to-

* Poznámka korektora: V současné době je primárním zdrojem linuxových zdrojových textů server `ftp://ftp.kernel.org` V České republice je zrcadlo např. na `ftp://ftp.fi.muni.cz/pub/linux/kernel`

hoto důvodu se tato kategorie nazývá *obálka*. V závislosti na tom, kudy prochází daná zpráva, mohou být data z této kategorie transportním softwarem pozměněna.

Druhou kategorii představují všechna data, která jsou nutná pro manipulaci s poštovní zprávou a která se nevztahují k žádnému transportnímu mechanismu. Příkladem může být řádka s předmětem zprávy, seznam všech příjemců nebo datum zaslání dané zprávy. V mnoha sítích se stalo standardem, že tato kategorie dat je uvedena před vlastní poštovní zprávou a tvoří tzv. *poštovní hlavičku*. Od *textu zprávy* je oddělena prázdným řádkem.[1]

Ve světě Unixu používá většina transportních softwarů formát hlavičky, který byl definován v dokumentu RFC 822. Jeho původním záměrem bylo vytvořit standard pro použití v sítích ARPANET, ale protože byl navržen jako nezávislý na prostředí, byl jednoduše upraven pro použití v dalších sítích, včetně mnoha sítí založených na protokolu UUCP.

Nicméně dokument RFC 822 je pouze největším obecným standardem. Vznikly i novější standardy, a to z důvodu zvyšujících se potřeb, jako je šifrování dat, podpora mezinárodní znakové sady a podpora multimediálního rozšíření pošty (MIME).

Ve všech těchto standardech se hlavička zprávy skládá z několika řádek, které jsou odděleny znakem nový řádek. Řádku tvoří název pole, které začíná ve sloupci jedna, a vlastní pole, které je odděleno dvojtečkou a bílým místem. Formát a sémantika každého pole jsou odlišné a závisí na názvu daného pole. Pole hlavičky může pokračovat i na novém řádku v případě, že následující řádek začíná znakem TAB. Pole mohou být uspořádána v libovolném pořadí.

Typická hlavička poštovní zprávy vypadá asi takto:

```
From brewhq.swb.de!ora.com!andyo Wed Apr 13 00:17:03 1994
Return-Path:
Received: from brewhq.swb.de by monad.swb.de with uucp
         (Smail3.1.28.1 #6) id m0pqqlT-00023aB; Wed, 13 Apr 94 00:17
Received: from ora.com (ruby.ora.com) by brewhq.swb.de with smtp
         (Smail3.1.28.1 #28.6) id ; Tue, 12 Apr 94 2
Received: by ruby.ora.com (8.6.8/8.6.4) id RAA26438; Tue, 12 Apr 94
Date: Tue, 12 Apr 1994 15:56:49 -0400
Message-Id:
From: andyo@ora.com (Andy Oram)
To: okir@monad.swb.de
Subject: Re: Your RPC section
```

[1] K poštovní zprávě se obvykle připojuje podpis neboli signatura, jenž obvykle obsahuje informace o autorovi zprávy společně s nějakým vtipem nebo motem. Podpis je oddělen od vlastního textu zprávy řádkem obsahujícím řetězec „--“.

Všechna potřebná pole hlavičky jsou obvykle vygenerována používaným rozhraním maileru, což může být například `elm`, `pine`, `mush` nebo `mailx`. Avšak některá pole jsou volitelná a může je přidat sám uživatel. Například mailer `elm` vám umožní upravit část hlavičky zprávy. Další pole jsou přidána poštovním transportním softwarem. Následuje seznam obecných polí hlavičky a jejich význam:

`From:` Toto pole obsahuje adresu elektronické pošty odesílatele a může eventuálně obsahovat i jeho „skutečné jméno“. Pro toto pole se používá obrovské množství formátů.

`To:` Toto pole obsahuje adresu elektronické pošty příjemce.

`Subject:` Toto pole obsahuje popis obsahu zprávy vyjádřený několika slovy. Pole `Subject` by mělo obsahovat přinejmenším *účel* dané zprávy.

`Date:` Toto pole obsahuje datum, kdy byla zpráva odeslána.

`Reply-to:` Toto pole určuje adresu, na kterou chce odesílatel směrovat odpověď od příjemce. Pole může být užitečné v případě, že máte několik účtů, ale přitom chcete dostávat poštu pouze na adresu, kterou používáte nejčastěji. Toto pole je volitelné.

`Organization:` Toto pole obsahuje společnost, která vlastní počítač a z něhož pochází daná pošta. Máte-li svůj počítač v soukromém vlastnictví, nechte toto pole buď volné, nebo sem zadejte řetězec „private“, případně nějaký nesmysl. Toto pole je volitelné.

`Message-ID:` Toto pole obsahuje řetězec, který vygeneroval transport pošty v původním systému. Vzhledem k této zprávě je to jedinečný řetězec.

`Received:` Toto pole vloží do hlavičky každý systém, který zpracoval vaši poštu (včetně počítačů odesílatele a příjemce). V něm je uveden název daného systému, číslo id zprávy, čas a datum, kdy daný systém obdržel tuto zprávu, dále od kterého systému tato zpráva pochází a který transportní software byl použit k jejímu doručení. Tyto informace se uvádějí z toho důvodu, abyste mohli sledovat směrování pošty a případně si stěžovat příslušné zodpovědné osobě.

`X-`*anything:* Žádný z poštovních programů by si neměl stěžovat na hlavičku, která začíná řetězcem `X-`. Tento řetězec se používá kvůli implementaci doplňkových vlastností, jež zatím nebyly vloženy do standardu RFC, nebo které do něj ani vloženy nebudou. Tento řetězec používá poštovní konference Linux Activists, kde se například pomocí pole hlavičky `X-Mn-Key:` vybírá požadovaný kanál.

Jedinou výjimku z této struktury představuje úplně první řádek. Ten začíná klíčovým slovem `From`, za kterým následuje místo dvojtečky pouze mezera. Aby se toto pole odlišilo od běžného pole `From:`, označuje se často jako pole `From_`. Toto pole obsahuje směrování, kudy musí daná zpráva projít. Směrování je uvedeno pomocí vykřičníkové notace (bude vysvětleno níže), která je charakteristická pro protokol UUCP. Dále obsahuje čas a datum, kdy byla zpráva přijata posledním počítačem, který ji zpracovával, a volitelně může obsahovat i hostitele, ze kterého byla daná zpráva přijata. Protože je toto pole obnovováno každým systémem, který zpracovává danou zprávu, zahrnuje se někdy do dat obálky.

Pole `From_` se uvádí z důvodu zpětné kompatibility s některými staršími mailery, ale jinak se už příliš nepoužívá. Používají ho pouze poštovní rozhraní uživatelů, která na něj spoléhají při označení začátku zprávy v poštovní schránce uživatele. Abyste se vyhnuli potížím s řádkami uvnitř textu zprávy, které také začínají řetězcem „From“, stalo se standardem uvození každého výskytu tohoto řetězce znakem „>“.

13.2 Jakým způsobem se pošta doručuje?

Obecně vytvoříte poštu pomocí nějakého rozhraní maileru, například pomocí programu `mail` nebo `mailx`; nebo pomocí dokonalejších programů, jako je `elm`, `mush` nebo `pine`. Takový program se nazývá *poštovní uživatelský agent* (*mail user agent*), zkráceně MUA. Když se posílá nějaká poštovní zpráva, předá program tuto zprávu ve většině případů rozhraní dalšího programu, který se postará o její doručení. Tento program se nazývá *poštovní přenosový agent* (*mail transport agent*), zkráceně MTA. U některých systémů existují pro místní a vzdálené doručování různí poštovní přenosoví agenti; na dalších systémech existuje pouze jeden agent. Program pro vzdálené doručení se obvykle nazývá `rmail`, dalším obdobným programem je `lmail` (pokud existuje).

Místní doručování znamená samozřejmě více, než jen pouhé přidání příchozí zprávy do poštovní schránky příjemce. Místní agent MTA bude obvykle rozumět schodovitým odkazům (což je nastavení, kdy místní adresy příjemce odkazují na další adresy) a doručování (což je přesměrování pošty uživatele na nějakou jinou destinaci). Také zprávy, které nemohou být doručeny, musí být *odmítnuty*, což znamená, že je systém musí vrátit odesílateli společně s nějakou chybovou zprávou.

U vzdáleného doručování závisí použitý transportní software na povaze daného spojení. Musí-li být pošta doručena po síti na bázi protokolu TCP/IP, použije se protokol SMTP. Zkratka SMTP znamená jednoduchý poštovní přenosový protokol (Simple Mail Transfer Protocol), který je definován v dokumentu RFC 788 a v RFC 821. Protokol SMTP se obvykle přímo spojí s počítačem příjemce a přenos pošty se sjedná s démonem SMTP, který je spuštěn na vzdálené straně.

V sítích na bázi protokolu UUCP nebývá obvykle pošta doručována přímo, ale je doručena požadovanému hostiteli přes několik zprostředkujících systémů. Posílá-li se zpráva pomocí spojení na bázi protokolu UUCP, spustí agent MTA odesílatele obvykle pomocí příkazu `uux` na doručujících systémech program `rmail` a pošle mu na standardní vstup danou zprávu.

Protože se tento proces provádí samostatně pro každou zprávu, může způsobit jak značné pracovní zatížení na hlavním poštovním serveru, tak i přeplnění dočasné fronty protokolu UUCP se stovkami malých souborů, které zabírají neúměrné množství místa na disku.[2] Proto vám někteří agenti MTA umožní seskupit několik zpráv pro vzdálený systém do jednoho dávkového souboru. Dávkový soubor obsahuje příkazy protokolu SMTP, které by za normálních okolností spustil místní hostitel, kdyby se použilo přímé spojení. Tento postup se označuje jako tzv. protokol BSMTP neboli *dávkový* protokol SMTP. Dávka je potom předána programům *tsmtp* nebo `bsmtp` na vzdáleném systému, které zpracují vstup stejným způsobem, jako kdyby došlo k normálnímu spojení.

13.3 Adresy elektronické pošty

U elektronické pošty se adresa skládá přinejmenším z názvu počítače, na kterém je uložena pošta dané osoby, a z identifikace uživatele. Tou může být buď přihlašovací jméno příjemce,, nebo cokoliv jiného. Jiná poštovní adresní schémata, jako je X.400, používají obecnější množinu „atributů“, které slouží k vyhledání hostitele příjemce na adresářovém serveru X.500.

Způsob, jakým je interpretován název počítače, tj. na jakém systému nakonec skončí vaše zpráva a jak se zkombinuje tento název se jménem uživatele příjemce, značně závisí na typu sítě, jíž jste účastníkem.

Systémy v síti Internet dodržují standard popsaný v RFC 822, který vyžaduje notaci **user@host.domain**, kde **host.domain** je plně kvalifikované doménové jméno daného hostitele. Spojovacím členem je znak „zavináč“ (@). Protože tato notace neobsahuje směrování na cílového hostitele, ale (jedinečný) název hostitele, nazývá se tento typ notace *absolutní* adresa.

V původním prostředí protokolu UUCP se běžně používá forma **path!host!user**, kde cesta **path** popisuje pořadí hostitelů, kterými musí zpráva projít, než dorazí k cílovému hostiteli **host**. Tato forma zápisu se nazývá *vykřičníková* notace podle znaku vykřičníku, který se používá v jejím zápisu. Dnes již mnoho sítí založených na protokolu UUCP adoptovalo standard z dokumentu RFC 822 a tudíž bude rozumět i absolutní adrese.

[2] To proto, že diskový prostor se obvykle alokuje v blocích o velikosti 1 024 bajtů. Takže i zpráva o velikosti 400 bajtů zabere celý 1 KB.

Tyto dva typy adres se příliš dobře neslučují. Předpokládejme adresu ve tvaru **hostA!user@hostB**. Není zcela jasné, zda znak „@“ bude mít přednost před cestou nebo tomu bude naopak. Pošleme zprávu hostiteli **hostB**, který ji pošle uživateli na adrese **hostA!user,** nebo pošleme zprávu hostiteli **hostA**, který ji doručí uživateli na adrese **user@hostB**?

Adresy, v nichž jsou zastoupeny různé typy operátorů adres, se nazývají *hybridní adresy*. Nejznámější z nich vidíte ve výše uvedeném příkladu. Tato adresa je obvykle vyřešena tak, že operátor „@“ dostane přednost před cestou. Ve výše uvedeném příkladu to znamená, že zpráva bude odeslána nejdříve hostiteli **hostB**.

Existuje však způsob, jak zadat směrování, které by vyhovovalo standardu uvedenému v dokumentu RFC 822: zápis **<@hostA,@hostB:user@hostC>** určuje adresu uživatele **user** na hostiteli **hostC**, kde hostitel **hostC** je dosažitelný přes hostitele **hostA** a **hostB** (v uvedeném pořadí). Tento typ adresy se často označuje jako tzv. *směrovací adresa*.

Dále existuje ještě adresový operátor „%“: U adresy **user%hostB@hostA** bude zpráva nejprve poslána hostiteli **hostA**, který nahradí znak procenta vpravo (pouze v našem případě) znakem „@“. Takže nyní budeme mít adresu **user@hostB** a mailer šťastně doručí vaši zprávu hostiteli **hostB**, který ji doručí uživateli **user**. Tento typ adresy se někdy označuje jako tzv. „Ye Olde ARPANET Kludge“ a jeho používání se v současné době snažíme zabránit. Přesto však tento typ adresy i nadále generuje mnoho poštovních přenosových agentů.

Ostatní sítě stále ještě používají odlišné způsoby adresování. Například sítě na bázi DEC používají jako adresový operátor dvě dvojtečky, takže adresa má tvar **host::user**.[3] A konečně standard X.400 používá zcela odlišné schéma, které popisuje příjemce pomocí skupiny párů atribut-hodnota, například pár země a organizace.

V sítích FidoNet je každý uživatel identifikován kódem, například **2:320/204.9**, který se skládá ze čtyř znaků označujících zónu (2 je pro Evropu), síť (320 označuje Paříž a Banlieue), uzel (což je místní hub) a koncový bod (počítač PC individuálního uživatele). Adresy sítě FidoNet mohou být namapovány na standard definovaný v dokumentu RFC 822; výše uvedená adresa může být zapsána jako **Thomas.Quinot@p9.f204.n320.z2.fidonet.org**. Neříkal jsem náhodou, že názvy domén se dají lehce zapamatovat?

Použití odlišných typů adres má určité důsledky, které si popíšeme dále. Avšak v prostředí, kde platí standard definovaný v dokumentu RFC 822, budete používat jen velmi zřídka nějaké jiné typy adres, než je typ absolutních adres, jako je **user@host.domain**.

[3] Když se snažíte dostat z prostředí, kde platí standard definovaný v dokumentem RFC 822, na adresu v síti DEC, můžete k tomu použít formuli **„host::user“@relay**, kde **relay** je název známé brány mezi sítěmi Internet a DEC.

13.4 Jak pracuje směrování pošty?

Proces doručování zprávy hostiteli příjemce se nazývá *směrování*. Kromě nalezení cesty ze systému odesilatele do cílového systému v sobě tento proces zahrnuje i kontrolu chyb a optimalizaci rychlosti a nákladů.

Existuje obrovský rozdíl ve způsobu, jakým se stará o poštu systém na bázi protokolu UUCP a systém v Internetu. V Internetu provede hlavní práci související se směrováním dat na hostitele příjemce (který je známý pomocí své IP-adresy) síťová hladina IP, zatímco v zóně protokolu UUCP musí být směrování provedeno uživatelem nebo ho musí vygenerovat poštovní přenosový agent.

13.4.1 Směrování pošty v Internetu

V Internetu záleží pouze na cílovém hostiteli, zda se vůbec uskuteční nějaké konkrétní směrování pošty. Implicitně je nastaveno přímé doručení zprávy cílovému hostiteli. To se provede tak, že systém vyhledá jeho IP-adresu a skutečné směrování dat ponechá na transportní hladině IP.

Většina systémů bude obvykle chtít směrovat veškerou příchozí poštu na dostupný poštovní server, který je schopen postarat se o veškerou dopravu pošty a který ji potom předá místním hostitelům. Pro svoji místní doménu oznámí systém tuto službu v databázi systému DNS pomocí tzv. záznamu MX. Zkratka MX znamená *poštovní server* (*Mail Exchanger*) a obvykle je v tomto záznamu uvedeno, že si hostitel serveru přeje pracovat jako doručovatel pošty pro všechny počítače v rámci příslušné domény. Záznamy MX lze také použít při správě pošty určené pro hostitele, kteří nejsou připojeni k Internetu, příkladem budiž sítě na bázi protokolu UUCP nebo hostitelé v sítích společností, kteří obsahují důvěrná data.

Záznamy MX mají přiřazenu tzv. *prioritu*. Priorita je udávána celým číslem. Má-li některý hostitel několik poštovních serverů, pokusí se poštovní přenosový agent přenést zprávu na ten poštovní server, který má nejnižší prioritu, a pouze v případě, že na tomto serveru neuspěje, se bude snažit spojit s hostitelem, který má vyšší prioritu. Je-li místní hostitel zároveň poštovním serverem pro cílovou adresu, nesmí doručit zprávu na žádného hostitele uvedeného v záznamu MX, který má vyšší prioritu než má on sám; toto je bezpečný způsob, jak zabránit vytváření poštovních smyček.

Předpokládejme, že například organizace Foobar Inc. chce veškerou poštu spravovat na svém počítači s názvem **mailhub**. Potom bude mít v databázi DNS přibližně následující záznam MX:

```
foobar.com          IN   MX     5   mailhub.foobar.com
```

Tento záznam říká, že na adrese **mailhub.foobar.com** bude poštovní server pro doménu **foobar.com** s prioritou 5. Hostitel, který chce doručit zprávu na adresu **joe@greenhouse.foobar.com**, vyhledá pomocí systému DNS doménu **foobar.com** a zjistí, že záznam MX ukazuje na hostitele **mailhub**. Pokud neexistuje žádný záznam MX s prioritou menší než 5, bude zpráva doručena poštovnímu serveru **mailhub**, který ji posléze odešle na hostitele **greenhouse**.

Výše uvedený příklad je skutečně pouze náčrt toho, jak pracují záznamy MX. Chcete-li více informací o směrování v síti Internet, nahlédněte prosím do dokumentu RFC 974.

13.4.2 Směrování pošty v sítích na bázi protokolu UUCP

Směrování pošty v sítích na bázi protokolu UUCP je mnohem komplikovanější než v síti Internet, protože vlastní transportní software neprovádí žádné směrování. Dříve musela být veškerá pošta adresována pomocí vykřičníkové notace, která uváděla seznam hostitelů, pomocí kterých se doručovala pošta. Jednotliví hostitelé byli odděleni vykřičníkem, za kterým následovalo jméno uživatele. Pokud jste chtěli adresovat dopis uživateli Janet na počítač s názvem **moria**, museli jste zadat cestu **eek!swim!moria!janet**. Tato formule poslala poštu z vašeho hostitele na hostitele **eek**, z něj pak na hostitele **swim** a nakonec dorazila pošta z hostitele **swim** na hostitele **moria**.

Zcela zřejmá nevýhoda této techniky spočívá v tom, že si musíte pamatovat síťovou topologii, rychlá spojení atd. Ale ještě horší věcí je, že změna v uspořádání síťové topologie – například odstranění některých spojení nebo odstranění nějakého hostitele – může způsobit nedoručení zprávy, protože jste nebyli obeznámeni s provedenými změnami. A konečně v případě, že změníte místo svého působení, budete pravděpodobně muset aktualizovat veškerá směrování.

Z jednoho důvodu bylo nutné používat směrování od zdroje. Tím důvodem byla přítomnost nejednoznačných názvů hostitelů. Předpokládejme, že existují dva systémy s názvem hostitele **moria**, jeden systém se nachází ve Spojených státech amerických a druhý ve Francii. Na který systém ale adresa **moria!janet** odkazuje? To bude jasné pouze až tehdy, uvedete-li cestu, pomocí které se lze spojit s hostitelem **moria**.

Prvním krokem, který vedl k odstraňování nejednoznačných názvů hostitelů, bylo založení *projektu mapování názvů pro protokol UUCP* (*The UUCP Mapping Project*). Tento projekt je umístěn na Rutgers University a registruje všechny oficiální názvy protokolu UUCP společně se sousedy UUCP a s jejich geografickým umístěním. Projekt zajišťuje, aby nebyl žádný název hostitele použit dvakrát. Informace získané projektem mapování názvů jsou zveřejňovány jako tzv. *Usenetové mapy* (*Usenet Maps*), které jsou pravidelně distribuovány pomocí Usenetu.[4] Typická položka systému v mapě vypadá (po odstranění komentářů) následovně:

[4] Mapy systémů registrovaných v rámci projektu mapování názvů pro protokol UUCP jsou distribuovány prostřednictvím diskusní skupiny **comp.mail.maps**; ostatní organizace mohou pro své sítě zveřejňovat samostatné mapy.

```
moria
        bert(DAILY/2)
        swim(WEEKLY)
```

Tato položka uvádí, že hostitel **moria** má spojení s hostitelem **bert**, se kterým se spojuje dvakrát denně. Dále má spojení s hostitelem **swim**, s nímž se spojuje každý týden. K formátu souboru s mapami se dále ještě vrátíme.

Pomocí informací o jednotlivých spojeních, které jsou uvedeny v souborech s mapami, je možné automaticky vygenerovat celou cestu z vašeho hostitele k libovolnému cílovému systému. Tyto informace se obvykle ukládají v souboru `paths`, někdy se tento soubor také nazývá *databáze cest*. Předpokládejme, že v souboru map stojí, že s hostitelem **bert** se můžete spojit přes hostitele **ernie**. V tom případě by položka pro hostitele **moria** v souboru `paths`, který byl vygenerován z části mapy, mohla vypadat asi takto:

```
moria           ernie!bert!moria!%s
```

Pokud nyní zadáte cílovou adresu **janet@moria.uucp**, vybere agent MTA výše uvedené směrování a pošle zprávu na hostitele **ernie** a jako adresu na obálce uvede **bert!moria!janet.**

Není ovšem vhodné vytvářet soubor `paths` ze všech usenetových map. V nich jsou obvykle informace poměrně zkreslené a často i zastaralé. Z tohoto důvodu se vytváří soubor `paths` ze všech světových map protokolu UUCP pouze na několika hlavních hostitelích. Většina systémů spravuje pouze informace o systémech, které jsou v jejich okolí, a poštu pro hostitele, které nenajdou ve své databázi, pošlou chytřejším hostitelům, jež mají kompletnější směrovací informace. Toto schéma se nazývá *směrování na chytřejší hostitele* (*smart-host routing*). Hostitelé, kteří udržují pouze jediné poštovní spojení pomocí protokolu UUCP (takoví hostitelé se též nazývají *koncové systémy* (*leaf sites*)), sami neprovádí žádné směrování; plně se spoléhají na svého chytřejšího hostitele.

13.4.3 Kombinace protokolu UUCP se standardem definovaným v dokumentu RFC 822

Zatím je nejlepším lékem na problémy se směrováním pošty v sítích na bázi protokolu UUCP převzetí systému DNS do sítí UUCP. Samozřejmě, že pomocí protokolu UUCP se nemůžete dotazovat jmenného serveru. Některé systémy UUCP však vytvořily malé domény, které vnitřně koordinují směrování pošty. V mapách tyto domény zveřejní pouze jednoho nebo dva hostitele, kteří budou označení jako poštovní brány. Tím pádem nemusí v mapách existovat položka pro každého hostitele, který se nachází v příslušné doméně. Brány se starají o veškerou poštu, která přichází do domény nebo která z dané domény odchází. Směrovací schéma uvnitř domény je pro okolní svět neviditelné.

Tento princip funguje velmi dobře se směrováním na chytřejší hostitele, které jsme si popsali výše. O globální směrovací informace se starají pouze brány; vedlejší hostitelé příslušné domény vystačí pouze s ručně vytvořeným souborem `paths`, který uvádí směrování uvnitř jejich domény a směrování na poštovní server. Dokonce ani poštovní brány nemusí mít informace o směrování na každého hostitele protokolu UUCP, které ve světě existují. Kromě kompletních informací o směrování uvnitř jimi obsluhované domény potřebují mít ve svých databázích pouze směrování na všechny domény. Například níže uvedená položka cesty bude směrovat veškerou poštu určenou pro systémy v doméně **sub.org** na hostitele **smurf**:

```
.sub.org            swim!smurf!%s
```

Veškerá pošta směřující na adresu **claire@jones.sub.org** bude poslána na hostitele **swim** s adresou **smurf!jones!claire**.

Hierarchické uspořádání prostoru s názvy domén umožňuje poštovním serverům kombinovat přesnější směrování s méně přesným směrováním. Například systém ve Francii může mít přesné směrování pro subdomény v rámci domény **fr**, ale veškerou poštu určenou pro hostitele v doméně **us** bude směrovat na nějaký systém ve Spojených státech amerických. V tomto případě směrování na základě domén (takto se tato technika označuje) značně redukuje velikost směrovacích databází a stejně tak i potřebnou administrativní režii.

Avšak hlavním přínosem použití názvu domén v prostředí sítí UUCP je, že shoda se standardem uvedeným v dokumentu RFC 822 umožňuje mezi sítěmi na bázi protokolu UUCP a sítí Internet jednoduchou průchodnost dat. V dnešní době má spousta domén UUCP spojení s internetovou bránou, která se chová jako jejich chytřejší hostitel. Posílání zpráv po Internetu je rychlejší a směrování informací je mnohem spolehlivější, protože hostitelé v Internetu mohou používat místo usenetových map systém DNS.

Aby mohl být systém na bázi protokolu UUCP dosažitelný z Internetu, uvádí obvykle internetové brány záznam MX pro domény na bázi protokolu UUCP (záznamy MX byly popsány výše). Předpokládejme třeba, že hostitel **moria** patří do domény **orcnet.org**. Hostitel **gcc2.groucho.edu** se chová vůči této doméně jako internetová brána. Z toho důvodu použije hostitel **moria** jako svého chytřejšího hostitele adresu hostitele **gcc2**, takže veškeré zprávy pro cizí domény budou doručovány pomocí Internetu. Na druhou stranu hostitel **gcc2** uvede záznam MX pro doménu **orcnet.org** a tím pádem může doručit veškerou příchozí poštu určenou pro systémy v doméně **orcnet** na hostitele **moria**.

Nyní zůstal už jen poslední problém, totiž že transportní programy protokolu UUCP neumí obsloužit plně kvalifikovaná doménová jména. Většina balíků UUCP byla navržena tak, aby zvládla názvy systémů do délky osmi znaků, některé dokonce i méně, a použití nealfanumerických kláves, jako jsou tečky, je u většiny z nich absolutně nepřípustné.

Proto je nutná existence určitého převodu mezi názvy odpovídajícími standardu, který byl definován v dokumentu RFC 822, a názvy hostitelů protokolu UUCP. Způsob, jakým je tento převod prováděn, zcela závisí na typu implementace. Obecný způsob, jak namapovat názvy FQDN na názvy protokolu UUCP, spočívá v použití mapování přímo v souboru cest:

```
moria.orcnet.org    ernie!bert!moria!%s
```

Tato formule vytvoří z adresy, která udává plně kvalifikovaný název domény, čistokrevnou vykřičníkovou notaci pro použití v protokolu UUCP. Některé mailery k tomuto účelu poskytují speciální soubory; například program `sendmail` používá soubor `uucpxtable`.

Někdy se při posílání pošty ze sítě na bázi protokolu UUCP do Internetu vyžaduje zpětná transformace (která se hovorově označuje jako tzv. přiřazení domény). Pokud odesilatel pošty použije jako cílovou adresu plně kvalifikované doménové jméno, lze se tomuto problému vyhnout tak, že se při doručování zprávy chytřejšímu hostiteli neodstraní název domény z adresy na obálce. Nicméně stále ještě existují systémy, které nejsou součástí žádné domény. I takovéto systémy lze rozčlenit do domén, což se provede tak, že se k nim připojí fiktivní doména **uucp**.

13.5 Formát souborů map a cest

Databáze cest poskytuje v sítích na bázi protokolu UUCP hlavní informace o směrování. Typická položka vypadá asi takto (název systému je od cesty oddělen tabulátory):

```
moria.orcnet.org  ernie!bert!moria!%s
moria              ernie!bert!moria!%s
```

Tento záznam uvádí, že každá zpráva určená pro hostitele **moria** bude doručena přes hostitele **ernie** a **bert**. Je zde zadán jak plně kvalifikovaný název, tak i název pro protokol UUCP. Oba názvy jsou zde uvedeny pro případ, že by mailer neměl samostatný způsob pro mapování mezi těmito dvěma jmennými prostory.

Pokud chcete směrovat všechny zprávy určené pro hostitele v rámci nějaké domény na jejich poštovní brány, můžete v databázi cest zadat také cestu, která bude mít jako cíl název domény, před nímž bude uvedena tečka. Pokud mohou být například hostitelé v doméně **sub.org** dosažitelní přes poštovní bránu **swim!smurf**, mohla by položka v databázi cest vypadat následovně:

```
.sub.org        swim!smurf!%s
```

Vytvoření souboru cest je přijatelné pouze v případě, že provozujete systém, který neobsahuje příliš mnoho informací o směrování. Provádíte-li směrování pro velké množství hostitelů, bude lepší k tomuto účelu použít příkaz `pathalias`, který umí vytvořit soubor cest ze souborů map. Mapy lze spravovat daleko lépe, protože konkrétní systém lze přidat nebo odstranit tak, že v mapě upravíte jeho položku a necháte znovu vytvořit soubor map. I když se mapy zveřejňované usenetovým projektem mapování názvů zatím ke směrování moc nepoužívají, mohou menší sítě na bázi protokolu UUCP poskytovat informace o směrování pomocí svých vlastních skupin map.

Soubor map se skládá především ze seznamu systémů, ve kterém má každý systém uveden seznam systémů, s nimiž se může spojit nebo které se mohou spojit s ním. Název systému začíná ve sloupci jedna a za ním následuje seznam jednotlivých spojení, která jsou vzájemně oddělena čárkami. Seznam může pokračovat i na dalších řádcích v případě, že následující řádek začíná znakem tabulátor. Každý záznam o spojení je tvořen názvem systému, za kterým je v hranatých závorkách uvedena jeho režie. Režie představuje aritmetický výraz složený z čísel a symbolické režie. Řádky, začínající znakem (#), jsou ignorovány.

Jako příklad uvažujme hostitele **moria**, který se spojuje dvakrát denně s hostitelem **swim.twobirds.com** a jedenkrát týdně s hostitelem **bert.sesame.com**. Kromě toho pro spojení s hostitelem **bert** používá pouze pomalý modem s rychlostí 2 400 bps. Hostitel **moria** by měl v souboru map zveřejnit následující položku:

```
moria.orcnet.org
        bert.sesame.com(DAILY/2),
        swim.twobirds.com(WEEKLY+LOW)

moria.orcnet.org = moria
```

Poslední řádek říká, že hostitel **moria** může být rozpoznán i na základě svého názvu pro protokol UUCP. Všimněte si, že musí být uvedena režie *DAILY/2*, protože volání dvakrát denně ve skutečnosti snižuje režii tohoto spojení na polovinu.

Při použití takovýchto souborů map je schopen příkaz `pathalias` vypočítat optimální směrování na libovolný cílový systém, který je uveden v souboru cest. Dále je schopen z těchto souborů vytvořit databázi cest, která může být poté používána pro směrování do těchto systémů.

Příkaz `pathalias` poskytuje množství dalších vlastností, například skrytí systému (tj. že systémy mohou být přístupné pouze pomocí brány) atd. Viz manuálové stránky příkazu `pathalias`, kde naleznete více detailů a dále i kompletní seznam režií jednotlivých spojení.

Komentáře v souboru map obvykle obsahují doplňkové informace o systémech, které jsou v tomto souboru popsány. Tyto komentáře musí odpovídat pevně stanovenému formátu, takže je lze získat z map. Například program `uuwho` používá k zobrazení těchto informací, které jsou mimochodem naformátovány moc hezky, databázi vytvořenou ze souborů map.

Pokud svůj systém zaregistrujete u organizace, která distribuuje svým členům soubory map, budete obvykle muset v mapě vyplnit přibližně takovýto záznam.

Následuje vzorová položka mapy (de facto jde o položku, která popisuje můj systém):

```
#N       monad, monad.swb.de, monad.swb.sub.org
#S       AT 486DX50; Linux 0.99
#O       private
#C       Olaf Kirch
#E       okir@monad.swb.de
#P       Kattreinstr. 38, D-64295 Darmstadt, FRG
#L       49 52 03 N / 08 38 40 E
#U       brewhq
#W       okir@monad.swb.de (Olaf Kirch); Sun Jul 25 16:59:32 MET DST
#
monad    brewhq(DAILY/2)
# Domains
monad = monad.swb.de
monad = monad.swb.sub.org
```

Bílé místo následující za prvními dvěma znaky je znak tabulátoru. Význam většiny polí je zřejmý; detailní popis obdržíte od jakékoliv domény, u které se zaregistrujete. Největší zábavou je rozluštit význam pole *L*: To uvádí vaše geografické umístění ve formátu zeměpisná šířka / zeměpisná délka a používá se při vykreslování postscriptových map, které ukazují všechny systémy v rámci dané země nebo všechny systémy na světě.[5]

13.6 Konfigurace programu elm

Zkratka `elm` znamená „elektronická pošta“ (electronic mail) a tento program je jedním z nejvýstižněji pojmenovaných unixových nástrojů. Program `elm` poskytuje celoobrazovkové rozhraní s propracovanou nápovědou. Nebudeme se zde zabývat způsobem jeho použití, ale zdržíme se pouze u jeho konfiguračních voleb.

5 Tyto mapy jsou pravidelně posílány na adresu **news.lists.ps-maps**. Předem vás varuji, jsou poměrně objemné!

Teoreticky můžete provozovat program `elm` bez jakékoliv konfigurace a přitom bude vše pracovat správně – kéž byste byli šťastní. Existuje však několik voleb, které je potřeba nastavit, i když budou potřeba jen při určitých událostech.

Při startu si program `elm` přečte několik konfiguračních proměnných ze souboru `elm.rc`, který se nachází v adresáři `/usr/lib/elm`. Potom se pokusí přečíst z vašeho domovského adresáře soubor `.elm/elmrc`. Tento soubor si obvykle nebudete vytvářet sami. Program ho vytvoří za vás, když z nabídky options vyberete volbu „save options“.

Sada voleb použitá v souboru `elmrc` je také dostupná v globálním souboru `elm.rc`. Většina nastavení uvedená v soukromém souboru `elmrc` potlačí nastavení uvedená v globálním souboru.

13.6.1 Volby v globálním souboru programu elm

V globálním souboru `elm.rc` musíte nastavit volby, které se týkají názvu vašeho hostitele. Například ve společnosti Virtual Brewery by mohl soubor pro hostitele **vlager** obsahovat následující informace:

```
#
# Jméno hostitele
hostname = vlager
#
# Jméno domény
hostdomain = .vbrew.com
#
# Plně kvalifikované doménové jméno
hostfullname = vlager.vbrew.com
```

Tyto volby poskytují programu `elm` informace o názvu místního hostitele. I když se tyto informace používají jen zřídka, měli byste je mít nastaveny. Poznamenejte si, že tyto informace mají význam pouze v případě, že je uvedete v globálním konfiguračním souboru; nacházejí-li se ve vašem soukromém souboru `elmrc`, budou ignorovány.

13.6.2 Mezinárodní znakové sady

V minulosti byly navrženy doplňky standardu definovaného v dokumentu RFC 822, které by umožnily podporu různých typů zpráv, například prostý text, binární soubory, postscriptové soubory atd. Skupina standardů a dokumentů RFC, které vyhovují těmto aspektům, se obecně označují zkratkou MIME (Multipurpose Internet Mail Extensions). Kromě jiného tyto

standardy umožňují příjemci zjistit, zda byla při psaní zprávy použita jiná znaková sada než standardní znaková sada ASCII, například francouzské akcenty nebo německé přehlásky. V určitých mezích tyto standardy podporuje i program `elm`.

Znaková sada, kterou vnitřně používá operační systém Linux, se obvykle označuje jako ISO-8859-1, což je název standardu, který tato znaková sada splňuje. Tento standard je také známý pod označením Latin-1. Všechny zprávy používající znaky z této znakové sady by měly mít ve své hlavičce následující řádek:

```
Content-Type:  text/plin; charset=iso-8859-1
```

Přijímací systém by měl toto pole rozeznat a při zobrazování zprávy by měl provést příslušná opatření. Pro zprávy typu *text/plain* (prostý text) je implicitně nastaveno pole znakové sady *charset* na hodnotu *us-ascii*.

Aby bylo možné zobrazit zprávy napsané v jiné znakové sadě, než je znaková sada ASCII, musí program `elm` vědět, jak má tyto znaky zobrazit. Pokud program `elm` obdrží zprávu, ve které má pole *charset* jinou hodnotu než *us-ascii* (nebo při zachování stejné znakové sady bude nastaven jiný typ obsahu zprávy než *text/plain*), pokusí se implicitně zobrazit zprávu pomocí příkazu s názvem `metamail`. Zprávy vyžadující pro zobrazení program `metamail` mají na obrazovce přehledu zobrazeno v prvním sloupci písmeno ‘`M`‘.

Protože je implicitní znakovou sadou operačního systému Linux znaková sada ISO-8859-1, není nutné pro zobrazení zpráv vytvořených pomocí této znakové sady používat program `metamail`. Je-li programu `elm` sděleno, že zobrazení odpovídá standardu ISO-8859-1, pak se program `metamail` nepoužije, ale zpráva bude zobrazena přímo. To zajistíte nastavením následující volby v globálním souboru `elm.rc`:

```
displaycharset = iso-8859-1
```

Pamatujte, že tuto volbu byste měli nastavit i v případě, že neplánujete posílání nebo přijímání zpráv, které obsahují jiné znaky než definuje standard ASCII. Tato volba se uvádí proto, že lidé posílající takovýto typ zpráv obvykle nakonfigurují svůj mailer tak, aby vložil do hlavičky pošty patřičné pole `Content-Type:`, a to bez ohledu na skutečnost, zda je či není daná zpráva napsána ve znakové sadě ASCII.

Ale pouze nastavení této volby v souboru `elm.rc` nestačí. Problém spočívá v tom, že při zobrazování zprávy pomocí vestavěného prohlížeče zavolá program `elm` pro každý zobrazovaný znak příslušnou funkci knihovny, aby zjistil, zda daný znak je či není zobrazitelný. Implicitně bude tato funkce považovat za zobrazitelné pouze znaky splňující standard ASCII a všechny ostatní znaky zobrazí jako „`^?`“. Tento problém lze vyřešit tak, že nastavíme pro-

měnnou prostředí *LC_CTYPE* na hodnotu *ISO-8859-1*. Tato proměnná sdělí knihovně, aby považovala za zobrazitelné všechny znaky ze znakové sady Latin-1. Podpora této i dalších vlastností je dostupná od verze knihovny *libc-4.5.8*.

Při posílání zpráv, které obsahují speciální znaky ze znakové sady ISO-8859-1, byste se měli ujistit, že jste v souboru `elm.rc` nastavili ještě další dvě proměnné:

```
charset = iso-8859-1
textencoding = 8bit
```

Tyto volby způsobí, že program `elm` uvede v hlavičce pošty, že příslušná zpráva byla vytvořena pomocí znakové sady ISO-8859-1 a že bude poslána ve formě 8bitových hodnot (implicitně se veškeré znaky překládají na 7bitové hodnoty).

Samozřejmě, že jakoukoliv z těchto voleb lze místo v globálním konfiguračním souboru uvést v soukromém souboru `elmrc`.

Kapitola

14

Připravení a spuštění programu smail

Tato kapitola je rychlým úvodem do problematiky nastavení programu `smail` a obsahuje také přehled funkcí, které tento program nabízí. I když je program `smail` ve svém chování značně kompatibilní s programem `sendmail`, jsou jejich konfigurační soubory naprosto odlišné.

Hlavním konfiguračním souborem je `/usr/lib/smail/config`. Tento soubor musíte vždy upravit tak, aby obsahoval hodnoty charakteristické pro váš systém. Je-li váš systém pouze koncovým systémem protokolu UUCP, pak nebudete mít moc práce. Ke konfiguraci směrování a transportních voleb lze použít i další soubory, o nichž se také krátce zmíníme.

Program `smail` implicitně ihned zpracuje a doručí veškerou příchozí poštu. Jestliže máte poměrně značné dopravní zatížení, můžete sdělit programu `smail`, aby seskupoval veškeré zprávy do tzv. *fronty*, kterou bude potom zpracovávat pouze v pravidelných intervalech.

Spravujete-li poštu v prostředí sítě na bázi protokolu TCP/IP, běží často program `smail` v režimu démona: při zavádění systému je spuštěn ze skriptu `rc.inet2` a přesune se na pozadí, kde čeká na příchozí spojení TCP na portu pro protokol SMTP (což je obvykle port číslo 25). To je velmi prospěšné v případě, kdy očekáváte značné množství pošty, protože program `smail` se nebude zvlášť spouštět při každém příchozím spojení. Alternativní možnost představuje přenechání správy portu pro protokol SMTP super serveru `inetd`, který při jakémkoliv výskytu příchozího spojení spustí program `smail`.

Program `smail` obsahuje množství příznaků, které ovlivňují jeho chování; kdybychom je zde všechny detailně popsali, stejně by vám to asi moc nepomohlo. Naštěstí program `smail` podporuje množství standardních operačních režimů, které jsou povoleny, pokud ho vyvoláte pomocí speciálního názvu příkazu, například pomocí příkazu `rmail` nebo `smtpd`. Tyto přezdívky obvykle představují symbolické odkazy na vlastní binární kód programu `smail`. S většinou z nich se setkáme, když se budeme bavit o různých vlastnostech programu `smail`.

V každém případě by měly existovat dva symbolické odkazy na program `smail`; konkrétně jsou to příkazy `/usr/bin/rmail` a `/usr/sbin/sendmail`.[1] Když sestavíte a pošlete poštovní zprávu pomocí uživatelského agenta, například pomocí programu `elm`, bude zpráva předána programu `rmail`, který ji doručí. Seznam příjemců je předáván na příkazové řádce. To samé se děje s příchozí poštou při použití protokolu UUCP. Avšak některé verze programu `elm` spustí místo programu `rmail` program `/usr/sbin/sendmail`, takže budete potřebovat oba tyto programy. Máte-li například uložen program `smail` v adresáři `/usr/local/bin`, napište na příkazovém řádku následující příkazy:

```
# ln -s /usr/local/bin/smail /usr/bin/rmail
# ln -s /usr/local/bin/smail /usr/sbin/sendmail
```

Chcete-li se při konfiguraci programu `smail` pouštět do složitějších úkolů, nahlédněte do manuálových stránek programů `smail`*(1)* a `smail`*(5)*. Pokud nejsou tyto informace součástí vaší oblíbené distribuce operačního systému Linux, můžete je získat ze zdrojového kódu programu `smail`.

14.1 Nastavení protokolu UUCP

V případě použití programu `smail` pouze v prostředí sítí na bázi protokolu UUCP je základní instalace poměrně jednoduchá. Nejprve se musíte ujistit, že máte dva symbolické odkazy na výše zmíněné příkazy `rmail` a `sendmail`. Očekáváte-li z ostatních systémů příjem dávkového protokolu SMTP, musíte mít navíc spojení mezi příkazem `rsmtp` a programem `smail`.

V distribuci programu `smail` od Vince Skahana najdete vzorový konfigurační soubor. Jeho název je `config.sample` a je umístěn v adresáři `/usr/lib/smail`. Musíte ho zkopírovat do souboru s názvem *config* a upravit ho tak, aby obsahoval hodnoty charakteristické pro váš systém.

Předpokládejme, že váš systém se nazývá **swim.twobirds.com** a je registrován v mapě protokolu UUCP pod názvem **swim**. Váš chytřejší (smart) hostitel je hostitel **ulysses**. V takovém případě by měl váš soubor `config` vypadat asi takto:

```
#
# Naše doménová jména
visible domain=two.birds:uucp
#
# Naše jméno pro odchozí poštu
```

[1] Toto je nové standardní umístění programu `sendmail`, které vyhovuje standardu linuxového souborového systému (Linux File System Standard). Další běžné umístění je v adresáři `/usr/lib`.

```
visible name=swim.twobirds.com
#
# UUCP jméno
uucp name=swim.twobirds.com
#
# Náš chytřejší hostitel
smart host=ulysses
```

První příkaz sděluje programu `smail` domény, ke kterým patří váš systém. Zde zadejte názvy domén, které budou navzájem odděleny dvojtečkami. Je-li název vašeho systému registrován v mapě protokolu UUCP, měli byste sem přidat i doménu **uucp**. V případě výskytu poštovní zprávy si program `smail` zjistí pomocí systémového volání *hostname(2)* název vašeho hostitele, pak porovná adresu příjemce s názvem hostitele, přitom bude postupně zkoušet všechny názvy uvedené v tomto seznamu. Jestliže se bude adresa shodovat s některým z těchto názvů nebo s některým z nekvalifikovaných názvů hostitelů, bude považovat příjemce za místního a program `smail` se pokusí doručit zprávu místnímu hostiteli. V opačném případě bude příjemce považován za vzdáleného a program `smail` se pokusí doručit zprávu cílovému hostiteli.

Volba *visible_name* by měla uvádět jeden plně kvalifikovaný název domény vašeho systému, který chcete použít u odcházející pošty. Tento název je používán pro veškerou odcházející poštu při generování adresy zasilatele. Musíte se ujistit, že používáte název, u kterého program `smail` pozná, že odkazuje na místního hostitele (tj. na název hostitele uvnitř některé z domén, které jsou vypsány ve volbě *visible_domain*). V opačném případě by odpovědi na vaši poštu nenašly správnou cestu k vašemu systému.

Poslední příkaz nastavuje cestu, která se používá při směrování na chytřejšího hostitele (bylo popsáno ve stati 13.4). U tohoto vzorového nastavení doručí program `smail` chytřejšímu hostiteli veškerou poštu adresovanou na vzdáleného hostitele. Cesta zadaná ve volbě *smart_path* bude použitá pro směrování na chytřejšího hostitele. Protože budou zprávy doručeny pomocí protokolu UUCP, musí volba udávat systém, který zná váš software UUCP. Viz 12. kapitola, kde najdete informace o tom, jak zveřejnit název systému protokolu UUCP.

Ve výše uvedeném souboru se vyskytuje ještě jedna volba, kterou jsme zatím nevysvětlili; tou je *uucp_name*. Důvod pro použití této volby je následující: Program `smail` implicitně používá hodnotu vrácenou příkazem *hostname(2)*. Příkladem může být například návratová cesta, která je v hlavičce zprávy uvedena na řádku *From_*. Pokud váš název hostitele není zaregistrován pomocí projektu mapování názvů pro protokol UUCP, měli byste sdělit programu *smail*, aby místo tohoto názvu hostitele používal vaše plně kvalifikované doménové jméno.[2] To lze provést pomocí volby *uucp_name*, kterou přidáte do souboru `config`.

V adresáři `/usr/lib/smail` se nachází ještě další soubor s názvem `paths.sample`. Tento soubor uvádí vzor, jak by mohl vypadat soubor `paths`. Avšak tento soubor budete potřebovat pouze v případě, že máte poštovní spojení s více než jedním systémem. Je-li to váš případ, musíte si tento soubor sami napsat nebo ho musíte vygenerovat z usenetových map. Soubor `paths` bude popsán dále v kapitole.

14.2 Nastavení pro lokální síť typu LAN

Pokud provozujete systém, který je spojen se dvěma nebo více hostiteli pomocí lokální sítě typu LAN, musíte určit jednoho hostitele, který se bude starat o spojení s okolním světem na bázi protokolu UUCP. Mezi hostiteli uvnitř vaší sítě lokální sítě LAN si budete asi chtít s největší pravděpodobností vyměňovat poštu pomocí protokolu SMTP, který bude pracovat v síti na bázi protokolu TCP/IP. Předpokládejme, že jsme zpět ve společnosti Virtual Brewery a hostitel **vstout** je nastaven jako brána pro protokol UUCP.

V síťovém prostředí je nejlepší uchovávat veškeré uživatelské poštovní schránky v jediném souborovém systému, který mají pomocí systému NFS připojen všichni ostatní hostitelé. To umožňuje uživatelům přesuny z jednoho počítače na druhý, aniž by museli přenášet svou poštu sem a tam (nebo v horším případě museli každé ráno kontrolovat tři nebo čtyři počítače, zda jim nepřišla nějaká nová pošta). Proto budete také požadovat, aby adresa odesilatele byla nezávislá na počítači, na kterém byla pošta napsána. Obvyklou praxí je použít v adrese odesilatele místo názvu hostitele pouze vlastní název domény. Například uživatel Janet by měl zadat místo adresy **janet@vale.vbrew.com** adresu **janet@vbrew.com** Dále si vysvětlíme, jak server pozná, že název domény je korektní název pro váš systém.

Další způsob, jak uchovávat všechny poštovní schránky na centrálním hostiteli, spočívá v použití protokolu POP nebo protokolu IMAP. Protokol POP znamená *poštovní protokol* (*Post Office Protocol*) a umožňuje uživatelům přistupovat ke svým schránkám pomocí jednoduchého

[2] Důvod je následující: Předpokládejme, že název vašeho hostitele je **monad**, avšak tento název není zaregistrován v mapách. V mapách je však zaregistrován nějaký jiný systém **monad**, takže veškerá pošta na adresu **monad!root**, kterou může dokonce poslat i váš přímý soused z hlediska protokolu UUCP, bude doručena druhému systému **monad**. To je rozhodně nepříjemné pro všechny.

spojení na bázi protokolu TCP/IP. Protokol IMAP znamená *interaktivní protokol pro přístup k poště* (*Interactive Mail Access Protocol*) a je podobný protokolu POP, ale trošku obecnější. Na platformu operačního systému Linux byly přeneseny jak servery pro protokol IMAP, tak i servery pro protokol POP. Servery jsou dostupné na adrese **sunsite.unc.edu** v adresáři `/pub/Linux/system/Network`.

14.2.1 Zápis konfiguračních souborů

Konfigurace pro společnost pivovaru pracuje následovně: všichni hostitelé kromě vlastního poštovního serveru s názvem **vstout** směrují veškerou odcházející poštu na poštovní server, k tomu použijí směrování na chytřejšího hostitele. Samotný poštovní server s názvem **vstout** posílá veškerou odcházející poštu na skutečného chytřejšího hostitele, který směruje veškerou poštu ze společnosti pivovaru; tento hostitel se nazývá **moria**.

Standardní soubor `config` určený pro všechny hostitele kromě poštovního serveru s názvem **vstout** vypadá následovně:

```
#
# Naše doména
visible domain=vbrew.com
#
# Naše jméno pro odchozí poštu
visible name=vbrew.com
#
# Cesta k chytřejšímu hostiteli: pomocí SMTP na vstout
smart path=vstout
smart transport=smtp
```

Je to velmi podobné způsobu, který jsme používali u systémů, jež byly připojeny pouze pomocí protokolu UUCP. Hlavní odlišnost spočívá v tom, že transportem používaným k posílání pošty na chytřejšího hostitele je samozřejmě protokol SMTP. Volba *visible_domain* sdělí programu `smail`, aby pro veškerou odcházející poštu používal místo názvu místního hostitele název domény.

Na poštovní bráně pro protokol UUCP s názvem **vstout** vypadá soubor `config` trochu jinak:

```
#
# Naše doména
hostnames=vbrew.com:vstout.vbrew.com:vstout
#
```

```
# Naše jméno pro odchozí poštu
visible name=vbrew.com
#
# Jméno pro UUCP
uucp name=vbrew.com
#
# Cesta k chytřejšímu hostiteli: pomocí UUCP na moria
smart path=moria
smart transport=uux
#
# Spravujeme poštu pro naši doménu
auth domains=vbrew.com
```

Tento soubor `config` používá ke sdělování informací programu `smail` o způsobu volání místního hostitele odlišné schéma. Místo toho, aby mu byl předal seznam domén a aby si nechal vyhledat název hostitele pomocí systémového volání, zadává seznam explicitně. Výše uvedený seznam obsahuje jak plně kvalifikované názvy hostitelů, tak i nekvalifikované názvy hostitelů a navíc obsahuje i samotný název domény. To umožní programu `smail` rozpoznat adresu **janet@vbrew.com** jako místní adresu a doručit zprávu uživateli **janet**.

Volba *auth_domains* uvádí domény, pro něž je směrodatný poštovní server **vstout**. To znamená, že když program `smail` obdrží jakoukoliv adresu směrovanou na adresu `host`.**vbrew.com**, kde hostitel `host` neodpovídá žádnému z místních počítačů, měl by ji odmítnout a vrátit zpět odesilateli. Není-li tato položka uvedena, bude každá taková zpráva poslána chytřejšímu hostiteli, který ji vrátí zpět poštovnímu serveru **vstout,** a tak to pude pokračovat, dokud nebude zlikvidována z důvodu překročení maximálního počtu skoků.

14.2.2 Spuštění programu smail

Nejprve se musíte rozhodnout, zda chcete spouštět program `smail` jako samostatného démona nebo zda necháte spravovat port protokolu SMTP super serverem `inetd`, který při jakémkoliv požadavku ze strany klienta na spojení pomocí protokolu SMTP spustí program `smail`. U poštovního serveru obvykle dáte přednost práci v režimu démona, protože to bude zatěžovat počítač mnohem méně, než stále opakované spouštění programu `smail`, kdykoliv se vyskytne nějaké spojení. Protože poštovní servery doručují také většinu příchozí pošty přímo uživatelům, zvolíte na většině hostitelů práci v režimu super serveru `inetd`.

Ať už zvolíte kterýkoliv z těchto režimů, musíte se ujistit, že máte v souboru `/etc/services` následující položku:

```
smtp            25/tcp          # Simple Mail Transfer Protocol
```

Tato položka definuje číslo portu pro protokol TCP, které by měl použít program `smail` při spojeních pomocí protokolu SMTP. V dokumentu RFC Assigned Numbers je definována implicitní hodnota 25.

Když je program `smail` spuštěn v režimu démona, přejde do pozadí a bude čekat na nějaké spojení na portu pro protokol SMTP. Když k tomuto spojení dojde, spustí druhý proces a bude řídit komunikaci, která probíhá na bázi protokolu SMTP. Démon programu `smail` se většinou spouští pomocí následujícího příkazu ze skriptu `rc.inet2`:

```
/usr/local/bin/smail -bd -q15m
```

Příznak `-bd` zapíná režim démona a příznak `-q15m` sdělí programu `smail`, aby každých 15 minut zpracovával zprávy, které se mezitím nahromadily ve frontě.

Chcete-li k tomuto účelu použít raději super-server `inetd`, měl by váš soubor `/etc/inetd.conf` obsahovat následující řádek:

```
smtp     stream  tcp nowait  root  /usr/sbin/smtpd smtpd
```

Příkaz `smtpd` by měl být symbolickým odkazem na binární soubor programu `smail`. Pamatujte, že po provedení těchto změn musíte donutit super-server *inetd*, aby znovu načetl konfigurační soubor `inetd.conf`. To provedete tak, že mu pošlete signál *HUP*.

Jak režim démona, tak i režim super-serveru `inetd` jsou režimy výhradní. Jestliže máte spuštěný program `smail` v režimu démona, musíte se ujistit, že jste v souboru `inetd.conf` zakázali veškeré řádky, které se vztahují ke službě `smtp`. Podobně se v případě, že necháte super-server `inetd` spravovat program `smail`, musíte ujistit, že ze skriptu `rc.inet2` nespouštíte démona `smail`.

14.3 Když se vám nepodaří prorazit...

Pokud se při instalaci v něčem zmýlíte, máte k dispozici spoustu vlastností, jež vám mohou pomoci nalézt jádro problému. V prvé řadě musíte zkontrolovat všechny log-soubory programu `smail`. Tyto soubory jsou uloženy v adresáři `/var/spool/smail/log` a jejich názvy jsou `logfile`, resp. `paniclog`. Soubor `logfile` vypisuje všechny provedené úkony, zatímco soubor `paniclog` je určen pouze pro výpis chybových zpráv, které se vztahují k chybám konfigurace atp.

Typický záznam v souboru `logfile` vypadá následovně:

```
04/24/94 07:12:04: [m0puwU8-00023UB] received
|              from: root
```

```
|          program: sendmail
|             size: 1468 bytes
 04/24/94 07:12:04: [m0puwU8-00023UB] delivered
|              via: vstout.vbrew.com
|               to: root@vstout.vbrew.com
|          orig-to: root@vstout.vbrew.com
|           router: smart host
|        transport: smtp
```

Tento výpis ukazuje, že zpráva od uživatele **root** směrovaná na **root@vstout.vbrew.com**, byla protokolem SMTP v pořádku doručena na hostitele **vstout**.

U zpráv, které nemůže program `smail` doručit, se vygeneruje v souboru log podobný záznam, jen místo klíčového slova `delivered` v něm bude uvedena chybová zpráva:

```
04/24/94 07:12:04: [m0puwU8-00023UB] received
|             from: root
|          program: sendmail
|             size: 1468 bytes
04/24/94 07:12:04: [m0puwU8-00023UB] root@vstout.vbrew.com ... defer
 (ERR 148) transport smtp: connect: Connection refused
```

Výše uvedená chyba je typická pro situaci, kdy program `smail` správně rozpozná, že má zprávu doručit hostiteli **vstout**, avšak nemůže se spojit se službou SMTP na hostiteli **vstout**. V takovém případě je buď chyba v konfiguraci, anebo ve vašich binárních souborech programu `smail` schází podpora protokolu TCP.

Tento problém není zase tak ojedinělý, jak se mohlo na první pohled zdát. Existují předkompilované binární soubory `smail`, dokonce i v některých distribucích operačního systému Linux, které nemají podporu pro sítě na bázi protokolu TCP/IP. Je-li to právě váš případ, musíte si sami zkompilovat program `smail`. Máte-li nainstalován program `smail`, pak si na něm můžete otestovat, zda podporuje sítě na bázi protokolu TCP. Stačí jen spustit program `telnet` na portu, který je určen pro protokol SMTP. Úspěšné připojení k serveru SMTP je ukázáno níže (vaše vstupy jsou zobrazeny *tímto stylem*):

```
$ telnet localhost smtp
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 monad.swb.de Smail3.1.28.1 #6 ready at Sun, 23 Jan 94
```

```
19:26 MET
QUIT
221 monad.swb.de closing connection
```

Pokud tento test nezobrazí hlavičku protokolu SMTP (to je ten řádek, co začíná hodnotou 220), pak se předtím, než začnete sestavovat program `smail`, ujistěte, že je vaše konfigurace *skutečně* v pořádku. Kompilace vlastního programu `smail` bude popsána dále.

Narazíte-li u programu `smail` na problém, který nepůjde lokalizovat z chybové zprávy vygenerované programem `smail`, můžete vyzkoušet povolit ladicí informace. To lze provést pomocí volby `-d`, za kterou může následovat číslo určující úroveň rozsahu výpisů (mezi volbou a číselným argumentem nesmí být mezera). V takovém případě vypíše program `smail` na obrazovce zprávu o své činnosti, která vám může pomoci při hledání příslušného problému.

[**No, já nevím … Snad to lidé nebudou považovat za příliš směšné:**] Pokud nic nepomůže, pak zkuste spustit program `smail` pomocí volby `-bR` v režimu „darebáka“. Tato volba se uvádí na příkazové řádce. Na manuálové stránce se o této volbě píše následující: „Vstupte do nepřátelské domény, kterou tvoří obrovské poštovní zprávy a svitky standardů ve formě RFC. Pokuste se ji zmenšit na protokolovou úroveň 26 a potom ji zase vraťte zpět.“ I když tato volba vaše problémy nevyřeší, může vás trochu povzbudit a poskytnout vám alespoň malou útěchu.[3]

14.3.1 Sestavení programu smail

Pokud s jistotou víte, že program `smail` nemá zabudovanou podporu pro sítě na bázi protokolu TCP, musíte si obstarat jeho zdrojový kód. Jestliže jste vaši distribuci operačního systému Linux dostali na CD-ROM, pak bude zdrojový kód programu `smail` asi její součástí, v opačném případě ho můžete získat ze sítě pomocí služby FTP.[4]

Při sestavování programu `smail` uděláte nejlépe, když začnete se skupinou konfiguračních souborů od Vince Skahana, které jsou součástí distribuce balíku `newspak`. Abyste ho zkompilovali s podporou síťového ovladače protokolu TCP, musíte nastavit v souboru `conf/EDITME` makro *DRIVER_CONFIGURATION* buď na hodnotu *bsd-network*, nebo na hodnotu *arpa-network*. Hodnota *bsd-network* je vhodná pro instalace v lokálních sítích typu LAN, zatímco instalace v Internetu vyžadují hodnotu *arpa-network*. Rozdíl v těchto dvou hodnotách spočívá v tom, že hodnota *arpa-network* přidá speciální ovladač pro službu BIND, který je schopen rozpoznat záznamy typu MX. Při zadání volby *bsd-network* nebudou tyto záznamy rozpoznány.

3 Pokud máte skutečně špatnou náladu, pak tuto volbu nezkoušejte.

4 Pokud jste si ho koupili od vašeho dodavatele společně s distribucí Linuxu, máte podle licenčních podmínek programu `smail` právo na tento zdrojový kód za cenu „nepatrného poštovního poplatku“.

14.4 Režimy doručování pošty

Jak jsme se již zmínili, program `smail` je schopen okamžitě doručovat zprávy nebo je může ukládat do fronty, kterou zpracuje později. Pokud si zvolíte ukládání zpráv do fronty, bude program `smail` ukládat veškerou poštu do podadresáře `messages`, který je umístěn v adresáři `/var/spool/smail`. Tyto zprávy nebude zpracovávat, dokud mu to explicitně nepřikážete (tento proces se označuje jako tzv. „zpracování fronty“).

Pomocí volby *delivery_mode* umístěné v souboru `config` si můžete vybrat jeden ze tří doručovacích režimů, a to buď režim *foreground*, *background* nebo *queued*. Tyto režimy zvolí doručování na popředí (příchozí zprávy se zpracují okamžitě), na pozadí (zprávy budou doručeny potomkem přijímacího procesu, rodičovský proces skončí okamžitě po vyvolání potomka přijímacího procesu) nebo zvolí doručování pomocí fronty. Pokud je v souboru `config` nastavena booleovská proměnná *queue_only*, bude příchozí pošta vždy posílána do fronty, a to bez ohledu na nastavení volby *delivery_mode*.

Pokud zapnete posílání do fronty, musíte se ujistit, že tato fronta je pravidelně kontrolována; například každých 10 nebo 15 minut. Spouštíte-li program `smail` v režimu démona, musíte na příkazovou řádku přidat volbu `-q10m`, aby se fronta zpracovávala každých 10 minut. Další možností je v těchto intervalech spouštět příkaz `runq` z programu `cron`. `runq` by měl být odkaz na programu `smail`.

Aktuální poštovní frontu si můžete zobrazit spuštěním programu `smail` s volbou `-bp`. Další možností je vytvořit odkaz `mailq` na program `smail`; pak stačí volat příkaz `mailq`:

```
$ mailq -v
m0pvB1r-00023UB From: root  (in /var/spool/smail/input)
                Date: Sun, 24 Apr 94 07:12 MET DST
                Args: -oem -oMP sendmail root@vstout.vbrew.com
Log of transactions:
Xdefer:  reason: (ERR 148) transport smtp:
connect: Connection refused
```

Tento příkaz ukazuje jednotlivé zprávy, které jsou umístěny v poštovní frontě. Záznam provedených transakcí (které jsou zobrazeny pouze v případě, že spustíte příkaz `mailq` s volbou `-v`) vám může poskytnout dodatečný důvod, proč daná zpráva stále čeká na doručení. Pokud zatím nebyl učiněn žádný pokus o doručení zprávy, nebude zobrazen žádný záznam provedených transakcí.

Dokonce i v případě, že nechcete používat poštovní frontu, vloží občas program *smail* zprávy do fronty, pokud zjistí, že okamžité doručení neuspělo z přenosových důvodů. U spojení pomocí protokolu SMTP může být například tímto důvodem nedosažitelnost požadovaného hostitele; avšak zprávy se mohou zpozdit i v případě, kdy se zjistí, že je souborový systém téměř zaplněn. Proto byste měli nechat frontu zpracovávat přibližně každou hodinu (pomocí příkazu `runq`). V opačném případě by veškeré odložené zprávy zůstaly ve frontě navždy viset.

14.5 Různé volby v souboru config

Existuje poměrně velký počet voleb, které lze zadat v konfiguračním souboru `config`. Ačkoliv jsou tyto volby užitečné, nejsou pro správnou funkci programu `smail` nezbytné, a proto se zde jimi nebudeme zabývat. Místo toho se zde zmíníme pouze o několika z nich, které by se vám mohly hodit:

error_copy_postmaster — Je-li tato booleovská proměnná nastavena, pak bude mít jakákoliv chyba za následek vygenerování zprávy pro účet postmaster. Obvykle se to provádí pouze u chyb, ke kterým došlo z důvodu chybné konfigurace. Tuto proměnnou lze zapnout v souboru `config` tak, že před její název vložíte symbol plus (+).

max_hop_count — Je-li počet skoků (hops) u dané zprávy (tj. počet hostitelů, jimiž daná zpráva prošla) rovný nebo vyšší než tato hodnota, bude výsledkem pokusů o vzdálené doručení chybová zpráva, která bude doručena odesilateli. Tato volba se používá k tomu, aby se zprávy nemohly dostat do nekonečné smyčky. Počet skoků se obvykle vypočítá z počtu polí *Received:*, které se nacházejí v hlavičce zprávy. Může však být také zadán ručně pomocí volby `-h` zadané na příkazové řádce.

Tato proměnná je implicitně nastavena na hodnotu 20.

postmaster — Adresa postmastera. Nemůže-li být adresa **Postmaster** nalezena jako korektní místní adresa, pak se tato volba použije jako poslední možnost. Implicitně je nastavena na hodnotu **root**.

14.6 Směrování a doručování zpráv

Program `smail` člení proces doručení na tři odlišné úkoly, směrovač, řídící modul a transportní modul.

Modul směrovače rozkládá veškeré vzdálené adresy a určuje, ke kterému hostiteli by měla být zpráva dále poslána a který transport se k tomu musí použít. V závislosti na povaze daného spojení mohou být použity rozdílné transporty, například jako transport může být použit protokol UUCP nebo protokol SMTP.

Místní adresy jsou předány řídicímu modulu, který rozloží veškerá směrování nebo veškeré používané přezdívky. Adresou může být například přezdívka nebo poštovní konference, případně může chtít uživatel směrovat svoji poštu na jinou adresu. Pokud je výsledná adresa vzdálená, bude předána modulu směrovače, který zjistí doplňkové směrování, v opačném případě bude podstoupena transportnímu modulu, který ji doručí místnímu hostiteli. Zdaleka nejběžnějším způsobem bude doručování do poštovní schránky, avšak zprávy mohou být také předány nějakému příkazu nebo připojeny k libovolnému souboru.

A konečně transportní modul je zodpovědný za libovolnou vybranou metodu doručení. Tento modul se pokusí doručit poštu a v případě neúspěchu buď vygeneruje skokovou zprávu, nebo odloží danou zprávu pro případné další pokusy.

Program `smail` vám poskytuje při konfiguraci těchto úkolů značnou volnost. Pro každý z těchto úkolů je dostupných několik ovladačů, z nichž si můžete vybrat pouze ty, které budete potřebovat. Tyto ovladače popíšete programu `smail` v několika souborech, konkrétně jsou to soubory `routers`, `directors` a `transports`, které se nacházejí v adresáři `/usr/lib/smail`. Pokud tyto soubory neexistují, budou se předpokládat rozumné implicitní hodnoty, které by měly být vhodné pro spoustu systémů používajících jako transportní modul protokol SMTP nebo protokol UUCP. Chcete-li změnit směrovací pravidla programu `smail` nebo chcete-li upravit používaný transportní modul, pak byste si měli obstarat vzorové soubory ze zdrojové distribuce,[5] zkopírovat tyto soubory do adresáře `/usr/lib/smail` a upravit je tak, aby vyhovovaly vašim potřebám. Vzorové konfigurační soubory jsou uvedeny v dodatku B.

14.7 Směrování zpráv

Když je zachycena zpráva, program `smail` nejprve zkontroluje, zda je cílovým hostitelem místní hostitel nebo vzdálený systém. Je-li cílovým hostitelem adresa jednoho z místních názvů hostitelů nakonfigurovaných v souboru `config`, bude zpráva předána řídicímu modulu. V opačném případě předá program `smail` cílovou adresu několika ovladačům směrovačů, aby zjistil, jakému hostiteli má být daná zpráva poslána. Ovladače směrovačů mohou být popsány v souboru `routers`; pokud tento soubor neexistuje, budou použity implicitní směrovače.

[5] Implicitní konfigurační soubory mohou být umístěny v podadresáři `samples/generic` adresáře se zdrojovým kódem.

Cílový hostitel je postupně předán všem směrovačům a zvolí se ten směrovač, který nalezne nejpřesnější směrování. Uvažme zprávu poslanou na adresu **joe@foo.bar.com**. Pak jeden ze směrovačů může znát implicitní směr na všechny hostitele v doméně **bar.com**, zatímco nějaký jiný směrovač může mít informace přímo o vlastním hostiteli **foo.bar.com**. Protože naposledy uvedený směrovač obsahuje přesnější směrování, bude mít přednost před dříve uvedeným směrovačem. Pokud existují dva směrovače, kteří poskytnou „nejlepší směrování“, bude vybrán ten z nich, který je uveden v souboru `routers` dříve.

Nyní tento směrovač určí typ použitého přenosu, například protokol UUCP, a vygeneruje novou cílovou adresu. Ta je pak předána transportnímu modulu společně s názvem hostitele, kterému by se měla zpráva poslat. Ve výše uvedeném příkladu by mohl program `smail` zjistit, že hostitel **foo.bar.com** je dosažitelný pomocí protokolu UUCP, pokud použije cestu **ernie!bert**. Potom program `smail` vygeneruje novou cílovou adresu **bert!foo.bar.com!user** a sdělí transportnímu modulu pro protokol UUCP, aby tuto adresu předal hostiteli **ernie** jako tzv. adresu na obálce.

Při použití implicitního nastavení jsou k dispozici následující směrovače:

- Může-li být adresa cílového hostitele resolvována pomocí volání funkce knihovny *gethostbyname(3)* nebo *gethostbyaddr*(3), pak bude zpráva doručena pomocí protokolu SMTP. Jedinou výjimkou z tohoto pravidla je případ, kdy se u adresy zjistí, že odkazuje na místního hostitele. V takovém případě bude předána řídicímu modulu.

  Program `smail` rozeznává IP-adresy, které jsou napsané jak s použitím tečkové notace, tak s využitím právoplatných názvů hostitelů v případě, že se dají rozložit pomocí systémového volání *gethostbyaddr(3)*. Například adresa **scrooge@[149.76.12.4]** je platnou adresou, i když je to značně neobvyklá poštovní adresa pro uživatele **scrooge** na hostiteli **quark.physics.groucho.edu**.

  Jestliže je váš počítač v Internetu, nepředstavují tyto směrovače nic z toho, co byste mohli potřebovat, protože nepodporují záznamy typu MX. Co máte v takovémto případě dělat, se dozvíte dále.

- Pokud existuje databáze cest `/usr/lib/smail/paths`, pokusí se program `smail` vyhledat v tomto souboru cílového hostitele (bez řetězce **.uucp**, který je uveden na konci). Pošta poslaná na adresu, která vyhovuje tomuto směrovači, bude doručena prostřednictvím protokolu UUCP s využitím cesty, jež byla nalezena v databázi.

- Adresa hostitele (bez řetězce **.uucp**, který je uveden na konci) bude porovnána s výstupem příkazu `uuname`, aby se zjistilo, zda je cílový hostitel skutečně sousedem UUCP. V tom případě bude zpráva doručena prostřednictvím transportního modulu pro protokol UUCP.

- Pokud adresa neodpovídala ani jednomu z výše uvedených směrovačů, bude doručena chytřejšímu hostiteli. Cesta na chytřejšího hostitele je společně s používaným transportem nastavena v souboru `config`.

Tato implicitní nastavení fungují správně u mnoha jednoduchých nastavení, avšak selhávají v případě, kdy jsou směrovací požadavky trošku komplikovanější. Při podobných potížích si budete muset nainstalovat svůj vlastní soubor `routers`, který potlačí implicitní nastavení. Vzorový soubor `routers`, ze kterého byste mohli vyjít, je uveden v příloze B. Některé distribuce operačního systému Linux také přicházejí se skupinou konfiguračních souborů, které jsou upraveny tak, aby se s těmito problémy vypořádaly.

Pravděpodobně nejhorší problémy vyvstávají v situacích, kdy je váš hostitel provozován ve dvojím prostředí, jak se spojeními na bázi protokolu UUCP, tak i se spojeními pomocí modemu na bázi protokolu IP. V tom případě budete mít v souboru `hosts` názvy hostitelů, se kterými budete komunikovat jen příležitostně pomocí protokolu SLIP, takže program `smail` se bude pokoušet doručovat veškerou poštu těmto hostitelům prostřednictvím protokolu SMTP. To ale nebudete potřebovat, protože i když je spojení pomocí protokolu SLIP aktivováno pravidelně, bude posílání pošty po protokolu SMTP mnohem pomalejší, než po síti na bázi protokolu UUCP. Při implicitním nastavení neexistuje žádný způsob, jak tento problém programu `smail` vyřešit.

Tomuto problému se můžete vyhnout, necháte-li program `smail` zkontrolovat soubor `paths` dříve, než se bude dotazovat resolveru. Pak můžete do souboru `paths` vložit všechny hostitele, u kterých chcete explicitně definovat doručování pošty po síti na bázi protokolu UUCP. Pokud nechcete posílat prostřednictvím protokolu SMTP *žádné* zprávy, můžete označit jako komentáře veškeré směrovače používající resolver.

Další problém spočívá v tom, že implicitní nastavení neposkytuje správné směrování pošty v síti Internet, protože směrovač založený na resolveru nevyhodnocuje záznamy typu MX. Chcete-li povolit úplnou podporu pro směrování pošty v Internetu, označte tento směrovač jako komentář a odstraňte značku komentáře z toho směrovače, jenž používá místo resolveru službu BIND. Nicméně některé distribuce operačního systému Linux obsahují binární soubory programu `smail`, do nichž není zakompilována podpora služby BIND. Pokud povolíte službu BIND, ale v souboru `paniclog` najdete zprávu „`router inet_hosts: driver bind not found`“, budete si muset sehnat zdrojový kód k programu `smail` a tento program znovu zkompilovat (viz výše uvedená stať 14.2).

A nakonec není obvykle příliš vhodné používat ovladač `uuname`. První problém spočívá v tom, že tento ovladač vygeneruje v případě, že nemáte nainstalován protokol UUCP, chybu konfigurace, protože nenajde žádný příkaz `uuname`. Druhý problém nastane, máte-li v souboru `Systems` protokolu UUCP uvedeno více systémů, s nimiž máte skutečné poštovní spo-

jení. Těmito systémy mohou být hostitelé, s nimiž si pouze vyměňujete news, nebo hostitelé, ze kterých si občas stahujete nějaké soubory pomocí anonymní služby UUCP, avšak na druhou stranu si s nimi nevyměňujete žádné soubory.

Abyste tento problém vyřešili, můžete nahradit program `uuname` skriptem příkazového interpretu, který bude provádět jednoduchou funkci `exit 0`. Avšak obecnější řešení spočívá v úpravě souboru `routers` a odstranění tohoto ovladače.

14.7.1 Databáze paths

Program `smail` očekává, že nalezne databázi cest v souboru `paths`, který se nachází v adresáři `/usr/lib/smail`. Tento soubor je volitelný, takže pokud nechcete provádět žádné směrování cest, stačí tento soubor odstranit.

Soubor `paths` musí být setříděným souborem ASCII, který obsahuje záznamy, jež mapují názvy cílových systémů na cesty protokolu UUCP zadané pomocí vykřičníkové notace. Soubor je nutno setřídit, protože program `smail` používá k vyhledání požadovaného systému binární hledání. V tomto souboru nejsou povoleny komentáře a název systému musí být oddělen od vlastní cesty znakem tabulátoru. Databáze cest jsou podrobněji probrány ve 13. kapitole.

Pokud jste tento soubor vygenerovali ručně, měli byste se ujistit, že jste do něj vložili všechny právoplatné názvy daného systému. Je-li například nějaký systém známý jak pod prostým názvem pro protokol UUCP, tak i pod plně kvalifikovaným doménovým jménem, musíte do tohoto souboru přidat položku pro každý z těchto názvů. Soubor může být setříděn například tak, že ho předáte příkazu `sort`*(1)*.

Je-li však váš systém pouze koncovým systémem, nebudete potřebovat vůbec žádný soubor `paths`: stačí pouze nastavit v souboru `config` vlastnosti týkající se chytřejšího hostitele a dále můžete nechat veškeré směrování pošty na svém poštovním doručovateli.

14.8 Doručování zpráv na místní adresy

Nejběžněji místní adresa odpovídá přihlašovacímu jménu uživatele. V takovémto případě je zpráva doručena do jeho poštovní schránky do adresáře `/var/spool/mail/`*user*, kde *user* je přihlašovací jméno daného uživatele. Dalšími případy je použití přezdívek, názvů poštovních konferencí nebo uživatelské směrování pošty. V těchto případech se místní adresa rozloží do nového seznamu adres, které mohou být buď místní, nebo vzdálené.

Kromě těchto „normálních“ adres se může program `smail` starat i o další typy místních destinací zpráv, jimiž mohou být například názvy souborů nebo převedení zpráv příkazům. Toto nejsou adresy v pravém slova smyslu, takže nemůžete poslat poštu například na „adresu“ **/etc/passwd@vbrew.com**; tyto adresy jsou korektní pouze v případě, že byly převzaty ze směrování nebo ze souborů přezdívek.

Za `název souboru` se považuje vše, co začíná znakem lomítka (/) nebo znakem vlnovky (~). Znak vlnovky odkazuje na domovský adresář příslušného uživatele a je přípustný pouze v případě, že byl název souboru převzat ze souboru `.forward` nebo ze směrovacího záznamu v poštovní schránce (viz níže). Pokud se doručuje do souboru, připojí program `smail` zprávu k danému souboru nebo v případě nutnosti tento soubor vytvoří.

Příkaz, na nějž má být daná zpráva převedena, může být jakýkoliv unixový příkaz, před kterým je uveden symbol propojení (|). Tento symbol zapříčiní, že program *smail* předá daný příkaz společně s jeho argumenty příkazového interpretu, ale tentokrát už bez uvozujícího symbolu (|). Vlastní zpráva je předána na standardní vstup tohoto příkazu.

Například k předávání poštovní konference do místní diskusní skupiny můžete použít skript příkazového interpretu s názvem `gateit` a potom nastavit místní přezdívku používající „|`gateit`“, která bude doručovat veškeré zprávy z poštovního seznamu do daného skriptu.

Obsahuje-li vyvolání příkazu bílé místo, musí být daný příkaz uzavřen do uvozovek. Vzhledem k objevujícím se bezpečnostním problémům jsou učiněna opatření, aby se příkaz nespustil v případě, že byla adresa získána nějakým pochybným způsobem (například pokud do souboru s přezdívkami, z něhož byla adresa získána, může zapisovat naprosto každý).

14.8.1 Místní uživatelé

Nejběžnějším případem místní adresy je uvedení poštovní schránky uživatele. Poštovní schránka je umístěna v adresáři `/var/spool/mail` a má přiřazeno jméno uživatele. Jejím vlastníkem je daný uživatel, který patří do skupiny **mail** a má nastavena přístupová práva 660. Pokud poštovní schránka neexistuje, pak ji program `smail` vytvoří.

Poznamenejte si, že ačkoliv je v současnosti adresář `/var/spool/mail` standardním místem pro umístění souborů poštovní schránky, mohou mít některé poštovní programy předkompilovány odlišné cesty, například do adresáře `/usr/spool/mail`. Pokud na vašem počítači neustále selhává doručování pošty, pak můžete zkusit, zda nepomůže vytvoření symbolického odkazu do adresáře `/var/spool/mail`.

Program `smail` vyžaduje existenci dvou adres: **MAILER-DAEMON** a **Postmaster**. U nedoručitelné pošty se vygeneruje skoková zpráva. Kopie této zprávy je poslána na účet **postmaster** k prozkoumání (to je kvůli možnému výskytu nějakého konfiguračního problému). Adresa **MAILER-DAEMON** se používá u skokové zprávy jako adresa odesílatele.

Pokud tyto adresy ve vašem systému neodpovídají korektním účtům, namapuje program `smail` účet **MAILER-DAEMON** na účet **postmaster**, resp. účet **postmaster** na účet **root**. Toto mapování můžete obvykle potlačit tak, že přezdívku k účtu **postmaster** přidělíte komukoliv, kdo je zodpovědný za údržbu poštovního softwaru.

14.8.2 Směrování

Uživatel si může přesměrovat svou poštu tak, že ji pošle na alternativní adresu. Program `smail` k tomuto účelu podporuje dvě metody. Jednou z možností je vložit následující položku na první řádek souboru poštovní schránky:

`Forward to` *recipient*,...

Tato položka zajistí, že bude veškerá příchozí pošta poslána všem příjemcům v zadaném seznamu. Další možností je vytvoření souboru `.forward` v domovském adresáři daného uživatele, který bude obsahovat čárkami oddělený seznam příjemců. U tohoto způsobu směrování jsou přečteny a zanalyzovány všechny uvedené řádky.

Všimněte si, že v souboru `.forward` můžete použít libovolný typ adresy. Praktická ukázka souboru `.forward`, který lze použít v případě dovolené, by mohla vypadat asi takto:

```
janet, "|vacation"
```

První uvedená adresa vždy doručí příchozí zprávu do poštovní schránky uživatele **janet**, avšak příkaz `vacation` pošle odesilateli krátké oznámení.

14.8.3 Soubory s přezdívkami

Program `smail` může spravovat takové soubory s přezdívkami, které jsou kompatibilní se soubory s přezdívkami, jež zná program `sendmail` vytvořený na univerzitě v Berkeley. Položky v souboru s přezdívkami mají následující syntaxi:

```
alias:   recipients
```

Pole `recipients` se skládá z čárkami odděleného seznamu adres, jež budou nahrazeny přezdívkou. Seznam příjemců může pokračovat i na více řádcích, pokud následující řádek začíná znakem tabulátor.

Dále existuje speciální vlastnost, která umožní programu *smail* přebírat poštovní konference ze souboru přezdívek: zadáte-li místo adresy příjemce řetězec „:include:filename“, přečte program `smail` soubor `filename` a použije seznam příjemců z tohoto souboru.

Hlavní soubor s přezdívkami má název `/usr/lib/aliases`. Umožníte-li zápis do tohoto souboru komukoliv, pak program `smail` nebude doručovat žádné zprávy příkazům příkazového interpretu, které jsou v tomto souboru uvedeny. Nyní následuje vzorový soubor `aliases`:

```
# soubor /usr/lib/aliases file pro vbrew.com
hostmaster: janet
postmaster: janet
usenet: phil
# Diskusní skupina development
development: joe, sue, mark, biff
          /var/mail/log/development
owner-development: joe
# Oznámení jsou zasílána všem zaměstnancům
announce: :include: /usr/lib/smail/staff,
          /var/mail/log/announce
owner-announce: root
# Brána mezi poštovní konferencí a lokální diskusní skupinou
ppp-list: "|/usr/local/lib/gateit local.lists.ppp"
```

Pokud se při doručování na adresu vygenerovanou ze souboru `alias` vyskytne nějaká chyba, pokusí se program `smail` poslat kopii chybové hlášky „vlastníkovi přezdívky“. Pokud například při doručování zprávy podle poštovního seznamu **development** neuspěje doručení zprávy uživateli **biff**, bude kopie chybové zprávy poslána odesilateli a stejně tak i na účty **postmaster** a **owner-development**. Jestliže adresa vlastníka neexistuje, nebude již vygenerována žádná další zpráva.

Při doručování do souborů nebo při vyvolávání programů uvedených v souboru `aliases` přejde z bezpečnostních důvodů program `smail` na účet uživatele **nobody**. Zvláště při doručování do souborů by mohly nastat skutečné bezpečnostní problémy. Například u výše uvedeného souboru musí být vlastníkem log-souborů uživatel **nobody**. Tento uživatel musí mít i právo do nich zapisovat. V opačném případě by doručování do těchto souborů neuspělo.

14.8.4 Poštovní konference

Poštovní konference mohou být spravovány nejen pomocí souboru *aliases*, ale také prostřednictvím souborů nacházejících se v adresáři `/usr/lib/smail/lists`. Poštovní konference s názvem *nag-bugs* je popsána v souboru `lists/nag-bugs`. Tento soubor by měl obsahovat čárkami oddělené adresy členů. Seznam může být uveden na několika řádcích a komentáře musí být uvozeny znakem (#).

Ke každé poštovní konferenci by měl existovat uživatel (nebo přezdívka) s názvem **owner-**`listname`, kde `listname` je název dané poštovní konference; veškeré chyby, které se vyskytnou při rozkládání adresy, budou oznámeny tomuto uživateli. Tato adresa se také používá jako adresa odesilatele u všech odcházejících zpráv. Je uvedena v hlavičce zprávy v poli `Sender:`.

14.9 Přenosy na bázi protokolu UUCP

V programu `smail` je zakompilováno množství transportů, které využívají balík UUCP. V prostředí protokolu UUCP jsou zprávy posílány na dalšího hostitele obvykle pomocí příkazu `rmail`. Zpráva je tomuto příkazu předána ze standardního vstupu a adresa na obálce mu je předána na příkazovém řádku. Na vašem hostiteli by měl být příkaz `rmail` symbolickým odkazem na příkaz `smail`.

Při předání zprávy transportu na bázi protokolu UUCP převede program `smail` cílovou adresu do vykřičníkové notace, jež se používá v prostředí protokolu UUCP. Například adresa **user@host** bude převedena na adresu **host!user**. Výskyt všech adresových operátorů '%' bude zachován, takže adresa **user%host@gateway** bude převedena na adresu **gateway!user%host**. Avšak samotný program `smail` nikdy takovouto adresu nevygeneruje.

Alternativně může program `smail` pomocí protokolu UUCP posílat a přijímat dávky protokolu BSMTP. U protokolu BSMTP se zabalí jedna nebo více zpráv do jediné dávky, jež bude obsahovat příkazy, které by spustil místní mailer v případě, že by se realizovalo skutečné spojení pomocí protokolu SMTP. Protokol BSMTP se často užívá v sítích typu store-and-forward (tj. například u sítí na bázi protokolu UUCP), aby se ušetřilo místo na disku. Vzorový soubor `transports` uvedený v dodatku B obsahuje přenos nazvaný `bsmtp`, který generuje v adresáři fronty dílčí dávky protokolu BSMTP. Později musí být dílčí dávky vloženy do konečných dávek pomocí skriptu příkazového interpretu, jenž do nich přidá příkazy *HELO* a *QUIT*.

Chcete-li u nějakého konkrétního spojení pomocí protokolu UUCP povolit transport *bsmtp*, musíte použít tzv. soubory metod (`method` files) (nahlédněte prosím na manuálovou stránku `smail`*(5)*, kde získáte více informací). Máte-li pouze jedno spojení na bázi protokolu UUCP

a pokud používáte směrovač chytřejšího hostitele, pak posílání dávek protokolu SMTP povolíte tak, že konfigurační proměnné *smart_transport* přiřadíte místo hodnoty *uux* hodnotu *bsmtp*.

Chcete-li přijímat dávky protokolu SMTP po spojení na bázi protokolu UUCP, musíte se ujistit, že máte příkaz na opětovné rozložení dávek, kterému bude vzdálený systém posílat své dávky. Pokud vzdálený systém také používá program `smail`, pak budete muset vytvořit odkaz `rsmtp` na program `smail`. Jestliže na vzdáleném počítači běží program `sendmail`, měli byste dodatečně nainstalovat skript příkazového interpretu s názvem `/usr/bin/bsmtp`, který bude provádět jednoduchý příkaz „`exec rsmtp`“ (protože symbolický odkaz nebude fungovat).

14.10 Přenosy na bázi protokolu SMTP

V současnosti podporuje program `smail` ovladač protokolu SMTP, který se používá k doručování pošty po sítích na bázi protokolu TCP.[6] Program `smail` je schopen doručit zprávu na libovolný počet adres nacházejících se na jednom samostatném hostiteli, jehož název je zadán buď jako plně kvalifikované doménové jméno, jež může být rozloženo pomocí síťového softwaru, nebo je zadán s respektováním tečkové notace, v tom případě ale musí být tento název uveden v hranatých závorkách. Adresy resolvované pomocí nějakého z ovladačů směrovačů BIND, *gethostbyname(3)* nebo *gethostbyaddr(3),* budou obvykle doručeny přenosu na bázi protokolu SMTP.

Ovladač protokolu SMTP se pokusí okamžitě spojit se vzdáleným hostitelem na portu *smtp*, který je uveden v souboru `/etc/services`. Není-li tento port dosažitelný nebo dojde-li k překročení časového limitu stanoveného pro dané spojení, bude nový pokus o doručení této zprávy učiněn někdy později.

Doručování v síti Internet vyžaduje, aby směrování na cílového hostitele bylo zadáno ve formě *route-addr*, která byla popsána ve 13. kapitole a nikoliv pomocí vykřičníkové notace.[7] Proto program `smail` převede adresu **user%host@gateway**, kde brána **gateway** je dosažitelná pomocí adresy **host1!host2!host3**, na adresu zdrojového směrování **<@host2,@host3:-user%host@gateway>**, jež bude poslána jako adresa na obálce na adresu **host1**. Chcete-li tyto transformace adres povolit (společně s ovladačem služby BIND), musíte v souboru `transports` upravit položku týkající se ovladače *smtp*. Vzorový soubor `transports` je uveden v dodatku B.

6 Autoři označují tuto podporu jako „jednoduchou“. Předesílají, že další verze programu `smail` bude úplně přepracovaná, což výrazně zlepší správu protokolu SMTP.

7 Nicméně v síti Internet se snažíme zabránit používání směrování. Místo toho by měla být použita plně kvalifikovaná doménová jména.

14.11 Omezení názvů hostitelů

Někdy je žádoucí odchytit veškeré nekvalifikované názvy hostitelů (to jsou takové názvy, které neobsahují název domény), které jsou zadány jako adresy odesilatele nebo příjemce. Je to užitečné například při provozování brány mezi dvěma sítěmi, z nichž jedna vyžaduje plně kvalifikovaná doménová jména. Na bráně spojující Internet se sítí na bázi protokolu UUCP by měly být nekvalifikované názvy hostitelů implicitně namapovány do domény **uucp**. Jakékoliv jiné úpravy adres jsou problematické.

Soubor `/usr/lib/smail/qualify` sděluje programu `smail`, které názvy domén se mají přiřadit jednotlivým názvům hostitelů. Položky souboru `qualify` jsou složeny z názvu hostitele, který začíná ve sloupci jedna a za ním následuje název domény. Řádky obsahující znak (#), který je uveden jako první znak, který není bílým místem, jsou považovány za komentáře. Položky se prohledávají v tom pořadí, v jakém jsou zadány.

Pokud neexistuje soubor `qualify`, nebude prováděno žádné překládání nekvalifikovaných názvů.

Speciální název hostitele * zastupuje všechny hostitele, což vám umožní mapovat do implicitní domény všechny hostitele, o kterých jste se v souboru `qualify` nezmínili. Tento název by měl být poslední položkou souboru.

Ve společnosti Virtual Brewery bylo pro všechny hostitele nastaveno, aby používali v adrese odesilatele plně kvalifikované názvy domény. Nekvalifikované adresy příjemce se překládají do implicitní domény **uucp**, takže v souboru *qualify* je nutná pouze jediná položka.

```
# /usr/lib/smail/qualify, poslední změna 12.února, 1994 janet
#
*              uucp
```

Kapitola

15

Program sendmail+IDA

15.1 Úvod do programu sendmail+IDA

Říká se, že *skutečným* správcem Unixu se stanete až v okamžiku, když se vám povede upravit soubor `sendmail.cf`. Ale také se říká, že musíte být blázen, když to chcete zkoušet podruhé.:-)

Program `sendmail` je neuvěřitelně výkonný nástroj. Většina lidí ho chápe jen velmi obtížně a ještě hůře se ho učí. Jakýkoliv program, jehož kompletní manuál (manuál k programu `sendmail` vydalo nakladatelství O'Reilly and Associates) má 792 stran, zcela pochopitelně vyděsí většinu lidí.

Program `sendmail+IDA` je odlišný. Odstraňuje nutnost úprav záhadného souboru `sendmail.cf` a správcům umožňuje definovat směrování a konfiguraci adres, které jsou specifické pro daný systém pomocí poměrně jednoduše srozumitelných podpůrných souborů tzv. *tabulek*. Zvolíte-li program `sendmail+IDA`, může vám to ušetřit spoustu hodin práce a stresu.

V porovnání s ostatními hlavními poštovními transportními agenty neexistuje asi nic, co by se nedalo pomocí programu `sendmail+IDA` udělat rychleji a jednodušeji. S jeho pomocí provedete poměrně jednoduše věci, které je potřeba vykonat před spuštěním normálního internetového systému nebo normálního systému na bázi protokolu UUCP. Konfigurace, jež jsou za normálních okolností velice obtížné, se v něm dají jednoduše vytvořit a spravovat.

V době sestavování této publikace je prostřednictvím anonymní služby FTP na adrese **vixen.cso.uiuc.edu** dostupná verze *`sendmail5.67b+IDA1.5`*. Je sestavena tak, aby na platformě operačního systému Linux nevyžadovala žádnou aktualizaci.

Všechny konfigurační soubory potřebné pro kompilaci zdrojového kódu programu `sendmail+IDA`, instalaci a spuštění pod operačním systémem Linux jsou obsaženy v balíku `newspak-2.2.tar.gz`. Ten je dostupný prostřednictvím anonymní služby FTP na adrese **sunsite.unc.edu** v adresáři `/pub/Linux/system/Mail`.

15.2 Konfigurační soubory – přehled

Tradiční program sendmail se nastavuje pomocí systémového konfiguračního souboru (obvykle je to soubor `/etc/sendmail.cf` nebo `/usr/lib/sendmail.cf`), který se nepodobá žádnému z jazyků, s nimiž jste se mohli až doposud setkat. Editace souboru *`sendmail.cf`* tak, aby se program sendmail choval požadovaným způsobem, může být hořkou zkušeností.

S programem `sendmail+IDA` je toto nelidské úsilí již věcí minulosti, protože všechny konfigurační volby jsou ovládány prostřednictvím tabulek, jež mají poměrně snadno pochopitelnou syntaxi. Tyto volby se konfigurují prostřednictvím programu `m4` (procesor maker) nebo programu `dbm` (databázový procesor), jenž se spustí na spoustě datových souborů prostřednictvím souboru `Makefile`, který je dodán společně se zdrojovými kódy.

V souboru `sendmail.cf` definujete pouze implicitní chování systému. Ve skutečnosti se všechny speciální úpravy uskutečňují prostřednictvím spousty doplňkových tabulek. Je to lepší, než přímo editovat soubor `sendmail.cf`. Seznam všech tabulek programu `sendmail` je uveden na obrázku 15.1.

`mailertable`	Definuje speciální chování pro vzdálené hostitele nebo domény.
`uucpxtable`	Přikazuje, že pošta bude doručována pomocí protokolu UUCP pouze na hostitele, kteří mají adresu uvedenou ve formátu systému DNS.
`pathtable`	Definuje cesty na bázi protokolu UUCP na vzdálené hostitele nebo domény, které splňují vykřičníkovou notaci.
`uucprelays`	Omezí cesty v souboru cest pouze na známé vzdálené hostitele.
`genericfrom`	Převede vnitřní adresy na všeobecně použitelné adresy, které jsou pro ostatní svět viditelné.
`xaliases`	Převede všeobecně použitelné adresy na korektní vnitřní adresy anebo převede korektní vnitřní adresy na všeobecně použitelné adresy.
`decnhetxtable`	Převede adresy odpovídající standardu definovanému v RFC-822 na adresy používané v sítích DEC.

Obrázek 15.1
Podpůrné soubory programu `sendmail`

15.3 Soubor sendmail.cf

U programu `sendmail+IDA` se soubor `sendmail.cf` neupravuje přímo, ale generuje se z konfiguračního souboru programu *m4*, který poskytuje správce místního systému. V následujícím výkladu ho budeme označovat jako soubor `sendmail.m4`.

Tento soubor obsahuje jen několik definic, ale převážně odkazy na tabulky, kde se provádí skutečná konfigurační práce. Obvykle je nutné zadat pouze:

- Názvy cest a názvy souborů používaných v místním systému.
- Název nebo názvy, pod kterými je systém znám v oblasti elektronické pošty.
- Požadovaný implicitní mailer (možná i požadovaný chytřejší hostitel).

Existuje velké množství parametrů, které lze definovat za účelem dosažení požadovaného chování místního systému nebo k potlačení předkompilovaných konfiguračních voleb. Tyto konfigurační volby jsou uloženy v souboru `ida/cf/OPTIONS`, který se nachází v adresáři zdrojových programů.

Soubor `sendmail.m4` může být při minimální konfiguraci (pro protokol UUCP nebo protokol SMTP platí, že veškerá nemístní pošta je přenášena na přímo připojeného chytřejšího hostitele) dlouhý jen 10 až 15 řádků, nepočítáme-li komentáře.

15.3.1 Příklad souboru sendmail.m4

Dále je uveden soubor `sendmail.m4` pro hostitele **vstout** ve společnosti Virtual Brewery. Hostitel **vstout** používá protokol SMTP ke komunikaci se všemi hostiteli sítě LAN pivovaru a veškerou poštu určenou pro ostatní místa posílá pomocí protokolu UUCP na hostitele **moria**, což je brána k síti Internet.

15.3.2 Obvykle používané parametry v souboru esndmail.m4

V souboru `sendmail.m4` se vždy vyžaduje pouze několik parametrů; \pokud vám vyhovují implicitní volby, můžete ostatní položky vynechat. Následující stati popisují podrobněji každou položku, která je uvedena v ukázkovém souboru `sendmail.m4`.

Položky definující cesty

```
dnl #define(LIBDIR,/usr/local/lib/mail)dnl # Kde jsou ostatní soubory?
```

Parametr *LIBDIR* definuje adresář, kde program `sendmail+IDA` očekává konfigurační soubory, různé tabulky dbm a speciální místní definice. V typické binární distribuci je tento parametr vestavěn do binárního souboru programu `sendmail` a tudíž nemusí být v souboru `sendmail.m4` explicitně určen.

Výše uvedený příklad obsahuje řádky uvozené řetězcem *dnl*. Tyto řádky jsou vlastně považovány za komentáře a jsou zde uvedeny pouze z informativních důvodů.

Chcete-li změnit umístění podpůrných souborů, odstraňte z výše uvedeného řádku uvozující řetězec *dnl*, nastavte cestu do požadovaného adresáře a znovu vytvořte a nainstalujte soubor `sendmail.cf`.

```
dnl #------------------SAMPLE SENDMAIL.M4 FILE------------------
dnl # (the string 'dnl' is the m4 equivalent of commenting out a line)
dnl # you generally don't want to override LIBDIR from the compiled in
paths
dnl #define(LIBDIR,/usr/local/lib/mail)dnl  # where all support files go
define(LOCAL_MAILER_DEF, mailers.linux)dnl  # mailer for local delivery
define(POSTMASTERBOUNCE)dnl                 # postmaster gets bounces
define(PSEUDODOMAINS, BITNET UUCP)dnl       # don't try DNS on these
dnl #-------------------------------------------------------------
dnl #
define(PSEUDONYMS, vstout.vbrew.com vstout.UUCP vbrew.com)
dnl                                         # names we're known by
define(DEFAULT_HOST, vstout.vbrew.com)dnl   # our primary 'name' for mail
define(UUCPNAME,vstout)dnl                  # our uucp name
dnl #
dnl #-------------------------------------------------------------
dnl #
define(UUCPNODES, |uuname|sort|uniq)dnl     # our uucp neighbors
define(BANGIMPLIESUUCP)dnl                  # make certain that uucp
define(BANGONLYUUCP)dnl                     #  mail is trated correctly
define(RELAY_HOST, moria)dnl        # our smart relay host
define(RELAY_MAILER, UUCP-A)dnl             # we reach moria via uucp
dnl #
dnl #-------------------------------------------------------------
```

```
dnl #
dnl # the various dbm lookup tables
dnl #
define(ALIASES, LIBDIR/aliases)dnl          # system aliases
define(DOMAINTABLE, LIBDIR/domaintable)dnl  # domainize hosts
define(PATHTABLE, LIBDIR/pathtable)dnl      # paths database
define(GENERICFROM, LIBDIR/generics)dnl     # generic from addresses
define(MAILERTABLE, LIBDIR/mailertable)dnl  # mailers per host or domain
define(UUCPXTABLE, LIBDIR/uucpxtable)dnl    # paths to hosts we feed
define(UUCPRELAYS, LIBDIR/uucpreslays)dnl   # short-circuit paths
dnl #
dnl #-------------------------------------------------------------
dnl #
dnl # include the 'real' code that makes it all work
dnl # (provided with the source code)
dnl #
include(Sendmail.mc)dnl                     # REQUIRED ENTRY !!!
dnl #
dnl #------------------ END OF SAMPLE SENDMAIL.M4 FILE------------------
```

Obrázek 15.2

Vzorový soubor `sendmail.m4` pro hostitele **vstout**

Definice místního maileru

```
# mailer pro místní doručování
define(LOCAL_MAILER_DEF, mailers.linux)dnl
```

Většina operačních systémů disponuje speciálním programem, který se stará o místní doručování pošty. V binárním kódu programu sendmail je již vestavěno několik typických programů pro většinu hlavních variant operačního systému Unix.

V Linuxu se musí explicitně definovat patřičný místní mailer, protože místní doručovací program nemusí bezpodmínečně být součástí distribuce, kterou jste nainstalovali. To lze provést přidáním parametru *LOCAL_MAILER_DEF* v souboru `sendmail .m4`.

Například u běžně používaného programu `deliver`,[1] který poskytuje tuto službu, byste měli parametru *LOCAL_MAILER_DEF* přiřadit hodnotu `mailers.linux`.

[1] Program `deliver` napsal Chip Salzenberg (**chip%tct@ateng.com**). Tento program je součástí některých distribucí a navíc ho lze najít v běžných anonymních FTP archívech, například na adrese **ftp.uu.net**.

Dále by měl být do adresáře, na který ukazuje parametr *LIBDIR*, nainstalován soubor s názvem `mailers.linux`. Tento soubor explicitně definuje program `deliver` ve vnitřním maileru *Mlocal* s patřičnými parametry, které zajistí, že program `sendmail` bude korektně doručovat poštu adresovanou do místního systému. Nejste-li expertem v problematice programu sendmail, nebudete zřejmě chtít následující příklad nijak upravovat.

```
# -- /usr/local/lib/mail/mailers.linux -
#      místní mailery pro Linux
Mlocal, P=/usr/bin/deliver, F=SlsmFDMP, S=10, R=25/10, A=deliver $u
Mprog,  P=/bin/sh,          F=lsDFMeuP,   S=10, R=10, A=sh -c $u
```

V souboru `Sendmail.mc` existuje také vestavěná implicitní volba pro program *`deliver`*, kterou lze začlenit do konfiguračního souboru `sendmail.cf`. Chcete-li ji uvést, nesmíte použít soubor `mailers.linux`, ale místo něho musíte v souboru `sendmail.m4` definovat následující záznam:

```
dnl --- (soubor sendmail.m4) ---
define(LOCAL_MAILER_DEF, DELIVER)dnl  # mailer pro místní doručování
```

Naštěstí makro `Sendmail.mc` předpokládá, že je doručovací program nainstalován v adresáři `/bin`, což ale neplatí pro balík Slackware 1.1.1 (který ho nainstaluje do adresáře `/usr/bin`). V takovémto případě se budete muset s tímto problémem vypořádat buď tak, že vytvoříte symbolický odkaz, nebo tak, že znovu sestavíte ze zdrojového kódu doručovací program, který bude moci být uložen v adresáři `/bin`.

Vypořádání se s odmítnutou poštou

```
# odmítnutá pošta se pošle postmasterovi
define(POSTMASTERBOUNCE)dnl
```

Většina systémů zjistila, že je důležité, aby byla pošta poslána a doručena s téměř 100% úspěšností. I když prověřování log-souborů démona `syslogd`*(8)* je užitečné, potřebují obvykle správci místní pošty vidět hlavičky odmítnuté pošty, aby mohli zjistit, zda byla pošta nedoručitelná z důvodu chyby na straně uživatele, nebo z důvodu konfigurační chyby na jednom ze zúčastněných systémů.

Pokud definujete parametr *POSTMASTERBOUNCE*, bude kopie každé odmítnuté zprávy poslána osobě, která má v daném systému na starosti účet **Postmaster**.

Bohužel nastavení tohoto parametru bude mít vliv i na *text* zprávy, protože i ten je poslán na účet **Postmaster**. Tento text může obsahovat osobní věci týkající se lidí, kteří používají poštu ve vašem systému.

Poštmistři by proto neměli číst poštu (přesněji by měli používat skripty příkazového interpretu, které z odmítnutých příchozích zpráv vymažou text), která jim není určena.

Položky vztahující se k systému DNS

```
# pro tyto domény nepoužívej DNS
define(PSEUDODOMAINS, BITNET UUCP)dnl
```

Existuje několik známých sítí, na které jsou v poštovních adresách často odkazy, ale které nejsou korektní z hlediska systému DNS. Pokud definujete parametr *PSEUDODOMAINS*, zabráníte tak zbytečným pokusům při vyhledávání pomocí systému DNS, jež by stejně nikdy neuspěly.

Definice názvů, pod nimiž je místní systém známý

```
define(PSEUDONYMS, vstout.vbrew.com  vstout.UUCP vbrew.com
dnl                                          # naše jména
# naše primární poštovní jméno
define(DEFAULT_HOST, vstout.vbrew.com)dnl
```

Systémy chtějí často skrýt svoji pravou totožnost, chtějí sloužit jako poštovní brány nebo dostávat a zpracovávat poštu adresovanou na ‘staré‘ názvy, pod nimiž byly dříve známé.

Parametr *PSEUDONYMS* uvádí seznam všech názvů hostitelů, pro něž bude místní systém přijímat poštu.

Parametr *DEFAULT_HOST* uvádí název hostitele, který se zobrazí ve zprávě, jež byla vytvořena na místním hostiteli. Je důležité, aby byl tento parametr nastaven na korektní hodnotu, protože jinak by veškerá zpáteční pošta byla nedoručitelná.

Položky vztahující se k protokolu UUCP

```
define(UUCPNAME, vstout)dnl              # naše UUCP-jméno
define(UUCPNODES, |uuname|sort|uniq)dnl  # naši UUCP-sousedé
define(BANGIMPLIESUUCP)dnl               # je nutné pro korektní
define(BANGONLYUUCP)dnl                  # zpracování UUCP-pošty
```

Systém DNS často zná určité místo jiným názvem, než pod jakým ho zná síť na bázi protokolu UUCP. Parametr *UUCPNAME* vám dovoluje definovat odlišný název hostitele, který se objeví v hlavičkách odcházející pošty určené pro protokol UUCP.

Parametr *UUCPNODES* definuje příkazy, které vrátí seznam názvů hostitelů v systémech, se kterými jste přímo spojeni prostřednictvím spojení na bázi protokolu UUCP.

Parametry *BANGIMPLIESUUCP* a *BANGONLYUUCP* zajišťují, že pošta adresovaná pomocí syntaxe s vykřičníkovou notací, která je charakteristická pro protokol UUCP, bude zpracována po způsobu protokolu UUCP, a nikoliv podle běžnějšího chování systému DNS, který se nyní používá v Internetu.

Přenosové systémy a mailery

```
define(RELAY_HOST, moria)dnl      # náš chytřejší hostitel
define(RELAY_MAILER, UUCP-A)dnl   # cesta k hostiteli moria přes UUCP
```

Mnoho správců systému se nechce obtěžovat s takovou konfigurací, která by umožnila spojení se všemi sítěmi (a tedy i se všemi systémy) ve všech sítích po celém světě. Místo toho budou raději posílat veškerou výstupní poštu na další systém, o kterém ví, že je věru „chytřejší“.

Parametr *RELAY_HOST* definuje název hostitele tohoto chytřejšího sousedního systému. Název je definován pro protokol UUCP.

Parametr *RELAY_MAILER* definuje mailer, který se použije pro přenos zpráv na chytřejšího hostitele.

Je třeba poznamenat, že nastavení těchto parametrů způsobí, že vaše odcházející pošta bude směrována na tento vzdálený systém, což samozřejmě bude mít vliv na zatížení tohoto vzdáleného systému. Dříve, než nakonfigurujete váš systém tak, aby využíval vzdálený systém jako všeobecně použitelnou bránu, musíte získat od poštmistra tohoto systému explicitní souhlas.

Různé konfigurační tabulky

```
define(ALIASES, LIBDIR/aliases)dnl            # systémové přezdívky
define(DOMAINTABLE, LIBDIR/domaintable)dnl    # tabulka domaintable
define(PATHTABLE, LIBDIR/pathtable)dnl        # databáze cest
define(GENERICFROM, LIBDIR/generics)dnl
define(MAILERTABLE, LIBDIR/mailertable)dnl    # mailery pro hostitele
  a domény
define(UUCPXTABLE, LIBDIR/uucpxtable)dnl
define(UUCPRELAYS, LIBDIR/uucprelays)dnl
```

Pomocí těchto maker můžete změnit adresáře, ve kterých program `sendmail+IDA` hledá různé tabulky dbm, jež definují „skutečné“ chování systému. Obvykle je moudré nechat tyto tabulky v adresáři *LIBDIR*.

Hlavní soubor Sendmail.mc

```
include(Sendmail.mc)dnl                    # NUTNÝ ZÁZNAM!!!
```

Autoři programu `sendmail+IDA` dodávají soubor `Sendmail.mc`, který obsahuje pravou „podstatu“ toho, co se později stane se souborem `sendmail.cf`. Pravidelně jsou uvolňovány nové verze, které obsahují opravené chyby a přidávají další funkce, aniž by vyžadovaly plnou verzi a překompilování programu `sendmail` ze zdrojových kódů.

Je velice důležité, abyste tento soubor *neupravovali*.

Jaké položky jsou tedy vyžadovány?

Pokud nepoužíváte žádnou z doplňkových tabulek dbm, doručí program `sendmail+IDA` poštu v souladu s nastavenou volbou *DEFAULT_MAILER* (eventuálně v souladu s nastavenými volbami *RELAY_HOST* a *RELAY_MAILER*), která je definována v souboru *sendmail.m4*. Tento soubor se používá ke generování souboru `sendmail.cf`. Toto chování lze jednoduše potlačit pomocí záznamů uvedených v tabulce `domaintable` nebo `uucpxtable`.

Všeobecný systém umístěný v Internetu a využívající služeb systému DNS nebo systém založený pouze na protokolu UUCP, jenž směruje veškerou poštu pomocí protokolu UUCP na chytřejšího hostitele, který je definován prostřednictvím volby *RELAY_HOST*, nebude pravděpodobně vyžadovat žádné specifické záznamy v tabulkách.

Všechny systémy by měly mít ve skutečnosti nastaveny makra *DEFAULT_HOST* a *PSEUDONYMS*, která definují kanonický název daného systému a jeho přezdívky, a dále by měly mít nastaveno makro *DEFAULT_MAILER*. Máte-li nastaveny volby *RELAY_HOST* a *RELAY_MAILER*, nebudete muset tyto implicitní volby nastavovat, protože budou fungovat automaticky.

U hostitelů na bázi protokolu UUCP bude pravděpodobně potřeba nastavit parametr *UUCPNAME* na jejich oficiální název pro protokol UUCP. Dále pravděpodobně nastaví parametry *RELAY_HOST* a *RELAY_MAILER*, které povolují směrování přes chytřejšího hostitele prostřednictvím poštovní brány. Používaný poštovní transport je definován pomocí volby *RELAY_MAILER* a obvykle by měl být pro sítě na bázi protokolu UUCP nastaven na hodnotu *UUCP-A*.

Pokud váš systém pracuje pouze na bázi protokolu SMTP a komunikuje prostřednictvím systému DNS, měli byste změnit parametr *DEFAULT_MAILER* na hodnotu *TCP-A* a dále smazat řádky s parametry *RELAY_MAILER* a *RELAY_HOST*.

15.4 Přehled tabulek programu sendmail+IDA

Program `sendmail+IDA` podporuje množství tabulek, které vám umožňují potlačit implicitní chování programu `sendmail` (definované v souboru `sendmail.m4`) a dále vám umožňují definovat speciální chování pro zvláštní situace, vzdálené systémy a sítě. Tyto tabulky jsou později zpracovány pomocí programu `dbm` s využitím souboru `Makefile`, který je součástí distribuce.

Většina systémů bude potřebovat pouze několik těchto tabulek. Možná, že nebudete potřebovat vůbec žádné tabulky. Pokud váš systém tyto tabulky nevyžaduje, bude asi nejjednodušší, když je vytvoříte jako soubory s nulovou velikostí (pomocí příkazu `touch`) a použijete implicitní soubor `Makefile`, který najdete v adresáři *LIBDIR*. Bude to lepší, než kdybyste museli editovat vlastní soubor `Makefile`.

15.4.1 Tabulka mailertable

Tabulka `mailertable` definuje speciální zacházení s konkrétními hostiteli nebo doménami, které vychází z názvu vzdáleného hostitele nebo z názvu sítě. U systémů v síti Internet se často používá k výběru pomocného hostitele nebo k přenosu pošty nebo brány, jejichž prostřednictvím se můžete spojit se vzdálenými sítěmi. Dále se používá k definici konkrétního protokolu (UUCP nebo SMTP), který se bude používat. Systémy na bázi protokolu UUCP obvykle nebudou muset tento soubor používat.

Pořadí položek v souboru je důležité. Program `sendmail` čte tento soubor směrem shora dolů a zpracovává zprávu podle prvního odpovídajícího pravidla, které najde. Takže je dobré umístit nejjednoznačnější pravidla na začátek souboru a obecnější pravidla na konec souboru.

Předpokládejme, že chcete směrovat veškerou poštu určenou pro oddělení počítačových věd na Groucho Marx University pomocí protokolu UUCP na bránu **ada**. Aby to bylo možné, musíte mít v tabulce `mailertable` položku, která vypadá asi následovně:

```
# (soubor mailertable)
#
# poštu pro doménu.cs.groucho.edu posílej přes UUCP-hostitele ada
UUCP-A,ada          .cs.groucho.edu
```

Dále předpokládejme, že chcete z důvodu rozlišení adres a z důvodu doručování veškerou poštu určenou pro rozsáhlejší doménu **groucho.edu** směrovat na zvláštní bránu **bighub**. Doplněné záznamy by mohly v tabulce `mailertable` vypadat asi takto:

```
# (soubor mailertable)
#
```

```
# poštu pro doménu cs.groucho.edu posílej přes UUCP-hostitele ada
UUCP-A,ada.cs.groucho.edu
#
# poštu pro doménu cs.groucho.edu posílej přes UUCP-hostitele bighub
UUCP-A,bighub.cs.groucho.edu
```

Jak jsme se již zmínili, je pořadí velmi důležité. Kdybychom zaměnili pořadí těchto dvou pravidel, pak by se veškerá pošta určená pro doménu **.cs.groucho.edu** směrovala na obecnější bránu **bighub** místo toho, aby putovala na jednoznačně určenou bránu **ada**, což bylo naším záměrem.

```
# (soubor mailertable)
#
# poštu pro doménu.groucho.edu posílej přes UUCP-hostitele bighub
UUCP-A,bighub .groucho.edu
#
# (další řádek je zbytečný,
# protože je použito předchozí pravidlo)
UUCP-A,ada    .cs.groucho.edu
```

Ve výše uvedených příkladech je definován mailer *UUCP-A*, který sděluje programu `sendmail`, aby používal k doručování pošty protokol UUCP s hlavičkami respektujícími doménový systém.

Čárka mezi mailerem a vzdáleným systémem sděluje maileru, že má zprávy směrovat na bránu **ada**, která se postará o rozlišení adres a doručení zpráv.

Záznamy v tabulce `mailertable` mají následující syntaxi:

```
mailer delimiter relayhost                    host_or_domain
```

Existuje několik druhů mailerů. Rozdíl mezi nimi obvykle spočívá v tom, jakým způsobem zacházejí s adresami. Běžné mailery jsou *TCP-A* (pro protokol TCP/IP s adresami typickými pro síť Internet), *TCP-U* (pro protokol TCP/IP s adresami typickými pro sítě na bázi protokolu UUCP) a *UUCP-A* (pro sítě na bázi protokolu UUCP, ale s adresami typickými pro síť Internet).

Znak nacházející se na levé straně řádky tabulky `mailertable`, který odděluje mailer od části s názvem hostitele, definuje způsob, jakým bude tabulka `mailertable` upravovat příslušnou adresu. Jedinou věcí, kterou je zde třeba upravit, je přepsat adresu na obálce (aby se pošta mohla dostat do vzdáleného systému). Přepisování čehokoliv jiného se nedoporučuje, protože by to mohlo vést k poškození konfigurace pošty.

! Znak vykřičník odstraní název hostitele příjemce předtím, než je zpráva směrována na mailer. Tento oddělovač by se měl použít v tom případě, kdy chcete nařídit odeslání pošty do špatně nakonfigurovaného vzdáleného systému.

, Znak čárka adresu vůbec neupravuje. Zpráva je většinou směrována pomocí zadaného maileru na zadanou bránu.

: Znak dvojtečka odstraní název hostitele příjemce pouze v případě, že mezi vaším hostitelem a cílovým hostitelem existují nějací mezilehlí hostitelé. Bude-li tedy adresa **foo!bar!joe**, bude hostitel **foo** odstraněn, avšak bude-li adresa **xyzzy!janet**, pak zůstane nezměněná.

15.4.2 Tabulka uucpxtable

Pošta na hostitele s plně kvalifikovanými doménovými jmény je obvykle doručena buď prostřednictvím směrování charakteristického pro síť Internet (protokol SMTP) s využitím systému DNS, nebo prostřednictvím přenosových hostitelů (bran). Tabulka `uucpxtable` zajistí doručení pomocí směrování na bázi protokolu UUCP, protože převede doménový název na bezdoménový název vzdáleného hostitele, který je charakteristický pro protokol UUCP.

Tato tabulka se často používá v případech, kdy jste dopravcem pošty pro daný systém nebo doménu, nebo když chcete poslat poštu pomocí přímého a spolehlivého spojení na bázi protokolu UUCP a nikoliv přes implicitní mailer a několik mezilehlých systémů a sítí, kde by eventuálně mohlo dojít k několika skokům.

Systémy na bázi protokolu UUCP komunikující se svými UUCP-sousedy, kteří používají doménové poštovní hlavičky, by měly používat tento soubor, aby si vynutili doručení pošty prostřednictvím přímého spojení typu point-to-point na bázi protokolu UUCP, protože jinak by musely používat méně přímé směrování prostřednictvím parametrů *RELAY_MAILER* a *RELAY_HOST* nebo pomocí parametru *DEFAULT_MAILER*.

Internetové systémy, které nepoužívají komunikaci na bázi protokolu UUCP, nebudou muset tabulku `uucpxtable` používat.

Předpokládejme, že poskytujete doručovací službu systému s názvem **sesame.com** prostřednictvím systému DNS a systému **sesame** prostřednictvím map protokolu UUCP. Do tabulky `uucpxtable` budete muset vložit následující položku, abyste poštu směrovanou na jejich hostitele mohli poslat po přímém spojení na bázi protokolu UUCP.

```
#=============== /usr/local/lib/mail/uucpxtable ==============
# Pošta poslaná na adresu joe@sesame.com bude přepsána na sesame!joe
# a poté doručena přes UUCP
```

```
#
sesame sesame.com
#
#-----------------------------------------------------------
```

15.4.3 Tabulka pathtable

Tabulka `pathtable` slouží k definici explicitního směrování na vzdálené hostitele nebo sítě. Soubor tabulky `pathtable` by měl mít syntaxi souboru cest `paths` a měl by být setříděn podle abecedy. Dvě pole na každé řádce musí být oddělena znakem tabulátoru, jinak by si mohl program `dbm` stěžovat.

Většina systémů nebude potřebovat žádné záznamy v tabulce `pathtable`.

```
#=============== /usr/local/lib/mail/pathtable ==============
#
# this is a pathalias-style paths file to let you kick mail to
# UUCP neighbors to the direct UUCP path so you don't have to
# go the long way through your smart host that takes other traffic
#
# you want real tabs on each line or m4 might complain
#
# route mail through one or more intermediate sites to a remote
# system using UUCP-style addressing
#
sesame!ernie!%s                 ernie
#
# forwarding to a system that is a UUCP neighbor of a reachable
# internet site.
```

```
#

swim!%s@gcc.groucho.edu         swim

# The following sends all mail for two networks through different

# gateways (see the leading '.' ?).

# In this example, "uugate" and "byte" are specific systems that
server

# as mail gateways to the .UUCP and .BITNET pseudo-domains respec-
tively

#

%s@uugate.groucho.edu           .UUCP
byte!%s@mail.shift.com          .BITNET
```

15.4.4 Tabulka domaintable

Tabulka `domaintable` se používá k dosažení jistého chování při vyhledávání pomocí systému DNS. Správci povoluje vytvořit zkrácené názvy, které lze použít u běžných odkazů na systémy nebo domény. Tyto zkrácené názvy tabulka `domaintable` automaticky nahradí odpovídajícími plnými názvy. Dále může být tabulka použita k nahrazení nekorektních názvů hostitelů nebo domén „korektními“ informacemi.

Většina systémů nebude potřebovat žádné záznamy v tabulce `domaintable`.

Následující příklad ukazuje, jak nahradit nekorektní adresu, kterou se snaží lidé v poště používat, korektní adresou:

```
#=============== /usr/local/lib/mail/domaintable ==============
#
#
brokenhost.correct.domain                  brokenhost.wrong.domain
#
#
#=============== konec domaintable ============================
```

15.4.5 Tabulka aliases

Přezdívky umožňují hned několik věcí:

- Pro směrování pošty poskytují zkrácený název nebo veřejný název, takže pošta může být doručena jedné nebo více osobám.
- Vyvolávají program, kterému je poštovní zpráva předána na vstup.
- Posílají poštu do souboru.

Všechny systémy vyžadují přezdívky pro účty **Postmaster** a **MAILER-DAEMON**, aby splňovaly standardy definované v RFC.

Při definování přezdívek, které vyvolávají programy nebo zapisují do programů, buďte vždy opatrní, protože program `sendmail` má zpravidla nastavenou hodnotu uid na **root**.

Změny provedené v souboru `aliases` se projeví až po spuštění následujícího příkazu, který vytvoří požadované tabulky dbm:

```
# /usr/lib/sendmail -bi
```

Totéž lze také obvykle provést spuštěním příkazu `newaliases` z programu `cron`.

Detaily týkající se poštovních přezdívek můžete nalézt na manuálové stránce *aliases(5)*.

```
#=============== /usr/local/lib/mail/aliases ==============
#
# příklad obvyklých typů přezdívek
#
usenet:         janet                 # přezdívka pro jednu osobu
admin:          joe, janet            # přezdívka pro více lidí
newspak-users:  :include:/usr/lib/lists/newspak
                                      # příjemci jsou v souboru
changefeed:     | /usr/local/lib/gup  # přezdívka, která spustí
                                        program
complaints:     /var/log/complaints   # přezdívka, která zapíše
                                        poštu do souboru
#
# Následující přezdívky jsou podle RFC povinné.
# Je důležité, aby se pošta směrovaná na tyto adresy dostala do
  přihrádky někoho, kdo čte poštu pravidelně.
```

```
#
postmaster:     root                    # nutné
MIALER-DAEMON:  postmaster              # nutné
#
#-------------------------------------------------------------
```

15.4.6 Zřídka používané tabulky

Následující tabulky jsou sice dostupné, ale používají se jen velmi zřídka. Chcete-li o nich získat více detailů, nahlédněte prosím do dokumentace, která je součástí zdrojového kódu programu `sendmail+IDA`.

`uucprelays` — Soubor `uucprelays` se používá ve zvláště známých systémech ke „zkrácení“ cesty protokolu UUCP. Je to lepší, než používat víceskokové nebo nespolehlivé cesty, které jsou vygenerovány z map protokolu UUCP tabulkou `pathalias`.

`genericfrom` a `xaliases` — Soubor `genericfrom` skrývá před okolním světem názvy místních uživatelů a jejich adresy. Toho může dosáhnout tak, že převede jména místních uživatelů na obecné adresy odesilatele, které neodpovídají interním jménům uživatelů.

Přidružená utilita s názvem `xalparse` automatizuje generování tabulky `genericfrom` a souboru `aliases`, takže převod jak odcházejícího, tak i přicházejícího jména uživatele se může dít z hlavního souboru `xaliases`.

`decnetxtable` — Tabulka `decnetxtable` přepisuje doménové adresy na adresy vyhovující standardu, který se používá v sítích DEC. Tento převod je podobný přepisu bezdoménových adres na doménové adresy, které lze použít v sítích s protokolem SMTP.

15.5 Instalace programu sendmail

V této stati se zaměříme na to, jak lze nainstalovat typickou distribuci binárního kódu programu `sendmail+IDA` a projdeme si kroky, které je zapotřebí učinit, aby byl tento program lokalizovaný a funkční.

Současnou verzi binárního kódu programu `sendmail+IDA` pro platformu Linux můžete získat na adrese **sunsite.unc.edu** v adresáři `/pub/Linux/system/mail`. Dokonce i když máte dřívější verzi programu `sendmail`, vřele doporučuji přejít na verzi `sendmail5.67b+IDA1.5`[2], protože všechny požadované aktualizace specifické pro platformu Linux jsou obsaženy ve zdrojovém kódu „vanilla sources" a navíc bylo odstraněno i několik závažných bezpečnostních děr, které se nacházely ve verzích, jež byly uvolněny před 1. prosincem 1993.

Jestliže sestavujete program `sendmail` ze zdrojového kódu, měli byste postupovat podle instrukcí uvedených v souborech `README`, které jsou součástí distribuce zdrojového kódu. Aktuální verze zdrojového kódu programu `sendmail+IDA` jsou k dispozici na adrese **vixen.cso.uiuc.edu**. Chcete-li sestavit program `sendmail+IDA` na linuxové platformě, budete k tomu potřebovat i konfigurační soubory specifické pro operační systém Linux, které se nacházejí v balíku `newspak-2.2.tar.gz`. Tento balík je k dispozici na adrese **sunsite.unc.edu** v adresáři `/pub/Linux/system/Mail`.

Pokud jste již dříve nainstalovali program `smail` nebo nějakého jiného poštovního doručovacího agenta, budete pravděpodobně chtít z bezpečnostních důvodů odstranit (nebo přejmenovat) veškeré soubory vztahující se k programu `smail`.

15.5.1 Rozbalení distribuce binárního kódu

Nejdříve musíte rozbalit archívní soubor na nějaké bezpečné místo:

```
$ gunzip -c sendmail5.65b+IDA1.5+mailx5.3b.tgz | tar xvf -
```

Máte-li „novější" verzi programu `tar`, například z poslední verze balíku Slackware, stačí pravděpodobně napsat pouze `tar -zxvf filename.tgz`, kde `filename` je název souboru, a dostanete stejný výsledek.

Při rozbalování daného archívu se vytvoří adresář s názvem `sendmail5.65b+IDA1.5-+mailx5.3b`. V tomto adresáři naleznete kompletní instalaci programu `sendmail+IDA` a navíc v něm bude i binární kód programu `mailx`, což je uživatelský agent. Všechny cesty k souborům nacházející se pod tímto adresářem určují umístění, kam by se měly soubory nainstalovat, takže je vhodné navrhnout takový příkaz `tar`, který přesune tyto soubory na požadované místo.

```
# cd sendmail5.65b+IDA1.5+mailx5.3b
# tar cf - . | (cd /; tar xvvpoof -)
```

2 Pozn. korektora: Poslední verze programu sendmail má číslo 8.9.1.

15.5.2 Sestavení souboru sendmail.cf

Abyste mohli sestavit soubor `sendmail.cf`, který bude přizpůsoben potřebám vašeho systému, musíte nejdříve napsat soubor `sendmail.m4`, jenž zpracujete pomocí programu `m4`. V adresáři `/usr/local/lib/mail/CF` naleznete ukázkový soubor s názvem `sample.m4`. Zkopírujte ho do souboru *název_vašeho_hostitele*`.m4` a upravte ho tak, aby odrážel situaci panující ve vašem systému.

Vzorový soubor je nastaven pro systém založený pouze na protokolu UUCP, který má doménové hlavičky a komunikuje s chytřejším hostitelem. U systémů podobných tomuto nastavení se musí v tomto souboru provést jen mírné úpravy.

V této stati vám poskytnu pouze krátký přehled maker, které musíte pozměnit. Chcete-li kompletní popis jejich funkce, nahlédněte prosím do dřívějších statí, kde jsme se zabývali souborem `sendmail.m4`.

LOCAL_MAILER_DEF Určuje soubor, ve kterém jsou definovány mailery pro místní doručování pošty. Chcete-li vědět, co vše do tohoto souboru patří, nahlédněte prosím do stati „Definice místního maileru“.

PSEUDONYMS Definuje všechny názvy, pod kterými je známý váš místní hostitel.

DEFAULT_HOST Sem zadejte svoje plně kvalifikované doménové jméno. Tento název se objeví jako název vašeho hostitele ve veškeré odcházející poště.

UUCPNAME Sem zadejte svůj nekvalifikovaný název hostitele.

RELAY_HOST a *RELAY_MAILER*
Jestliže komunikujete se svým chytřejším hostitelem pomocí protokolu UUCP, nastavte parametr *RELAY_HOST* na název pro protokol UUCP, který bude definovat vašeho „chytřejšího přenosového“ souseda v síti UUCP. Jestliže chcete doménové hlavičky, nastavte mailer na hodnotu *UUCP-A*.

DEFAULT_MAILER Jestliže jste v Internetu a komunikujete pomocí systému DNS, měli byste tento parametr nastavit na hodnotu *TCP-A*. Řetězec *TCP-A* sdělí programu `sendmail`, aby použil mailer *TCP-A*, který doručuje poštu pomocí protokolu SMTP, a aby použil pro adresu na obálce normální adresu kompatibilní se standardy uvedenými v RFC. Systémy v Internetu pravděpodobně nebudou muset definovat parametry *RELAY_HOST* a *RELAY_MAILER*.

Chcete-li vytvořit soubor `sendmail.cf`, spusťte následující příkaz:

```
# make název_vašeho_hostitele.cf
```

Tento příkaz zpracuje váš soubor *název_vašeho_hostitele*`.m4` a vytvoří z něho soubor *název_vašeho_hostitele*`.cf`.

Dále musíte otestovat, zda vytvořený konfigurační soubor provádí přesně to, co jste od něj očekávali. Testování konfiguračního souboru bude vysvětleno v následujících dvou statích.

Jste-li teď již spokojeni s chováním konfiguračního souboru, zkopírujte ho pomocí následujícího příkazu na patřičné místo:

```
# cp název_vašeho_hostitele.cf /etc/sendmail.cf
```

Od tohoto okamžiku může váš program `sendmail` začít pracovat. Do příslušného startovního souboru (obvykle je to soubor `/etc/rc.inet2`) vložte následující řádku. Chcete-li tento proces rozběhnout okamžitě, můžete ručně spustit následující příkaz.

```
# /usr/lib/sendmail -bd -q1h
```

15.5.3 Testování konfiguračního souboru sendmail.cf

Chcete-li spustit program `sendmail` v „testovacím“ režimu, spusťte ho s parametrem `-bt`. Implicitní konfigurační soubor se nazývá `sendmail.cf`, ten je také nainstalován ve vašem systému. Alternativní soubor můžete otestovat pomocí volby `-C` *filename*, kde *filename* je název daného souboru.

V následujících příkladech budeme testovat soubor `vstout.cf`, což je konfigurační soubor vygenerovaný ze souboru `vstout.m4`, který je zobrazen na obrázku 15.2.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
>
```

Následující testy zaručí, že je program `sendmail` schopen doručit veškerou poštu uživatelům ve vašem systému. Ve všech případech by měl být výsledek testu stejný a měl by ukazovat na místní název systému s nastaveným mailerem *LOCAL*.

Nejprve otestujme, jakým způsobem by měla být doručena pošta místnímu uživateli.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 me
rewrite: ruleset  3   input: me
rewrite: ruleset  7   input: me
rewrite: ruleset  9   input: me
rewrite: ruleset  9 returns: < me >
rewrite: ruleset  7 returns: < >, me
rewrite: ruleset  3 returns: < >, me
rewrite: ruleset  0   input: < >, me
rewrite: ruleset  8   input: < >, me
rewrite: ruleset 20   input: < >, me
rewrite: ruleset 20 returns: < >, @ vstout . vbrew . com , me
rewrite: ruleset  8 returns: < >, @ vstout . vbrew . com , me
rewrite: ruleset 26   input: < >, @ vstout . vbrew . com , me
rewrite: ruleset 26 returns: $# LOCAL @$ vstout . vbrew . com $: me
rewrite: ruleset  0 returns: $# LOCAL @$ vstout . vbrew . com $: me
>
```

Výstup ukazuje, jak program `sendmail` vnitřně zpracovává adresu. Tato adresa je předána různým skupinám pravidel, které ji analyzují, spustí postupně další skupiny pravidel a rozloží ji na jednotlivé komponenty.

V našem příkladu jsme předali adresu **me** skupinám pravidel 3 a 0 (to je význam hodnot 3, 0, které jsou uvedeny před adresou). Poslední řádka ukazuje rozloženou adresu, která je výsledkem použití skupiny pravidel 0. Tato rozložená adresa se skládá z maileru, pomocí něhož by měla být daná zpráva doručena, a ze jmen hostitele a uživatele, které jsou maileru předány.

Dále otestujme poštu adresovanou uživateli vašeho systému, adresace bude splňovat syntaxi protokolu UUCP.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 vstout!me
rewrite: ruleset  3  input: vstout ! me
```

```
[...]
rewrite: ruleset   0   returns: $# LOCAL $@ vstout . vbrew . com $: me
>
```

Dále otestujme adresaci pošty uživateli vašeho systému, adresace bude splňovat syntaxi typickou pro Internet, tedy pošta bude adresována na váš plně kvalifikovaný název hostitele.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 me@vstout.vbrew.com
rewrite: ruleset  3 input: me @ vstout . vbrew . com
[...]
rewrite: ruleset  0 returnes: $# LOCAL $@ vstout . vbrew . com $: me
>
```

Výše uvedené dva testy byste měli zopakovat pro každý název, který jste uvedli v parametrech *PSEUDONYMS* a *DEFAULT_NAME*, jež se nacházejí v souboru `sendmail.m4`.

A konečně otestujme, zda můžete posílat poštu na svého přenosového hostitele (na svou bránu).

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 fred@moria.com
rewrite: ruleset   3 input:   fred @ moria . com
rewrite: ruleset   7 input:   fred @ moria . com
rewrite: ruleset   9 input:   fred @ moria . com
rewrite: ruleset   9 returns: < fred > @ moria . com
rewrite: ruleset   7 returns: < @ moria . com > , fred
rewrite: ruleset   3 returns: < @ moria . com > , fred
rewrite: ruleset   0 input:   < @ moria . com > , fred
rewrite: ruleset   8 input:   < @ moria . com > , fred
rewrite: ruleset   8 returns: < @ moria . com > , fred
rewrite: ruleset  29 input:   < @ moria . com > , fred
rewrite: ruleset  29 returns: < @ moria . com > , fred
rewrite: ruleset  26 input:   < @ moria . com > , fred
```

```
rewrite: ruleset  25  input:    < @ moria . com > , fred
rewrite: ruleset  25  returns:  < @ moria . com > , fred
rewrite: ruleset   4  input:    < @ moria . com > , fred
rewrite: ruleset   4  returns:  fred @ moria . com
rewrite: ruleset  26  returns:  < @ moria . com > , fred
rewrite: ruleset   0  returns:  @# UUCP-A $@ moria $:
  < @ moria . com > , fred
>
```

15.5.4 Všechno dohromady – integrované testování souboru sendmail.cf a jeho tabulek

V tomto okamžiku máte již ověřeno, že pošta dodržuje požadované implicitní chování a že můžete posílat i přijímat korektně adresovanou poštu. K dokončení instalace vám už chybí jen vytvořit patřičné tabulky dbm, abyste získali požadované konečné výsledky.

Po vytvoření tabulky nebo tabulek, které jsou potřebné pro váš systém, je musíte zpracovat pomocí programu `dbm`. To lze provést tak, že v adresáři s tabulkami napíšete příkaz `make`.

Pokud provozujete systém pouze na bázi protokolu UUCP, nemusíte vytvářet žádnou z tabulek, o kterých se zmiňuje soubor `README.linux`. Stačí pouze vytvořit soubory tabulek s nulovou velikostí, aby soubor `Makefile` pracoval správně.

Pokud provozujete systém pouze na bázi protokolu UUCP a pokud komunikujete i s jinými systémy, než je váš chytřejší hostitel, budete muset pro každý z těchto systémů vložit do tabulky `uucpxtable` (protože jinak by pošta adresovaná na ně šla přes chytřejšího hostitele) příslušnou položku a znovu spustit příkaz `dbm`, aby se tabulka `uucpxtable` zaktualizovala.

Nejprve se musíte ubezpečit, že pošta poslaná na tyto systémy přes přenosového hostitele, který je definován pomocí parametru *RELAY_HOST*, bude poslaná pomocí maileru, který je definován v parametru *RELAY_MAILER*.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 fred@sesame.com
rewrite: ruleset    3      input:   fred @ sesame . com
rewrite: ruleset    7      input:   fred @ sesame . com
rewrite: ruleset    9      input:   fred @ sesame . com
```

```
rewrite: ruleset    9    returns:   < fred > @ sesame . com
rewrite: ruleset    7    returns:   < @ sesame . com > , fred
rewrite: ruleset    3    returns:   < @ sesame . com > , fred
rewrite: ruleset    0      input:   < @ sesame . com > , fred
rewrite: ruleset    8      input:   < @ sesame . com > , fred
rewrite: ruleset    8    returns:   < @ sesame . com > , fred
rewrite: ruleset    29     input:   < @ sesame . com > , fred
rewrite: ruleset    29   returns:   < @ sesame . com > , fred
rewrite: ruleset    26     input:   < @ sesame . com > , fred
rewrite: ruleset    25     input:   < @ sesame . com > , fred
rewrite: ruleset    25   returns:   < @ sesame . com > , fred
rewrite: ruleset    4      input:   < @ sesame . com > , fred
rewrite: ruleset    4    returns:   fred @ sesame . com
rewrite: ruleset    26   returns:   < @ sesame . com > , fred
rewrite: ruleset    0    returns:   @# UUCP-A $@ sesame $: < @ sesa-
me . com > , fred
>
```

Máte-li i jiné sousedy v síti UUCP, než je váš přenosový hostitel definovaný pomocí parametru *RELAY_HOST*, musíte se ujistit, že se pošta adresovaná na tyto hostitele bude chovat korektně. Pošta adresovaná pomocí vykřičníkové notace na hostitele, se kterým komunikujete na bázi protokolu UUCP, by měla být poslána přímo na tohoto hostitele (pokud to výslovně nezakážete prostřednictvím položky uvedené v tabulce `domaintable`). Předpokládejme, že hostitel **swim** je vaším přímým sousedem sítě UUCP. V tom případě, pokud programu `sendmail` pošlete adresu **swim!fred**, by se měl objevit následující výsledek:

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 swim!fred
rewrite: ruleset  3   input: swim ! fred
[...vynecháno...]
rewrite: ruleset  0   returnes: $# UUCP $@ swim $: < >, fred
>
```

Pokud máte v tabulce `uucpxtable` uvedeny položky, jež přikazují doručit poštu pomocí protokolu UUCP určitým sousedům sítě UUCP, kteří používají u své pošty doménové hlavičky splňující standardy definované v síti Internet, je potřeba otestovat také tyto transportní cesty.

```
# /usr/lib/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE
Enter <ruleset> <address>
[Note: No initial ruleset 3 call]
> 3,0 dude@swim.2birds.com
rewrite: ruleset 3 input: dude @ swim . 2birds . com
[...vynecháno...]
rewrite: ruleset 0 returns: $# UUCP $@ swim . 2 birds $: < >, dude
>
```

15.6 Administrace a jednoduché poštovní triky

Když jsme nyní probrali teorii konfigurace, instalace a testování programu `sendmail+IDA`, zastavme se na chvíli u věcí, které se často stávají v každodenním životě správce pošty.

Vzdálené systémy se někdy porouchají. Modemy nebo telefonní linky selžou, definice systému DNS jsou díky chybě člověka nastaveny chybně. Sítě naprosto neočekávaně spadnou. V takových situacích musí správci pošty vědět, jak mají rychle, efektivně a *bezpečně* zasáhnout, aby pošta stále mohla chodit přes alternativní směrování, dokud vzdálené systémy nebo poskytovatelé služeb neobnoví normální provoz.

Zbytek této kapitoly budeme věnovat řešení nejčastěji se vyskytujících případů „nouzových stavů elektronické pošty“.

15.6.1 Směrování pošty na přenosového hostitele

Směrujete-li poštu pro konkrétního hostitele nebo doménu na určitého přenosového hostitele, používáte k tomu obvykle tabulku `mailertable`.

Chcete-li například směrovat poštu pro doménu **backwood.org** na její bránu UUCP s názvem **backdoor**, pak vložíte do tabulky `mailertable` následující záznam:

```
UUCP-A,backdoor    backwood.org
```

15.6.2 Vynucené poslání pošty do špatně zkonfigurovaných vzdálených systémů

Hostitelé v Internetu mají často problémy s posláním pošty do špatně zkonfigurovaných vzdálených systémů. Existuje několik variant tohoto problému, avšak obecným příznakem těchto problémů je, že pošta je vzdáleným systémem odmítnuta nebo se do něj vůbec nedostane.

Tyto problémy staví správce místního systému do těžké pozice, protože vaše uživatele zpravidla nezajímá, že osobně nespravujete všechny systémy na celém světě (nebo že nevíte, jak přimět vzdáleného správce k tomu, aby tento problém odstranil). Oni pouze vědí, že se jejich pošta nemůže prokousat na druhou stranu k příslušnému příjemci a že vy jste vhodnou osobou, které by si mohli stěžovat.

Konfigurace vzdáleného systému je jejich problém, a ne váš. Ve všech případech se ale ujistěte, že *nepoškodíte* svůj systém jen kvůli tomu, abyste mohli komunikovat se špatně nakonfigurovaným vzdáleným systémem. Nemůžete-li sdělit poštmistrovi vzdáleného systému, aby v časově přijatelném období opravil svoji konfiguraci, máte jen dvě možnosti.

- Většinou je možné si úspěšně vynutit poslání pošty do vzdáleného systému, ale protože je vzdálený systém špatně nakonfigurovaný, nemusí na vzdáleném konci správně fungovat odpovědi…ale to už je vlastně problém správce vzdáleného systému.

  Špatné hlavičky na obálce vaší odcházející pošty můžete pro daného hostitele nebo doménu upravit pouze prostřednictvím položky v tabulce `domaintable`. Ta způsobí opravu nekorektních informací v poště pocházející z vašeho systému:

  ```
  braindead.correct.domain.com        braindead.wrong.domain.com
  ```

- Často špatně zkonfigurované systémy odmítnou přijetí pošty zpět do systému odesilatele a zobrazí působivou zprávu „that mail isn't for this site" (tato pošta není určena pro daný systém), protože tyto systémy nemají ve své konfiguraci správně nastaveny své přezdívky nebo jejich ekvivalenty pomocí parametru *PSEUDONYMS*. U veškeré pošty adresované na takovéto systémy je možné odstranit z adresy na obálce všechny názvy hostitelů a informace o doménách.

  Znak vykřičník (*!*) uvedený v následujícím záznamu tabulky `mailertable` doručí poštu na patřičný vzdálený systém. Tato pošta se bude jevit jejich programu `sendmail` stejně tak, jako kdyby byla vytvořena na daném vzdáleném systému. Pamatujte si, že tato položka mění pouze adresu na obálce, takže ve zprávě se bude i nadále zobrazovat správná zpáteční adresa.

  ```
  TCP!braindead.correct.domain.com    braindead.wrong.domain.com
  ```

Bez ohledu na to, zda se vám povede poslat poštu do špatně nakonfigurovaného systému, nebudete mít žádnou záruku, že uživatelé ze vzdáleného systému budou moci na vaši zprávu odpovědět (pamatujte, že adresy jsou poškozené...). Avšak v takovém případě si budou spíše stěžovat uživatelé vzdáleného systému svým správcům, nežli vaši uživatelé vám.

15.6.3 Jak dosáhnou toho, aby byla pošta poslána prostřednictvím protokolu UUCP

V ideálním světě (z perspektivy sítě Internet) by měli všichni hostitelé uvedeny záznamy v doménové jmenné službě (v systému DNS) a posílali by poštu s plně kvalifikovanými názvy hostitelů.

Pokud budete muset komunikovat s takovým systémem na bázi protokolu UUCP, můžete zajistit posílání pošty po spojení na bázi protokolu UUCP typu point-to-point a nikoliv prostřednictvím implicitního maileru, který by v podstatě pomocí tabulky `uucpxtable` vytvořil z plně kvalifikovaného názvu hostitele nekvalifikovaný název hostitele.

Chcete-li si vynutit doručení na bázi protokolu UUCP na adresu **sesame.com**, měli byste do tabulky `uucpxtable` vložit následující záznam:

```
# uprav adresu sesame.com a pro doručení použij UUCP
sesame     sesame.com
```

Výsledkem je, že program `sendmail` zjistí (na základě parametru *UUCPNODES*, který se nachází v souboru `sendmail.m4`), že jste přímo spojeni se vzdáleným systémem a proto bude řadit poštu tak, aby se mohla doručit pomocí protokolu UUCP.

15.6.4 Zákaz doručování pošty na bázi protokolu UUCP

Občas se dostanete i do opačné situace. Systémy mohou mít často spoustu přímých spojení na bázi protokolu UUCP, které se používají jen zřídka, nebo které nejsou natolik spolehlivé a trvale dostupné jako implicitní mailer nebo přenosový hostitel.

Například v okolí města Seattle existuje spousta systémů, které si vyměňují různé distribuce operačního systému Linux prostřednictvím anonymní služby UUCP. Tato výměna se ale děje pouze v době, kdy je uvolněna nová verze distribuce. Tyto systémy komunikují pomocí protokolu UUCP pouze tehdy, když je to zapotřebí, takže je zpravidla rychlejší a spolehlivější posílat poštu přes několik spolehlivějších skoků prostřednictvím veřejných (vždy dostupných) přenosových hostitelů.

Doručování pošty na bázi protokolu UUCP na hostitele, s nímž jste přímo propojeni, zabráníte velmi snadno. Pokud má vzdálený systém plně kvalifikovaný název domény, můžete do tabulky `domaintable` přidat obdobnou položku:

```
# neposílej poštu svým sousedům přes UUCP
snorkel.com        snorkel
```

Tento záznam nahradí každý výskyt názvu pro protokol UUCP plně kvalifikovaným doménovým jménem (FQDN) a tím zabrání splnění řádky s parametrem *UUCPNODES*, která je definována v souboru `sendmail.m4`. Výsledkem bude obvykle to, že se pošta pošle na základě hodnot parametrů *RELAY_HOST* a *RELAY_MAILER* (nebo na základě hodnoty parametru *DEFAULT_MAILER*).

15.6.5 Spuštění programu sendmail v režimu okamžitého zpracování fronty

Chcete-li zprávy čekající ve frontě zpracovat okamžitě, zadejte pouze příkaz `/usr/lib/runq`. Tento příkaz vyvolá program `sendmail` s patřičně nastavenými volbami, které zajistí, že program `sendmail` spustí frontu nevyřízených úkolů okamžitě a nebude čekat na další plánované spuštění.

15.6.6 Zápis poštovních statistik

Mnoho správců systému (a lidí, kteří pro ně pracují) se zajímá o množství pošty poslané do místního systému, z místního systému nebo přes místní systém. Existuje několik způsobů, jak sledovat kvantitu poštovní dopravy.

- Součástí programu `sendmail` je utilita s názvem `mailstats`, která čte soubor `/usr/local/lib/mail/sendmail.st` a vypisuje počet zpráv a množství bajtů, které přenesl každý z mailerů definovaných v souboru `sendmail.cf`. Aby se mohly zapisovat statistiky o odeslané poště, musí být tento soubor ručně vytvořen správcem místního systému. Pokud odstraníte a znovu vytvoříte soubor `sendmail.st`, pak se součty jednotlivých statistik vymažou. Jeden z možných způsobů je následující:

  ```
  # cp /dev/null /usr/lib/local/mail/sendmail.st
  ```

- Pravděpodobně nejlepší způsob, který zpracovává kvalitní statistiky bez ohledu na to, kdo používá poštu a jaké množství pošty se posílá do místního systému, z místního systému nebo přes místní systém, představuje povolení poštovních ladicích informací. To lze provést prostřednictvím démona `syslogd`*(8)*. Zpravidla je nutné spustit soubor démona `/etc/syslogd` ze svého systémového startovacího souboru (což byste stejně měli udělat) a přidat řádek do souboru `/etc/syslog.conf`*(5)*, který vypadá asi následovně:

  ```
  mail.debug                                /var/log/syslog.mail
  ```

  Používáte-li soubor `mail.debug` a máte-li poměrně velký objem pošty, může soubor log nabýt značné velikosti. Výstupní soubory z démona `syslogd` se zpravidla musí v pravidelných intervalech nahrazovat nebo „promazávat“ prostřednictvím démona `crond`*(8)*.

Existuje množství běžně dostupných utilit, které vytvářejí souhrn z výstupu poštovních log-souborů vytvořených démonem `syslogd`. Jedněmi z nejznámějších utilit je perlový skript `syslog-stat.pl`, který je součástí distribuce zdrojového kódu programu `sendmail+IDA`.

15.7 Kombinace a srovnávání jednotlivých distribucí binárního kódu

Konfigurace přenosu elektronické pošty a doručovacích agentů nemá žádný přesný standard a navíc ani nemá „jednotnou strukturu adresářů“.

Z toho důvodu je nutné se ujistit, že všechny odlišné části systému (USENET news, pošta, protokol TCP/IP) souhlasí s umístěním doručovacího programu pro místní poštu (například programy `lmail`, `deliver` atd.), doručovacího programu pro vzdálenou poštu (například program `rmail`) a poštovního transportního programu (například program `sendmail` nebo `smail`). Tyto předpoklady nejsou zpravidla uvedeny, i když použití příkazu `strings` vám může pomoci zjistit, jaké soubory a adresáře se očekávají. Následuje několik problémů, se kterými jsme se v minulosti setkali u některých běžně dostupných distribucí binárního kódu a zdrojového kódu operačního systému Linux.

- Některé verze distribuce balíku NET-2 protokolu TCP/IP mají definované služby pro program `umail` a nikoliv pro program `sendmail`.
- Existuje několik importovaných verzí programů *elm* a *mailx*, které hledají doručovacího agenta pod názvem `/usr/bin/smail` a nikoliv pod názvem `sendmail`.
- V programu `sendmail+IDA` je vestavěn místní mailer pro program `deliver`, avšak program `sendmail+IDA` očekává, že se tento program bude nacházet v adresáři `/bin` a nikoliv v adresáři `/usr/bin`, což je adresář typický pro operační systém Linux.

Abyste neměli při kompilaci všech poštovních klientů ze zdrojových kódů potíže, doporučujeme raději místo tohoto postupu vytvořit mezi jednotlivými programy symbolické odkazy.

15.8 Kde získat více informací

Existuje mnoho míst, kde byste mohli hledat informace o programu `sendmail`. Máte-li zájem o jejich seznam, nahlédněte prosím do dokumentu Linux Mail HOWTO, který je pravidelně posílán na adresu **comp.answers**. Tento informační bulletin je také dostupný prostřednictvím anonymní služby FTP na adrese **rtfm.mit.edu**. Avšak nejlepším zdrojem informací

jsou zdrojové kódy k programu `sendmail+IDA`. Podívejte se do souborů `DBM-GUIDE`, `OPTIONS` a `Sendmail.mc`, které se nachází v podadresáři `ida/cf` adresáře se zdrojovými kódy.

Kapitola

16

Síťové news (Netnews)

16.1 Historie Usenetu

Myšlenka síťových news se zrodila v roce 1979, kdy dva absolventi univerzity, Tom Truscott a Jim Ellis, začali uvažovat o využití protokolu UUCP ke spojení počítačů za účelem výměny informací mezi uživateli Unixu. Ve státě North Carolina vytvořili malou síť tvořenou třemi počítači.

Původně byl provoz sítě obsluhován několika skripty příkazového interpretu (které byly později přepsány do jazyka C), ale ty se nikdy nedostaly na veřejnost. Rychle je nahradily tzv. „A“ news, které byly prvním veřejným vydáním softwaru news.

Systém „A“ news nebyl navržen pro denní práci s více články ve skupině. Když začal jejich objem narůstat, rozhodli se ho Mark Horton a Matt Glickman přepsat, a výsledek svého snažení nazvali vydání „B“ (tzv. Bnews). První veřejné vydání Bnews mělo číslo verze 2.1 a stalo se tak v roce 1982. Postupně byl systém rozšiřován o některé nové rysy. Současná verze Bnews má číslo 2.11. Pomalu však zastarává, nehledě k tomu, že oficiální správce přešel na INN.

O další přepis se postarali Geoff Colyer a Hanry Spencer v roce 1987; šlo o verzi „C“, neboli C News. Následovala spousta oprav, z nichž nejvýznamnější bylo vydání C News Performance Release. V systémech, které spravují velké množství diskusních skupin, se režie způsobená častým spouštěním programu `relaynews`, který je zodpovědný za odesílání příchozích článků jiným hostitelům, stává významnou. Verze Performance Release přidává novou volbu `relaynews`, která umožňuje spouštět stejnojmenný program, v režimu démona, kdy tento program umístí sám sebe na pozadí.

Verze Performance Release je verzí C News, která je v současné době součástí většiny linuxových balíků.

Všechna vydání až po verzi „C“ jsou určena především pro sítě UUCP, i když je lze stejně dobře používat i v jiných prostředích. Efektivní přenos v sítích typu TCP/IP, DECNet nebo jim podobných vyžaduje nové schéma. Z toho důvodu byl v roce 1986 zaveden protokol NNTP (Network News Transfer Protocol – Protokol pro přenos síťových news). Je založen na síťových spojeních a specifikuje množství příkazů pro interaktivní předávání a získávání článků.

Na Síti existuje spousta aplikací založených na protokolu NNTP. Jednou z nich je i balík `nntpd`, jehož autory jsou Brian Barber a Phil Lapsley. Kromě jiného může sloužit také jako služba zajišťující čtení news hostitelům v rámci lokální počítačové sítě. Balík `nntpd` byl navržen jako doplněk balíků Bnews nebo C News, které měl rozšířit o vlastnosti protokolu NNTP.

Jiným balíkem vycházejícím z protokolu NNTP je `INN`, neboli Internet News. Nejedná se o systém front end, ale spíše o plnohodnotný systém news. Obsahuje důmyslného démona pro přenos news, který je schopen efektivně spravovat několik souběžných spojení NNTP a z toho důvodu je vhodným serverem pro mnoho systémů připojených k Internetu.

16.2 Čím je Usenet v každém případě?

Jedním z ohromujících faktů ohledně Usenetu je skutečnost, že není součástí žádné organizace ani jej neřídí žádná centralizovaná autorita pro správu sítě. Ve skutečnosti je součástí usenetové lidové tradice, kterou bez ohledu na technický popis nelze nijak definovat. Můžete jen říci, čím není. Kdybyste měli po ruce knihu Brendana Kehoe „Zen and the Art of the Internet“ (je dostupná on-line nebo přes Prentice Hall, viz [Kehoe92]), našli byste zde zábavný seznam usenetových „nevlastností“.

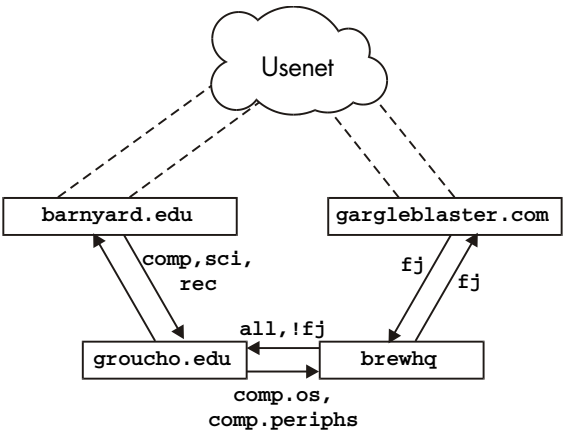
Budeme-li riskovat, že to bude znít hloupě, mohli bychom Usenet definovat jako sdružení vzájemně oddělených míst, které si mezi sebou vyměňují news. Aby se váš systém stal součástí Usenetu, potřebujete nalézt jiný takový systém a sjednat si dohodu s jeho vlastníky nebo správci o výměně news. Poskytování news jinému systému se někdy také říká „krmení“, z čehož vychází další obecný axiom usenetové filosofie: „Dej mu nažrat a jsi v něm.“

Nejmenší jednotkou usenetových news je článek. Jedná se o zprávu, kterou uživatel „odešle do sítě“. Aby s ním mohl systém news pracovat, je opatřen administrativní informací, takzvanou hlavičkou článku. Ta je podobná formátu poštovní hlavičky specifikované standardem Internet mail standard RFC 822 v tom, že je tvořena několika řádky textu, z nichž každý začíná názvem pole ukončeným dvojtečkou, za nímž následuje hodnota tohoto pole.[1]

[1] Formát zpráv usenetových news specifikuje standard RFC 1036, „Standard for interchange of USENET messages“.

Články jsou postupovány jedné nebo více diskusním skupinám (newsgroup). Diskusní skupina se zabývá články týkajícími se jednoho tématu. Všechny diskusní skupiny jsou hierarchicky organizovány, přičemž název každé skupiny zároveň udává její pozici v této hierarchii. Například z názvu diskusní skupiny **comp.os.linux.announce** může každý poznat, že slouží pro oznámení týkající se počítačového operačního systému jménem Linux.

Tyto články si pak mezi sebou vymění všechna usenetová místa, která mají zájem o news z příslušné diskusní skupiny. Pokud dvě místa souhlasí s výměnou news, mohou si mezi sebou předávat libovolné diskusní skupiny a mohou do nich přidávat své místní hierarchicky uspořádané skupiny. Například server **groucho.edu** může mít odkaz na server **bernard.edu**, který je hlavním zdrojem news, a několik odkazů na menší servery, které naopak zásobuje svými news. Takto může Bernyard College obdržet všechny usenetové skupiny, zatímco GMU má zájem jen o některé hlavní skupiny typu **sci**, **comp**, **rec** atd. Některé z následujících serverů, řekněme například UUCP-server nazvaný **brewhq**, bude mít zájem o ještě menší počet skupin, z důvodu nedostatku síťových nebo hardwarových zdrojů. Na druhou stranu může mít server **brewhq** zájem o diskusní skupiny z větve **fj**, které GMU nepřijímá. Pro takové případy udržuje jiný odkaz na **gargleblaster.com**, kde je umístěna celá hierarchie **fj** a ten předá serveru **brewhq**. Tok news ukazuje obrázek 16.1.



Obrázek 16.1
Tok usenetových news v Groucho Marx Univerzity

Popisky šipek vedoucích ze serveru **brewhq** vyžadují bližší vysvětlení. Implicitně chce totiž všechny místní news posílat na adresu **groucho.edu**. Avšak na tomto serveru nejsou uchovávány skupiny **fj** a neexistuje žádný ukazatel pro posílání jakýchkoliv zpráv z těchto skupin. Proto je předávání zpráv z **brewhq** do GMU označeno jako **all,!fj**, což znamená, že jsou univerzitě posílány všechny diskusní skupiny s výjimkou **fj**.

16.3 Jak Usenet obsluhuje news?

Dnes se Usenet rozrostl do obrovských rozměrů. Systémy, které uchovávají celé síťové news obvykle přenášejí něco kolem šedesáti megabajtů dat denně[2]. Samozřejmě, že to vyžaduje mnohem více než jen postrkování souborů. Podívejme se tedy na způsob, jakým většina unixových systémů obsluhuje usenetové news.

News jsou po síti šířeny různými druhy přenosů. Historickým médiem je UUCP, ale hlavní dopravu dnes obstarává Internet. Použitému směrovacímu algoritmu se říká flooding (zaplavování): každý systém udržuje několik spojení (přítoků news – news feeds) s jinými systémy. Každý článek vytvořený nebo obdržený lokálním systémem news jim takto předá, výjimku tvoří jen případ, kdy již příslušný článek existuje, a v takovém případě je tento článek zrušen. Systém může na základě hlavičkového pole `Path:` zjistit, kterými systémy již daný článek prošel. Tato hlavička obsahuje seznam všech systémů, které článek předaly dále, v podobě tzv. notace bang path.

Aby bylo možné články rozlišovat a odhalovat duplicity, nese si s sebou každý usenetový článek id zprávy (uvedené v hlavičkovém poli `Message-Id`), které kombinuje název odesílatele a sériové číslo do tvaru „<`sériové číslo@odesílatel`>“. Systém news uloží id každého článku, který zpracoval, do souboru `history`, s nímž jsou pak porovnávány všechny nově příchozí články.

Tok článků mezi dvěma místy může být omezen dvěma kritérii: zaprvé může být článku přiřazena tzv. distribuce (v hlavičkovém poli `Distribution:`), která může omezit jeho doručení jen do určitých skupin míst. Na druhé straně mohou být předávané skupiny news omezeny odesílatelem i příjemcem. Sada diskusních skupin a distribucí je obvykle uchovávána v souboru *sys*.

Velký počet článků zpravidla vyžaduje zdokonalení výše uvedeného schématu. V sítích UUCP je přirozené sesbírat články za určité období, zkombinovat je do jednoho souboru, ten potom zkomprimovat a poslat vzdálenému systému. Této proceduře říkáme *dávkování* (*batching*)[3].

Jinou alternativou je protokol *ihave/sendme*, který v prvé řadě zabraňuje přenosu duplicitních článků, čímž šetří i šířku pásma. Místo aby umístil do dávkových souborů všechny články a poslal je dál, zkombinuje pouze id zpráv článků do jedné velké zprávy „ihave“ a pošle ji vzdálenému systému. Ten si ji přečte, porovná její obsah s obsahem souboru history a v souboru „sendme“ vrátí seznam článků, které požaduje. Pak mu jsou poslány pouze tyto články.

2 Moment: 60 megabajtů při rychlosti 9600 bps, to je 60 milionů krát 1200, to je…šrum…šrum…, to je ale 34 hodin!

3 Zlaté pravidlo síťových news podle Geoffa Collyera zní: „Měl bys své články dávkovat.“

Samozřejmě, že protokol ihave/sendme má smysl jen u dvou velkých systémů, z nichž každý získává news z několika nezávislých zdrojů, které sčítají nové zprávy dostatečně často pro efektivní tok news.

Systémy, které jsou připojeny k Internetu, obecně spoléhají na software založený na protokolu TCP/IP, který využívá protokol NNTP (Network News Transfer Protocol)[4]. Zprostředkuje přenos news mezi zásobníky a zajišťuje jednotlivým uživatelům přístup ke vzdáleným hostitelům.

Protokol NNTP definuje tři různé způsoby přenosu news. Jedním z nich je verze ihave/sendme v reálném čase, které se také říká *tlačení* (*pushing*). Druhým způsobem je metoda *tažení* (*pulling*) news, při které klient požaduje seznam článků v dané diskusní skupině nebo hierarchii, které dorazily na příslušný server po specifikovaném datu, a vybere si ty, jež nenajde ve svém souboru history. Třetí metoda slouží k interaktivnímu čtení news a dovoluje vám nebo vašemu prohlížeči news získávat články ze specifikovaných diskusních skupin, stejně jako posílat články s neúplnými hlavičkovými informacemi.

Na každém serveru jsou news uchovávány v adresářové hierarchii `/var/spool/news`, každý článek je uložen v samostatném souboru a každá diskusní skupina má svůj vlastní adresář. Název adresáře je odvozen z názvu diskusní skupiny, přičemž jednotlivé komponenty tohoto názvu tvoří cestu. Takto budou články ze skupiny **comp.os.linux.misc** uloženy v adresáři `/var/spool/news/comp/os/linux/misc`. Článkům jsou v jednotlivých skupinách přidělována čísla podle pořadí, v jakém přišly. Toto číslo slouží jako název souboru. Rozsah čísel článků, které jsou aktuálně k dispozici, je uchováván v souboru *active*, který současně slouží jako seznam známých diskusních skupin ve vašem systému.

Jelikož je diskový prostor omezeným zdrojem[5], musí někdo po určité době články mazat. Tomuto mechanismu říkáme *vypršení platnosti* (*expiring*). Doba platnosti článků z určitých skupin a hierarchií zpravidla vyprší po pevném počtu dnů od jejich doručení. Toto pravidlo může odesílatel potlačit za pomoci pole `Expires:` v hlavičce článku, ve kterém uvede datum vypršení platnosti.

4 Jeho popis najdete v RFC 977.

5 Někteří lidé tvrdí, že Usenet je spiknutím dodavatelů modemů a pevných disků.

Kapitola

17

C News

Jedním z nejoblíbenějších softwarových balíků pro síťové news je balík C News. Byl navržen pro systémy, které přenášejí news prostřednictvím protokolu UUCP. Tato kapitola se zabývá základními vlastnostmi C News a základny instalace a údržby tohoto balíku.

Program C News ukládá své konfigurační soubory do adresáře `/usr/lib/news` a většina jeho binárních souborů je umístěna v adresáři `/usr/lib/news/bin`. Články jsou uchovávány pod adresářem `/var/spool/news`. Měli byste se ujistit, že všechny soubory v těchto adresářích jsou vlastněny uživatelem **news** ze skupiny **news**. Většina následujících problémů totiž souvisí s tím, že systém C News nemá přístup k příslušným souborům. Zvykněte si, že dříve než se čehokoliv v tomto adresáři dotknete se musíte přihlásit jako uživatel **news** prostřednictvím `su`. Jedinou výjimkou je program `setnewsids`, který slouží k nastavování skutečných uživatelských id některých programů news. Vlastníkem těchto programů musí být uživatel **root** a musí mít nastaven setuid bit.

V následujících statích si podrobně popíšeme všechny konfigurační soubory systému C News a řekneme si, co je třeba udělat pro to, aby váš systém správně fungoval.

17.1 Doručování news

Články lze do systému C News dodávat několika způsoby. Když pošle článek místní uživatel, program pro čtení news (newsreader) ho obvykle předá příkazu `inews`, který doplní hlavičkové informace. Články ze vzdálených systémů, ať už se jedná o jediný článek nebo o celou dávku článků, jsou předány příkazu `rnews`, který je uloží do adresáře `/var/spool/news/in.coming`, odkud si ho později vyzvedne program `newsrun`. Při použití obou výše zmíněných postupů však bude článek nakonec předán programu `relaynews`.

U každého článku příkaz `relaynews` nejprve vyhledá id zprávy v souboru `history`, čímž zjistí, zda již místní systém daný článek zná. Duplicitní články zruší. Potom se program `relaynews` podívá do hlavičky na řádek `Newsgroups:`, aby zjistil, zda daný systém přijímá články z některé z těchto skupin. Pokud ano a příslušná diskusní skupina je uvedena v souboru `active`, pokusí se program příslušný článek uložit do odpovídajícího adresáře v adresáři `spool`. Pokud takový adresář neexistuje, pak ho vytvoří. Id zprávy následně zaznamená do souboru `history`. V opačném případě program `relaynews` článek smaže.

Pokud se programu `relaynews` nepodaří uložit příchozí článek, protože skupina, do které byl zaslán, není uvedena v souboru `active`, umístí článek do skupiny **junk**.[1] Program *relaynews* také vyhledá staré články nebo články se špatným datem a zruší je. Příchozí dávky článků, jejichž zpracování z nějakého důvodu selže, umístí do adresáře `/var/spool/news/in.coming/bad` a do log-souboru zapíše chybu.

Poté je článek přesměrován na všechny ostatní systémy, které požadovaly news z těchto skupin, přičemž se využije přenos specifikovaný pro konkrétní místo. Aby nebyl článek posílán serveru, který ho již viděl, je každé místo určení porovnáno s hlavičkovým polem `Path:`, které obsahuje seznam míst (zapsaný v bang path style), přes která článek zatím prošel. Příslušný článek mu bude poslán pouze v případě, že se název cílového místa nevyskytuje v tomto seznamu.

Systém C News je běžně používán k předávání news mezi systémy, které využívají protokol UUCP. Lze ho však používat i v prostředí NNTP. Pro doručování news (buď jednotlivých článků, nebo celých dávek) na vzdálený UUCP-systém se za pomoci `uux` spustí na vzdáleném systému program `rnews` a pošle článek nebo dávku na jeho standardní vstup.

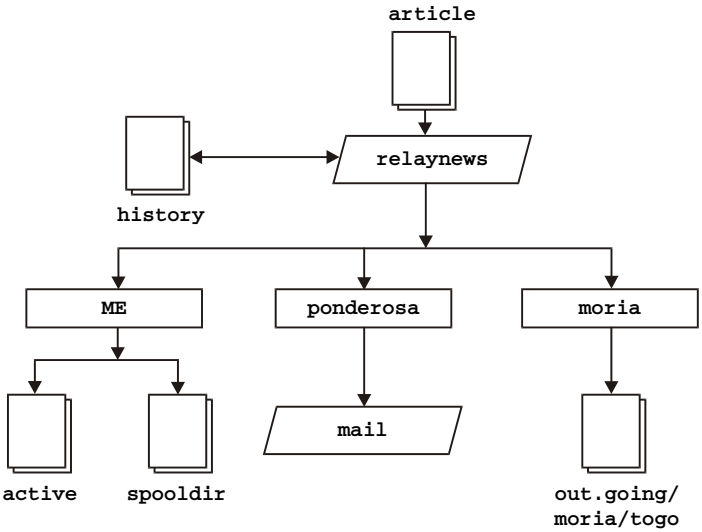
Je-li pro daný systém povoleno dávkování, nepošle systém C News příchozí článek okamžitě, ale připojí název jeho cesty do souboru, který se většinou nazývá `out.going/system/togo`. Periodicky je ze záznamu `crontab` spouštěn program pro vytváření dávek[2], který umístí články do jednoho nebo více souborů, případně je i zkomprimuje a pošle je programu `rnews` na vzdáleném systému.

Tok news zprostředkovaný programem `relaynews` ukazuje obrázek 17.1. Články mohou být přenášeny do lokálního systému (označeného jako *ME*) a prostřednictvím elektronické pošty do systému nazvaného **panderosa** nebo místa nazvaného **moria**, pro které je zapnuto dávkování.

---

1 Mezi skupinami, které existují ve vašem systému a skupinami, o které máte zájem, mohou být značné rozdíly. Například seznam zapsaných skupin může uvádět skupinu **comp.all**, což znamená všechny skupiny pod hierarchií **comp**, ale vy máte v souboru `active` uvedeno jen několik skupin **comp**. Články zaslané do těchto skupin budou umístěny do skupiny **junk**.

2 Všimněte si, že by to měl být crontab systému **news**, aby nebyla porušena přístupová práva k souborům.



Obrázek 17.1
Tok news zprostředkovaný programem `relaynews`

17.2 Instalace

Při instalaci systému C News rozbalte soubory do příslušných adresářů (pokud jste tak již neučinili) a upravte níže uvedené konfigurační soubory. Všechny je najdete v adresáři `/usr/lib/news`. Jejich formáty si nyní popíšeme.

`sys` Pravděpodobně bude třeba upravit řádek *ME*, který popisuje váš systém, i když parametr *all/all* je bezpečný vždy. Doplňte do něj také řádek pro každý systém, kterému předáváte news.

Pokud jste koncovým uzlem, stačí vám jen řádek, který pošle všechny místně vytvořené články do vašeho zásobníku. Předpokládejme, že vaším dodavatelem je server **moria**, pak by váš soubor *sys* vypadal následovně:

```
ME:all/all::
moria/moria.orcnet.org:all/all,!local:f:
```

`organization` Název vaší organizace. Například „`Virtual Brewery, Inc`“. Na vašem domovském počítači zadejte něco na způsob „private site“. Většina lidí nebude žádat správnou konfiguraci vašeho systému, pokud tento soubor neupravíte.

`newsgroups`

`mailname` Poštovní adresa vašeho systému, tj. například **vbrew.com**.

`whoami` Název vašeho systému pro účely news. Často se používá UUCP-název místa, například **vbrew**.

`explist` Tento soubor byste asi měli upravit, protože definuje dobu platnosti některých speciálních diskusních skupin. Zde může hrát důležitou roli i disková kapacita.

Počáteční hierarchii vytvoříte tak, že si ze systému, který vás zásobuje novými news, obstaráte soubory `active` a `newsgroups` a nainstalujete je do adresáře `/usr/lib/news`. Také se ujistěte, že je jejich vlastníkem uživatel news a mají nastavena přístupová práva 644. Ze souboru `active` odstraňte všechny skupiny **to.***, **to**.*mysite* a **to**.*feedsite*, stejně jako **junk** a **control**. Skupiny **to.*** slouží k výměně zpráv protokolu ihave/sendme, ale měli byste je vytvořit bez ohledu na to, zda s použitím tohoto protokolu počítáte, či nikoli. Dále nahraďte pomocí následujícího příkazu všechna čísla článků ve druhém a třetím poli v souboru `active`.

```
# cp active active.old
# sed 's/ [0-9]* [0-9]* / 0000000000 00001 /' active.old > active
# rm active.old
```

Druhý příkaz, který je mým oblíbeným, spouští program `sed`*(1)*. Nahradí dva řetězce číslic řetězcem nul a řetězcem 000001.

Nakonec vytvoříme adresář `spool` a jeho podadresáře používané pro ukládání příchozích a odchozích news:

```
# cd /var/spool
# mkdir news news/in.coming news/out.going
# chown -R news.news news
# chmod -R 755 news
```

Používáte-li pozdější verzi systému C news, bude možná ještě nutné vytvořit v adresáři `spool` podadresář `out.master`.

Jestliže používáte pro čtení news jiný program, než jaký je dodáván se systémem C News, zjistíte, že váš adresář spool je místo v adresáři `/var/spool/news` vytvořen v adresáři `/usr/spool/news`. Nepodaří-li se vašemu programu pro čtení news najít příslušné články, vytvořte symbolický odkaz jménem `/usr/spool/news`, který bude ukazovat na adresář `/var/spool/news`.

Nyní jste připraveni na příjem news. Všimněte si, že už nemusíte vytvářet žádné jiné adresáře, ale pokaždé, když systém C News obdrží článek patřící do skupiny, pro kterou ještě neexistuje příslušný adresář, tento vytvoří.

Konkrétně to platí pro skupiny *alt*. Takže po nějaké době zjistíte, že váš adresář spool bude přeplněn adresáři s názvy skupin, které jste si nikdy nezapsali, například **alt.lang.teco**. Tomu zabráníte tak, že buď odstraníte ze souboru `active` všechny nechtěné skupiny, nebo budete pravidelně spouštět skript příkazového interpretu, který odstraní všechny prázdné adresáře pod adresářem `/var/spool/news` (vyjma `out.going` a `in.coming` samozřejmě).

Systém C News potřebuje účet, na který by mohl posílat chybové a stavové zprávy. Implicitně se tento účet nazývá **usenet**. Využijete-li toto implicitní nastavení, musíte nastavit alias, který by veškerou poštu došlou na tento účet přesměroval na odpovědné osoby. (Jak to provést u programů `smail` a `sendmail` popisujeme v 14. a 15. kapitole.) Toto chování lze také potlačit pomocí proměnné prostředí *NEWMASTER*, které přiřadíte příslušný název. Proměnnou je třeba nastavit v souboru crontab systému **news**, stejně jako při každém manuálním spuštění nějakého administrativního nástroje, takže zřízení aliasu je zřejmě jednodušší.

Při úpravě souboru `/etc/passwd` se ujistěte, že každý uživatel má v poli *pw_gecos* (jde o čtvrté pole) zapsáno svoje skutečné jméno. Patří k etiketě sítě (netiquette), že se v poli From: příslušného článku objeví skutečné jméno odesílatele. Stejně to musíte udělat, když budete používat elektronickou poštu.

17.3 Soubor sys

Soubor `sys`, umístěný v adresáři `/usr/lib/news`, řídí hierarchie skupin, které dostáváte, a jejich další směrování. I když existují speciální nástroje pro práci s tímto souborem, jmenovitě to jsou `addfeed` a `delfeed`, myslím, že je lepší upravovat tento soubor ručně.

Soubor *sys* obsahuje záznam pro každý systém, kterému předáváte news, stejně jako popis skupin, které přijímáte. Každý záznam má následující formát:

```
site[/exclusions]:grouplist[/distlist][:flags[:cmds]]
```

Jako pokračovací znak na novém řádku se používá znak zpětné lomítko. Znak (#) znamená komentář.

site Toto je název systému, kterého se záznam týká. Zpravidla se použije UUCP-název příslušného místa. V souboru *sys* musí mít záznam i váš systém, jinak byste nedostávali žádné články.

Speciální název *ME* označuje váš systém. Tento záznam definuje všechny skupiny, které chcete mít uloženy lokálně. Články, které nevyhovují řádku se záznamem *ME*, putují do skupiny **junk**.

Protože systém C News porovnává pole `site` s názvy systémů v hlavičkovém poli `Path:`, musí si tyto přesně odpovídat. Některé systémy předávají v tomto poli plně kvalifikované názvy domén nebo alias typu **news**.*systém.doména*. Aby nebyly těmto systémům vráceny žádné články, je třeba přidat tyto názvy do seznamu výjimek, ve kterém jsou jednotlivé položky odděleny čárkami.

V záznamu týkajícím se systému **moria** by mohlo příslušné pole obsahovat například řetězec **moria/moria.ocnet.org**.

grouplist Toto je seznam skupin a hierarchií oddělených čárkami, které si zapsal konkrétní systém. Hierarchii je možné specifikovat prostřednictvím předpony (například **comp.os** pro všechny skupiny, jejichž název začíná touto předponou), za níž může volitelně následovat klíčové slovo **all** (tj. **comp.os.all**).

Předání příslušné hierarchie nebo skupiny zabráníte tak, že před její název přidáte vykřičník. Při porovnávání názvu diskusní skupiny s tímto seznamem má nejvyšší prioritu nejdelší shoda. Když například pole *grouplist* obsahuje

```
!comp,comp.os.linux,comp.folklore.computers
```

nebudou příslušnému systému předány žádné skupiny z **comp** s výjimkou podskupin skupiny **comp.folklore.computers** a **comp.os.linux**.

Pokud daný systém požaduje všechny news, které jste sami obdrželi, zadejte do pole *grouplist* položku *all*.

distlist je offset oddělený od pole *grouplist* lomítkem a obsahující seznam distribucí, který se má předávat dále. I v tomto případě můžete některé distribuce vyřadit tím, že před ně umístíte znak vykřičník. Všechny distribuce označuje řetězec *all.* Pokud toto pole vynecháte, znamená to, že jste zvolili *all*.

Můžete například použít seznam distribucí *all,!local*, čímž zabráníte šíření news, které jsou určeny jen pro lokální použití.

Toto pole většinou obsahuje minimálně dvě distribuce: *world*, což je vždy implicitní distribuce, která se použije, když uživatel žádnou distribuci neuvede, a *local*. Existují také další distribuce, které se týkají určité oblasti, státu, země atd. Nakonec zde máme dvě distribuce, které používá pouze systém C News. Jsou to *sendme* a *ihave* a využívá je protokol ihave/sendme.

Využití distribucí je předmětem stálých sporů. Podle někoho vytváří některé programy pro čtení news falešné distribuce podle nejvyšší hierarchie, například **comp** při posílání news do skupiny **comp.os.linux**. Distribuce, které se týkají regionů, jsou také sporné, protože news, pokud je pošlete po Internetu, mohou cestovat mimo váš region.[3] Distribuce aplikované na organizace mají na druhé straně velký význam, zabraňují například šíření tajných informací mimo příslušnou organizaci. Tohoto cíle se však dosáhne lépe vytvořením samostatné diskusní skupiny nebo hierarchie.

flags Toto pole popisuje parametry přenosu. Může být prázdné nebo může obsahovat kombinaci následujících znaků:

- *F* Tento příznak povoluje dávkování.
- *f* Tento příznak je podobný výše uvedenému, ale umožňuje systému C News přesnější výpočet velikosti odcházejících dávek.
- *I* Tento příznak způsobí, že budou C News vytvářet seznam článků vhodný pro protokol ihave/sendme. Aby bylo možné tento protokol použít, jsou nutné další úpravy v souborech `sys` a `batchparams`.
- *n* Na základě tohoto příznaku budou vytvářeny dávkové soubory pro přenosové klienty využívající protokol NNTP, jako například `nntpxmit` (viz 18. kapitola). Dávkové soubory obsahují název souboru článku a id zprávy.
- *L* Tento příznak říká systému C News, aby přenášel pouze články zaslané vašemu systému. Příznak může být následován desítkovým číslem *n*, které způsobí, že budou C News přenášet pouze články, které dorazily během *n* hops. Systém C News určí počet hops z pole `Path:`.
- *u* Systém C News bude vytvářet dávky pouze z článků v neřízených (nemoderovaných) skupinách.
- *m* Systém C News bude vytvářet dávky pouze z článků v řízených skupinách.

Současně lze použít pouze jeden z příznaků *F*, *f*, *I* nebo *n*.

cmds Toto pole obsahuje příkaz, který se provede nad každým článkem, pokud není povoleno dávkování. Příslušný článek bude předán na standardní vstup tohoto příkazu. Tuto možnost byste měli využívat jen u malých dávek news, protože jinak by došlo k velkému zatížení obou systémů.

[3] Je zcela běžné, když článek poslaný řekněme z Hamburgu do Frankfurtu putuje přes uzel **reston.ans.net** v Holandsku, nebo dokonce přes některý uzel v USA.

Implicitní příkaz má formát

```
uux - -r -z system!rnews
```

a spustí na vzdáleném systému program `rnews`, jemuž předá na standardní vstup příslušný článek.

Implicitní vyhledávací cesta pro příkazy předané v tomto poli je `/bin:/usr/bin:/usr/lib/news/bin/batch`. Poslední z těchto adresářů obsahuje několik skriptů příkazového interpretu, jejichž názvy začínají slovem *via*. Podrobněji si je popíšeme dále v kapitole.

Je-li povoleno dávkování jedním z parametrů *F* nebo *f*, *I* nebo *n*, bude systém C News očekávat, že najde v tomto poli název souboru a ne příkaz. Pokud název souboru nezačíná lomítkem (/), předpokládá se, že jde o relativní cestu k adresáři `/var/spool/news/out.going`. Je-li toto pole prázdné, vezme se implicitní cesta *systém*`/togo`.

Při nastavování systému C News budete zřejmě muset vytvořit svůj vlastní soubor *sys*. Abychom vám to ulehčili, nabízíme vám vzorový soubor serveru **vbrew.com**, ze kterého si můžete zkopírovat, co potřebujete.

```
# Bereme vše, co nám ostatní poskytnou.
ME:all/all::

# Vše co dostaneme posíláme na moria kromě lokálních článků.
# Používáme dávkování.
moria/moria.orcnet.org:all,!to,to.moria/all,!local,!brewery:f:

# Skupinu comp.risks posíláme na adresu jack@ponderosa.uucp
ponderosa:comp.risks/all::rmail jack@ponderosa.uucp.

# Hostitel swim dostává několik skupin.
swim/swim.twobirds.com:comp.os.linux,rec.humor.oracle/all,!local:f:

# Zaznamenávej poštovní mapy pro pozdější zpracování.
usenet-maps:comp.mail.maps/all:F:/var/spool/uumaps/work/batch
```

17.4 Soubor active

Soubor `active` najdete v adresáři `/usr/lib/news`. Obsahuje seznam všech skupin, které zná váš systém, a všech článků, které jsou zde v současné době uloženy. Jen výjimečně ho budete přímo upravovat, ale kvůli úplnosti si ho zde probereme. Záznamy v tomto souboru mají následující formát:

```
newsgroup high low perm
```

Pole *newsgroup* je samozřejmě název skupiny. Pole *low* a high udávají nejnižší a nejvyšší čísla článků, které jsou právě dostupné. Není-li dostupný žádný článek, je pole *low* rovno high+1.

Přinejmenším to objasňuje účel pole low. Nicméně systém C News kvůli efektivitě toto pole neaktualizuje. To by nebyla tak velká ztráta, kdyby na něm nezávisely některé programy pro prohlížení news. Například program `trn` se na základě tohoto pole rozhoduje, zda může odstranit nějaké články ze své databáze vláken. Kvůli aktualizaci pole *low* je proto nutné pravidelně spouštět program `updatemin` (nebo v dřívějších verzích systému C News skript `upact`).

Parametr *perm* řídí přístup uživatelů, kteří mají povolen přístup k dané skupině. Může nabývat jedné z následujících hodnot:

y — Uživatelé mohou do této skupiny přispívat.

n — Uživatelé nemohou do této skupiny přispívat. Nicméně mohou z ní číst články.

x — Tato skupina byla lokálně znepřístupněna. K tomu občas dochází, když se administrátoři news (neboli superuživatelé) kvůli nějakému poslanému článku urazí.

Články určené pro tuto skupinu nejsou ukládány lokálně, nicméně jsou předávány dále systémům, které je požadují.

m — Tato hodnota označuje řízenou (moderovanou) skupinu. Pokusí-li se uživatel zaslat do této skupiny nějaký článek, inteligentní program pro čtení news ho na to upozorní a místo do skupiny ho pošle moderátorovi. Adresu moderátora zjistí ze souboru `moderators`, který najde v adresáři `/usr/lib/news`.

=real-group — Takto označíte skupinu *newsgroup* jako lokální alias (přezdívku) jiné skupiny, jmenovitě *real-group*. Všechny články zaslané do skupiny *newsgroup* budou přesměrovány do této skupiny.

V systému C News ve skutečnosti nebudete muset přímo pracovat s tímto souborem. Skupiny lze mazat lokálně i globálně pomocí nástrojů `addgroup` a `delgroup` (viz dále popis ve

stati Nástroje pro údržbu a úlohy). Je-li do Usenetu přidána nebo z něj naopak odstraněna nějaká skupina, dozví se o tom všechny systémy na základě řídicí zprávy *newgroup* nebo *rmgroup*. Nikdy však neposílejte tuto zprávu sami! Máte-li zájem o podrobnosti týkající se vytváření diskusních skupin, pak si přečtěte měsíční příspěvky do konference **news.announce.newusers**.

Se souborem `active` je úzce spjat soubor `active.times`. Kdykoliv je vytvořena nová skupina, zaznamená systém C News do tohoto souboru zprávu, která obsahuje název vytvořené skupiny, datum vytvoření, zda se tak stalo na základě zprávy *newgroup* nebo lokálně a kdo tak učinil. To kvůli programům pro čtení news, které tak mohou informovat uživatele o každé nově vytvořené skupině. Tuto informaci využívá také příkaz *NEWGROUPS* protokolu NNTP.

17.5 Dávkování článků

Dávky news dodržují konkrétní formát, který je stejný pro Bnews, C News a INN. Každému článku předchází následující řádek:

```
#! rnews count
```

kde *count* je počet bajtů článku. Použije-li se dávková komprese, je výsledný soubor zkomprimován jako celek a na začátek je umístěn další řádek, který udává příkaz, jež se použije pro rozbalení. Standardním kompresním nástrojem je `compress`, který je označen zprávou

```
#! cunbatch
```

Někdy, když je potřeba poslat dávky za pomocí poštovního programu, který ze všech dat odstraňuje osmý bit, je možné zkomprimovanou dávku chránit prostřednictvím tzv. c7-kódování. Takovéto dávky budou označeny `c7unbatch`.

Když je dávka na vzdáleném systému předána programu `rnews`, vyhledá tyto značky a dávku příslušným způsobem zpracuje. Některé systémy používají ještě jiné kompresní nástroje, například `gzip`, a před takto zkomprimované soubory předřazují zprávu `zunbatch`. Systém C News těmto nestandardním hlavičkám nerozumí. Má-li je podporovat, je třeba upravit zdrojový kód.

V systému C News je dávkování článků prováděno programem `/usr/lib/news/bin/batch/sendbatches`, který vezme seznam článků ze souboru `system/togo` a umístí je do několika dávek. Měl by být spouštěn jednou za hodinu případně ještě častěji, v závislosti na objemu provozu.

Jeho práci řídí soubor `batchparams` umístěný v adresáři `/usr/lib/news`. Tento soubor popisuje maximální velikost dávky pro každý systém, dávkovací a volitelně i komprimační

program, který se má použít, a přenos, který zajistí doručení příslušné dávky vzdálenému systému. Dávkovací parametry je možné zadat pro každý systém jednotlivě nebo jako sadu implicitních parametrů, kde není explicitně zmiňován žádný systém.

Dávkování pro specifický systém spustíte následujícím příkazem

```
# su news -c "/usr/lib/news/bin/batch/sendbatches site"
```

Spustíte-li program *sendbatches* bez argumentů, bude obsluhovat veškeré dávkovací dotazy. Interpretace slova „all" závisí na přítomnosti implicitního záznamu v souboru `batchparams`. Pokud existuje, budou se kontrolovat všechny adresáře uvedené v adresáři `/var/spool/news/out.going`. V opačném případě bude program procházet všechny záznamy uvedené v souboru `batchparams`. Všimněte si, že program *sendbatches* bere při procházení adresáře `out.batches` v úvahu pouze adresáře, které nemají v názvu tečku nebo znak at (@).

Při instalaci systému C News pravděpodobně zjistíte, že součástí distribuce je i soubor `batchparams`, který obsahuje rozumné implicitní záznamy, takže zřejmě tento soubor nebudete muset upravovat. Jeho formát si zde popíšeme jen pro úplnost. Každý řádek je složen ze šesti polí, které jsou navzájem odděleny mezerami nebo tabulátory:

```
site size max batcher muncher transport
```

Následuje popis významu těchto polí:

Pole *site* udává název systému, kterého se daný záznam týká. Soubor `togo` musí být v tomto systému umístěn v adresáři `spool` jako `out.going/togo`. Název místa `/default/` označuje implicitní záznam.

Pole *size* udává maximální velikost vytvořených dávek článků (před kompresí). Je-li velikost jednoho článku větší než tento údaj, povolí systém C News výjimku a umístí ho do samostatné dávky.

Pole max udává maximální počet dávek vytvořených a naplánovaných pro přenos, než je dávkování pro tento konkrétní systém zrušeno. To je užitečné v případě, že vzdálený systém dlouhou dobu nefunguje, protože se tak vyhnete zahlcení UUCP-adresářů `spool` miliony dávek news.

Systém C News určuje velikost čekajících dávek za pomoci skriptu `queulen`, který najdete v adresáři `/usr/lib/news/bin`. Balík `newspak`, jehož autorem je Vince Skahan, by měl obsahovat skript pro UUCP kompatibilní s BNU. Používáte-li jiné uspořádání adresáře spool, například Taylor UUCP, budete si možná muset napsat svůj vlastní skript.[4]

[4] Nezajímá-li vás počet souborů v adresáři `spool` (protože jste jediná osoba, která používá daný počítač, a nepíšete články o velikostech řádově megabajtů), můžete obsah tohoto skriptu nahradit jednoduchým příkazem `exit 0`.

Pole *batcher* obsahuje příkaz, který slouží k vytváření dávky ze seznamu článků v souboru `togo`. Pro účely normálního přenosu je jím zpravidla program `batcher`. Pro jiné účely mohou být poskytovány jiné dávkovací programy. Například protokol ihave/sendme vyžaduje, aby byl seznam článků převeden do řídicích zpráv protokolu ihave nebo sendme, které jsou pak odeslány do skupiny **to**.`site`. To mají na starosti programy `batchih` a `batchsm`.

Pole *muncher* specifikuje příkaz, který se použije při kompresi. Zpravidla to bývá skript **compcun**, který vytvoří komprimovanou dávku.[5] Pro stejné účely je možné využít také programy `gzip`, řekněme `gzipcun` (aby bylo jasno, musíte si ho sami napsat). Je třeba se ujistit, že program `uncompress` na vzdáleném systému je příslušným způsobem upraven, aby rozpoznal soubory komprimované programem `gzip`.

Pakliže vzdálený systém nedisponuje příkazem `uncompress`, stačí použít volbu `ncomp`, která kompresi zakáže.

Poslední pole *transport* popisuje přenos, který se má použít. K dispozici je několik příkazů, jejichž názvy začínají řetězcem *via* a slouží pro různé druhy přenosů. Program `sendbatches` jim předá na příkazové řádce název cílového systému. Pokud jste nenastavili hodnotu `batchparams` na */default/*, odvodí název systému z pole *site*, přičemž odstraní veškeré znaky za první tečkou nebo lomítkem včetně. V případě položky */default/* se použijí názvy podadresářů adresáře `out.going`.

Program `uux` používá ke spouštění `rnews` dva příkazy: `viauux` a `vaiuuxz`. Druhý z nich nastavuje příznak `-z` pro (starší verzi) programu `uux`, který způsobí, že tento program nebude vracet zprávy o úspěšném doručení každého článku. Jiný příkaz, `viamail`, posílá dávky článků uživateli **rnews** na vzdáleném systému prostřednictvím elektronické pošty. Samozřejmě, že tento vzdálený systém musí veškerou poštu pro uživatele **rnews** přesměrovat do svého lokálního systému news. Úplný seznam všech přenosů naleznete v manuálových stránkách příkazu `newsbatch`*(8)*.

Všechny příkazy z posledních tří polí musí být umístěny v adresáři `out.going`*/systém* nebo `/usr/lib/news/bin/batch`. Většina z nich jsou skripty, takže si můžete nové nástroje snadno přizpůsobit svým potřebám. Jsou spouštěny jako roura. Seznam článků je předáván na standardní vstup dávkovacího programu, jehož výstupem je příslušná dávka. Ta je předána programu na zpracování dávek atd.

[5] Skript `compcun`, který je dodáván jako součást balíku C News, používá program `compress` s 12bitovou kompresí, protože to je standard většiny systémů. Můžete vytvořit kopii tohoto souboru, řekněme `compcun16`, který by používal 16bitovou kompresi. Toto vylepšení však není tak působivé.

Následuje ukázkový soubor:

```
# soubor batchparms
# site      | size     |max   |batcher    |muncher    |transport
#-------------+--------+------+---------+-----------+-----------
/default/     100000     22  batcher       compcun    viauux
swim           10000     10  batcher       nocomp     viauux
```

17.6 Vypršení platnosti news

V systému Bnews se o aktuálnost news stará program nazvaný `expire`, kterému předáte jako argumenty seznam diskusních skupin a časový údaj, po jehož uplynutí vyprší platnost článků. Pokud chcete, aby různým hierarchiím vypršela platnost v jinou dobu, potřebujete skript, který by pro každou skupinu spouštěl program `expire` samostatně. Systém C News nabízí pohodlnější řešení: v souboru `explist` stačí uvést diskusní skupiny a příslušné časové intervaly. Jednou denně je z `cron` spouštěn příkaz `doexpire`, který podle tohoto seznamu zpracuje všechny skupiny.

Někdy si možná budete chtít ponechat články z určitých skupin i po uplynutí doby jejich platnosti; například si budete chtít archivovat programy zaslané do skupiny **comp.sources.unix**. Soubor `explist` umožňuje označit skupiny, které chcete takto archivovat.

Záznam v souboru `explist` vypadá následovně:

```
grouplist perm times archive
```

Pole *grouplist* je čárkami oddělený seznam diskusních skupin, kterých se daný záznam týká. Hierarchii skupin zadáte celým názvem skupiny, za nímž může nepovinně následovat řetězec *all*. Například záznam týkající se všech podskupin skupiny **comp.os** můžete v poli *grouplist* uvést jako **comp.so** nebo **comp.os.all**.

Při zjišťování vypršení platnosti news z nějaké skupiny je její název v příslušném pořadí porovnán se všemi záznamy v souboru `explist`. Použije se první vyhovující záznam. Chcete-li například po čtyřech dnech zrušit většinu obsahu diskusní skupiny **comp** s výjimkou skupiny **comp.os.linux.announce**, jejíž obsah si chcete ponechat týden, vytvoříte záznam pro druhou skupinu, který bude udávat sedmidenní platnost článků v ní, a za ním bude následovat záznam pro skupinu **comp**, který bude udávat platnost pouze čtyři dny.

Pole *perm* popisuje, zda se příslušný záznam týká moderovaných, nemoderovaných nebo všech skupin. Může nabývat hodnot *m*, *u* nebo *x*, které označují moderované, nemoderované a všechny skupiny.

Třetí pole, *times*, většinou obsahuje pouze jediné číslo. To je počet dnů, po jejichž uplynutí vyprší doba platnosti článků, které ji neměly explicitně nastavenu v poli `Expires:` v hlavičce článku. Všimněte si, že se jedná o počet dnů od data, kdy článek dorazil do vašeho systému, nikoliv od data jeho odeslání.

Pole *times* ovšem může být i složitější. Může obsahovat kombinaci až tří čísel, navzájem oddělených pomlčkou. První určuje počet dnů, které musí uplynout, aby se článek stal kandidátem na vyřazení. Jen zřídka se používá jiná hodnota než nulová. Druhé pole je výše zmiňovaný implicitní počet dnů, po kterém vyprší platnost souboru. Třetí číslo specifikuje počet dnů, po kterých bude článek bezpodmínečně smazán, bez ohledu na to, zda má v hlavičce uvedeno pole `Expires:`. Uvedete-li pouze prostřední číslo, použijí se pro zbylé dvě pole implicitní hodnoty. Ty je možné specifikovat prostřednictvím speciálního záznamu */bounds/*, který bude popsán dále.

Čtvrté pole, *archive*, určuje, zda má být příslušná diskusní skupina archivována, či nikoli. Pokud si archivaci nepřejete, pak zde uveďte pomlčku. V opačném případě zadejte buď úplnou cestu (odkazující na adresář), nebo znak at (@). Ten zastupuje implicitní archivní adresář, který pak musíte programu `doexpire` předat pomocí argumentu `-a` v příkazové řádce. Archivní adresář by měl vlastnit uživatel **news**. Když program `doexpire` archivuje článek řekněme ze skupiny **comp.sources.unix**, uloží ho do adresáře `comp/sources/unix` pod archivním adresářem a pokud neexistuje, vytvoří nový. Vlastní archivní adresář ovšem nevytvoří.

Program `doexpire` spoléhá na dva speciální záznamy ze souboru `explist`. Místo názvů diskusních skupin používají klíčová slova */bounds/* a */expired/*. Záznam */bounds/* obsahuje implicitní hodnoty tří položek pole *times*, které jsme si před chvílí popsali.

Pole */expired/* určuje, jak dlouho bude systém C News uchovávat řádky v souboru `history`. Tato položka je nutná z toho důvodu, že systém C News neodstraní řádek s názvem příslušného článku ze souboru `history` ihned po jeho smazání, ale ponechá ho tam pro případ, že by po tomto datu přišel duplicitní článek. Pokud dostáváte články pouze z jediného systému, může být tato hodnota malá. V ostatních případech se pro sítě založené na protokolu UUCP doporučují použít hodnotu odpovídající dvěma týdnům, v závislosti na zkušenostech s prodlevami článků z těchto systémů.

Vzorový soubor `explist` s krátkými intervaly vypršení platnosti zde reprodukujeme:

```
# keep history lines for two weeks. Nobody gets more than three month.
/expired/                       x         14        -
/bounds/                                  0-1-90    -

# groups we want to keep longer than the rest
comp.os.linux.announce          m         10        -
```

```
comp.os.linux                   x       5       -
alt.folklore.computers          u       10      -
rec.humor.oracle                m       10      -
soc.feminism                    m       10      -

# Archive *.sources groups
comp.sources,alt.sources        x       5       @

# defaults for tech groups
comp,sci                        x       7       -

# enough for a long weekend
misc,talk                       x       4       -

# throw away junk quickly
junk                            x       1       -

# Archive *.sources groups
comp.sources,alt.sources        x       5       @

# defaults for tech groups
comp,sci                        x       7       -

# enough for a long weekend
misc,talk                       x       4       -

# throw away junk quickly
junk                            x       1       -

# control messages are of scant interest, too
control                         x       1       -

# catch-all entry for the rest of it
all                             x       2       -
```

V souvislosti s vypršením vyvstává několik potenciálních problémů. Váš program pro čtení news může například spoléhat na třetí pole aktivního souboru, které obsahuje číslo nejnižšího článku, který je k dispozici. Při vyřazování článků systém C News toto pole neaktualizuje. Pokud potřebujete (nebo chcete), aby toto pole odráželo aktuální stav, pak musíte po každém spuštění programu `doexpire`[6] spustit i program `updatemin`.

Systém C News dále neprovádí kontrolu platnosti souborů procházením adresáře příslušné diskusní skupiny, ale pouze na základě souboru `history`.[7] Budou-li v souboru history nesprávné údaje, mohou články zůstat na vašem systému navždy, protože C News na ně prostě zapomenou.[8] Tento problém lze opravit pomocí skriptu `addmissing`, který najdete v adresáři `/usr/lib/news/bin/maint` a který přidá do souboru `history` chybějící články. Jiný skript, `mkhistory`, znovu vytvoří celý soubor `history`. Dříve než tento skript spustíte se nezapomeňte přihlásit jako uživatel **news**, jinak získáte soubor `history`, který bude pro systém C News nečitelný.

17.7 Různé soubory

Chování systému C News řídí více souborů, ale jejich existence není pro jeho fungování životně důležitá. Všechny sídlí v adresáři `/usr/lib/news`. Nyní si je krátce popíšeme.

`newsgroups` Tento soubor je společníkem souboru `active` a obsahuje seznam názvů diskusních skupin společně s popisem jejich hlavního tématu. Tento soubor je automaticky aktualizován, jakmile systém C News obdrží řídicí zprávu *checknews* (viz stať 17.8).

`localgroups` Máte-li více lokálních skupin, na které nechcete, aby vás systém C News upozorňoval pokaždé, když obdrží zprávu *checknews*, umístěte jejich názvy a popisy do tohoto souboru stejným způsobem, jak byste je zařadili do souboru `newsgroups`.

`mailpath` Tento soubor obsahuje adresu moderátorů všech moderovaných skupin. Každý řádek obsahuje název skupiny následovaný poštovní adresou moderátora (odsazenou tabulátorem).

6 U starších verzí systému C News to měl na starosti skript `upact`.

7 Datum, kdy článek dorazil, je uchováváno v prostředním poli na řádku v souboru history a je vyjádřeno v sekundách od 1. ledna 1970.

8 Nevím, *proč* k tomu dochází, ale na mém počítači se to občas stává.

Implicitní jsou dva speciální záznamy – *backbone* a *internet*. Oba označují (v notaci bang-path) cestu k nejbližšímu páteřnímu místu a systému, který rozumí adrese podle standardu RFC 822 (**uživatel@hostitel**). Implicitními záznamy jsou

```
internet      backbone
```

Máte-li nainstalován jeden z programů `smail` nebo `sendmail`, pak nemusíte záznam *internet* měnit, protože tyto programy rozumí adresování podle standardu RFC.

Záznam *backbone* se využije, když uživatel přispívá do moderované skupiny, jejíž moderátor není explicitně uveden. Je-li například název skupiny **alt.sewer** a záznam *backbone* obsahuje řetězec `path!%s`, pošle systém C News tento článek na adresu `path`**!alt-sewer** doufaje, že počítač backbone bude schopen článek předat dál. Jakou máte použít cestu zjistíte od správců news systému, který vás jimi zásobuje. Jako poslední útočiště můžete také použít adresu **uunet.uu.net!%s**.

`distributions`

Tento soubor ve skutečnosti nepatří systému C News, ale používají ho některé programy pro čtení news a `nntpd`. Obsahuje seznam distribucí, které rozpozná váš systém, a popis jejich (zamýšleného) efektu. Například společnost Virtual Brewery používá soubor, který obsahuje následující záznamy:

```
world      everywhere in the world
local      Only local to this site
nl         Netherlands only
mugnet     MUGNET only
fr         France only
de         Germany only
brewery    Virtual Brewery only
```

`log` Tento soubor obsahuje záznam všech aktivit systému C News. Pravidelně ho vybírá program `newsdaily`. Kopie starších log-souborů jsou pojmenovávány `log.o`, `log.oo` atd.

`errlog` Do tohoto souboru jsou zaznamenávány všechny chybové zprávy vyprodukované systémem C News. To se netýká článků, které byly odhozeny kvůli špatné skupině apod. Když program `newsdaily` zjistí, že je tento soubor neprázdný, automaticky ho pošle správci news (implicitně na účet **usenet**).

Soubor `errlog` vyprazdňuje program `newsdaily`. Starší kopie tohoto souboru jsou uchovávány pod názvy typu `errlog.o`.

`batchlog` Sem jsou zaznamenávána veškerá spuštění programu *sandbatches*. Tento soubor má zpravidla jen malý význam. Navštěvuje ho také program `newsdaily`.

`watchtime` Jedná se o prázdný soubor, který se vytvoří vždy při spuštění programu `newsmatch`.

17.8 Řídicí zprávy

Protokol usenetových news rozeznává speciální kategorii článků, které ze strany systému news evokují jisté odpovědi nebo akce. Takovýmto článkům říkáme *řídicí zprávy*. Poznáte je podle přítomnosti pole `Control:` v hlavičce článku, které obsahuje název řídicí operace, která se má provést. Existuje jich několik typů a všechny jsou obsluhovány skripty umístěnými v adresáři `/usr/lib/news/ctl`.

Většina z nich provede příslušnou akci automaticky v době, kdy je článek zpracován systémem C News, aniž by o tom informovaly správce news. Implicitně jsou správci předány pouze zprávy *checkgroups*, ale toto chování lze úpravou příslušných skriptů změnit.

17.8.1 Zpráva cancel

Nejznámějším typem zprávy je zpráva *cancel*, s jejíž pomocí může uživatel zrušit článek, který dříve poslal. Pakliže tento článek dosud existuje, zajistí tato zpráva jeho efektivní odstranění z adresářů *spool*. Zpráva *cancel* je předána všem systémům, které dostávají news z příslušné skupiny, bez ohledu na to, zda příslušný článek již viděli či nikoli. To pro případ, že by se původní článek opozdil za rušící zprávou. Některé systémy news dovolují uživatelům rušit zprávy jiných osob; to je samozřejmě definitivní ne-ne.

17.8.2 Zprávy newgroup a rmgroup

Tyto dvě zprávy se týkají vytváření a rušení diskusních skupin. Nová skupina může být v „obvyklé“ hierarchii vytvořena až poté, co to odsouhlasí uživatelé Usenetu. Pravidla týkající se hierarchie skupiny **alt** dovolují vzniknout něčemu, co se hodně podobá anarchii. Více podrobností najdete ve skupinách **news.announce.newusers** a **news.announce.newgroups**. Nikdy neposílejte zprávu *newgroup* nebo *rmgroup*, pokud bezpodmínečně nevíte, že k tomu máte oprávnění.

17.8.3 Zpráva checkgroups

Zprávy *checkgroups* posílají správci news, aby všechny systémy v síti provedly vzájemnou synchronizaci svých souborů `active` s realitou Usenetu. Komerční poskytovatelé internetových služeb by mohli například poslat tuto zprávu svým zákazníkům. Jednou měsíčně pošle moderátor skupiny **comp.announce.newgroup** „oficiální“ zprávu *checkgroups* pro hlavní hierarchie. Tato zpráva je ovšem poslána jako běžný článek, nikoliv jako řídicí zpráva. Chcete-li provést operaci *checkgroup*, uložte tento článek do souboru, řekněme `/tmp/check`, odstraňte z něj vše kromě vlastní řídicí zprávy a předejte ho následujícím způsobem skriptu *checkgroups*:

```
#   su   news -c "/usr/lib/news/bin/ctl/checkgroups"  < /tmp/check
```

Tento příkaz aktualizuje váš soubor `newsgroups`, do kterého přidá skupiny uvedené v souboru `localgroups`. Původní soubor `newsgroups` bude přejmenován na `newsgroups.bac`. Všimněte si, že pošlete-li tuto zprávu lokálně, bude jen zřídka fungovat, protože program `inews` odmítne tak velký článek přijmout.

Když systém C News zjistí rozpory mezi seznamem *checkgroups* a souborem `active`, vytvoří seznam příkazů, které zajistí aktualizaci vašeho systému, a pošle ho správci news. Výstup má zpravidla následující podobu:

```
From news Sun Jan 30 16:18:11 1994
Date: Sun, 30 Jan 94 16:18 MET
From: news (News Subsystem)
To: usenet
Subject: Problems with your active file

The following newsgroups are not valid and should be removed.
alt.ascii-art
bionet.molbio.gene-org
comp.windows.x.intrisics
de.answers

You can do this by executing the commands:
/usr/lib/news/bin/maint/delgroup alt.ascii-art
/usr/lib/news/bin/maint/delgroup bionet.molbio.gene-org
/usr/lib/news/bin/maint/delgroup comp.windows.x.intrisics
/usr/lib/news/bin/maint/delgroup de.answers
```

```
The following newsgroups were missing.
comp.binaries.cbm
comp.databases.rdb
comp.os.geos
comp.os.qnx
comp.unix.user-friendly
misc.legal.moderated
news.newsites
soc.culture.scientists
talk.politics.crypto
talk.politics.tibet
```

Pokud od systému news obdržíte zprávu tohoto typu, pak jí slepě nevěřte. V závislosti na tom, kdo poslal zprávy *checkgroups*, může postrádat několik skupin až celé hierarchie. Skupiny byste tedy měli odstraňovat jen opatrně. Pokud zjistíte, že některé skupiny, jež chcete vést ve vašem systému, chybí, pak je musíte doplnit pomocí skriptu `addgroup`. Uložte seznam chybějících skupin do souboru a předejte ho následujícímu malému skriptu:

```
#!/bin/sh
cd /usr/lib/news

while read group; do
if grep -si "^$group[[:space:]].*moderated" newsgroup; then
mod=m
else
mod=y
fi
/usr/lib/news/bin/maint/addgroup $group $mod
done
```

17.8.4 Zprávy sendsys, version a senduuname

Nakonec zde máme tři zprávy, které lze využít k získávání informací o síťové topologii. Jsou to zprávy *sendsys*, *version* a *senduuname*. Systém C News pošle po jejich obdržení odesílateli buď soubor `sys`, řetězec s verzí softwaru nebo výstup programu `uuname`*(1)*. C News jsou při poskytování zpráv ohledně verzí velmi skoupé; vrátí pouze písmeno „C“.

I zde platí, že tento typ zpráv byste neměli posílat, pokud si nejste absolutně jistí, že nemohou opustit vaši (regionální) síť. Odpovědi na zprávy *sendsys* mohou snadno shodit síť UUCP.[10]

17.9 C News v prostředí NFS

Jednoduchý způsob, jak šířit news v rámci lokální počítačové sítě, je uchovávat všechny news na ústředním hostiteli a relevantní soubory exportovat prostřednictvím souborového systému NFS, kdy programy pro čtení news mohou přímo procházet články. Výhodou této metody oproti využití protokolu NNTP je mnohem nižší režie spojená se získáváním a řazením článků. Protokol NNTP na druhé straně vítězí v heterogenní síti, kde se vybavení jednotlivých hostitelů výrazně liší nebo kde uživatelé nemají ekvivalentní účty na serveru.

Při použití souborového systému NFS musí být články zaslané lokálnímu hostiteli přesměrovány na centrální počítač, protože přístup k administrativním souborům by v opačném případě mohl vystavit systém nebezpečným podmínkám, které by způsobily nekonzistenci souborů. Nebo se můžete rozhodnout chránit vaši oblast spool tím, že ji exportujete jen pro čtení, což také vyžaduje přesměrování na centrální počítač.

Systém C News provádí tyto úkoly transparentně. Jakmile pošlete nějaký článek, váš program pro čtení news zpravidla spustí program `inews`, který vloží daný článek do systému news. Tento příkaz podrobí článek několika kontrolám, doplní hlavičku a projde soubor `server` v adresáři `/usr/lib/news`. Pokud tento soubor existuje a obsahuje jiný název hostitele, než je název lokálního hostitele, spustí na tomto serveru prostřednictvím `rsh` program `inews`. Protože tento skript používá několik binárních příkazů a podporuje soubory systému C News, musíte mít buď lokálně nainstalován C News, nebo si připojit software news ze serveru.

Aby spojení `rsh` správně fungovalo, musí mít každý uživatel na serveru ekvivalentní účet, tj. takový, na který se může přihlásit bez hesla.

Ujistěte se, že název hostitele v souboru `server` přesně odpovídá výstupu příkazu `hostname`*(1)* na serveru. Jinak by se totiž systém C News při doručování článku navždy zacyklil.

17.10 Nástroje pro údržbu

Navzdory složitosti systému C News může být život jeho správce poměrně snadný, protože tento systém nabízí širokou paletu nástrojů pro údržbu. Některé z nich jsou určeny pro pravidelné spouštění z `cron`, například `newsdaily`. Používání těchto skriptů významně snižuje nároky na denní péči o vaši instalaci C News.

10 Ani na Internetu bych to nezkoušel.

Nebude-li uvedeno jinak, jsou následující příkazy umístěny v adresáři `/usr/lib/news/bin/maint`. Nezapomeňte, že před spuštěním těchto příkazů, se musíte přihlásit jako uživatel **news**. Pokud byste tyto příkazy spustili jako superuživatel, mohlo by dojít k tomu, že se tyto soubory stanou pro systém C News nepřístupnými.

`newsdaily` Význam tohoto souboru je zřejmý z jeho názvu. Jde o důležitý skript, který pomáhá udržet log-soubory přijatelně velké a ponechává poslední tři kopie každého z nich. Také se pokouší napravit různé anomálie, jako jsou staré dávky v příchozích a odchozích adresářích, příspěvky do neznámých nebo moderovaných diskusních skupin atd. Výsledné chybové zprávy pak pošle správci news.

`newswatch` Tento skript byste měli spouštět pravidelně, zhruba jednou za hodinu, protože hledá anomálie v systému news. Má vystopovat problémy, které by měly okamžitý vliv na funkčnost systému news a poslat o tom správci hlášení. Mezi kontrolované oblasti patří zamykací soubory, které se nepodařilo odstranit, neobsloužené vstupní dávky a nedostatek místa na disku.

`addgroup` Přidá do vašeho lokálního systému novou skupinu. Správný formát tohoto příkazu je

```
addgroup groupname y|n|m|=realgroup
```

Druhý argument má stejný význam, jako příznak v souboru `active` a může znamenat, že do dané skupiny může přispívat kdokoliv (g), nikdo (n), že je moderovaná (m) nebo že se jedná o alias pro jinou skupinu (*=realgroup*).

Tento příkaz možná využijete také v případě, kdy první články nově vytvořené skupiny dorazí dříve, než řídicí příkaz *newgroup*, který má vyvolat její vytvoření.

`delgroup` Umožňuje lokální smazání nějaké skupiny. Spustíte ho jako

```
delgroup groupname
```

Pořád ještě ale musíte smazat články, které zůstaly v adresáři spool. Anebo můžete jejich odstranění svěřit přirozenému běhu věcí (tj. programu *expire*).

`addmissing` Přidá chybějící články do souboru `history`. Tento skript použijte, máte-li podezření, že se nějaké články v systému zasekly.[11]

[11] Vždycky mě zajímalo, jak se zbavit článku „Pomoc, nemůžu rozchodit X11 s 0.97.2!!!“.

`newsboot` Tento skript by se měl spouštět při zavádění systému. Odstraní všechny zamykací soubory, které zbyly po „zabití“ procesů news při ukončování systému a uzavře a provede všechny dávky, které zde zůstaly z NNTP-spojení, jež byly přerušeny při ukončování systému.

`newsrunning` Tento skript najdete v adresáři `/usr/lib/news/bin/input` a lze ho použít k zakázání rozbalování příchozích news, například v pracovní době. Rozbalování dávek vypnete následujícím způsobem

```
/usr/lib/news/bin/input/newsrunning off
```

Rozbalování znovu zapnete, když místo řetězce *off* použijete *on*.

Kapitola

18

Popis protokolu NNTP

18.1 Úvod

Z důvodu existence různých přenosových protokolů poskytuje protokol NNTP zcela odlišný způsob výměny news, než jaký používá systém C News. NNTP znamená „Network News Transfer Protocol“ (Protokol pro přenos síťových news) a není vlastním softwarovým balíkem, nýbrž internetovým standardem[1]. Je založen na „proudově orientovaném“ spojení – většinou prostřednictvím protokolu TCP – mezi klientem na libovolném místě v síti a serverem, který uchovává síťové news ve svém diskovém prostoru. Toto proudové spojení umožňuje klientovi a serveru interaktivní přenos článků s takřka nulovým zpožděním, čímž se daří udržovat nízký počet duplicitních článků. Společně s vysokými přenosovými rychlostmi Internetu se dosahuje přenosu news, který daleko překonává původní sítě UUCP. Zatímco před dvěma roky bylo zcela běžné, když článek dorazil do posledního uzlu Usenetu za dva týdny, podařilo se nyní tuto dobu zkrátit na méně než dva dny a v samotném Internetu je to často otázka několika minut.

K získávání, posílání a předávání článku slouží klientům různé příkazy. Rozdíl mezi odesláním a předáním je ten, že druhá metoda se může týkat článků s neúplnými hlavičkovými informacemi[2]. Nalezení článku mohou využívat klienti pro přenos news stejně jako programy pro čtení news. Tím se stává protokol NNTP vynikajícím nástrojem, který umožňuje mnoha klientům v lokální síti přístup k news, přičemž se vyhnou zkomoleninám, k nimž dochází při použití NFS.

---

1 Formálně je specifikován v RFC 977.

2 Při předávání článku prostřednictvím protokolu NNTP k němu server vždy přidá alespoň jedno hlavičkové pole, které se jmenuje `Nntp-Posting-Host:`. Obsahuje hostitelský název klienta.

Protokol NNTP umožňuje také aktivní a pasivní způsob přenosu news, kterému se hovorově říká „pushing“ (tlačení) a „pulling“ (tažení). Tlačení používá v podstatě stejný způsob, jako protokol C News ihave/sendme. Klient nabízí serveru články prostřednictvím příkazu `IHAVE<varmsig>` a server mu vrátí odpověď, z níž vyplývá, zda již daný článek má, nebo ho požaduje. Pokud o něj má zájem, klient mu článek pošle a ukončí ho tečkou na samostatném řádku.

Tlačení news má jednu nevýhodu v tom, že poměrně výrazně zatěžuje systém serveru, protože ten musí kvůli každému článku prohledávat databázi historie.

Opačnou technikou je tažení news, kdy klient požaduje seznam všech (dostupných) článků diskusní skupiny, které dorazily po specifikovaném datu. Takovýto dotaz předává pomocí příkazu `NEWNEWS`. Ze získaného seznamu id-zpráv si klient vybere ty články, které ještě nevlastní a postupně o ně požádá server příkazem *ARTICLE*.

Problémem tažení news je, že server potřebuje mít těsnou kontrolu nad skupinami a distribucemi, které povoluje klientovi požadovat. Musí se například ujistit, že nebude neautorizovaným klientům poslán tajný materiál z lokální diskusní skupiny.

Programy pro čtení news také disponují několika pohodlnými příkazy, které jim dovolují odděleně získání hlaviček a těl článků, případně i jednotlivých řádků článků. Takto je možné udržovat všechny news na centrálním hostiteli, k němuž mohou všichni uživatelé lokální sítě přistupovat prostřednictvím klientských programů pro čtení a posílání news, které pracující na bázi protokolu NNTP. Jedná se o alternativní řešení pro exportování adresářů news prostřednictvím systému NFS, což bylo popsáno v kapitole 17.

Celkový problém protokolu NNTP tkví v tom, že umožňuje informovanému člověku vložit do proudu news články s falešnou specifikací odesílatele. Tomu se říká *falšování news* (*news faking*)[3]. Rozšíření protokolu NNTP umožňuje u určitých příkazů vyžadovat autentifikaci uživatele.

V současné době existuje několik balíků protokolu NNTP. Jedním z nejpoužívanějších je démon NNTP, který je referenční implementací tohoto protokolu. Jeho původními autory jsou Stan Barber a Phil Lapsley, kteří chtěli jeho prostřednictvím ilustrovat detaily RFC 977. Nejaktuálnější verze má číslo `nntpd-1.5.11`, a tu si nyní popíšeme. Můžete si buď sehnat zdrojové soubory a sami si je přeložit, nebo použít program `nntpd` z balíku binárních souborů *net-std* od Freda van Kempena. Z důvodu různých konfiguračních nastavení, která jsou specifická pro jednotlivé systémy, nejsou poskytovány binární soubory balíku `nntpd`.

[3] Stejný problém se vyskytuje v souvislosti s protokolem SMTP, Simple Mail Transfer Protocol.

Balík `nntpd` tvoří server a dva klienti pro tlačení a tažení news a najdete zde také náhradu programu `inews`. Vše je určeno pro prostředí Bnews, ale po několika úpravách bude fungovat i v prostředí C News. Pokud však plánujete používat protokol NNTP i k jiným účelům, než pro zprostředkování přístupu k vašemu serveru news, není pro vás probíraná implementace tou pravou volbou. Proto budeme probírat pouze démona NNTP z balíku `nntpd` a vynecháme popis klientských programů.

K dispozici je také balík nazvaný „InterNet News“, zkráceně INN, jehož autorem je Rich Salz. Umožňuje přenos jak NNTP, tak i UUCP-news a je vhodnější pro velké systémy news. Pro přenos news prostřednictvím protokolu NNTP je rozhodně lepší než `nntpd`. Balík INN je v současné době ve verzi `inn-1.4sec`. Existuje také sada nástrojů pro sestavení INN na linuxovém počítači. Jejím autorem je Arjan de Vet a je dostupná na serveru **sunsite.unc.edu** v adresáři `system/Mail`. Při konfiguraci balíku INN se řiďte dokumentací, která je dodávána společně se zdrojem, případně INN FAQ, které jsou pravidelně posílány do konference **news.software.b**.

18.2 Instalace serveru NNTP

Server NNTP se nazývá `nntpd` a lze ho zkompilovat dvěma způsoby, v závislosti na očekávaném zatížení systému news. Kvůli implicitním hodnotám specifickým pro některá místa, jež jsou zakódovány do spustitelného souboru, nejsou k dispozici žádné zkompilované verze. Veškerou konfiguraci provádí makro, které je definováno v souboru `common/conf.h`.

Server `nntpd` může být nakonfigurován buď jako samostatný server, který je spouštěn při zavádění systému z `rc.inet2`, nebo jako démon řízený `inetd`. Ve druhém případě je třeba mít v souboru `/etc/inetd.conf` následující záznam:

```
nntp   stream  tcp nowait   news   /usr/etc/in.nntpd   nntpd
```

Pokud konfigurujete `nntpd` jako samostatný server, ujistěte se, že je každý podobný řádek řádně okomentován. V každém případě se musíte přesvědčit, že v souboru `/etc/services` nechybí následující řádek:

```
nntp   119/tcp    readnews untp    # Network News Transfer Protocol
```

Aby bylo možné ukládat příchozí články apod., potřebuje mít server `nntpd` v adresáři `spool` také adresář `.tmp`. Vytvoříte ho následovně:

```
# mkdir       /var/spool/news/.tmp
# chown news.news /var/spool/news/.tmp
```

18.3 Omezení přístupu k NNTP

Přístup ke zdrojům NNTP je řízen souborem `nntp_access`, který je uložen v adresáři `/usr/lib/news`. Řádky v tomto souboru specifikují přístupová práva přidělená vzdáleným hostitelům. Každý řádek má následující formát:

site `read|xfer|both|no post|no [!exceptgroup]`

Jakmile se klient připojí na NNTP-port, pokusí se server `nntpd` pomocí zpětného výhledávání podle IP-adresy získat plně kvalifikovaný název domény. Hostitelský název klienta a jeho IP-adresa jsou porovnány s polem `site` každého záznamu v takovém pořadí, v jakém jsou uvedeny v souboru. Shoda může být buď částečná, nebo úplná. V případě úplné shody ji použije. V případě jen částečné shody ji použije jen pokud se nenajde žádná další částečná shoda. Pole `site` je možné specifikovat jedním z následujících způsobů:

hostname Toto je plně kvalifikovaný název domény hostitele. Pokud přesně odpovídá kanonickému hostitelskému názvu klienta, použije se tento záznam a všechny další záznamy budou ignorovány.

IP address Toto je IP-adresa ve tečkové notaci. Pokud jí IP-adresa klienta odpovídá, použije se tento záznam a všechny následující záznamy budou ignorovány.

domain name Toto je název domény zadaný ve formě *.domain. Pokud hostitelský název klienta odpovídá názvu domény, pak záznam vyhovuje.

network name Toto je název sítě specifikovaný v souboru `/etc/networks`. Pokud číslo sítě klientovy IP-adresy odpovídá číslu sítě sdružené s názvem sítě, pak záznam vyhovuje.

default Této volbě vyhovuje každý klient.

Záznamy s obecnější specifikací adres je dobré uvést na začátku souboru, protože případné shody budou pozdějšími přesnějšími shodami potlačeny.

Druhé a třetí pole popisují přístupová práva přidělená danému klientovi. Druhé pole uvádí přístupová práva pro získávání news metodou pull (*read*) a předávání news metodou push (*xfer*). Hodnota *both* povoluje oba přístupy, zatímco hodnota *no* přístup úplně zakazuje. Třetí pole uděluje klientovi právo předávat články, což znamená doručovat články s neúplnými hlavičkovými informacemi, které pak doplní software news. Pokud druhé pole obsahuje hodnotu *no*, je třetí pole ignorováno.

Čtvrté pole je volitelné a obsahuje čárkami oddělený seznam skupin, k nimž má klient zakázaný přístup.

Následuje ukázkový soubor `nntp_access`:

```
#
# všichni budou přebírat news, ale ne číst ani posílat
default                    xfer          no
#
# hostitel public.vbrew.com poskytuje veřejný přístup po modemu
# povolíme čtení a posílání do všech skupin kromě lokálních
public.vbrew.com           read          post   !local
#
# ostatní počítače pivovaru mohou číst i posílat
*.vbrew.com                read          post
```

18.4 Autorizace NNTP

Při realizaci přístupových symbolů typu *xfer* nebo *read* v souboru `nntp_access` vyžaduje server `nntpd` po klientovi autorizaci příslušných operací. Když například specifikujete přístupové právo *Xfer* nebo *XFER*, nepovolí server `nntpd` klientovi přenos článku do vašeho systému, pokud neprojde autentifikací.

Proces autentifikace je implementován prostřednictvím nového příkazu protokolu NNTP, který se nazývá *AUTHINFO*. Za pomoci tohoto příkazu předá klient serveru NNTP uživatelské jméno a heslo. Server `nntpd` je porovná s databází `/etc/passwd` a ověří, že příslušný uživatel patří do skupiny **nntp**.

18.5 Interakce serveru nntpd a systému C News

Po přijetí článku ho musí server `nntpd` doručit do subsystému news. V závislosti na tom, zda jej obdržel jako výsledek příkazu *IHAVE* nebo *POST*, je článek předán systému `rnews` nebo `inews`. Místo spouštění programu `rnews` jej také můžete (v době překladu) nastavit tak, aby zpracovával příchozí články do dávky a výsledné dávky pak posílal do souboru `/var/spool/news/in.coming`, kde si je při příštím dotazu vyzvedne program `relaynews`.

Aby mohl řádně používat protokol ihave/sendme, musí mít server `nntpd` přístup k souboru `history`. V době překladu se proto musíte ujistit, že máte správně nastavenu příslušnou cestu. Také je třeba se přesvědčit, že se systém C News a `nntpd` dohodli na formátu vašeho souboru history. Systém C News používá při přístupu k němu hašovací funkce `dbm`, která má

poměrně hodně různých a mírně odlišných implementací. Pokud by byl systém C News slinkován s jinou knihovnou `dbm`, než kterou máte ve vaší knihovně `libc`, musíte s touto knihovnou slinkovat také server `nntpd`.

Typickým symptomem nesourodosti databázového formátu serveru `nntpd` a systému C News jsou chybové zprávy v systémovém log-souboru, že server `nntpd` nemohl tuto databázi otevřít nebo prostřednictvím protokolu NNTP obdržel duplicitní články. Vhodným testem je vzít článek z adresáře `spool`, navázat telnetové spojení se serverem `nntpd` a nabídnout ho serveru `nntpd`, jak je demonstrováno v níže uvedeném příkladu (váš vstup je označen *takto*). Samozřejmě, že musíte nahradit řetězec <msg@id> řetězcem id zprávy článku, který chcete předat serveru `nntpd`.

```
$ telnet localhost nntp
Trying 127.0.0.1...
Connected to localhost
Escape characters is '^]'.
201 vstout NNTP[auth] server version 1.5.11t (16 November
1991) ready at Sun Feb 6 16:02:32 1194 (no posting)
IHAVE <msg@id>
435 Got it.
QUIT
```

Tento rozhovor ukazuje příslušnou reakci serveru `nntpd`; zpráva „Got it“ říká, že již příslušný článek má. Pokud místo toho obdržíte zprávu „335 Ok“, pak vyhledávání v souboru history z nějakého důvodu selhalo. Ukončete rozhovor stiskem kombinace kláves Ctrl+D. Co se stalo, zjistíte na základě systémového log-souboru; server `nntpd` zapisuje všechny druhy zpráv do souboru `syslog`. Nekompatibilní knihovnu `dbm` zpravidla zjistíte podle zprávy se stížností na selhání programu `dbminit`.

Kapitola

19

Konfigurace programu pro čtení news (Newsreader)

Cílem programů pro čtení news je zprostředkovat uživatelům snadný a komfortní přístup k funkcím systému news, jako je předávání článků nebo sbírání obsahu diskusní skupiny. Kvalita tohoto rozhraní je předmětem vášnivých debat.

Na platformu Linuxu bylo portováno několik programů pro čtení news. Dále si popíšeme základní nastavení tří nejoblíbenějších z nich, jmenovitě to budou `tin`, `trn` a `nn`.

Jedním z nejefektivnějších nástrojů pro čtení news získáte následujícím zápisem:

```
$ find /var/spool/news -name '[0-9]*' -exec cat {} \; | more
```

Tímto způsobem čtou news unixoví nadšenci.

Většina programů pro čtení news je však mnohem kultivovanější. Obvykle nabízí celoobrazovkové rozhraní s oddělenými úrovněmi nabízejícími zobrazení všech skupin, do kterých je uživatel přihlášen, zobrazení přehledu všech článků v jedné skupině a konečně i jednotlivých článků.

Na úrovni diskusní skupiny zobrazuje většina programů seznam článků, ve kterém je uveden předmět článku a jeho autor. Ve velkých skupinách nemůže uživatel sledovat vzájemnou provázanost článků, i když je možné vystopovat odpovědi na dřívější články.

V odpovědi je většinou zopakován předmět původního článku, před nějž je předsunut řetězec „`Re:`“. Kromě toho je id zprávy článku vytvořeno tak, aby přímo následovalo po předchozím a mohlo být umístěno do hlavičky `References`:. Seřazením článků podle těchto dvou kri-

terií vzniknou malé shluky (ve skutečnosti se jedná o stromy) článků, kterým se říká *vlákna* (*threads*). Jedním z úkolů při psaní programu pro čtení news je vymyslet efektivní schéma pro tvorbu vláken, protože čas potřebný pro takovéto seřazení je úměrný druhé mocnině celkového počtu článků

V této kapitole se však nebudeme hlouběji zabývat způsobem, jakým jsou vystavěna uživatelská rozhraní. Všechny programy pro čtení news, které jsou v současné době v Linuxu dostupné, disponují dobrou nápovědou, takže by vám práce s nimi neměla činit potíže.

Dále se budeme zabývat pouze administrativními úkoly. Většinou bude řeč o tvorbě databází vláken a evidencí.

19.1 Konfigurace programu tin

Nejvšestrannějším programem pro čtení news s ohledem na tvorbu vláken je program `tin`. Napsal ho Iain Leas a je volně odvozen od staršího programu jménem `tass`[1]. Propočet vláken provádí v době, kdy uživatel vstoupí do diskusní skupiny, a kromě případu, kdy používáte spojení NNTP, je poměrně rychlý.

Na počítači 486DX50 zabere setřídění 1000 článků při čtení přímo z disku zhruba 30 sekund. Při spojení NNTP se zatíženým serverem news by tato operace zabrala více než 5 minut.[2] Tuto dobu je možné zkrátit pravidelnou aktualizací indexového souboru, které dosáhnete pomocí volby `-u` nebo spuštěním programu `tin` s parametrem `-U`.

Program `tin` obvykle ukládá svou databázi vláken do souboru `.tin/index` v domovském adresáři uživatele. Tato konfigurace však může příliš zatěžovat systémové zdroje, takže možná budete chtít udržovat jedinou kopii těchto souborů na nějakém centrálním místě. Toho dosáhnete tak, že programu `tin` přidělíte například účet **news** nebo nějaký jiný účet s omezenými právy.[3] Program `tin` pak bude udržovat všechny databáze v souboru `/var/spool/news/.index`. Při každém přístupu k souboru nebo ukončení příkazového interpretu nastaví efektivní uid na skutečné uid uživatele, který ho spustil.[4]

1 Jeho autorem je Rich Skrenta.

2 K výraznému zlepšení dojde, pokud NNTP-server provede setřídění sám a klientovi předá přímo databáze vláken; umí to například program INN verze 1.4.

3 Pro tento účel však nepoužívejte účet **nobody**. Mělo by se dodržovat nepsané pravidlo, a totiž nesdružovat s tímto účtem žádné soubory nebo příkazy.

4 Z toho důvodu obdržíte nepěkné chybové zprávy, když tento program spustíte jako superuživatel. Kvůli tomu byste s ním neměli pracovat jako **root**.

Lepším řešením je nainstalovat indexovacího démona `tind`, který bude pravidelně aktualizovat indexové soubory. Tento démon však není součástí žádné distribuce Linuxu, takže si ho budete muset i sami přeložit. Pokud provozujete lokální síť s centrálním serverem news, můžete dokonce spouštět démona `tind` na tomto serveru a nechat klienty, aby si stahovali indexové soubory pomocí protokolu NNTP. K tomu samozřejmě potřebujete rozšíření protokolu NNTP. Doplňky démona `nntpd`, které zavádějí toto rozšíření, jsou součástí zdrojového kódu programu `tin`.

Verze programu `tin`, která byla součástí některých distribucí Linuxu, v sobě neměla zakomponovánu podporu protokolu NNTP, ale v současné době ji většina distribucí již má. Pokud program spustíte jako `rtin` nebo s volbou `-r`, pokusí se *tin* spojit s NNTP serverem uvedeným v souboru `/etc/nntpserver` nebo v proměnné prostředí `NNTPSERVER`. Soubor `nntpserver` obsahuje na jediném řádku pouze název serveru.

19.2 Konfigurace programu trn

Program `trn` je také následníkem staršího programu pro čtení news, jmenovitě je to program `rn` (což znamená *read news*). Písmeno „t“ v jeho názvu znamená „threaded“. Napsal ho Wayne Davidson.

Na rozdíl od programu `tin` neumí program `trn` generovat svou databázi vláken za běhu. Místo toho využívá databázi vytvořenou programem `mthreads`, který je pravidelně spouštěn z `cron` kvůli aktualizaci indexových souborů.

Když nespustíte program `mthreads`, neznamená to, že byste nemohli přistupovat k novým článkům, budete je ale všechny mít roztroušeny po výběrovém menu článků, místo jednoho vlákna, které by šlo lehce přeskočit.

Budete-li chtít zapnout tvorbu vláken pro konkrétní diskusní skupinu, spustíte program `mthreads` a jako argumenty mu na příkazové řádce předáte seznam diskusních skupin. Seznam vytvoříte stejným způsobem, jaký jste použili při tvorbě souboru `sys`:

```
mthreads comp,rec,!rec.games.go
```

Výše uvedený zápis povolí vytváření vláken pro všechny skupiny **comp** a **rec** s výjimkou skupiny **rec.games.go** (lidé, kteří hrají hru Go, nepotřebují líbivá vlákna). Potom spustíte stejný program bez argumentů, aby mohl zařadit nově příchozí články. Uspořádání všech skupin, které máte uvedeny v souboru `active`, lze zapnout pomocí argumentu **all**, který předáte programu `mthreads` v příkazové řádce.

Pokud dostáváte news přes noc, budete obvykle spouštět program `mthreads` jen jednou, a to ráno, ale dle potřeby to může být i častěji. Systémy s velkým provozem možná budou používat program `mthreads` v režimu démona. Pokud ho totiž spustíte při zavádění systému s parametrem `-d`, umístí sám sebe na pozadí a každých deset minut bude zjišťovat, zda nedošly nějaké nové články, a pokud ano, pak je zařadí. Chcete-li spouštět program `mthreads` jako démona, umístěte do skriptu `rc.news` následující řádek:

```
/usr/local/bin/rn/mthreads -deav
```

Parametr `-a` zapíná automatickou tvorbu vláken pro nové skupiny, parametr `-v` povoluje zápis zpráv do log-souboru programu `mthread`, který se nazývá `mt.log` a je umístěn v adresáři, ve kterém máte nainstalován program `trn`.

Staré články, které již nejsou k dispozici, je třeba z indexových souborů pravidelně mazat. Implicitně jsou odstraňovány pouze články, jejichž číslo je pod dolním vodoznakem (water mark).[5] Články nad tímto číslem, jejichž doba platnosti by přesto nevypršela (protože nejstaršímu článku mohla být za pomoci hlavičkového pole `Expires:` přidělena dlouhá doba platnosti), lze odstranit pomocí parametru `-e`, který spustí tzv. pokročilé vyhodnocování doby platnosti. Pokud program `mthreads` běží jako démon, způsobí parametr `-e`, že přejde do tohoto režimu jednou denně - krátce po půlnoci.

19.3 Konfigurace programu nn

Program `nn`, jehož autorem je Kim F. Storm, o sobě tvrdí, že je nástrojem, jehož cílem je nejenom umožňovat čtení news. Jeho název je zkratkou „No News“ a má následující moto: „No News is good news. `nn` is better.“ („Žádné zprávy – dobré zprávy. Program `nn` je lepší.“)

Aby dosáhl tohoto nesmělého cíle, přichází program `nn` s velkou zásobou podpůrných nástrojů, které dovolují nejenom generovat vlákna, ale umožňují také intenzívní kontrolu konzistence těchto databází a účtů, získávání statistiky využití a omezení přístupů. Součástí balíku je i administrativní program nazvaný `nnadmin`, který dovoluje provádět tyto úkoly interaktivně. Je velmi intuitivní, takže se jím zde nebudeme zabývat a zaměříme se pouze na vytváření indexových souborů.

Databáze vláken programu `nn` se nazývá `nnmaster`. Obvykle běží jako démon spuštěný ze skriptu `rc.news` nebo `rc.inet2`. Spouští se následovně:

```
/usr/local/lib/nn/nnmaster -l -r -C
```

[5] Všimněte si, že systém C News neaktualizuje tento dolní vodoznak automaticky, ale je třeba spustit program `updatemin`, který to provede za něj. Podrobnosti viz kapitola 17.

Tento zápis povolí vytváření vláken pro všechny diskusní skupiny, které jsou uvedeny v souboru *active*.

Jinou možností je pravidelně spouštět program `nnmaster` z `cron` a přitom mu předat seznam příslušných skupin. Tento seznam je podobný seznamu zapsaných skupin v souboru `sys`. Liší se jen v tom, že místo čárek používá jako oddělovače mezery. Místo falešného názvu skupiny **all** byste měli k označení všech skupin používat prázdný řetězec „“. Program lze spustit například takto:

```
# /usr/local/lib/nn/nnmaster !rec.games.go rec comp
```

Všimněte si, že v tomto zápisu záleží na pořadí prvků v seznamu. Pokud bychom uvedli řetězec `!rec.games.go` až za řetězcem `rec`, byly by vždy zpracovány všechny články z této skupiny.

Program `nn` nabízí několik metod pro odstraňování článků, jimž vypršela doba platnosti. První z nich provádí aktualizaci databáze tak, že projde adresáře všech diskusních skupin a zruší ty záznamy, k nimž již neexistují odpovídající záznamy. Tato operace je implicitní a lze ji aktivovat i pomocí parametru `-E`. Kromě případu, kdy ji provádíte prostřednictvím NNTP, je poměrně rychlá.

Druhá metoda se chová úplně stejně jako program `mthreads` při implicitním spuštění. Odstraní totiž pouze ty záznamy, které odkazují na články, jejichž číslo je nižší než dolní vodoznak v souboru `active`. Tuto metodu aktivujete pomocí parametru `-e`.

Konečně třetí strategie spočívá ve zrušení celé databáze a opětovném sesbírání všech článků. Aktivujete ji, když program `nnmaster` spustíte s parametrem `-E3`.

Seznam skupin, jejichž platnost již vypršela, se předává pomocí parametru `-F` stejným způsobem. Pokud však spouštíte program `nnmaster` jako démona, musíte ho před aktualizací databáze „zabít“ (pomocí parametru `-k`) a znovu ho spustit s původními parametry. Příslušný příkaz pro spuštění kontroly vypršení platnosti všech skupin podle první metody tedy vypadá takto:

```
# nnmaster -kF ""
# nnmaster -lrC
```

Chování programu `nn` je možné upravit prostřednictvím mnoha dalších parametrů. Chcete-li vědět, jak odstraňovat špatné články nebo provádět výtahy z článků, pak si přečtěte manuálové stránky programu `nnmaster`.

Program `nnmaster` spoléhá na soubor nazvaný *GROUPS*, který je umístěn v adresáři `/usr/local/lib/nn`. Pokud neexistuje, program si ho sám vytvoří. Pro každou diskusní skupinu obsahuje tento soubor řádek začínající názvem této skupiny, za nímž může volitelně

následovat časové označení a příznaky. Úpravou těchto příznaků je možné změnit zacházení programu s touto skupinou. Nelze však změnit pořadí, ve kterém se skupiny objevují.[6] Přípustné příznaky a jejich význam také najdete v manuálových stránkách programu `nnmaster`.

6 To proto, že pořadí skupin musí souhlasit s pořadím záznamů v (binárním) souboru `MASTER`.

A

Nulový tiskový kabel pro PLIP

K sestavení nulového tiskového kabelu pro spojení PLIP potřebujete dva 25pinové konektory (označení DB-25) a nějaký 11vodičový kabel. Tento kabel může být dlouhý maximálně 15 metrů.

Podíváte-li se pozorně na konektor, měli byste vedle každého pinu přečíst drobná čísla, přičemž jedničku má pin vlevo nahoře (držíte-li konektor širší stranou nahoru) a pin v levém dolním rohu má číslo 25. Nulový tiskový kabel vytvoříte vzájemným propojením následujících pinů obou konektorů:

```
D0          2   ---   15     ERROR
D1          3   ---   13     SLCT
D2          4   ---   12     PAPOUT
D3          5   ---   10     ACK
D4          6   ---   11     BUSY
GROUND     25   ---   25     GROUND
ERROR      15   ---    2     D0
SLCT       13   ---    3     D1
PAPOUT     12   ---    4     D2
ACK        10   ---    5     D3
BUSY       11   ---    6     D4
```

Zbylé piny zůstanou nezapojené. Jedná-li se o stíněný kabel, může být stínění na jednom konci připojeno ke kovovému pouzdru.

B

Vzorové konfigurační soubory programu smail

Tento dodatek popisuje vzorové konfigurační soubory pro koncový uzel UUCP v lokální síti. Vycházejí z ukázkových souborů, které jsou součástí distribuce programu `smail-3.1.28`. I když se zde pokusíme o stručný výklad způsobu, jakým tyto soubory fungují, doporučujeme vám přečíst si velmi hezky zpracovanou manuálovou stránku `smail`*(8)*, kde jsou tyto soubory popsány lépe. Jakmile porozumíte základní myšlence konfigurace programu `smail`, zjistíte, že stojí za to si ji přečíst.

První je soubor `routers`, který popisuje sadu směrovačů pro program `smail`. Když má program `smail` doručit zprávu na danou adresu, předá tuto adresu postupně všem směrovačům, dokud některému z nich nevyhovuje, což zde znamená, že příslušný směrovač našel cílového hostitele ve své databázi, ať už je to soubor `paths`, `/etc/hosts` nebo jiný směrovací mechanismus, který směrovač používá.

Záznamy v konfiguračních souborech programu `smail` začínají jedinečným názvem, který identifikuje příslušný směrovač, přenos nebo direktor. Následuje seznam atributů, které definují jeho chování. Tento seznam je tvořen globálními atributy, jako například použitým ovladačem a soukromými atributy, kterým rozumí pouze konkrétní ovladač. Jednotlivé atributy jsou odděleny čárkami a sady globálních a soukromých atributů jsou navzájem odděleny středníkem.

Aby vám byl tento rozdíl jasný, předpokládejme, že chcete udržovat dva oddělené soubory s aliasy; jeden bude obsahovat směrovací informace pro vaši doménu a druhý pak globální směrovací informace, generované pravděpodobně z map UUCP. Pomocí programu `smail` nyní můžete v souboru `routers` specifikovat dva směrovače, které budou oba používat ovladač `pathalias`. Tento ovladač bude vyhledávat názvy hostitelů v databázi aliasů. Očekává, že mu bude jako soukromý atribut předán název souboru:

```
#
# pathalias database for intra-domain routing
domain_paths:
   driver=pathalias,     # look up host in a paths file
   transport =uux;       # if matched, deliver over UUCP
   file=paths/domain,    # file is /usr/lib/smail/paths/domain
   proto=lsearch,        # file is unsorted (linear search)
   optional,             # ignore if the file does not exist
   required=vbrew.com,   # look up only *.vbrew.com hosts
#
# pathalias database for routing to hosts outside our domain
world_paths:
   driver=pathalias,     # look up host in a paths file
   transport =uux;       # if matched, deliver over UUCP

   file=paths/world,     # file is /usr/lib/smail/paths/world
   proto=bsearch,        # file is sorted with sort(1)
   optional,             # ignore if the file does not exist
   -required,            # no required domains
   domain=uucp,          # string ending ".uucp" before searching
```

Druhý globální atribut uvedený v obou záznamech směrovačů definuje přenos, který má být použit v případě, že směrovač vyhovuje dané adrese. V našem případě bude zpráva doručena přenosem `uux`. Přenosy jsou definovány v souboru `transports`, který bude popsán dále.

Způsob přenosu si můžete lépe přizpůsobit, když místo atributu `transport` specifikujete soubor metod. Soubory metod poskytují mapování cílových hostitelů na přenosy. Zde se jimi však nebudeme zabývat.

Následující soubor `routers` definuje směrovače pro lokální síť, která se dotazuje knihovny resolveru. Na internetovém hostiteli byste však měli používat směrovač, který umí zpracovávat MX-záznamy. V tom případě je nutné zrušit komentáře u alternativního směrovače *inet_bind*, který používá zabudovaný ovladač BIND programu `smail`.

V prostředí, které kombinuje protokoly UUCP a TCP/IP možná narazíte na problém, kdy v souboru `/etc/hosts` máte hostitele, s nimiž dochází k jen příležitostnému spojení pomocí protokolů SLIP nebo PPP. Většinou jim budete chtít i nadále posílat poštu prostřednictvím protokolu UUCP. Aby si ovladač *inet_hosts* těchto hostitelů nevšímal, musíte je umístit do souboru `paths/force`. Jedná se o další databázi aliasů, která je prohledána ještě předtím, než program `smail` předá dotaz resolveru.

```
# A sample /usr/lib/smail/routers file
#
# force - force UUCP delivery to certain hosts, even when they are
in out /etc/hosts:
force:
   driver=pathalias,          # look up host in a paths file
   transport=uux;             # if matched, deliver over UUCP

   file=paths/force,          # file is /usr/lib/smail/paths/force
   optional,                  # ignore if the file does not exist
   proto=lsearch,             # file is unsorted (linear search)
   -required,                 # no required domains
domain=uucp,                  # string ending ".uucp" before sear-
ching

# inet_addrs - match domain literals containing literal
#         IP addresses, such as in janet@[191.72.2.1]
inet_addrs:
   driver=gethostbyaddr,      # driver to match IP domain literals
   transport=smtp;            # deliver using SMTP over TCP/IP

   fail_if_error,             # fail if address is malformed
   check_for_local,           # deliver directly if host is ourself

# inet_hosts - match hostnames with gethostbyname (3a)
#        Comment this out if you wish to use the BIND version inste-
ad
inet_hosts:
   driver=gethostbyname,      # match hosts with the library function
   transport=smtp;            # use default SMTP

   - required,                # no required domains
   -domain                    # no defined domain suffixes
   -only_local_domain             # don't restrict to defined domains

# inet_hosts - alternate version using BIND to access DNS
#inet_hosts:
#  driver=bind,               # use buil/in BIND driver
#  transport=smtp;            # use TCP/IP SMTP for delivery
```

```
#  defnames,                  # use standard domain searching
#  defer_no_connect           # try again if the nameserver is down
#  local_mx_okay              #fail (don't pass through) an MX to the
local host
#

# pathalias database for intra-domain routing
domain_paths:
   driver=pathalias,          # look up host in a paths file
   transport=uux;             # if matched, deliver over UUCP

   file=paths/domain,         # file is /usr/lib/smail/paths/domain
proto=lsearch,                # file is unsorted (linear search)
   optional,                  # ignore if the file does not exist
   required=vbrew.com,        # look up only *.vbrew.com hosts
#
# pathalias database for routing to hosts outside our domain
world_paths:
   driver=pathalias,          # look up host in a paths file
   transport =uux;            # if matched, deliver over UUCP

   file=paths/world,          # file is /usr/lib/smail/paths/world
   proto=bsearch,             # file is sorted with sort(1)
   optional,                  # ignore if the file does not exist
   -required,                 # no required domains
   domain=uucp,               # string ending ".uucp" before sear-
ching

# smart_hosts - a partically specified smarthost director
#  If the smart_path attribute is not defined in
#  /usr/lib/smail/config, this router is ignored.
#  The transport attribute is overriden by the global
#  smart_transport varriable
smart_host:
   driver=smarthost,          # special-case driver
   transport=uux;             # by default deliver over UUCP

   -path,                     # use smart_path config file variable
```

Manipulace s poštou, určenou pro lokální adresu, je řízena souborem `directors`. Ten je vystavěn podobným způsobenm, jako soubor `routers`, přičemž každá položka definuje jednoho direktora. Direktoři nedoručují zprávy, starají se pouze o potřebná směrování, například přes aliasy, postoupení pošty apod.

Při doručování zprávy na lokální adresu, například uživateli **janet**, předá program `smail` toto uživatelské jméno postupně všem direktorům. Pokud některému z nich vyhovuje, pak buď specifikuje přenos, kterým má být daná zpráva doručena (například do schránky daného uživatele), nebo vygeneruje novou adresu (například po vyhodnocení aliasu).

Z důvodů bezpečnosti obvykle direktoři provádějí spoustu kontrol, zda soubory, které používají, nemohly být pozměněny. Adresy získané pochybným způsobem (například ze souboru `aliases`, do kterého může kdokoliv zapisovat), jsou považovány za nebezpečné. Některé přenosové ovladače takovéto adresy odmítnou, například přenos, který doručuje zprávy do souboru.

Program `smail` kromě toho také sdružuje uživatele s každou adresou. Každá operace zápisu nebo čtení je prováděna jménem daného uživatele. Při doručování do schránky uživatele **janet**, je adresa samozřejmě sdružena s uživatelem **janet**. S jinými adresami, získanými například ze souboru `aliases`, jsou sdružena jiná uživatelká jména, například **nobody**.

Podrobnosti najdete na manuálové stránce programu `smail`*(8)*.

```
# A sample /usr/lib/smail/directories file

# aliasinclude - expand ":include:filename" addresses produced
#        by alias files
aliasinclude:
   driver=aliasinclude,       # use this special-case driver
   nobody;                    # access file as nobody user if unsecu-
re

   copysecure;                # get permissions from alias director
   copyowners,                # get owners from alias director

# forwardinclude - expand ":include:filename" addrs produced
#  by forward files
forwardinclude:
   driver=forwardinclude,     # use this special-case driver
   nobody;                    # access file as nobody user if unsecure
```

```
   checkpath,                  # check path accessibility
   copysecure;                 # get permissions from alias director
   copyowners,                 # get owners from alias director

# aliases - search for alias expansions stored in a database
aliases:
   driver=aliasfile,           # general-purpose aliasing director
   -nobody,                    # all addresses are associated
                               # with nobody by default anyway
   sender_okay,                # don't remove sender from expansions
   owner=owner-$user;          # problems go to an owner address

   file=/usr/lib/aliases,      # default: sendmail compatible
   modemask=002,               # should not be globally writable
   optional,                   # ignore if file does not exist
   proto=lsearch,              # unsorted ASCII file

# dotforward - expand .forward files in user home directories
dotforward:
   driver=forwardfile,         # general-purpose forwarding director
   owner=real-$user,           # problems go to the users's mailbox
   nobody,                     # use nobody user, if unsecure
   sender_okay;                # sender never removed from expansion

   file=~/.forward,            # .forward file in home directories
   checkowner,                 # the user can own this file
   owners=root,                # or root can own the file
   modemask=002                # it should not be globally writable
   caution=0-10:uucp:daemon,   # don't run things as root or daemons
   # be extra careful of remotely accessible home directories
   unsecure="~ftp:~uucp:~nuucp:/tmp:/usr/tmp",

#forwardto - expand a "Forward to " line at the top of
# user's mailbox file
forwardto:
   driver=forwardfile,
   owner=Postmaster,           # errors go to Postmaster
```

```
   nobody,                    # use nobody user, if unsecure
   sender_okay;               # don't remove sender from expansion

   file=/var/spool/mail/${lc:user},            # location of
user's mailbox
   forwardto,                 # enable "Forward to " check
   checkowner,                # the user can own this file
   owners=root,               # or root can own the file
   modemask=0002              # under System V, group mail can write
   caution=0-10:uucp:daemon,  # don't run things as root or daemons

# user - match users on the local host with delivery to their mail-
boxes
user:  driver=user;           # driver to match usernames

   transport=local,           # local transport goes to mailboxes

#real_user - match usernames when prefixed with the string "real-"
real_user:
       driver=user;           # driver to match usernames

   transport=local,           # local transport goes to mailboxes
   prefix="real-",            # for example, match real-root

# lists  expand mailing lists stored below /usr/lib/smail/lists
lists: driver=forwardfile,
   caution,                   # flag all addresses with caution
   nobody,                    # and then associate the nobody user
   sender_okay,               # do NOT remove the sender
   owner=owner-$user;         # the list owner

   # map the name of mailing list to lower case
   file=lists/${lc:user}
```

Po úspěšném nasměrování zprávy předá program `smail` tuto zprávu přenosu, který specifikoval směrovač nebo direktor, jemuž tato zpráva vyhovovala. Tyto přenosy jsou definovány v souboru `transports`. Znovu opakujeme, že přenos je definován jako sada globálních a soukromých voleb.

Nejdůležitější volbou definovanou v každém záznamu je ovladač, který přenos obsluhuje, například ovladač *pipe*, který spustí příkaz uvedený v atributu *cmd*. Kromě toho existuje i několik globálních atributů, které může přenos využít a které provádějí různé transformace hlavičky zprávy a volitelně i těla zprávy. Atribut *return_path* například způsobí, že přenos vloží do hlavičky zprávy pole *return_fpath*. Atribut *unix_hack* způsobí, že před každý výskyt slova `From` na začátku řádku bude vložen znak >.

```
# A sample /usr/lib/smail/transports file

# local - deliver mail to local users
local:  driver=appendfile,# append message to a file
         return_path,      # include a Return-Path: field
         from,             # supply a From_ envelope line
         unix_from_hack,   # insert > before From in body
         local;            # use local forms for delivery

         file=/var/spool/mail/${lc:user}, # location of mailbox fi-
les
         group=mail,# group to own file for System V
         mode=0660, # group mail can access
         suffix="\n",      # append an extra newline

# pipe - deliver mail to shell commands
pipe:   driver=pipe,       # pipe message to another program
         return_path,      # include a Return-Path: field
         from,             # supply a From_ envelope line
         unix_from_hack,   # insert > before From in body
         local;            # use local forms for delivery

         cmd="/bin/sh -c $user", # send address to the Bourne Shell
         parent_env,       # environment info from parent addr
         pipe_as_user,     # use user-id associated with address
         ignore_status,    # ignore a non-zero exit status
         ignore_write_errors,     # ignore write errors, i.e., broken
pipe
         umask=0022,       # umask for child process
         -log_output,      # do not log stdout/stderr
```

```
# file - deliver mail to files
file:   driver=appendfile,
        return_path,      # include a Return-Path: field
        from,             # supply a From_ envelope line
        unix_from_hack,   # insert > before From in body
        local;            # use local forms for delivery

        file=$user,       # file is taken from address
        append_as_user,   # use user-id associated with address
        expand_user,      # expand ~ and $ within address
        suffix="\n",      # append an extra newline
        mode=0600,        # set permissions to 600

# uux - deliver to the rmail program on a remote UUCP site
uux:    driver=pipe,
        uucp,             # use UUCP-style addressing forms
        from,             # supply a From_ envelope line
        max_addrs=5,      # at most 5 addresses per invocation
        max_chars=200;    # at most 200 chars of addresses

        cmd="/usr/bin/uux - -r -a$sender -g$grade $host!rmail
$(($user)$)",
        pipe_as_sender,   # have uucp logs contain caller
        log_output,       # save error output for bounce messages
#       defer_child_errors,     # retry if uux returns an error

# demand - deliver to a remote rmail program, polling immediately
demand: driver=pipe,
        uucp,             # use UUCP-style addressing forms
        from,             # supply a From_ envelope line
        max_addrs=5,      # at most 5 addresses per invocation
        max_chars=200;    # at most 200 chars of addresses

        cmd="/usr/bin/uux - -a$sender -g$grade $host!rmail
$(($user)$)",
        pipe_as_sender,   # have uucp logs contain caller
        log_output,       # save error output for bounce messages
#       defer_child_errors,     # retry if uux returns an error
```

```
# hbsmtp - half-baked BSMTP. The output files must
#        be processed regularly and sent out via UUCP.
hbsmtp: driver=appendfile,
         inet,              # use RFC 822-addressing
         hbsmtp,            # batched SMTP w/o HELO and QUIT
         -max_addrs, -max_chars;  # no limit on number of addresses

         file="/var/spool/smail/hbsmtp/$host",
         user=root,         # file is owned by root
         mode=0600,         # only read-/writable by root.

# smtp - deliver using SMTP over TCP/IP
smtp:    driver=tcpsmtp,
         inet,
         -max_addrs, -max_chars; # no limit on number of addresses

         short_timeout=5m, # timeout for short operations
         long_timeout=2h,  # timeout for longer SMTP operations
         service=smtp,     # connect to this service port
# For internet use: uncomment the below 4 lines
#        use_bind,         # resolve MX and multiple A records
#        defnames,         # use standard domain searching
#        defer_no_connect, # try again if the nameserver is down
#        -local_mx_okay,   # fail an MX to the local host
```

C

Slovníček

Moto: Mohli bychom použít více záznamů a méně třpytu.
Nebojte se přispět svými návrhy.

Zapamatovat si, co znamenají všechny zkratky a termíny v oblasti sítí, je pro jednoho člověka nesmírně složité. Zde uvádíme ty z nich, které se v knize často objevovaly, a připojujeme k nim krátký popis.

ACU Automatic Call Unit. Modem.[1]

ARP Address Resolution Protocol. Protokol, který slouží k mapování IP-adresy na ethernetovou adresu.

ARPA Advanced Resarch Project Agency, později DARPA. Zakladatel Internetu.

ARPANET Praotec dnešního Internetu. Experimentální síť vytvořená v americké společnosti Defense Advanced Research Project Agency (DARPA).

Assigned Numbers (Přidělená čísla) Pravidelně zveřejňovaná část dokumentu *RFC*, která obsahuje seznam veřejně přidělených čísel používaných pro různé účely v sítích TCP/IP. Obsahuje například seznam všech čísel portů známých služeb jako je *rlogin*, *telnet* atd. Nejaktuálnější vydání tohoto dokumentu má název RFC 1340.

---

1 Eventuelně teenager s telefonem.

bang path V sítích UUCP se jedná o zvláštní notaci cesty z jednoho systému UUCP na jiný. Název vznikl podle vykřičníků (‘bangs‘), které slouží k oddělení názvů hostitelů. Například cesta **foo!bar!ernie!bert** určuje cestu k hostiteli **bert**, při níž se půjde (v tomto pořadí) přes hostitele **foo**, **bar** a **ernie**.

BBS Bulletin Board System. Vytáčený systém pro předávání zpráv elektronickou poštou.

BGP Border Gateway Protocol. Protokol pro výměnu směrovacích informací mezi autonomními systémy.

BIND Berkeley Internet Name Domain server. Implementace serveru DNS.

BNU Basic Networking Utilities. V současné době nejběžnější varianta protokolu UUCP. Také se jí někdy říká HoneyDanBer UUCP. Tento název je odvozen ze jmen autorů: P. Honeyman, D. A. Novitz a B. E. Redman.

broadcast network (vysílací síť) Síť, která umožňuje jedné stanici současnou adresaci datagramu pro všechny ostatní stanice v síti.

BSD Berkeley Software Distribution. Druh Unixu.

canonical hostname (kanonický název hostitele) Primární název hostitele v DNS. Je to jediný název hostitele, se kterým je sdružen A-záznam a který je vrácen při provádění zpětného vyhledávání.

CCITT Comiteé Consultatif International de Télegraphique et Téléphonique. Mezinárodní organizace telefonních služeb.

CSLIP Compressed Serial Line IP. Protokol pro výměnu IP-paketů po sériové lince, přičemž dochází ke kompresi hlaviček většiny datagramů TCP/IP.

DNS Domain name system. Distribuovaná databáze, která se používá v Internetu pro mapování názvů hostitelů na IP-adresy.

EGP External Gateway Protocol. Protokol pro výměnu směrovacích informací mezi autonomními systémy.

Ethernet V hovorové mluvě se používá pro označení druhu síťového vybavení. Z technického hlediska je Ethernet jedním ze sady standardů IEEE. Ethernet jako hardware používá ke spojení několika hostitelů jediný

kabel, často koaxiál, který umožňuje přenosové rychlosti až 10 Mbp. Protokol Ethernet definuje způsob, jakým mohou hostitelé komunikovat prostřednictvím tohoto kabelu.[2]

FQDN Fully Qualified Domain Name. Název hostitele s připojeným názvem domény, který je platným indexem v databázi názvů domén.

FTP File Transfer Protocol. Protokol, na kterém je založena a podle něj i pojmenována jedna z nejznámějších služeb pro přenos souborů.

FYI „For Your Information“ (Pro vaši informaci). Sada dokumentů obsahující neformální informace týkající se Internetu.

GMU Groucho March University. Fiktivní univerzita používaná v knize jako příklad.

GNU GNU není Unix – tato rekurzivní zkratka je názvem projektu nadace Free Software Foundation, jehož cílem je poskytovat kompletní sadu unixových nástrojů, které je možno volně používat a kopírovat. Na veškerý software GNU se vztahuje speciální poznámka o autorských právech, které se také říká GNU General Public License (GPL), neboli Copyleft. Tuto licenci najdete v dodatku D.

HoneyDanBer Název varianty protokolu UUCP. Viz též BNU.

host (hostitel) Obecně uzel sítě. Zařízení, které je schopno přijímat a vysílat síťové zprávy. Zpravidla se jedná o počítač, ale mohou jím být také X-terminály nebo chytré tiskárny.

ICMP Internet Control Message Protocol. Síťový protokol, který používá protokol IP, když vrací hostiteli-odesilateli chybové informace atd.

IEEE Institute of Electrical and Electronics Engineers. Další organizace zabývající se tvorbou standardů. Z pohledu unixového uživatele jsou jejím největším úspěchem pravděpodobně standardy POSIX, které definují vlastnosti unixových systémů od rozhraní a sémantiky systémových volání až po administrativní nástroje.

Kromě toho vyvinula firma IEEE specifikace pro sítě Ethernet, Token Ring a Token Bus. Dílem IEEE je také hojně používaný standard pro binární reprezentaci reálných čísel.

[2] Jen tak mimochodem, ethernetový protokol, který běžně využívá protokol TCP/IP, neodpovídá přesně standardu IEEE 802.3. Ethenetové rámce obsahují typové pole, zatímco rámce IEEE 802.3 obsahují délkové pole.

IETF Internet Engineering Task Force.

internet Počítačová síť tvořená několika menšími samostatnými sítěmi.

Internet Konkrétní celosvětový internet.

IP Internet Protocol. Síťový protokol.

ISO International Standards Organization. Společnost zabývající se prosazováním standardů.

ISDN Integrated Services Digital Network. Nová telekomunikační technologie, která používá místo analogového systému digitální.

LAN Local Area Network. Malá počítačová síť.

MX Mail Exchanger. Typ zdrojového záznamu DNS používaného k označení hostitele jako poštovní brány pro nějakou doménu.

network, packet-switched
(přenos paketů) Paleta sítí, která umožňuje okamžitý přenos dat po malých paketech, které jsou jednotlivě přeneseny na místo určení. Sítě typu packet-switched spoléhají na trvalé nebo skoro trvalé spojení.

network, store-and-forward
(ulož a pošli) Jedná se o protiklad sítí typu packet-switched. Tyto sítě přenášejí data po celých souborech a nepoužívají trvalá spojení. Místo toho se hostitelé navzájem spojují pouze v určitých intervalech a přenášejí všechna data najednou. Tento způsob vyžaduje dočasné uložení dat, než je spojení navázáno.

NFS Network File System. Standardní síťový protokol a softwarový balík zajišťující transparentní přístup k datům na vzdálených discích.

NIS Network Information System. Aplikace založená na protokolu RPC, která umožňuje sdílet konfigurační soubory (například soubor s hesly) mezi několika hostiteli. Viz též heslo YP.

NNTP Network News Transfer Protocol. Protokol používaný pro přenos news po síťových TCP-spojeních.

oktet V Internetu se jedná o technický termín, který označuje posloupnost osmi bitů. Používá se spíše než *bajt*, protože na některých počítačích v Internetu má bajt jiný počet bitů než osm.

OSI Open Systems Interconnection. Standard ISO pro síťový software.

path (cesta) Tento termín je v sítích UUCP často používán jako synonymum pro směr (*route*). Viz též *bang path*.

PLIP Parallel Line IP. Protokol pro výměnu IP-paketů po paralelní lince, například tiskový port.

port, TCP nebo UDP Porty jsou pro protokoly TCP a UDP abstrakcí koncového bodu služby. Dříve než může proces zprostředkovat přístup nebo přistupovat k nějakému síťovému zařízení, musí požádat o port. Společně s IP-adresou hostitele slouží porty k jedinečné identifikaci dvou účastníků TCP-spojení.

portmapper (mapovač portů) Jedná se o prostředníka mezi čísly programů, která používá RPC kvůli identifikaci jednotlivých RPC-serverů a čísel portů protokolů TCP a UDP, na kterých tyto služby odposlouchávají.

PPP Protokol point-to-point. Protokol PPP je flexibilní a rychlý podkladový protokol sloužící k posílání síťových protokolů typu IP nebo IPX po spojení typu point-to-point. Kromě využití v sériových (modemových) linkách lze tento protokol použít také jako podkladový protokol pro ISDN.

RARP Reverse Address Resolution Protocol. Povoluje hostiteli nalézt svou IP-adresu při zavádění systému.

resolver Tato knihovna je zodpovědná za mapování názvů hostitelů na IP-adresy a vice versa.

resource record (zdrojový záznam) Základní jednotka informace v databázi DNS, jejíž označení se často zkracuje na RR. Každý záznam má přidělen určitý typ a třídu, například záznam mapující název hostitele na IP-adresu má typ A (jako adresa) a třídu IN (jako Internet Protocol).

reverse lookup (zpětné vyhledávání) Proces vyhledávání názvu hostitele podle jeho IP-adresy. V rámci DNS se tak děje vyhledáním IP-adresy hostitele v doméně **in-addr.arpa**.

RFC Request For Comments. Sada dokumentů popisujících internetové standardy.

RIP Routing Information Protocol. Tento směrovací protokol se používá k dynamickému přizpůsobování tras uvnitř (malé) sítě.

route (trasa, směr) Posloupnost hostitelů, kterými musí informace projít při cestě od původního hostitele k cílovému. Hledání vhodné trasy se také říká *směrování*.

routing daemon (směrovací démon) Ve větších sítích se změny v topologii obtížně zavádějí, a proto se k šíření směrovacích informací členským hostitelům sítě používají jiné prostředky. Těmto prostředkům říkáme dynamické směrování, kdy si směrovací informace mezi sebou vyměňují *směrovací démoni*, kteří běží na centrálních hostitelích v síti. Protokoly, které k tomu využívají, se nazývají *směrovací protokoly*.

RPC Remote Procedure Call. Protokol pro provádění procedur uvnitř procesu na vzdáleném hostiteli.

RR Zkratka *resource record*.

RS-232 Běžný standard sériových rozhraní.

RTS/CTS Hovorový název hardwarového „podání ruky“, který provádí dvě zařízení komunikující spolu přes rozhraní RS-232. Název je odvozen ze dvou okruhů: RTS („Ready To Send“) a CTS („Clear To Send“).

RTM Internet Worm (Internetový červ) Virus, který se díky několika bezpečnostním dírám v Unixu VMS a BSD 4.3 rozšířil po Internetu. Několik „chyb“ v programu způsobilo jeho nekontrolovatelné množení a efektivní kolaps velkých částí Internetu. RTM jsou iniciály autora (Roberta T. Morrise), které nechal v těle programu.

site (systém, místo) Seskupení hostitelů, které se zvenku chová téměř jako jediný síťový uzel. Například z hlediska Internetu bychom mohli považovat Groucho Marx University za jeden systém bez ohledu na komplexnost jeho vnitřní sítě.

SLIP Serial Line IP. Protokol pro výměnu IP-paketů po sériové lince, viz též CSLIP.

SMTP Simple Mail Transfer Protocol. Používá se k poštovnímu přenosu po TCP-linkách, ale také pro přenos poštovních dávek po UUCP-linkách (dávkované SMTP).

SOA Start of Authority. Typ zdrojového záznamu DNS.

System V Druh Unixu.

TCP Transmission Control Protocol. Síťový protokol.

TCP/IP Nedbalý popis sady internetového protokolu jako celku.

UDP User Datagram Protocol. Síťový protokol.

UUCP Unix to Unix Copy. Sada síťových přenosových příkazů pro vytáčené sítě.

Version 2 UUCP Zastarávající varianta protokolu UUCP.

virtual beer (virtuální pivo) Oblíbený nápoj každého uživatele Linuxu. První zmínka o virtuálním pivu, pokud si pamatuji, byla v poznámkách k jádru Linuxu verze 0.98.X, kde se Linus zmiňoval o „Oxford Beer Trolls“ kvůli zaslání nějakého virtuálního piva.

well-known services (dobře známé služby) Tento termín se často používá pro běžné síťové služby, jako je *telnet* a *rlogin*. Z technického hlediska popisuje všechny služby, kterým bylo v dokumentu „Assigned Numbers“ RFC přiřazeno oficiální číslo portu.

YP Yellow Pages (Žluté stránky). Starší název služby systému NIS, který se již nepoužívá, protože se jedná o obchodní značku společnosti British Telecom. Nicméně většina utilit NIS si i nadále ponechala názvy s předponou *yp*.

D

Seznam literatury

Knihy

Z knih v následujícím seznamu se dozvíte více informací o některých tématech probíraných v této knize. Seznam není ani úplný, ani systematický, knihy zde uvedené jsem shodou okolností četl a připadaly mi poměrně užitečné. Jakékoliv doplňky a vylepšení tohoto seznamu jsou vítány.

Obecné knihy o Internetu

[Kehoe92] Brendan P. Kehoe: *Zen and the Art of the Internet.*

„Zen“ byl jedním z prvních, ne-li vůbec prvním intenetovým průvodcem, který seznamoval začínajícího uživatele s různými trendy, službami a folklórem kolem Internetu. Jakožto stostránkový svazek pokrýval témata od přispívání do usenetových news až po Internetového červa. Kniha je k dispozici prostřednictvím anonymní služby FTP na mnoha FTP serverech a lze ji volně šířit a publikovat. Tištěná verze je k dostání také v nakladatelství Prentice-Hall.

Administrativní problémy

[Hunt92] Craig Hunt: *TCP/IP Network Administration.* O'Reilly and Associates, 1992. ISBN 0-937175-82-X.

Není-li pro vás dost dobrý Network Administration Guide, sežeňte si tuto knihu. Najdete v ní vše od získání IP-adresy až po řešení síťových problémů a bezpečnostní aspekty.

Zaměřuje se na nastavení protokolu TCP/IP, tj. konfiguraci rozhraní, nastavení směrování a rozpoznávání názvů. Obsahuje podrobný popis vlastností, které nabízí směrovací démoni `routed` a `gated`, včetně dynamického směrování.

Najdete zde také popis konfigurace aplikačních programů a síťových démonů, jako je `inetd`, tzv. příkazy `r`, NIS a NFS.

Dodatek obsahuje podrobný popis démonů `gated` a `named` a popis konfigurace programu sendmail.

[Stern92] Hal Stern: *Managing NIS a NFS*. O'Reilly and Associates, 1992. ISBN 0-937175-75-7.

Jedná se o doprovodnou knihu ke knize Craiga Hunta „TCP/IP Network Administration". V plném rozsahu se zabývá použitím systémů NIS (Network Information System) a NFS (Network File System), včetně konfigurace automounteru a systému PC/NFS.

[OReilly89] Tim O'Reilly and Grace Todino: *Managing UUCP and Usenet, 10th ed.* O'Reilly and Associates, 1992. ISBN 0-93717593-5.

Toto je standardní kniha o sítích UUCP. Popisuje UUCP Version 2, stejně jako verzi BNU. Pomůže vám rozchodit UUCP-uzel od úplného začátku a najdete zde také praktické rady a řešení mnoha problémů, jako je testování spojení nebo psaní skriptů pro rozhovory. Kniha se zabývá také exotičtějšími tématy, například jak zřídit cestovní UUCP-uzel nebo jemnostmi různých odrůd protokolu UUCP.

Druhá část knihy se zabývá Usenetem a softwarem pro síťové news. Vysvětluje konfiguraci jak Bnews (verze 2.11), tak i C News a seznámí vás se základy údržby síťových news.

[Spaf93] Gene Spafford and Simson Garfinkel: *Practical UNIX Security*. O'Reilly and Associates, 1992. ISBN 0-937175-72-2.

Tato kniha je nezbytná pro každého správce systému s přístupem k síti a nejenom pro něj. Probírá všechny sporné body týkající se počítačové bezpečnosti, které jsou seřazeny od základních bezpečnostních funkcí, jež Unix nabízí. I když byste se měli snažit zabezpečit všechny části systému, je část věnovaná sítím a bezpečnosti z hlediska našeho kontextu nejzajímavější. Vedle základních bezpečnostních strategií, které se týkají služeb BSD (*telnet*, *rlogin* atd), systémů NFS a NIS, se kniha zabývá také pokročilými bezpečnostními rysy typu MIT Kerberos, Secure RPC od firmy Sun a použití firewallů k odstínění vaší sítě před útoky z Internetu.

[AlbitzLiu92] Paul Albitz and Cricket Liu: *DNS and BIND*. O'Reilly and Associates, 1992. ISBN 1-56592-010-4.

Tato kniha je užitečná pro ty, kteří se zabývají správou systému DNS. Podrobně vysvětluje všechny rysy služby DNS a poskytuje příklady, které vám zpříjemní dokonce i volby BIND, jež vypadají na první pohled zcela nepřirozeně. Skutečně se dobře čte a hodně jsem se toho z ní dozvěděl.

[NISPlus] Rick Ramsey: *All about Administering NIS+*. Prentice-Hall, 1993. ISBN 0-13-068800-2.

Pozadí

Následující seznam knih možná bude zajímat ty, kdo se chtějí dozvědět více o tom, jak funguje protokol TCP/IP a jeho implementace, ale nechtějí číst dokumenty RFC.

[Stevens90] Richard W. Stevens: *UNIX Network Programming*. Prentice-Hall International, 1990. ISBN 0-13-949876-X.

Toto je zřejmě nejpoužívanější kniha o programování v sítích TCP/IP, z níž se současně dozvíte vše možné o internetových protokolech.[1]

[Tanen89] Andrew S. Tanenbaum: *Computer Networks*. Prentice-Hall International, 1989. ISBN 0-13-166836-6.[2]

Tato kniha vám dá nahlédnout do obecných síťových problémů. Za pomoci referenčního modelu OSI vysvětluje problémy spojené s návrhem každé vrstvy a algoritmy, které k tomu slouží. Na úrovni každé vrstvy jsou navzájem srovnávány implementace několika sítí, mezi nimiž nechybí ani ARPANET. Jedinou nevýhodou této knihy je příliš velké množství zkratek, takže je někdy obtížné sledovat, co autor říká. Tento problém je ale síťovému prostředí vrozený.

[Comer88] Douglas R.Comer: *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. Prentice-Hall International, 1988.

Dokumenty RFC

Na následující seznam dokumentů RFC se odkazujeme v průběhu celé knihy. Všechny dokumenty RFC jsou k dispozici prostřednictvím anonymní služby FTP na adrese **nic.ddn.mil, ftp.uu.net**. Chcete-li dostat RFC elektronickou poštou, pošlete zprávu na adresu **service@nic.ddn.mil** a jako předmět této zprávy uveďte `RFC-number.TXT`.

1340 Assigned Numbers, *Postel, J.,* and *Reynolds, J.* Tento dokument RFC definuje významná čísla používaná v různých protokolech, například standardní čísla portů, které odposlouchávají servery TCP a UDP, a čísla protokolů používaná v hlavičce IP-datagramu.

1 Nezapomeňte, že Stevens napsal novou knihu o protokolu TCP/IP nazvanou *TCP/Illustrated, Volume 1, The Protocols*, která vyšla v nakladatelství Addison Wesley. Zatím jsem neměl čas se na ni podívat.

2 ISBN, pod kterým je k dostání v Severní Americe, se může lišit.

1144 Compressing TCP/IP headers for low-speed seriál links, *Jacobson, V.* Tento dokument popisuje algoritmus používaný při kompresi hlaviček TCP/IP v protokolech CSLIP a PPP. Velice hodnotné čtení.

1033 Domain Administrators Operations Guide, *Lottor, M*. Společně s doprovodnými dokumenty, RFC 1034 a RFC 1035, tvoří ucelený zdroj informací o systému DNS, Domain Name System.

1034 Domain Names – Concepts and Facilites, *Mockapetris, P.V*. Doprovodný dokument k dokumentu RFC 1033.

1035 Domain Names – Implementation and Specification, *Mockapetris, P.V*. Doprovodný dokument k dokumentu RFC 1033.

974 Mail Routing and the Domain System, *Partridge, C*. Tento dokument popisuje směrování pošty na Internetu. Zájemci zde najdou úplný popis MX-záznamů.

977 Network News Transfer Protocol, *Kantor, B*., and *Lapsley, P*. Definice NNTP, všeobecně používaného protokolu pro přenos news v Internetu.

1094 NFS: Network File System Protocol Specification, *Nowicki, B*. Formální specifikace protokolu NFS a připojovacích protokolů (verze 2).

1055 Nonstandard for Transmission of IP Datagrams Over Seriál Lines: SLIP, *Romkey, J.L.* Popisuje protokol SLIP, Seriál Line Internet Protocol.

1057 RPC: Remote Procedure Call Protocol Specification: Version 2, *Sun Microsystems, Inc*.

1058 Routing Information Protocol, *Hedrick, C.L.* Popisuje protokol RIP, který se používá při dynamické výměně směrovacích informací mezi sítěmi LAN a MAN.

821 Simple Mail Transfer Protocol, *Postel, J.B.* Definuje SMTP, poštovní přenosový protokol používaný v sítích TCP/IP.

1036 Standard for the Interchange of USENET messages, *Adams, R.,* and *Horton, M.R.*. Tento dokument RFC popisuje formát zpráv usenetových news a způsob jejich výměny v Internetu a v sítích UUCP. V brzké době bychom se měli dočkat revize tohoto dokumentu.

822 Standard for the Format of ARPA Internet text messages, *Crocker, D*. Úplný zdroj vědomostí co se týče pošty přizpůsobené standardu RFC. Každý ho zná, ale jen málo lidí ho skutečně četlo.

968 Twas the Night Before Start-up, *Cerf, V.* Kdo říká, že hrdinové sítí nejsou opěvováni?

LINUX

Průvodce jádrem operačního systému Linux

Linux je fenoménem Internetu. Vznikl jako zájmový studentský projekt a postupem času se stal nejoblíbenějším zdarma dostupným operačním systémem. Pro mnoho lidí to je záhada. Jak může zdarma dostupný produkt za něco stát? Jak může ve světě, v němž dominuje několik silných softwarových společností, obstát něco, co vytvořila parta „hackerů“? Jak může program, na jehož vývoji se podílí spousta lidí v řadě zemí, být stabilní a efektivní? Faktem ale je, že Linux obstál a opravdu je stabilní a efektivní. Ke své každodenní činnosti jej používá řada univerzit a výzkumných pracovišť. Řada lidí jej používá na svých domácích počítačích a vsadil bych se, že jej nějakým způsobem používá většina společností, i když si to ne vždy uvědomují. Linux se používá při procházení webem, jako hostitelský systém webovských sítí, k psaní článků a jako u všech počítačů a systémů také ke hraní her. Je třeba zdůraznit, že Linux není v žádném případě nějaká hračka - je to kvalitně navržený a profesionálně napsaný operační systém, používaný nadšenci na celém světě.

Kořeny Linuxu je možno vystopovat už od doby vzniku Unixu. V roce 1969 začal Ken Thompson, výzkumný pracovník Bell Laboratories, experimentovat s víceuživatelským a víceúlohovým operačním systémem na jinak nevyužitém počítači PDP-7. Záhy se k němu připojil Dennis Richie a spolu s dalšími pracovníky brzy vytvořili první verzi Unixu. Richie byl značně ovlivněn dřívějším projektem, systémem MULTICS, a samo jméno Unix je vlastně narážka na tento projekt. První verze byly napsány v assembleru, třetí verze systému však už byla přepsána do nového programovacího jazyka - jazyka C. Jazyk C Richie navrhl a napsal speciálně jako jazyk určený pro tvorbu operačních systémů. Tento přechod umožnil přenesení Unixu na podstatně výkonnější počítač PDP-11/45 a 11/70 společnosti Digital. Dál už to šlo rychle. Unix opustil prostředí laboratoří a výzkumných pracovišť a zanedlouho dodávala vlastní verze Unixu většina významných počítačových společností.

Linux vznikl jako z nouze ctnost. Linus Torvalds, autor Linuxu a tvůrce jeho návrhu, si mohl dovolit koupit pouze jediný program - Minix. Minix je velmi jednoduchý operační systém unixovského typu, často používaný jako výukový nástroj. Jeho schopnosti ovšem Linuse nijak nenadchly, takže se rozhodl pro vytvoření vlastního systému. Jako základní model si vybral Unix, s nímž byl jako student velmi dobře seznámen. Začal na počítači PC Intel 386 a pustil se do psaní. Šlo to velmi rychle a nadšený Linus nabídl výsledky své práce ostatním studentům pomocí celosvětové počítačové sítě, používané hlavně v akademickém prostředí. S programem se seznámili další lidé a začali k němu přispívat. Většina nově vzniklých kódů byla vlastně řešením nějakého problému, který měl jeho autor. Zanedlouho se Linux stal operačním systémem. Je třeba podotknout, že Linux neobsahuje žádný původní kód Unixu, představuje vlastní implementaci normy POSIX a používá velkou část kódu GNU (GNU není Unix) nadace Free Software Foundation, Cambridge, Massachusetts.

Řada lidí používá Linux jako jednoduchý nástroj; často jej pouze nainstalují z některého dobrého distribučního balíku CD-ROM. Většina uživatelů Linuxu jej používá k vytváření dalších aplikací nebo ke spouštění aplikací vytvořených někým jiným. Řada uživatelů Linuxu hltavě

čte dokumenty HOWTO,[1] pociťují jak nadšení, když se jim podaří nějakou část systému správně nakonfigurovat, tak i zklamání, když se jim to nepodaří. Pouze malá skupina uživatelů je schopna aktivně vytvářet ovladače zařízení a nabízet Linusi Torvaldsovi, který vytvořil a udržuje jádro Linuxu, návrhy na opravy jádra. Linus přijímá doplňky a úpravy jádra od kohokoliv. Může to znít jako spolehlivá cesta k anarchii, Linus však provádí velmi pečlivé kontroly a veškerý nový kód do jádra vkládá sám. Vždy existovala jen malá skupina lidí, kteří přispívali k jádru Linuxu.

Většina uživatelů Linuxu se nestará jak jejich systém funguje, ani jak drží pohromadě. Je to trochu škoda, protože studium Linuxu představuje velmi dobrý způsob, jak se naučit více o fungování operačních systémů. Nejen, že je Linux velmi dobře napsán, všechny jeho zdrojové kódy jsou navíc zdarma k dispozici. Je to dáno tím, že i když si autoři ponechávají autorská práva na své programy, svolili k jejich volné distribuci podle veřejné licence GNU nadace Free Software Foundation. Na první pohled mohou být zdrojové kódy matoucí - uvidíte adresáře jako `kernel`, `mm` a `net` - co ale obsahují a jak kód funguje? Je nutné hlubší pochopení celkové struktury a návrhu Linuxu. A to by mělo být cílem této knihy: jasně vysvětlit, jak operační systém Linux pracuje. Měli byste získat představu o tom, co se v systému děje, když například kopírujete soubor z jednoho místa na druhé, nebo když čtete elektronickou poštu. Dobře si vzpomínám na vzrušení, které jsem zažil, když jsem poprvé zjistil, jak operační systém skutečně pracuje. O toto vzrušení bych se chtěl podělit se čtenáři této knihy.

Můj zájem o Linux se datuje od roku 1994, kdy jsem navštívil Jima Paradise, který pracoval na přenosu Linuxu na platformu Alpha AXP. V té době jsem pracoval ve společnosti Digital Equipment Corporation. Od roku 1984 jsem působil převážně v oblasti sítí a komunikací, od roku 1992 jsem začal pracovat v nové divizi Digital Semiconductor. Cílem divize bylo uplatnit se na trhu procesorů a rozšířit procesory Alpha AXP a desky pro tyto procesory i mimo Digital. Když jsem o Linuxu poprvé slyšel, ihned jsem pochopil možnost, která se pro systémy Alpha otevírá. Jimovo nadšení bylo nakažlivé a začal jsem s ním na přenosu spolupracovat. Při této práci jsem musel stále více obdivovat nejen samotný operační systém, ale i jeho autory. Je to bezesporu pozoruhodná skupina lidí a to, že jsem se stal jejím členem, pro mne představovalo největší uspokojení za celou dobu, co pracuji na vývoji programů. Lidé se mne na Linux velmi často ptají jak v práci, tak doma a já velmi rád odpovídám. Čím více Linux používám jak v profesionálním, tak v osobním životě, tím více se stávám linuxovým nadšencem. Všimněte si, že jsem použil raději termín „nadšenec“ a ne „fanatik“; linuxového nadšence definuji jako člověka, který si je vědom skutečnosti, že existují i jiné operační systémy, raději je ale nepoužívá. Má manželka Gill, která používá Windows 95, svého času poznamenala: „Nikdy jsem si nemyslela, že budeme mít každý svůj operační systém.“ Mým potřebám inženýra Linux perfektně vyhovuje. Jedná se o vynikající, pružný a adaptabilní nástroj.

---

[1] Dokumenty HOWTO představují přesně to, co naznačuje jejich název - návod, jak něco udělat. Na téma Linuxu jich bylo napsáno velmi mnoho a všechny jsou užitečné.

Většina zdarma distribuovaných programů je k dispozici ve verzi pro Linux a často mi stačí pouze nahrát si předpřeložené spustitelné soubory nebo nainstalovat program z CD. Co jiného bych měl používat, abych se zadarmo naučil programovat v jazycích C++, Perl, nebo abych se dozvěděl něco o Javě?

Alpha AXP je pouze jednou z mnoha hardwarových platforem, na nichž Linux pracuje. Většina různých Linuxů pracuje na procesorech Intel, objevuje se však stále širší nabídka verzí pro jiné procesory. Sem patří Intel, MIPS, Sparc a PowerPC. Tuto knihu jsem mohl orientovat na kterýkoliv ze zmíněných procesorů, mé zkušenosti s Linuxem však vycházejí z verze pro procesor Alpha AXP, takže při ilustraci některých klíčových bodů se orientuji právě na tento procesor. Je však třeba říct, že 95 % zdrojového kódu jádra Linuxu je nezávislých na hardwarové platformě. Proto je i 95 % této knihy věnováno strojově nezávislým částem jádra Linuxu.

Komu je kniha určena

V této knize nedělám žádné předpoklady o znalostech a zkušenostech čtenářů. Věřím tomu, že zájem o problematiku je dostatečnou motivací k sebevzdělávání. Při čtení vám může pomoci jistá míra zkušenosti s počítači, nejlépe s počítači třídy PC, a také určitá znalost jazyka C.

Členění knihy

Tato kniha *není* psána jako dokumentace nitra Linuxu. Namísto toho se zabývá operačními systémy obecně a Linuxem zvláště. Jednotlivé kapitoly se řídí pravidlem „od obecného ke konkrétnímu“. Nejprve vždy obecně popisují činnost určitého subsystému jádra a poté se pouštějí do větších podrobností.

Záměrně jsem se rozhodl při popisu algoritmů jádra a metod, jak jádro řeší různé problémy, nepoužívat formulace jako `routine_X()` volá `routine_Y()`, která inkrementuje položku `foo` datové struktury `bar`. Takovéto věci zjistíte čtením kódu. Kdykoliv potřebuji vysvětlit část kódu nebo popsat něco podobného, začínám od čistého stolu popisem příslušných datových struktur. V knize je tedy poměrně podrobně popsána řada důležitých datových struktur jádra a jejich vzájemné vazby.

Jednotlivé kapitoly jsou poměrně nezávislé, stejně jako jednotlivé subsystémy jádra, které kniha popisuje. Někdy se však na sebe odkazují, protože například není možno popisovat procesy bez pochopení funkce virtuální paměti.

Kapitola *Základy hardwaru* představuje stručný popis moderního PC. Operační systém musí úzce spolupracovat s hardwarem, který představuje smysl jeho existence. Operační systém potřebuje některé služby, které mohou být zajištěny pouze hardwarově. Aby bylo možno operační systém Linux plně pochopit, je nutné základní porozumění hardwarové problematice.

Kapitola *Základy softwaru* představuje základní úvod do programování a zabývá se assemblerem a jazykem C. Popisuje nástroje používané při vytvoření operačního systému a obsahuje přehled funkcí a služeb operačního systému.

Kapitola *Správa paměti* popisuje způsob, jakým Linux obsluhuje fyzickou a virtuální paměť systému.

Kapitola *Procesy* popisuje co to proces je a jak jádro Linuxu vytváří, spravuje a ruší procesy v systému.

Při koordinaci své práce komunikují procesy navzájem mezi sebou a jádrem. Linux podporuje řadu mechanismů pro meziprocesovou komunikaci (*Inter-Process Communication Mechanism - IPC*). Dvěma z nich jsou signály a roury, Linux ale podporuje také mechanismy IPC Systemu V, pojmenované po verzi Unixu, v níž se poprvé objevily. Tyto mechanismy jsou popsány v kapitole *Meziprocesová komunikace*.

Standard PCI (*Peripheral Component Interconnect*) je dnes uznáván jako levný a výkonný standard pro datové sběrnice počítačů PC. Kapitola *PCI* popisuje jak jádro Linuxu inicializuje a používá PCI sběrnice a zařízení v systému.

Kapitola *Přerušení a jejich obsluha* se zabývá tím, jak jádro Linuxu obsluhuje přerušení. I když jádro obsahuje obecné mechanismy a rozhraní pro obsluhu přerušení, některé podrobnosti obsluhy závisejí na hardware a architektuře systému.

Jednou ze silných stránek Linuxu je podpora řady hardwarových zařízení, které jsou dnes k dispozici. Kapitola *Ovladače zařízení* popisuje, jak jádro Linuxu řídí fyzická zařízení připojená k systému.

Kapitola *Souborový systém* popisuje, jak jádro Linuxu spravuje soubory v souborových systémech, které podporuje. Popisuje virtuální souborový systém (VFS) a rovněž podporu reálných souborových systémů jádrem.

Sítě a Linux jsou dva pojmy s prakticky stejným významem. Linux je bez nadsázky produktem Internetu a WWW. Jeho autoři i uživatelé si pomocí webu vyměňují nápady a programy, a Linux sám bývá mnohde používán k zajištění síťových potřeb organizace. Kapitola *Sítě* popisuje, jak Linux podporuje síťové protokoly, označované společně termínem TCP/IP.

Kapitola *Mechanismy jádra* se zaměřuje na některé obecné úlohy a mechanismy, které musí jádro Linuxu zajišťovat, aby mohly jeho jednotlivé části účinně spolupracovat.

Kapitola *Moduly* popisuje, jak jádro dokáže dynamicky podle potřeby nahrávat různé funkce, například souborové systémy.

Kapitola *Zdrojový kód* vysvětluje, kde máte ve zdrojovém kódu Linuxu hledat konkrétní funkce.

Příloha *Datové struktury* obsahuje okomentovaný definiční kód datových struktur, které byly v předchozích kapitolách popsány.

V příloze *Procesor Alpha* jsou uvedeny hlavní charakteristiky architektury Alpha AXP.

Příloha *Užitečné adresy WWW a FTP* uvádí seznam adres, na nichž můžete prostřednictvím Internetu získat další informace o problematice Linuxu.

Konečně *Slovníček* představuje stručné vysvětlení hlavních pojmů, s nimiž se v této knize setkáte.

V textu knihy naleznete odkazy na hierarchii zdrojového kódu jádra Linuxu. Uvádím je pro případ, že byste se chtěli podívat přímo na zdrojový kód. Všechny odkazy jsou relativní vzhledem k cestě `/usr/src/linux`. Pokud vezmeme jako příklad soubor `foo/bar.c`, naleznete jej jako `/usr/src/linux/foo/bar.c`. Pokud používáte Linux (což byste měli), představuje studium zdrojového kódu cennou zkušenost a tuto knihu můžete použít jednak jako příručku pro pochopení kódu a jednak jako průvodce různými datovými strukturami.

Ochranné známky

Poděkování

Musím poděkovat řadě lidí, kteří si udělali čas a poslali mi připomínky k této knize. Snažím se všechny připomínky zahrnovat do nových verzí. Zvláštní poděkování si zaslouží John Rigby a Michael Bauer, kteří důkladně přečetli a opravili celou knihu. Nebylo to nic jednoduchého.

Odkazy na zdrojové texty jádra

1. Viz `foo` v `foo/bar.c`

Kapitola

1

Základy hardwaru

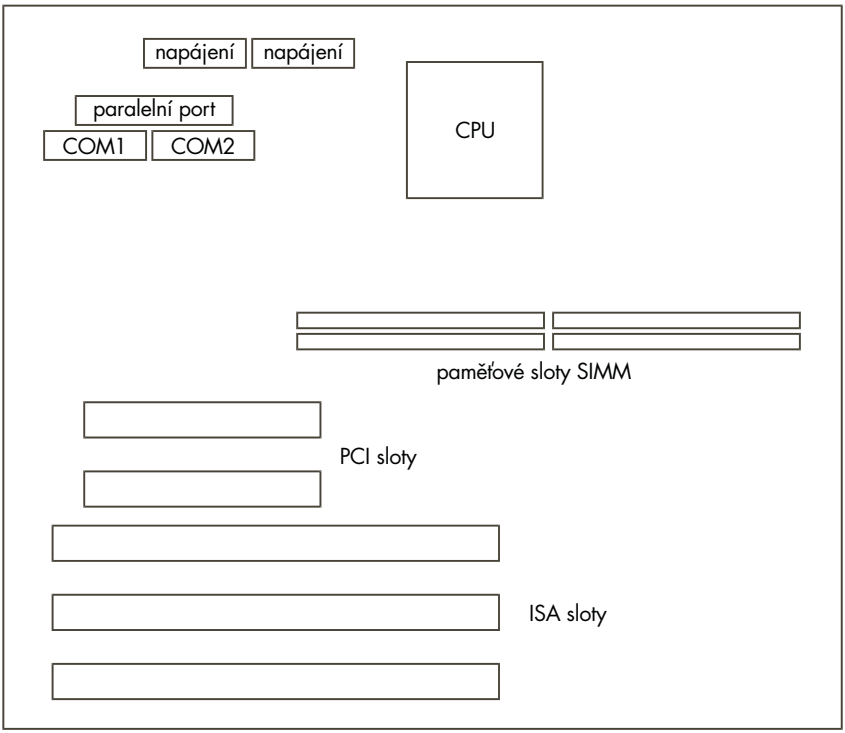
Operační systém musí úzce spolupracovat s hardwarem, který jej hostí. Operační systém potřebuje některé služby, které je možno zajistit pouze hardwarově. K úplnému pochopení činnosti operačního systému Linux je nutné porozumět základům hardwaru. Tato kapitola představuje stručný úvod do hardwaru moderního PC.

Revoluce začala v lednu roku 1975, kdy se na obálce časopisu „Popular Electronics“ objevil obrázek počítače Altair 8800.

Počítač Altair 8800, pojmenovaný podle jednoho z prvních dílů seriálu Star Trek, byl počítač, který si mohl zájemce o elektroniku postavit za pouhých 397 dolarů. S procesorem Intel 8080 a 256 bajty paměti bez obrazovky či klávesnice to bylo podle dnešních standardů úplné nic. Jeho autor, Ed Roberts, pojmenoval svůj vynález „personal computer“. V dnešní době se termínem PC označuje prakticky každý počítač, který můžete odnést v rukou. Podle této definice jsou počítači PC i některé velmi výkonné systémy Alpha AXP.

Různí nadšenci pochopili sílu Altairu a začali pro něj vytvářet programy a hardware. Pro tyto průkopníky představoval Altair svobodu - svobodu od dávkových úloh pro mainframové systémy, spravované elitou kněžstva. Na fenoménu počítače, který můžete mít doma na kuchyňském stole, vydělali závratné bohatství dva zběhlí vysokoškolští studenti. Objevila se řada různého hardwaru s různými rozdíly a softwaroví nadšenci radostně vytvářeli nové programy pro nové systémy. Paradoxně to byla společnost IBM, která ohlášením počítače IBM PC v roce 1981 (k zákazníkům se dostal v roce 1982) určila formu moderních osobních počítačů. S procesorem Intel 8088, 64 KB paměti (rozšiřitelnými na 256 KB), dvěma disketovými mechanikami a displejem Colour Graphics Adapter (CGA) s rozlišením 80 znaků na 25 řádků to podle dnešních měřítek nebylo nic moc, počítač se ale prodával velmi dobře. V roce 1983 následoval model IBM PC/XT, doplněný o neuvěřitelných 10 MB pevného disku. Krátce předtím vytvořily své klony architektury IBM PC i jiné společnosti, jako například Com-

paq, a architektura PC se tak stala de facto standardem. Tento standard umožnil, aby řada hardwarových společností začala soutěžit na stále se rozrůstajícím trhu, což vedlo ke snížení cen. Řada rysů architektury těchto prvních PC byla přenesena i do moderních PC. Například i ten nejvýkonnější systém s procesorem Intel Pentium Pro se spouští v adresačním režimu Intel 8086. Když Linus Torvalds začínal psát to, z čeho později vzniknul Linux, vybral si nejrozšířenější a cenově dostupnou platformu PC Intel 80386.



Obrázek 1.1
Základní deska typického PC

Když se na PC podíváme zvenčí, nejnápadnějšími komponentami je skříň počítače, klávesnice, myš a monitor. Na čelní stěně skříně jsou nějaká tlačítka, malý displej s nějakými čísly a disketová mechanika. Většina dnešních systémů má také mechaniku CD-ROM a pokud vám záleží na vašich datech, můžete mít také páskovou jednotku pro zálohování. Všechna tato zařízení se společně označují termínem *periferní zařízení*.

Přestože celkové řízení systému má na starosti procesor (CPU), není to jediné inteligentní zařízení. Jistou úroveň inteligence mají i řadiče jednotlivých periferií, například řadič IDE. Uvnitř PC (viz obrázek 1.1) najdeme základní desku, která obsahuje procesor či CPU, paměť a několik slotů pro připojení periferií ISA nebo PCI. Některé řadiče, například řadič IDE disku, mohou být vestavěny přímo na základní desku.

1.1 Procesor

Procesor, CPU nebo mikroprocesor je srdcem každého počítačového systému. Procesor načítá a vykonává instrukce uložené v paměti a provádí tak operace matematické, logické a řídí toky dat. V začátcích počítačové éry byly jednotlivé části procesoru samostatné (a fyzicky velké) moduly. V té době vznikl termín *Central Processing Unit* (CPU). Moderní mikroprocesor kombinuje všechny tyto moduly v jednom integrovaném obvodu, vytvořeném na malinkém kousku křemíku. V této knize budeme chápat termíny *CPU*, *procesor* a *mikroprocesor* jako rovnocenné.

Mikroprocesor pracuje s binárními daty, tedy daty, složenými z nul a jedniček.

Jedničky a nuly jsou reprezentovány zapnutím nebo vypnutím elektrického napětí. Stejně jako desítkové číslo 42 znamená vlastně „4 desítky a 2 jedničky“, je i binární číslo posloupnost binárních číslic, z nichž každá reprezentuje určitou mocninu dvou. 10 na první (10^1) je 10, 10 na druhou (10^2) je 10×10, 10^3 je 10×10×10 a tak dále. Binárně 0001 je desítkově 1, binárně 0010 je desítkově 2, binárně 0011 je 3, 0100 je 4 a tak dále. Takže desítkových 42 je binárně 101010 nebo také 2 + 8 + 32 nebo $2^1 + 2^3 + 2^5$. V počítačových programech se ale čísla neuvádějí ve dvojkové soustavě, namísto toho se obvykle používá soustava šestnáctková.

Při základu šestnáct reprezentuje každý řád jednu mocninu šestnácti. Desítkové číslice jsou pouze 0 až 9, číslice 10 až 15 se jedním znakem zapisují jako písmena A, B, C, D, E a F. Například šestnáctkově E je desítkově 14, šestnáctkově 2A je desítkově 42 (dvě šestnáctky plus deset). Podle konvencí programovacího jazyka C (kterou v knize používám) se šestnáctková čísla uvozují znaky „*0x*“; šestnáctkových 2A se tedy zapisuje jako *0x2A*.

Mikroprocesor může provádět matematické operace jako sčítání, násobení a dělení a logické operace typu „je X větší než Y?“.

Činnost procesoru je řízena vnějšími hodinami. Tyto hodiny, takzvané systémové hodiny, generují pravidelné hodinové impulsy a s každým impulsem provede procesor nějakou operaci. S každým tikem hodin může procesor například vykonat jednu instrukci. Rychlost procesoru se udává rychlostí „tikání“ systémových hodin. Procesor s rychlostí 100 MHz dostává každou sekundu 100,000,000 hodinových impulsů. Výkon procesoru se ale nedá pomocí hodinového kmitočtu posuzovat, protože různé procesory stihnou za jeden hodinový impuls různé množ-

ství práce. Obecně ale platí, že čím rychlejší hodiny, tím výkonnější procesor. Instrukce vykonávané procesorem jsou velmi jednoduché, například „přečti obsah paměti na adrese X do registru Y“. Registry jsou interní pamětí mikroprocesoru, slouží k ukládání dat a k manipulaci s nimi. Prováděná operace může způsobit, že procesor přeruší to, co momentálně dělá, a odskočí na úplně jinou instrukci někde jinde v paměti. Jednotlivé jednoduché instrukce dávají moderním procesorům prakticky neomezené možnosti, protože mohou zpracovat milióny nebo dokonce miliardy takovýchto instrukcí za sekundu.

Před vykonáním instrukce je nutné ji načíst z paměti. Sama instrukce se může odkazovat na data uvnitř paměti, která se pak podle potřeby načítají nebo ukládají.

Velikost, počet a typ registrů závisí výhradně na typu procesoru. Procesor Intel 80486 má jinou sadu registrů než procesor Alpha AXP, první rozdíl může být ten, že registry procesoru Intel jsou 32bitové, registry procesoru Alpha jsou 64bitové. Obecně ale platí, že procesory mají větší počet univerzálních registrů a menší počet speciálním registrů. Většina procesorů má následující speciální, vyhrazené registry:

Ukazatel instrukcí (*Program Counter, PC*)

Tento registr obsahuje adresu instrukce, která se bude provádět v dalším kroku. Obsah registru PC se automaticky inkrementuje po každém načtení instrukce.

Ukazatel zásobníku (*Stack Pointer, SP*)

Procesory mají přístup k většímu objemu externí paměti s náhodným přístupem (RAM), která slouží k dočasnému ukládání dat. Zásobník představuje jednoduchou metodu jak do externí paměti ukládat a načítat dočasné hodnoty. Procesory jsou obvykle vybaveny instrukcemi, které umožňují uložit jednu hodnotu na zásobník a přečíst ji později zpět. Zásobník pracuje na principu „poslední dovnitř, první ven“ (*Last In First Out, LIFO*). Jinak řečeno, pokud uložíte na zásobník dvě hodnoty, X a Y, a poté vyzvednete ze zásobníku jednu hodnotu, dostanete hodnotu Y.

U některých procesorů se zásobník rozrůstá nahoru směrem k vrcholu paměti, u jiných postupuje shora dolů ke dnu paměti. Některé procesory, například ARM, podporují oba typy zásobníků.

Stavový registr (*Processor Status, PS*)

Instrukce mohou dávat nějaký výsledek, například instrukce „je obsah registru X větší než obsah registru Y“ vrací hodnotu „ano“ nebo „ne“. Tyto a další informace o momentálním stavu procesoru jsou uloženy v registru PS. Například většina procesorů má minimálně dva pracovní režimy, režim jádra a uživatelský režim. Registr PS obsahuje i informaci o momentálním režimu činnosti.

1.2 Paměť

Všechny systémy mají nějakou hierarchii paměti, na různých úrovních této hierarchie se nacházejí paměti s různou rychlostí a velikostí. Nejrychlejší paměť se označuje jako *cache* (skladiště, zásobárna) a její funkce přesně odpovídá jejímu jménu – slouží k dočasnému uložení obsahu hlavní paměti. Tato paměť je sice velmi rychlá, ale také velmi drahá, takže většina procesorů obsahuje malou interní cache a na základní desce je pak větší externí cache. Některé procesory používají stejnou paměť cache pro uložení instrukcí i dat, jiné mají dvě paměti - jednu pro instrukce, druhou pro data. Procesory Alpha AXP obsahují dvě interní paměti cache, jednu pro data (D-Cache) a druhou pro instrukce (I-Cache). Externí cache (B-Cache) je pro obě společná. Kromě toho systém ještě obsahuje hlavní paměť, která je v porovnání s externí pamětí cache velmi pomalá. V porovnání s interní cache procesoru se hlavní paměť přímo plouží.

Obsah pamětí cache a hlavní paměti se musí shodovat, musí být *koherentní*. Jinak řečeno, pokud se nějaké slovo z hlavní paměti nachází v některé paměti cache, musí systém zajistit, aby obsah cache a hlavní paměti byl stejný. Zajištění koherence pamětí cache je částečně úkolem hardwaru, částečně úkolem operačního systému. To je ovšem společné i pro řadu jiných systémových úloh, kdy musí hardware a software při zajišťování nějaké činnosti úzce spolupracovat.

1.3 Sběrnice

Jednotlivé obvody na systémové desce jsou propojeny systémem, nazývaným sběrnice. Systémová sběrnice se dělí na tři logické celky, na adresovou sběrnici, datovou sběrnici a řídící sběrnici. Datová sběrnice udává místo v paměti (adresu) pro datový přenos. Datová sběrnice slouží k přenosu dat, je obousměrná a umožňuje načíst data do procesoru nebo je z procesoru zapsat. Řídící sběrnice obsahuje různé linky, které slouží k přenosu časovacích a řídících signálů pro celý systém. Existuje řada různých sběrnic, oblíbené sběrnice pro připojení periferií k systému jsou například sběrnice ISA a PCI.

1.4 Řadiče a periferie

Periferie jsou reálná zařízení, jako například grafické karty nebo disky, ovládané čipem řadiče, který může být na základní desce nebo na kartě, zapojené do základní desky. IDE disky jsou řízeny řadičem IDE, SCSI disky řídí řadič SCSI a podobně. Řadiče jsou připojeny k procesoru a k sobě navzájem různými sběrnicemi. Většina dnešních systémů používá k propojení hlavních komponent systému sběrnice ISA a PCI. Řadiče jsou rovněž procesory, podobně jako CPU, a je možno je chápat jako inteligentní pomocníky procesoru, který zajišťuje celkové řízení systému.

Každý řadič je jiný, obvykle však mají registry, jejichž prostřednictvím se ovládají. Program spuštěný na CPU musí být schopen do těchto registrů zapisovat a číst z nich. Jeden registr může například obsahovat příznak chyby. Další může být použit k různým řídicím účelům podle režimu řadiče. CPU může individuálně adresovat každý řadič v systému, což znamená, že musí existovat softwarový ovladač zařízení, který manipuluje s registry řadiče. Dobrým příkladem je sběrnice IDE, která umožňuje individuálně ovládat každý k ní připojený disk. Dalším příkladem může být sběrnice PCI, která umožňuje zvlášť ovládat každé k ní připojené zařízení (například grafickou kartu).

1.5 Adresový prostor

Systémová sběrnice spojuje CPU s hlavní pamětí, je to jiná sběrnice než ta, kterou jsou k CPU připojeny hardwarové periferie. Paměťový prostor, na němž jsou souhrnně připojeny periferie, se označuje jako V/V prostor. V/V prostor může být dále dělen, to nás ale v této chvíli nemusí zajímat. CPU může přistupovat jak k adresovému prostoru systémové paměti, tak k adresovému prostoru V/V zařízení, zatímco řadiče mohou k systémové paměti přistupovat pouze nepřímo, a to pouze s podporou procesoru. Z pohledu zařízení, řekněme ovladače disketových mechanik, je vidět pouze ta část adresového prostoru, v němž se zobrazují registry tohoto zařízení (ISA), není vidět systémová paměť. CPU má typicky odlišné instrukce pro přístup k paměti a k V/V prostoru. Může například existovat instrukce s významem „přečti bajt z V/V adresy *0x3f0* do registru X“. To je přesně způsob, jakým CPU řídí hardwarové periferie, čtením a zápisem registrů v V/V prostoru. Která část V/V prostoru je přidělena kterým periferiím (řadiči IDE, sériovým portům, řadiči disket a podobně) je dáno konvencí už od doby vzniku architektury PC. Adresa *0x3f0* z V/V prostoru je shodou okolností adresa řídícího registru jednoho ze sériových portů (COM1).

Jsou chvíle, kdy řadič potřebuje přečíst nebo zapsat větší objem dat přímo z nebo do systémové paměti. Příkladem může být zápis dat na pevný disk. V takovém případě se používá řadič přímého přístupu do paměti (Direct Memory Access, DMA), který umožňuje hardwarovým zařízením přímý přístup k systémové paměti, ovšem pouze pod přísným řízením a dohledem CPU.

1.6 Hodiny

Všechny operační systémy potřebují znát čas, takže moderní PC jsou vybaveny zvláštní periferií, zvanou hodiny reálného času (Real Time Clock, RTC). Tyto hodiny zajišťují dvě věci: informaci o přesném čase a generování pravidelných časových intervalů. RTC mají vlastní baterii, takže pracují, i když je počítač vypnut, proto tedy PC vždy „zná“ správné datum a čas. Intervalový časovač umožňuje operačnímu systému přesně plánovat základní úkoly.

Kapitola

2

Základy softwaru

Program je posloupnost počítačových instrukcí, které vykonávají nějakou úlohu. Program může být napsán v assembleru, počítačovém jazyce nízké úrovně, nebo v nějakém vyšším, na platformě nezávislém jazyce, jako je například C. Operační systém je speciální program, který uživatelům umožňuje spouštět aplikace jako jsou tabulkové procesory a textové editory. Tato kapitola představuje úvod do základních programovacích principů a vysvětluje úkol a funkci operačního systému.

2.1 Počítačové jazyky

2.1.1 Assemblery

Instrukce, které CPU načítá z paměti a provádí, nejsou pro člověka vůbec srozumitelné. Jedná se o strojový kód, který počítači přesně říká, co má dělat. Šestnáctkové číslo *0x89e5* je pro procesor Intel 80486 instrukce, která zkopíruje obsah registru ESP do registru EBP. Jedním z prvních programátorských nástrojů vyvinutých už pro první počítače byl assembler, program, který bere pro člověka srozumitelný zdrojový kód a přeloží jej do strojového kódu. Jazyk assembler provádí manipulace s registry a daty explicitně, je jiný pro každý mikroprocesor. Assembler pro procesory Intel *x*86 se výrazně liší od assembleru pro mikroprocesor Alpha AXP. Následující ukázka assembleru procesoru Alpha AXP ukazuje, co může takový program dělat:

```
    ldr r16, (r15)        ; řádek 1
    ldr r17, 4(r15)       ; řádek 2
    beq r16,r17,100       ; řádek 3
    str r17, (r15)        ; řádek 4
100:                      ; řádek 5
```

První příkaz (na řádku 1) naplní registr 16 hodnotou na adrese, uložené v registru 15. Druhý příkaz naplní registr 17 hodnotou z následující adresy. Třetí příkaz porovná hodnotu registrů 16 a 17 a pokud se shodují, skočí na návěští *100*. Pokud registry neobsahují stejnou hodnotu, pokračuje se řádkem 4, na němž se obsah registru 17 uloží do paměti. Pokud registry obsahují stejnou hodnotu, není nutné ji ukládat. Programy v assembleru se píší obtížně a zdlouhavě, navíc se v nich snadno dělají chyby. Pouze velmi malá část jádra Linuxu je napsána přímo v assembleru. Tyto části jsou závislé na konkrétním procesoru a důvodem použití assembleru je dosažení maximálního výkonu.

2.1.2 Programovací jazyk C a kompilátor

Psát v assembleru větší programy je obtížné a zdlouhavé. Často vznikají chyby a výsledné programy nejsou přenositelné, protože jsou vázány na určitou rodinu procesorů. Daleko lepší je použít strojově nezávislý jazyk, jako je například C. Jazyk C umožňuje popisovat program pomocí jeho logického algoritmu a pomocí dat, nad nimiž operuje. Speciální programy zvané kompilátory čtou program v jazyce C a překládají jej do assembleru, generují z něj konkrétní strojový kód. Dobrý kompilátor dokáže překládat tak, že výsledný kód je téměř stejně efektivní jako kdyby byl napsán zkušeným programátorem přímo v assembleru. Většina jádra Linuxu je napsána v jazyce C. Vezměme si následující příkaz jazyka C:

```
if (x != y)
        x = y ;
```

Tento příkaz provádí přesně stejnou operaci jako výše uvedený příklad v assembleru. Pokud se obsah proměnné `x` neshoduje s obsahem proměnné `y`, překopíruje se obsah proměnné `y` do `x`. Program v jazyce C je organizován do rutin, které provádějí jednotlivé funkce. Každá rutina může vracet jakoukoliv hodnotu nebo datový typ podporovaný jazykem C. Velké programy jako například jádro Linuxu se mohou skládat z mnoha samostatných C-modulů, každý z nich má své vlastní rutiny a datové struktury. Jednotlivé moduly zdrojového kódu plní příbuzné logické funkce, jako například obsluhu souborového systému.

Jazyk C podporuje mnoho typů proměnných; proměnná je místo v paměti, na něž se dá odkazovat symbolickým jménem. Ve výše uvedeném fragmentu programu označují `x` a `y` místa v paměti. Programátor se nestará o to, kam v paměti budou data uložena, to je záležitostí linkeru (viz dále). Některé proměnné obsahují různé typy dat, jako například celá nebo reálná čísla, jiné představují ukazatele.

Ukazatel je proměnná, která obsahuje adresu - místo v paměti, kde jsou uložena jiná data. Představme si proměnnou `x`. Ta může být v paměti uložena na adrese *0x80010000*. Můžete mít ukazatel, řekněme `px`, který bude ukazovat na `x`. Ukazatel `px` může „bydlet“ na adrese *0x80010030*. Hodnota ukazatele `px` bude *0x8001000*, tedy adresa proměnné `x`.

Jazyk C umožňuje shromažďovat do jedné datové struktury spolu související proměnné. Například takto:

```
struct {
         int i ;
         char b ;
} my_struct ;
```

Toto je datová struktura nazvaná `my_struct`, která obsahuje dva prvky, celé číslo (32 bitů) zvané `i` a znak (8 bitů) zvaný `b`.

2.1.3 Linker

Linker je program, který „slinkuje“ dohromady několik modulů a knihoven a výsledkem je jediný program. Moduly obsahují strojový kód vygenerovaný assemblerem nebo překladačem a najdeme v nich jednak spustitelný kód a data a dále informace, které říkají linkeru, jak jednotlivé moduly zkombinovat do výsledného programu. Jeden modul může například obsahovat všechny databázové funkce programu, jiný modul může obsahovat funkce pro obsluhu řádkových příkazů. Linker propojí odkazy mezi těmito moduly v místech, kde se rutina nebo data, na něž se jeden modul odkazuje, nacházejí v modulu jiném. Jádro Linuxu je jediný velký program, vzniklý slinkováním řady modulů.

2.2 Co je to operační systém

Bez příslušného softwaru je počítač jenom hromada elektroniky, která vydává teplo. Pokud je hardware srdcem počítače, je software jeho duší. Operační systém je soubor systémových programů, které uživateli umožňují spouštět aplikační programy. Operační systém abstrahuje od skutečného hardwaru systému a přináší uživatelům a aplikacím virtuální stroj. V zásadě platí, že teprve operační systém představuje charakteristiku počítače. Většina uživatelů PC používá jeden nebo více operačních systémů, z nichž každý může mít úplně jiný vzhled a chování. Linux je tvořen řadou samostatných dílů, které dohromady tvoří operační systém. Nutným dílem Linuxu je jeho jádro, ani to by však nebylo k ničemu bez knihoven nebo příkazových interpretů.

Abychom si lépe ukázali, co to operační systém vlastně je, zkusme si představit co se stane, když zadáte jednoduchý příkaz:

```
$ ls
Mail            c               images          perl
docs            tcl
$
```

Znak $ je prompt přihlašovacího příkazového interpretu (v tomto případě bash). Znamená to, že se čeká na vás, na uživatele, až zadáte nějaký příkaz. Když zadáváte příkaz ls, ovladač klávesnice rozpozná, že jsou zadávány nějaké znaky. Ovladač klávesnice je předá příkazovému interpretu, který příkaz zpracuje tak, že se pokouší najít spustitelný soubor stejného jména. Najde jej jako soubor /bin/ls. Zavolají se služby jádra, které přemístí kód obsažený v souboru do virtuální paměti a zahájí jeho provádění. Příkaz ls volá souborové služby jádra a zjišťuje, jaké soubory existují. Souborový systém může využít informací uložených ve vyrovnávací paměti souborového systému, nebo použije ovladač diskového zařízení a načte údaje z disku. Může dojít i k tomu, že síťový ovladač začne komunikovat se vzdáleným počítačem aby zjistil podrobnosti o vzdálených souborech, k nimž má váš systém přístup (souborové systémy je možné vzdáleně připojit pomocí služby Network File System, NFS). Ať už se informace zjistí jakýmkoliv způsobem, ls je vypíše a ovladač displeje je zobrazí na obrazovce.

Celé to vypadá dost komplikovaně, nicméně takovýto jednoduchý příklad nám ukazuje, že operační systém je ve skutečnosti řada spolupracujících funkcí, které dohromady dávají uživateli souvislý pohled na systém.

2.2.1 Správa paměti

Pokud bychom měli neomezené prostředky, řekněme paměť, řada funkcí, které operační systém provádí, by byla zbytečných. Jedním ze základních triků každého operačního systému je zajistit, aby se malý objem fyzické paměti choval jako paměť daleko větší. Této obrovské paměti říkáme virtuální paměť. Vtip je v tom, že programy, které v systému běží, se domnívají, že mají k dispozici celou tuto velikou paměť. Systém v době běhu rozděluje paměť na snadno manipulovatelné stránky a tyto stránky podle potřeby odkládá na disk. Programy si ničeho nevšimnou díky dalšímu triku, multiprocessingu.

2.2.2 Procesy

Proces můžeme chápat jako spuštěný program. Každý proces je samostatná entita, která vykonává nějaký program. Pokud se podíváte na procesy v Linuxu, uvidíte jich celou řadu. Například zadáním příkazu ps uvidíte následující seznam procesů:

```
$ ps
  PID TTY STAT   TIME COMMAND
  158 pRe 1     0:00 -bash
  174 pRe 1     0:00 sh /usr/X11R6/bin/startx
  175 pRe 1     0:00 xinit /usr/X11R6/lib/X11/xinit/xinitrc --
  178 pRe 1 N   0:00 bowman
  182 pRe 1 N   0:01 rxvt -geometry 120x35 -fg white -bg black
```

```
  184 pRe 1 <   0:00 xclock -bg grey -geometry -1500-1500 -padding 0
  185 pRe 1 <   0:00 xload -bg grey -geometry -0-0 -label xload
  187 pp6 1     9:26 /bin/bash
  202 pRe 1 N   0:00 rxvt -geometry 120x35 -fg white -bg black
  203 ppc 2     0:00 /bin/bash
 1796 pRe 1 N   0:00 rxvt -geometry 120x35 -fg white -bg black
 1797 v06 1     0:00 /bin/bash
 3056 pp6 3 <   0:02 emacs intro/introduction.tex
 3270 pp6 3     0:00 ps
$
```

Pokud by můj systém měl mnoho procesorů, mohl by každý proces (přinejmenším teoreticky) běžet na jednom procesoru. Bohužel ve skutečnosti mám pouze jediný procesor, takže operační systém se uchyluje k dalším trikům a nechává každý proces běžet pouze krátkou dobu. Tento čas se označuje jako časové kvantum. Celému triku se říká multiprocessing nebo také plánování a způsobuje, že každý proces si myslí, že on je jediný proces v systému. Procesy jsou jeden před druhým chráněny, takže když jeden proces zhavaruje nebo začne pracovat chybně, neovlivní to nijak ostatní procesy. Operační systém toho dosahuje tím, že každému procesu přidělí samostatný adresový prostor a proces může manipulovat pouze s ním.

2.2.3 Ovladače zařízení

Ovladače zařízení představují důležitou část jádra Linuxu. Stejně jako ostatní části operačního systému pracují v silně privilegovaném prostředí a pokud by něco dělaly špatně, mohly by způsobit závažné problémy. Ovladače zařízení řídí interakci mezi operačním systémem a tím hardwarovým zařízením, které ovládají. Například při zápisu na disk IDE používá souborový systém obecné rozhraní blokového zařízení. Ovladač se stará o detaily a zajišťuje provedení operací, specifických pro dané zařízení. Ovladače zařízení jsou určeny vždy pro konkrétní čip řadiče, s nímž spolupracují, takže pokud například máte SCSI řadič NCR810, budete potřebovat ovladač pro řadič NCR810.

2.2.4 Souborové systémy

V Linuxu, stejně jako v systému Unix, se k samostatným souborovým systémům, které systém může používat, nepřistupuje pomocí identifikátorů zařízení (jako jsou čísla nebo jména disků). Namísto toho jsou všechna zařízení zkombinována do jediné stromové hierarchické struktury, která reprezentuje souborový systém jako celek. Do tohoto jediného stromu přidává Linux nový souborový systém pokaždé, jakmile dojde k jeho připojení do připojovacího adresáře, řekněme `/mnt/cdrom`. Jednou z důležitých vlastností Linuxu je podpora řady roz-

dílných souborových systémů. Díky tomu je celý systém velmi pružný a je schopen spolupracovat s jinými operačními systémy. Nejoblíbenější souborový systém v Linuxu je systém `ext2`, který podporuje většina distribucí Linuxu.

Souborový systém dává uživateli rozumný pohled na soubory a adresáře na discích bez ohledu na typ souborového systému nebo jiné vlastnosti samotného fyzického zařízení. Linux transparentně podporuje řadu rozdílných souborových systémů (například *MS-DOS* a `ext2`) a všechny připojené soubory a souborové systémy představuje jako jediný integrovaný virtuální souborový systém. Obecně tedy platí, že uživatelé a procesy nepotřebují vědět typ souborového systému, v němž je uložen nějaký soubor, prostě jej jenom používají.

Blokové ovladače zařízení skrývají rozdíl mezi fyzickými typy blokových zařízení (například IDE a SCSI) a, z pohledu libovolného souborového systému, představují každé zařízení pouze jako lineární seznam bloků dat. Velikosti bloků se mohou pro různá zařízení lišit, například pro disketová zařízení je typický blok o velikosti 512 bajtů, pro zařízení IDE je typický blok o velikosti 1024 bajtů a tyto podrobnosti jsou před uživatelem systému skryty. Souborový systém `ext2` vypadá vždy stejně, bez ohledu na to, na jakém zařízení je fyzicky uložen.

2.3 Datové struktury jádra

Operační systém musí udržovat řadu informací o momentálním stavu systému. Jak se v systému různé věci mění, je nutné tyto datové struktury modifikovat tak, aby odrážely skutečnost. Když se například přihlásí uživatel, může se vytvořit nový proces. Jádro musí vytvořit datovou strukturu reprezentující nový proces a zapojit jej mezi datové struktury, reprezentující všechny ostatní procesy v systému.

Tyto datové struktury povětšinou existují pouze ve fyzické paměti a přistupovat k nim může pouze jádro a jeho subsystémy. Datové struktury obsahují data a ukazatele – adresy jiných datových struktur nebo rutin. Všechny dohromady mohou datové struktury používané jádrem Linuxu působit zmatečně. Každá datová struktura však má svůj účel a přestože některé jsou používány několika subsystémy jádra, jsou podstatně jednodušší, než by mohlo vypadat na první pohled.

Pochopení jádra Linuxu je vázáno pochopením jeho datových struktur a funkcí, k nimž je jádro Linuxu používá. Tato kniha zakládá popis jádra Linuxu na jeho datových strukturách. O každém subsystému jádra se zde hovoří v termínech jeho algoritmů, metod, jakými plní svou funkci, a způsobu použití datových struktur jádra.

2.3.1 Lineární seznamy

Při údržbě svých datových struktur používá Linux řadu technik softwarového inženýrství. V řadě příležitostí používá *propojené* nebo *zřetězené* datové struktury. Pokud každou datovou strukturu chápeme jako jednu instanci nebo výskyt něčeho, řekněme procesu nebo síťového zařízení, musí být jádro schopno nalézt všechny instance. V lineárním seznamu obsahuje kořenový ukazatel adresu první datové struktury (*prvku)*, v seznamu a každá struktura obsahuje ukazatel na následující prvek seznamu. Ukazatel posledního prvku má hodnotu 0 nebo NULL, čímž se signalizuje konec seznamu. V *obousměrně propojeném* seznamu obsahuje každý prvek ukazatel jednak na následující prvek a jednak také ukazatel na předchozí prvek seznamu. Pomocí obousměrně propojených seznamů se usnadňuje přidávání a odstraňování prvků uprostřed seznamu, je k tomu ale zapotřebí více paměti. To je typické dilema každého operačního systému: rozhodování mezi zatížením paměti a procesoru.

2.3.2 Hashovací tabulky

Lineární seznamy jsou užitečný nástroj jak organizovat datové struktury, průchod takovýmto seznamem ale může být neefektivní. Pokud hledáte určitý prvek, může se vám snadno stát, že budete muset procházet celým seznamem. K obejití tohoto omezení používá Linux další techniku, *hashování*. *Hashovací tabulka* je *pole* nebo *vektor* ukazatelů. Pole nebo vektor je jednoduše soustava věcí, které leží v paměti jedna za druhou. Knihovnu můžeme chápat jako pole knih. K polím se přistupuje pomocí *indexu*, který představuje offset v poli. Pokud se budeme dále držet analogie s knihovničkou, můžete každou knihu popsat její pozicí v knihovně, můžete například chtít pátou knihu.

Hashovací tabulka je pole ukazatelů na datové struktury, přičemž její index se odvozuje od informací v těchto strukturách. Pokud budete mít datovou strukturu obsahující informace o obyvatelích nějaké vesnice, můžete jako index použít věk obyvatel. Při hledání dat o určité osobě použijete její věk jako index do hashovací tabulky obyvatel a pak už budete pouze sledovat ukazatel na datovou strukturu obsahující informace o určité osobě. Bohužel je velmi pravděpodobné, že určitý věk bude mít více obyvatel ve vesnici, takže ukazatel v hashovací tabulce je ukazatelem na řetězec či seznam datových struktur, které popisují obyvatele stejného věku. Nicméně průchod těmito kratšími seznamy je stále rychlejší než prohledávání všech datových struktur.

Hashovací tabulky urychlují přístup k často používaným datovým strukturám, Linux používá hashovací tabulky velmi často při implementaci *vyrovnávacích pamětí*. Vyrovnávací paměť obsahuje užitečné informace, k nimž je nutné přistupovat rychle, a často obsahuje pouze podmnožinu všech dostupných dat. Datové struktury se ukládají a udržují ve vyrovnávacích pamětech, protože k nim jádro přistupuje velmi často. Vyrovnávací paměti mají ovšem i nevýhodu, protože jejich použití a údržba je mnohem složitější než prostá manipulace s lineár-

ními seznamy nebo hashovacími tabulkami. Pokud se nějakou datovou strukturu podaří ve vyrovnávací paměti najít (takzvaný *zásah*), udělá to radost. Pokud tam ale požadovaná struktura není, je nutné prohledat všechny příslušné struktury, a pokud požadovaná struktura existuje, musí se přidat do vyrovnávací paměti. Při přidávání nové struktury do vyrovnávací paměti může být nezbytné odstranit z ní nějakou jinou strukturu. Linux musí rozhodnout o tom, kterou strukturu odstranit, přičemž hrozí nebezpečí, že odstraněnou strukturu bude potřebovat hned vzápětí.

2.3.3 Abstraktní rozhraní

Jádro Linuxu používá velmi často abstraktních rozhraní. Rozhraní je soubor rutin a datových struktur, které nějakým způsobem fungují. Například všechny ovladače síťových zařízení musejí nabízet určité rutiny manipulující nad určitými datovými strukturami. Řešením mohou být obecné vrstvy kódu, které používají služeb (rozhraní) nižších vrstev. Síťová vrstva je obecná a ve spolupráci s kódem pro konkrétní zařízení vytváří standardní rozhraní.

Velmi často se nižší vrstvy při zavádění systému *registrují* u vyšších vrstev. Registrace obvykle obnáší přidání nějaké datové struktury do nějakého seznamu. Například každý souborový systém vestavěný v jádře se v jádru registruje při zavádění systému nebo, pokud používáte moduly, při prvním použití daného souborového systému. Které souborové systémy jsou registrovány můžete zjistit v souboru `/proc/filesystems`. Registrační datová struktura velmi často obsahuje ukazatele na jednotlivé funkce. Jedná se o adresy programových funkcí, které zajišťují určité úkony. Když použijeme opět jako příklad registraci souborového systému, v datové struktuře, kterou každý souborový systém předává jádru při své registraci, je obsažena adresa rutiny specifické pro daný souborový systém, která musí být zavolána vždy, když se systém připojuje.

Kapitola

3

Správa paměti

Subsystém pro správu paměti je jednou z nejdůležitějších částí operačního systému. Už od prvních dnů výpočetní techniky bylo vždy zapotřebí více paměti, než kolik v systému fyzicky existovalo. Byly vyvinuty různé strategie jak toto omezení obejít, jednou z nejúspěšnějších je použití virtuální paměti. Virtuální paměť způsobuje, že se systém tváří jako kdyby měl více paměti než kolik ve skutečnosti má díky tomu, že se paměť podle potřeby sdílí mezi jednotlivými procesy.

Virtuální paměť toho ovšem zajišťuje více než jenom zvětšení objemu skutečné paměti. Subsystém správy paměti zajišťuje následující funkce:

Velký adresový prostor

Operační systém způsobuje, že se systém chová, jako kdyby měl více paměti, než kolik ve skutečnosti má. Velikost virtuální paměti může být mnohonásobně větší než velikost skutečné fyzické paměti v systému.

Ochrana

Každý proces v systému má vlastní virtuální adresový prostor. Virtuální adresové prostory jsou vzájemně úplně odděleny a běžící proces jedné aplikace tak nemůže nijak ovlivnit jiné procesy. Navíc mohou hardwarové mechanismy virtuální paměti chránit určité oblasti paměti před zápisem. Tím se kód a data chrání před přepsáním chybnými aplikacemi.

Mapování paměti

Mapování paměti slouží k mapování obrazů programu a datových souborů do adresového prostoru procesu. Při použití paměťového mapování je obsah souboru přímo svázán s virtuálním adresovým prostorem procesu.

Alokace fyzické paměti

Subsystém správy paměti umožňuje každému spuštěnému procesu alokovat spravedlivý díl fyzické paměti systému.

Sdílení virtuální paměti

Přestože virtuální paměť umožňuje, aby procesy měly oddělené (virtuální) adresové prostory, jsou situace, kdy je vhodné sdílet paměť mezi více procesy. V systému může být například více procesů, které provádějí příkaz `bash`. Není nutné mít v paměti více kopií programu `bash`, každý v adresovém prostoru jednoho procesu, je lepší mít ve fyzické paměti pouze jedinou kopii, kterou budou příslušné procesy sdílet. Dalším běžným příkladem sdílení kódu mezi více procesy jsou například dynamické knihovny.

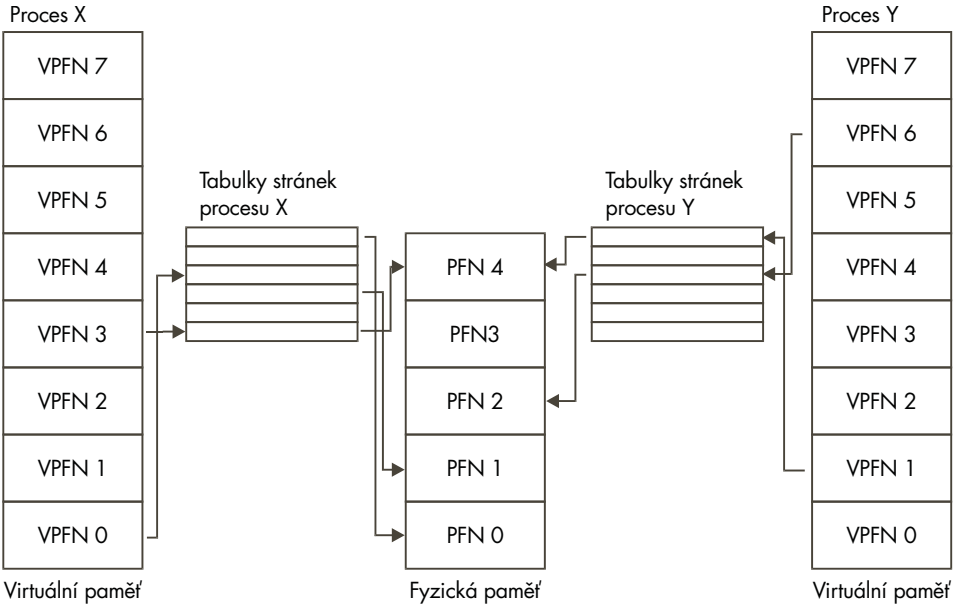
Sdílenou paměť je možno využít také jako mechanismus pro meziprocesovou komunikaci, kdy si dva nebo více procesů vyměňují informace prostřednictvím paměťové oblasti, kterou společně sdílejí. Linux podporuje meziprocesovou komunikaci sdílením paměti mechanismem Unix System V IPC.

3.1 Abstraktní model virtuální paměti

Před popisem metod, které Linux používá pro podporu virtuální paměti, bude užitečné seznámit se s abstraktním modelem, který nebude zatížen přílišnými detaily.

Když procesor provádí program, načítá z paměti instrukce a dekóduje je. Při dekódování instrukcí může potřebovat načíst nebo zapsat z nebo na určité místo paměti. Pak procesor instrukci provede a přesune se na následující instrukci programu. Procesor tedy neustále přistupuje k paměti, ať už načítáním instrukce nebo načítáním či zápisem dat.

Při použití virtuální paměti jsou všechny adresy virtuální, nejsou to adresy fyzické. Virtuální adresy překládá na fyzické adresy procesor podle informací v tabulkách, udržovaných operačním systémem.



Obrázek 3.1
Abstraktní model mapování virtuálních adres na fyzické adresy

Aby se překlad zjednodušil, je virtuální i fyzická paměť rozdělena na pohodlně manipulovatelné části, takzvané *stránky*. Všechny stránky jsou stejně veliké. Teoreticky by sice nemusely být, ale kdyby nebyly, celý systém by se jen velmi obtížně spravoval. Linux na systémech s procesorem Alpha AXP používá stránky o velikosti 8 KB, systémy s procesorem Intel *x*86 používají stránky o velikosti 4 KB. Každá stránka je označena jedinečným číslem, číslem rámce stránky (PFN).

V takovémto stránkovém modelu se virtuální adresa skládá ze dvou částí: offsetu a čísla rámce virtuální stránky. Vezmeme-li stránku o velikosti 4 KB, bity 11 až 0 virtuální adresy jsou offset a bity 12 a výše představují číslo rámce stránky. Procesor musí přeložit číslo rámce virtuální stránky na fyzickou a poté ve fyzické stránce přistoupit na místo se správným offsetem. K této činnosti procesor používá *tabulky stránek*.

Na obrázku 3.1 vidíme virtuální adresový prostor dvou procesů, procesu *X* a procesu *Y*, každý má svou vlastní tabulku stránek. Tyto tabulky mapují každou virtuální stránku procesu na fyzickou stránku paměti. Vidíme, že virtuální stránka 0 procesu *X* je mapována na fyzickou stránku 1, virtuální stránka 1 procesu *Y* se mapuje na fyzickou stránku 4. Každá položka v naší pomyslné tabulce stránek musí obsahovat následující informace:

- Příznak platnosti, který udává, zda je položka tabulky platná.
- Číslo rámce fyzické stránky, kterou tato položka popisuje.
- Informace pro řízení přístupu, které popisují, jak se stránka může používat. Smí se do ní zapisovat? Obsahuje spustitelný kód?

K tabulce stránek se přistupuje tak, že číslo rámce virtuální stránky slouží jako offset v tabulce stránek. Virtuální stránka 5 tedy bude šestou položkou tabulky (první položkou je stránka 0).

Při překladu virtuální adresy na fyzickou musí procesor nejprve zjistit číslo rámce virtuální stránky a poté offset v tabulce virtuálních stránek. Když je velikost stránky mocninou dvou, dají se tyto operace snadno provádět pomocí bitového maskování a posuvů. Podívejme se znovu na obrázek 3.1 a předpokládejme stránku o velikosti *0x2000* bajtů (desítkově 8192) a adresu *0x2194* ve virtuálním adresovém prostoru procesu *Y*, kterou procesor přeloží jako adresu na offsetu *0x194* ve virtuální stránce 1.

Číslo rámce virtuální stránky procesor použije jako index do tabulky stránek a získá tak položku tabulky pro danou stránku. Pokud je položka tabulky s daným offsetem platná, procesor z ní převezme číslo rámce fyzické paměti. Pokud položka není platná, proces se pokouší o přístup k neexistující oblasti virtuální paměti. V takovém případě procesor nemůže provést překlad adresy a musí předat řízení operačnímu systému, který problém vyřeší.

Jak ale procesor doručí operačnímu systému zprávu o tom, že se nějaký proces pokusil o přístup k oblasti virtuální paměti, kterou procesor nemohl přeložit na fyzickou adresu? Ať už je mechanismus předání zprávy jakýkoliv, této události se obecně říká *výpadek stránky* a operační systém je upozorněn na výpadek virtuální paměti a na okolnosti, za kterých k tomu došlo.

Pokud se proces odkazuje na platnou položku tabulky stránek, procesor vezme číslo fyzického rámce stránky a vynásobí jej velikostí stránky, takže získává *bázovou* adresu stránky ve fyzické paměti. K této adrese se přičte offset instrukce nebo dat, která se požadují.

Když se budeme držet našeho předchozího příkladu, mapuje se virtuální stránka 1 procesu *Y* na fyzickou stránku 4, která začíná na adrese *0x8000* ($4 \times 0x2000$). Přičtením bajtového offsetu *0x194* se dostáváme na fyzickou adresu *0x8194*.

Tento mechanismus mapování virtuálních adres na fyzické umožňuje, aby se stránky virtuální paměti mapovaly na fyzické stránky v libovolném pořadí. Například na obrázku 3.1 se virtuální stránka 0 procesu *X* mapuje na fyzickou stránku 1, zatímco virtuální stránka 7 se mapuje na fyzickou stránku 0, přestože se jedná o vyšší virtuální adresu, než je adresa stránky 0. To nám ukazuje zajímavý vedlejší efekt virtuální paměti - stránky virtuální paměti nemusejí být ve fyzické paměti přítomny v žádném pevném pořadí.

3.1.1 Vynucené stránkování

Protože fyzické paměti je v systému méně než paměti virtuální, musí být operační systém velmi opatrný na to, aby neplýtval fyzickou pamětí. Jedna možnost jak ušetřit fyzickou paměť je nahrávat pouze ty virtuální stránky, které právě spuštěný program opravdu potřebuje. Mějme například spuštěn nějaký databázový program, který něco hledá v databázi. V takovém případě není nutné, aby byla do paměti nahrána celá databáze, stačí nahrát pouze ty záznamy, které se momentálně zkoumají. Pokud databázi prohledáváme, není nutné mít v paměti zároveň nahránu tu část programu, která zajišťuje přidávání nových záznamů. Této metodě nahrávání pouze potřebných stránek se říká vynucené stránkování.

Když se proces pokusí o přístup k virtuální adrese, která momentálně není v paměti, procesoru se nepodaří pro požadovanou stránku najít platný záznam v tabulce stránek. Například podle obrázku 3.1 není v tabulce stránek procesu *X* platná položka pro virtuální stránku 2, takže pokud by se proces *X* pokusil o čtení virtuální adresy ve druhé stránce, procesoru by se nepodařilo provést překlad na fyzickou adresu. V této chvíli procesor upozorní operační systém na fakt, že došlo k výpadku stránky.

Pokud je virtuální adresa neplatná, znamená to, že aplikace se pokusila o přístup k adrese, k níž by přistupovat neměla. Aplikace třeba někam zabloudila a pokouší se nyní zapisovat na nějakou náhodnou adresu. V takovém případě ji operační systém ukončí, čímž chrání ostatní procesy v systému před chybnou aplikací.

Pokud je ale adresa platná a odkazovaná stránka momentálně není v paměti, musí operační systém přesunout tuto stránku do paměti z obrazu paměti uloženého na disku. Diskové přístupy trvají relativně dlouho, takže proces bude muset počkat, než dojde k načtení stránky. Pokud jsou v systému současně jiné procesy, které běžet mohou, nechá operační systém mezitím pracovat tyto procesy. Načtená stránka se zapíše do některé volné stránky fyzické paměti a do tabulky stránek procesu se doplní záznam ke dříve požadované virtuální stránce. Proces se pak znovu spustí na stejné instrukci, která vyvolala výpadek stránky. Tentokrát se při přístupu do virtuální paměti procesoru podaří přeložit virtuální adresu na fyzickou a proces tedy může pokračovat v práci.

Linux používá vynucené stránkování při nahrávání obrazů spustitelných souborů do virtuální paměti procesů. Když je spuštěn nějaký příkaz, otevře se soubor, který jej obsahuje, a jeho obsah se namapuje do virtuální paměti procesu. Dosáhne se toho modifikací datových struktur jádra, které popisují paměťovou mapu procesu a celý postup se označuje jako *mapování paměti*. Nicméně do fyzické paměti se nahraje pouze první část obrazu programu. Zbytek zůstává na disku. Když se obraz provádí, generuje výpadky stránek a Linux pomocí paměťové mapy procesu zjišťuje, které části obrazu je potřeba nyní přenést do paměti a nechat je proběhnout.

3.1.2 Odkládání na disk

Když proces potřebuje přenést do fyzické paměti nějakou stránku a ve fyzické paměti není žádná stránka volná, musí operační systém pro požadovanou stránku vytvořit místo tím, že ve fyzické paměti zruší nějakou jinou stránku.

Pokud rušená stránka pochází z obrazu spustitelného souboru nebo z datového souboru do nějž nebylo zapisováno, není nutné stránku ukládat. Je možno ji přímo zrušit a pokud by ji příslušný proces potřeboval znovu, načetla by se zpátky do paměti z příslušného obrazu nebo datového souboru.

Pokud ale stránka byla modifikována, musí operační systém zachovat její obsah tak, aby k ní bylo možno později opět přistupovat. Tomuto typu stránek se říká *modifikované* (*dirty*) stránky a při jejich odstraňování z paměti se ukládají do speciálního souboru, zvaného odkládací soubor. Přístupy k odkládacímu souboru jsou výrazně pomalejší než přístupy k fyzické paměti a operační systém tedy musí pečlivě balancovat mezi potřebami zapisovat stránky na disk a potřebami znovu je načítat při dalším použití.

Pokud je algoritmus používaný k rozhodnutí o tom, které stránky zrušit či odkládat (takzvaný *odkládací algoritmus*) neefektivní, dochází k situaci zvané *thrashing* - stránky se neustále zapisují na disk a čtou z disku a operační systém je vytížen natolik, že nemá čas na pořádnou práci. Pokud by například bylo často přistupováno k fyzické stránce 1 na obrázku 3.1, pak tato stránka není vhodným kandidátem pro odložení na disk. Množina stránek, které proces momentálně používá, se nazývá *pracovní prostor*. Efektivní odkládací mechanismus musí zajistit, aby pracovní prostor každého procesu byl přítomen ve fyzické paměti.

Linux používá k volbě stránek, které se mají z paměti odsunout, mechanismus zvaný LRU - Least Recently Used, tedy česky „nejdávněji použitá“. Podle tohoto schématu má každá stránka v systému svůj „věk“) který se mění s přístupem ke stránce. Čím častěji se ke stránce přistupuje, tím je „mladší“) méně navštěvovaná stránka je starší a stále stárne. Dobrým kandidátem na odložení jsou právě staré stránky.

3.1.3 Sdílená virtuální paměť

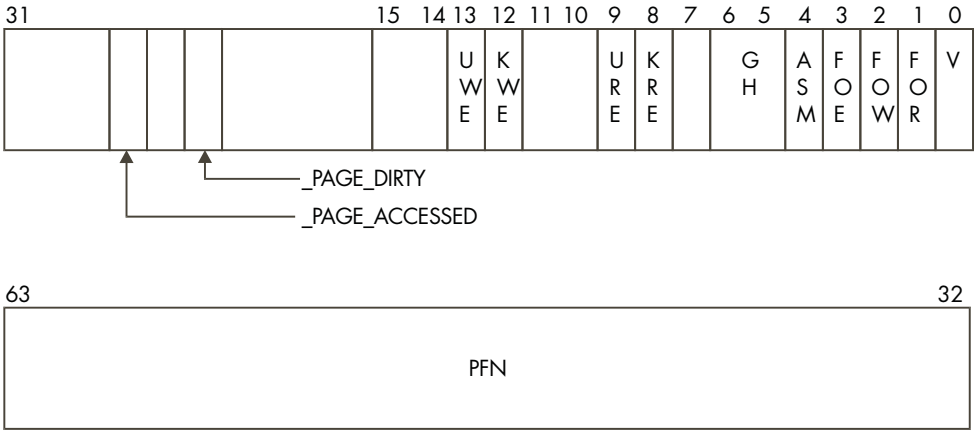
Virtuální paměť zjednodušuje sdílení paměti mezi několika procesy. Všechny přístupy do paměti se provádějí přes tabulky stránek a každý proces má svou vlastní tabulku stránek. Pokud mají dva procesy sdílet stejný kus fyzické paměti, objeví se v tabulkách stránek obou procesů stejné číslo fyzického paměťového rámce.

Na obrázku 3.1 vidíme dva procesy, které sdílejí fyzickou stránku 4. Pro proces *X* je to virtuální stránka 4, zatímco pro proces *Y* je to virtuální stránka 6. Tím se ukazuje zajímavý rys sdílení stránek: sdílené fyzické stránky se nemusejí ve virtuálním adresovém prostoru procesů, které je sdílejí, nacházet na stejných místech.

3.1.4 Fyzické a virtuální adresovací režimy

Není příliš rozumné, aby samotný operační systém běžel ve virtuální paměti. Bylo by dost příšerné, kdyby operační systém musel udržovat tabulku stránek i sám pro sebe. Většina univerzálních procesorů podporuje jak fyzický adresovací režim, tak virtuální adresovací režim. Fyzický adresovací režim nepotřebuje žádné tabulky stránek a v tomto režimu procesor neprovádí žádné překlady adres. Jádro Linuxu je sestaveno tak, aby pracovalo ve fyzickém adresovacím režimu.

Procesor Alpha AXP nemá žádný zvláštní fyzický adresovací režim. Namísto toho rozděluje paměťový prostor na několik oblastí a dvě z nich určuje jako oblasti s fyzicky mapovanými adresami. Tento adresový prostor jádra se označuje jako segment KSEG a zahrnuje všechny adresy od *0xfffffc0000000000* nahoru. Aby bylo možno provádět kód v segmentu KSEG (podle definice kód jádra) a přistupovat k datům v této oblasti, musí kód běžet v režimu jádra. Jádro Linuxu pro procesory Alpha je sestaveno tak, aby se spouštělo od adresy *0xfffffc0000310000*.



Obrázek 3.2
Položka tabulky stránek procesoru Alpha AXP

3.1.5 Řízení přístupu

Záznamy v tabulkách stránek obsahují také informace o řízení přístupu. Protože procesor tyto tabulky využívá pro mapování virtuálních adres procesů na fyzické, může při té příležitosti použít i informace o řízení přístupu ke kontrole, zda proces nepřistupuje k paměti nepovoleným způsobem.

Existuje řada důvodů proč omezovat přístup do různých oblastí paměti. Některé části paměti, například ty, kde je uložen spustitelný kód, jsou přirozeně určeny pouze pro čtení, operační systém by neměl procesu dovolit přepisovat data přes svůj spustitelný kód. Naopak stránky obsahující data je možno modifikovat, neměl by se však podařit pokus o provedení obsahu těchto stránek jako programu. Většina procesorů má nejméně dva režimy běhu programů: režim *jádra* a *uživatelský* režim. Zřejmě nebudete chtít, aby byl kód jádra a jeho datové struktury přístupné, pokud proces neběží v režimu jádra.

Informace pro řízení přístupu jsou udržovány v PTE (položkách tabulek stránek) a jsou procesorově závislé. Na obrázku 3.2 vidíme strukturu PTE procesoru Alpha AXP. Jednotlivé bity mají následující význam:

V „Valid“; je nastaven, pokud je PTE platná.

FOE „Fault on Execute“; když dojde k pokusu o provádění instrukcí v této stránce, ohlásí procesor výpadek stránky a předá řízení operačnímu systému.

FOW „Fault on Write“; jako výše, k výpadku však dochází při pokusu o zápis do této stránky.

FOR „Fault on Read“; jako výše, k výpadku však dochází při pokusu o čtení z této stránky.

ASM „Address Space Match“; tento příznak se používá pokud chce operační systém vyčistit pouze některé položky z překladového bufferu.

KRE Kód běžící v režimu jádra může tuto stránku číst.

URE Kód běžící v uživatelském režimu může tuto stránku číst.

GH Příznak granularity slouží při mapování celého bloku jedním překladovým bufferem a ne několika.

KWE Kód běžící v režimu jádra může do této stránky zapisovat.

UWE Kód běžící v uživatelském režimu může do této stránky zapisovat.

PFN Číslo rámce stránky. U PTE s nastaveným příznakem V obsahuje toto pole číslo fyzického rámce stránky. Pokud příznak V není nastaven a toto pole není nulové, obsahuje informace o tom, kde je stránka uložena ve odkládacím souboru.

Následující dva bity jsou definovány a využívány Linuxem:

_PAGE_DIRTY Pokud je příznak nastaven, stránku je nutno uložit do odkládacího souboru.

_PAGE_ACCESSED Tímto příznakem Linux označuje stránku, k níž se přistupovalo.

3.2 Vyrovnávací paměti

Pokud byste chtěli systém implementovat podle právě popsaného teoretického modelu, systém by sice mohl fungovat, nepracoval by ale příliš efektivně. Návrháři operačních systémů i procesorů usilují o co největší zvýšení výkonu systému. Kromě zrychlení samotného procesoru, paměti a dalších prvků je nejlepší řešení použití vyrovnávacích pamětí pro užitečné informace a data, které zajistí rychlejší provádění některých operací. Při správě paměti využívá Linux řadu vyrovnávacích pamětí.

Vyrovnávací paměť bufferů

Vyrovnávací paměť bufferů obsahuje datové buffery, které využívají ovladače blokových zařízení. Tyto buffery mají pevnou délku (například 512 bajtů) a obsahují bloky informací, které už byly z blokového zařízení načteny, nebo které se na něj mají zapsat. S blokovými zařízeními se dá pracovat pouze tím způsobem, že se z nich přečtou nebo se na ně zapíší bloky pevné délky. Všechny pevné disky jsou bloková zařízení.

Buffery jsou indexovány identifikátorem zařízení a číslem požadovaného bloku a slouží k rychlému nalezení bloku dat. K blokovým zařízením se vždy přistupuje pouze přes buffery. Pokud se data nacházejí v bufferu, není nutné je načíst z fyzického blokového zařízení (řekněme pevného disku) a přístup se tak výrazně zrychluje.

Vyrovnávací paměť stránek

2 Slouží ke zrychlení přístupu k obrazům programů a k datům na disku.

Ukládá se do ní logický obsah souborů stránek a přistupuje se k ní prostřednictvím identifikace souboru a offsetu. Při načítání stránek z disku do paměti se stránky uchovávají v této vyrovnávací paměti.

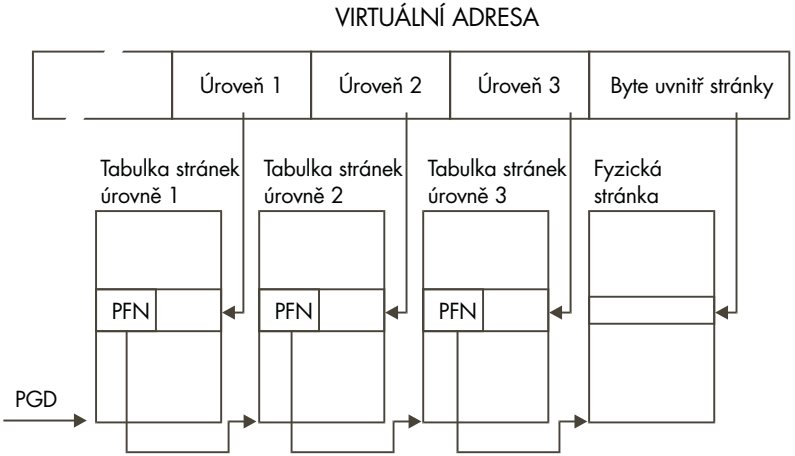
Odkládací paměť

3 V odkládacím souboru se ukládají pouze modifikované stránky.

Pokud nedojde k modifikaci stránky od posledního zápisu do odkládacího souboru, pak při následující potřebě stránku odkládat není nutné ji do souboru zapisovat, protože už v něm zapsána je. Namísto toho je možno stránku pouze zrušit. V silně odkládacím systému se tak ušetří zbytečné a náročné diskové operace.

Hardwarové vyrovnávací paměti

Jednou běžně implementovanou hardwarovou vyrovnávací pamětí je přímo v procesoru vestavěná vyrovnávací paměť položek tabulek stránek. V takovém případě procesor nemusí pokaždé přímo načítat položku tabulky stránek, podle potřeb využívá informace uložené ve své vyrovnávací paměti. Tato paměť se označuje jako překladový buffer (Translation Look-aside Buffer, TLB) a obsahuje kopie položek tabulek stránek jednoho nebo více procesů v systému.



Obrázek 3.3
Tříúrovňové tabulky stránek

Když dojde k odkazu na virtuální adresu, procesor se pokusí najít potřebnou položku TLB. Pokud ji najde, může virtuální adresu přímo přeložit na fyzickou adresu a provést požadovanou datovou operaci. Pokud se procesoru nepodaří najít potřebnou položku TLB, musí požádat o pomoc operační systém. Provede to tak, že signalizuje výpadek bloku TLB. Systémově závislé mechanismy zajistí doručení této výjimky operačnímu systému, který může problém napravit. Operační systém vygeneruje novou položku TLB pro požadované mapování. Po ošetření výjimky se procesor opět pokusí o překlad virtuální adresy. Tentokrát už všechno bude fungovat, protože pro požadovanou adresu existuje potřebný blok TLB.

Nevýhodou použití vyrovnávacích pamětí, ať už hardwarových nebo jiných, je, že kvůli ušetření práce musí Linux věnovat více času a prostoru údržbě vyrovnávacích pamětí a pokud by došlo k porušení obsahu vyrovnávacích pamětí, celý systém se zhroutí.

3.3 Tabulky stránek Linuxu

Linux předpokládá tři úrovně tabulky stránek. Každá tabulka stránek obsahuje číslo rámce stránky tabulky stránek další úrovně. Na obrázku 3.3 je vidět, jak se virtuální adresa rozpadá na řadu položek, každá z nich představuje offset v určité tabulce stránek. Při překladu virtuální adresy na fyzickou musí procesor vzít obsah položky na každé úrovni, konvertovat jej na offset ve fyzické paměti obsahující tabulku stránek a načíst číslo rámce stránky další úrovně tabulky stránek. Toto se opakuje třikrát až dojde k nalezení čísla rámce fyzické stránky, která obsahuje požadovanou virtuální adresu. V tom okamžiku se použije poslední pole virtuální adresy, bajtový offset, kterým se naleznou požadovaná data ve stránce.

Každá platforma, na níž Linux běží, musí zajistit překladová makra pro průchod tabulkou stránek procesu. Díky tomu jádro nemusí znát formát položek tabulky stránek ani jejich uspořádání.

Toto řešení je natolik úspěšné, že Linux může používat stejný kód pro manipulaci s tabulkami stránek pro procesor Alpha, který pracuje se třemi úrovněmi tabulky stránek, i pro procesor Intel *x*86, který používá pouze dvouúrovňovou tabulku.

3.4 Alokace a dealokace stránek

V systému se vyskytuje velká řada požadavků na fyzické stránky. Když například dochází k nahrávání obrazu kódu do paměti, systém potřebuje alokovat stránky. Tyto stránky budou uvolněny až kód skončí svou práci. Další použití fyzických stránek je pro uložení datových struktur jádra, jako například samotných tabulek stránek. Mechanismy a datové struktury používané pro alokování a dealokování stránek jsou pravděpodobně nejkritičtější pro efektivnost celého subsystému virtuální paměti.

5 Všechny fyzické stránky jsou popsány datovou strukturou `mem_map`, což je seznam struktur `mem_map_t` [1], které se inicializují při zavádění systému. Každá struktura `mem_map_t` popisuje jednu fyzickou stránku v systému. Důležitými položkami struktury (co se týče správy paměti) jsou následující položky:

count Počitadlo počtu uživatelů této stránky. Pokud je hodnota počitadla větší než jedna, je stránka sdílena mezi více procesy.

age Tato položka popisuje věk stránky a slouží k rozhodování, zda je stránka vhodným kandidátem na zrušení a odložení.

map_nr Číslo rámce fyzické stránky, kterou tato struktura `mem_map_t` popisuje.

Vektor `free_area` slouží kódu alokace stránek k nalezení volných stránek. Tímto mechanismem je podporováno celé schéma správy bufferů a co se týče samotného kódu, velikost stránky a fyzický stránkovací mechanismus konkrétního procesoru jsou irelevantní.

Každý prvek struktury `free_area` obsahuje informace o bloku volných stránek. První prvek pole popisuje samostatné stránky, druhý prvek bloky o dvou stránkách, následující prvek bloky o čtyřech stránkách a tak dále s krokem po mocninách dvou. Položka `list` se používá jako začátek fronty a obsahuje ukazatel na datové struktury `page` v poli `mem_map`. Zde se řadí do fronty bloky stránek. `map` je ukazatel na bitovou mapu, která obsahuje záznamy o alokovaných skupinách stránek dané velikosti. N-tý bit této bitové mapy je nastaven, pokud je N-tý blok stránek volný.

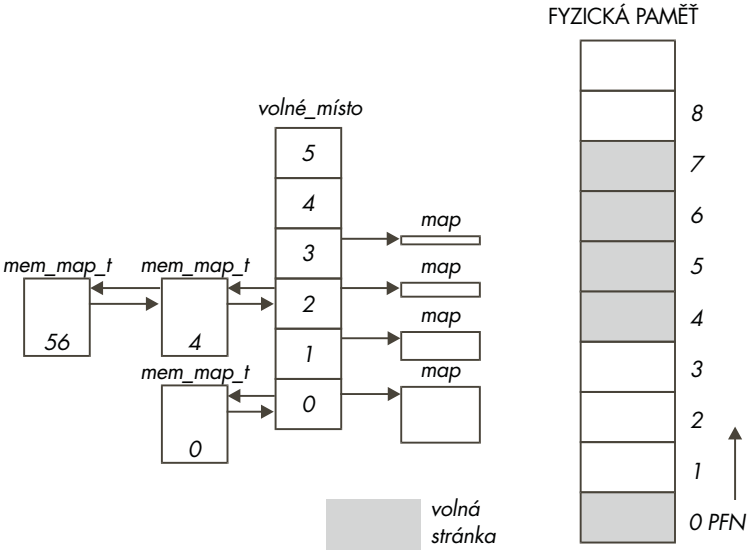
Na obrázku 3.4 vidíme strukturu `free_area`. Prvek 0 má jednu volnou stránku (stránku s číslem rámce 0) a prvek 2 má dva volné bloky o čtyřech stránkách, první začíná stránkou s číslem rámce 4 a druhý stránkou s číslem rámce 56.

3.4.1 Alokace stránek

6 K efektivní alokaci a dealokaci bloků stránek používá Linux Buddyho algoritmus. Kód pro alokování stránek se pokouší alokovat blok jedné nebo více fyzických stránek. Stránky se alokují v blocích o velikosti mocniny dvou. Znamená to tedy, že můžete alokovat blok o jedné stránce, dvou stránkách, čtyřech stránkách a tak dále. Dokud je v systému dostatek volných stránek, aby mohl být požadavek splněn (nr_free_pages > min_free_pages), bude alokační kód prohledávat strukturu `free_area` a bude hledat blok o požadovaném počtu stránek. Každý prvek seznamu `free_area` je mapa alokovaných a volných bloků stránek dané velikosti. Například prvek 2 pole je paměťová mapa, která popisuje volné a alokované bloky o velikosti 4 stránky.

[1] Je poněkud matoucí, že tato struktura se označuje také jako *stránka*.

Alokační algoritmus nejprve hledá bloky stránek požadované velikosti. Prochází seznamem volých stránek který je seřazen v prvku `list` datové struktury `free_area`. Pokud není nalezen žádný volný blok požadované velikosti, hledá se volný blok o větší velikosti (tedy blok dvakrát větší než požadovaný). Tento proces pokračuje do doby, než je prohledána celá struktura `free_area` nebo než bude nalezen vhodný blok stránek. Pokud je nalezen blok stránek o větší než požadované velikosti, musí se rozdělit na blok požadované velikosti. Protože velikosti bloků jsou vždy mocninami dvou, dělení bloků na menší části je prostě jenom jejich půlení.Volné bloky se zařadí do příslušné fronty a alokovaný blok stránek se vrátí volajícímu procesu.



Obrázek 3.4
Datová struktura `free_area`

Pokud si vezmeme například obrázek 3.4 a budeme požadovat blok o velikosti dvě stránky, dojde k rozdělení prvního bloku o čtyřech stránkách (který začíná stránkou s číslem rámce 4) na dva dvoustránkové bloky. První z nich, začínající na stránce s rámcem 4, bude vrácen volajícímu procesu jako požadovaný blok, druhý blok, začínající stránkou s číslem rámce 6, bude zařazen do fronty volných dvoustránkových bloků v prvku 1 pole `free_area`.

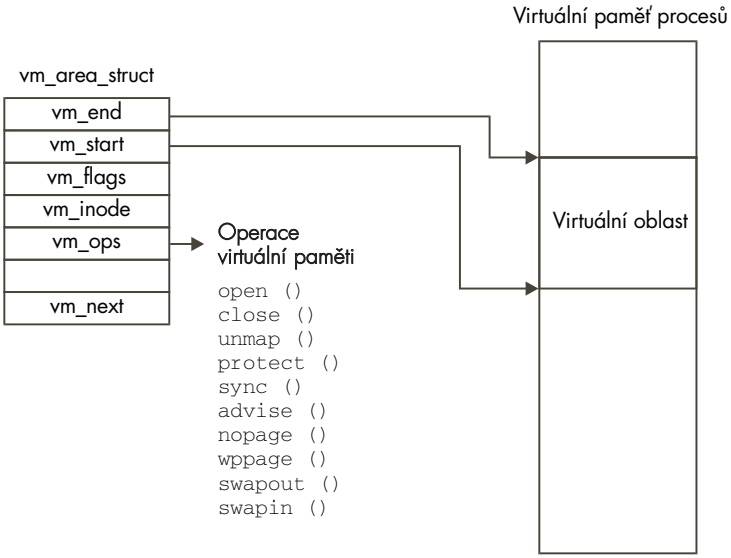
3.4.2 Dealokace stránek

Alokování bloků stránek vede k fragmentaci paměti, kdy se větší bloky volných stránek rozbíjejí na menší. Dealokační kód rekombinuje stránky zpět na větší bloky vždy když je to možné. Velikost vraceného bloku stránek je významná, protože umožňuje snadnou kombinaci bloků na větší bloky.

Vždy když dojde k uvolnění bloku stránek, kontroluje se, zda není volný sousední blok stránek o stejné velikosti. Pokud ano, pak se oba dva bloky zkombinují do nového volného bloku o dvojnásobné velikosti. Pokaždé když dojde k rekombinaci dvou bloků do většího bloku, pokusí se dealokační kód rekombinovat tento blok do ještě většího bloku. Díky tomu budou bloky volných stránek vždy tak velké, jak to jen využití paměti dovolí. Například pokud by na obrázku 3.4 došlo k uvolnění stránky s číslem rámce 1, zkombinovala by se s už volnou stránkou s číslem rámce 0 a výsledný blok by byl zařazen do prvního prvku struktury `free_area` jako blok o velikosti 2 stránky.

3.5 Mapování paměti

Když dochází k vykonávání obrazu programu, musí se obraz spustitelného kódu přenést do virtuálního adresového prostoru procesu. To platí i pro všechny sdílené knihovny, které jsou ke spustitelnému obrazu přilinkovány. Spustitelný soubor se fakticky nepřenáší do fyzické paměti, dojde pouze k jeho napojení na virtuální paměť procesu. Pak, když se spuštěná aplikace odkazuje na danou část programu, dojde k přenesení spustitelného obrazu do paměti. Toto napojení obrazu na virtuální adresový prostor procesu se označuje jako mapování paměti.



Obrázek 3.5
Oblasti virtuální paměti

Virtuální paměť každého procesu je reprezentována datovou strukturou `mm_struct`. Ta obsahuje informace o právě prováděném obrazu (například *bash*) a dále ukazatele na řadu datových struktur `vm_area_struct`. Každá datová struktura `vm_area_struct` popisuje začátek a konec oblasti virtuální paměti, přístupová práva procesu do této oblasti a operace pro tuto oblast paměti. Tyto operace jsou vlastně rutiny, které Linux musí používat při manipulaci s touto

oblastí virtuální paměti. Například jednou z operací nad oblastí virtuální paměti je provedení správné akce v případě, kdy se proces pokusí o přístup do této oblasti, ale zjistí (prostřednictvím výpadku stránky), že požadovaná oblast není momentálně ve fyzické paměti. Této operaci se říká operace *nopage*. Operace *nopage* se používá když Linux potřebuje vynutit přítomnost stránky spustitelného obrazu v paměti.

Když se spustitelný obraz mapuje do virtuální paměti procesu, generují se datové struktury `vm_area_struct`. Každá struktura `vm_area_struct` představuje část spustitelného obrazu: spustitelný kód, inicializovaná data (proměnné), neinicializovaná data a další. Linux podporuje řadu standardních operací nad virtuální pamětí a když dochází k vytváření struktury `vm_area_struct`, přiřadí se jí správná množina operací nad virtuální pamětí.

3.6 Vynucené stránkování

Jakmile je spustitelný obraz namapován do virtuální paměti procesu, může se začít provádět. Protože je v paměti fyzicky přítomen pouze úplný začátek spustitelného obrazu, dojde záhy k přístupu do oblasti virtuální paměti, která ještě není umístěna ve fyzické paměti. Když se proces pokouší o přístup k virtuální adrese, která nemá platnou položku tabulky stránek, procesor ohlásí Linuxu výpadek stránky.

Výpadek stránky oznamuje virtuální adresu, na níž došlo k výpadku, i typ paměťové operace, která výpadek způsobila.

Linux musí nalézt strukturu `vm_area_struct` té oblasti paměti, v níž došlo k výpadku stránky. Prohledávání struktur `vm_area_struct` je kritické pro efektivní obsluhu výpadků stránek, proto jsou struktury propojeny do takzvaného stromu AVL (Adělson-Velski a Landis) stromu. Pokud pro vypadlou virtuální adresu neexistuje datová struktura `vm_area_struct`, pokusil se proces o přístup na nelegální virtuální adresu. Linux odešle procesu signál `SIGSEGV` a pokud proces tento signál neobslouží, bude ukončen.

Dále Linux kontroluje typ vzniklého výpadku stránky proti typům přístupu povoleným v této oblasti virtuální paměti. Pokud se proces pokouší přistupovat k paměti nepovoleným způsobem, řekněme že chce zapisovat někam, kde je povoleno pouze čtení, bude mu rovněž signalizována paměťová chyba.

Když Linux zjistí, že výpadek stránky je legální, musí jej obsloužit.

Linux musí rozlišovat mezi stránkami uloženými v odkládacím souboru a mezi stránkami, které jsou součástí spustitelného obrazu někde na disku. Zjistí to podle položky tabulky stránek pro vypadlou stránku.

Pokud je položka tabulky stránek neplatná, ale neprázdná, znamená to, že vypadlá stránka je momentálně držena v odkládacím souboru. U položek tabulky stránek procesoru Alpha AXP jsou to ty položky, které nemají nastaven příznak platnosti, v poli PFN však obsahují nenulovou hodnotu. V tomto případě obsahuje PFN informaci o tom, kde v odkládacím souboru (a ve kterém odkládacím souboru) je stránka uložena. Jak jsou obsluhovány stránky uložené v odkládacím souboru bude popsáno dále v této kapitole.

Ne všechny datové struktury `vm_area_struct` mají přiděleny operace nad virtuální pamětí, a dokonce i ty, které je přiděleny mají, nemusejí mít přidělenu operaci *nopage*. Je to dáno tím, že implicitně Linux řeší přístup alokováním nové fyzické stránky a vytvořením platné položky tabulky stránek pro tuto stránku. Pokud není pro danou oblast paměti definována operace *nopage*, provede Linux implicitní obsluhu.

10 Obecná operace *nopage* se používá pro paměťově mapované spustitelné obrazy a používá vyrovnávací paměť stránek k přenesení požadovaného obrazu stránky do fyzické paměti.

Ať už dojde k přenosu stránky do fyzické paměti jakýmkoliv způsobem, provede se aktualizace tabulky stránek procesu. Při aktualizaci tabulky může být potřebné provést i nějaké hardwarově závislé operace, zejména pokud procesor používá interní překladové buffery. Nyní tedy byl výpadek stránky obsloužen, dále se o něj nestaráme a spouštíme proces znovu od instrukce, která výpadek stránky vyvolala.

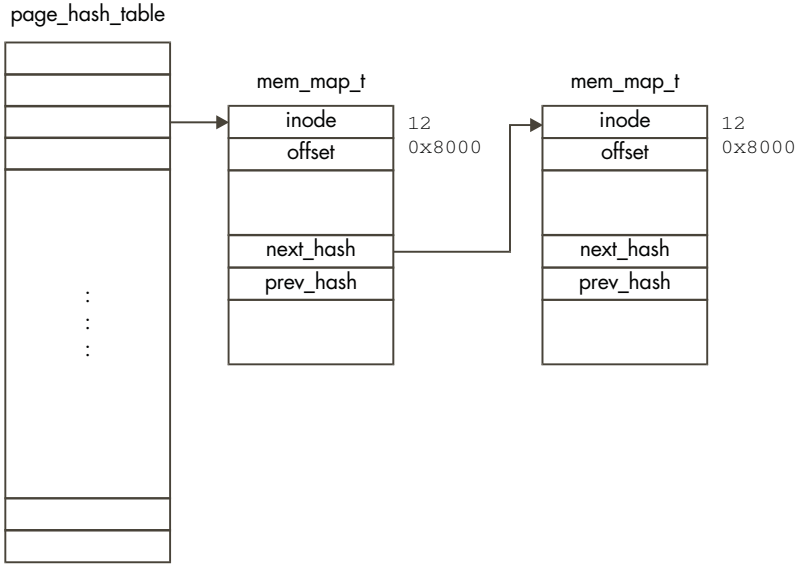
3.7 Vyrovnávací paměť stránek v Linuxu

11 Úkolem vyrovnávací paměti stránek je zrychlit přístup k souborům na disku. Paměťově mapované soubory se načítají po jednotlivých stránkách a ty se ukládají do vyrovnávací paměti stránek. Na obrázku 3.6 vidíme, že vyrovnávací paměť stránek se skládá ze struktury `page_hash_table`, vektoru ukazatelů na datové struktury `mem_map_t`.

Každý soubor v Linuxu je identifikován datovou strukturou *inode* subsystému VFS (popsaného v kapitole Souborový systém). Každý inode je jedinečné číslo a plně odpovídá jednomu a právě jednomu souboru. Index do tabulky stránek se odvozuje od *inode* souboru a offsetu v tomto souboru.

Vždy, když se z paměťově mapovaného souboru načítá stránka, například pokud vznikl požadavek na přenesení do paměti v důsledku vynuceného stránkování, načítá se stránka přes vyrovnávací paměť stránek. Pokud je stránka ve vyrovnávací paměti přítomna, vrací se kódu obsluhy výpadku stránky ukazatel na strukturu `mem_map_t`, která odpovídá požadované stránce. V opačném případě se musí stránka přenést do paměti ze souborového systému. Linux provede alokaci fyzické stránky a načte stránku ze souboru na disku.

Pokud je to možné, zahájí Linux rovněž načtení následující stránky ze souboru. Toto jednoduché dopředné čtení jedné stránky znamená, že pokud proces přistupuje ke stránkám v lineárním pořadí (což je pravděpodobné), bude mít v okamžiku požadavku na další stránku tuto stránku už připravenu.



Obrázek 3.6
Vyrovnávací paměť stránek v Linuxu

Postupem času se obsah vyrovnávací paměti stránek rozrůstá, tak jak dochází k načítání a vykonávání obrazů. Stránky se z vyrovnávací paměti odstraňují v okamžiku, kdy nejsou déle potřebné, tedy pokud není daný obraz používán žádným procesem. Když Linux pracuje s pamětí, mohou mu začít chybět fyzické stránky. V takovém případě provede snížení velikosti vyrovnávací paměti stránek.

3.8 Odkládání a rušení stránek

Když začne chybět fyzická paměť, musí se subsystém správy paměti pokusit o uvolnění fyzických stránek. Tento úkol plní odkládací démon jádra, *kswapd*.

Odkládací démon je speciální typ procesu, jde o vlákno jádra. Vlákna jádra jsou procesy, které nemají žádnou virtuální paměť a běží v režimu jádra ve fyzickém adresovém prostoru. Označení „odkládací démon“ je poněkud zavádějící, protože tento démon toho dělá daleko více, než jen prosté odložení stránek do systémových odkládacích souborů. Jeho úkolem je zajistit v systému dostatek volných stránek, aby mohl systém správy paměti pracovat efektivně.

Odkládací démon jádra (*kswapd*) je spuštěn procesem *init* jádra při startu systému a sedí a čeká na periodické spouštění odkládacím časovačem jádra.

12 Vždy, když přijde zpráva od časovače, démon se podívá, zda v systému nezačínají chybět fyzické stránky. K rozhodování, zda uvolnit nějaké stránky, používá dvě proměnné, `free_pages_high` a `free_pages_low`. Dokud je počet volných stránek větší než hodnota `free_pages_high`, nedělá démon nic, spí dál, dokud jej časovač příště neprobudí. Pro potřeby těchto testů pracuje odkládací démon s počtem stránek, které jsou momentálně zapisovány do odkládacího souboru. Tento počet je uchováván v proměnné `nr_async_pages`, jejíž hodnota se inkrementuje vždy při zařazení stránky do fronty na zápis do odkládacího souboru a dekrementuje se vždy, když se dokončí zápis stránky na odkládací zařízení. Hodnoty `free_pages_low` a `free_pages_high` se nastavují při startu systému a odvozují se od celkového počtu fyzických stránek v systému. Pokud počet volných stránek klesne pod hodnotu `free_pages_high` nebo dokonce pod hodnotu `free_pages_low`, pokusí se odkládací démon třemi způsoby snížit počet obsazených fyzických stránek:

- Redukcí velikosti bufferů a vyrovnávací paměti stránek.
- Odložením sdílených paměťových stránek Systemu V.
- Odložením a zrušením stránek.

Pokud počet volných stránek klesne pod hodnotu `free_pages_low`, pokusí se odkládací démon uvolnit šest paměťových stránek. V opačném případě se pokouší uvolnit pouze tři stránky. Všechny tři výše uvedené metody se cyklicky opakují dokud není uvolněno dostatečné množství stránek. Odkládací démon si pamatuje, kterou metodu použil při posledním pokusu o uvolnění fyzických stránek. Při každém spuštění začíná uvolňovat stránky tou metodou, která posledně uspěla.

Jakmile uvolní dostatek stránek, odkládací démon usíná a čeká na další probuzení časovačem. Pokud byl důvodem pro uvolňování stránek pokles počtu volných stránek pod hodnotu `free_pages_low`, bude démon spát pouze polovinu svého normálního spacího času. Jakmile se počet volných stránek vyhoupne nad hodnotu `free_pages_low`, spí už démon mezi jednotlivými kontrolami svou normální dobu.

3.8.1 Redukce velikosti bufferů a vyrovnávací paměti stránek

Stránky držené ve vyrovnávacích pamětech bufferů a stránek jsou dobrými kandidáty na uvolnění ve vektoru `free_area`. Vyrovnávací paměť stránek, která obsahuje stránky paměťově mapovaných souborů, může obsahovat nepotřebné stránky, které zbytečně zaplňují paměť. Podobně buffery, obsahující data čtená a zapisovaná z/na fyzická zařízení, mohou obsahovat nepotřebné údaje. Když začnou docházet fyzické stránky, je poměrně jednoduché zrušit stránky ve vyrovnávacích pamětech, protože k tomu není nutné provádět zápis na fyzická zařízení

(což by bylo nutné při odkládání stránek z paměti). Zrušení takovýchto stránek nemá žádný výrazný vedlejší efekt kromě toho, že se zpomalí přístup k fyzickým zařízením a paměťově mapovaným souborům. Pokud se ovšem rušení stránek z vyrovnávacích pamětí provede spravedlivě, dotkne se to stejným dílem všech procesů.

Vždy, když se odkládací démon jádra pokouší snížit velikost vyrovnávacích pamětí, prozkoumá blok stránek ve stránkovém vektoru `mem_map` a zjistí, zda je možno nějaké stránky uvolnit z fyzické paměti. Velikost zkoumaných bloků stránek je tím vyšší, čím více démon odkládá, tedy čím níže poklesl počet volných fyzických stránek. Bloky stránek se zkoumají cyklicky, při každém pokusu o uvolnění stránek se prověřují jiné bloky. Tento postup se označuje jako *hodinový* algoritmus, protože podobně jako u pohybu minutové ručičky hodinek dojde vždy po několika voláních nakonec k prohlédnutí celého vektoru `mem_map`.

U každé zkoumané stránky se zjistí, zda je uložena ve vyrovnávací paměti stránek nebo v bufferu. Všimněte si, že v tomto okamžiku se neuvažuje o rušení sdílených stránek a že stránka nemůže být současně v obou vyrovnávacích pamětech. Pokud stránka není v žádné z vyrovnávacích pamětí, prozkoumá se následující stránka vektoru `mem_map`.

Stránky se ukládají ve vyrovnávacích bufferech (nebo přesněji, stránky v sobě obsahují buffery), čímž se zefektivňuje alokace a dealokace bufferů. Kód snížení počtu obsazených stránek se pokouší uvolnit buffery uložené ve zkoumané stránce.

Pokud se podaří uvolnit všechny buffery, je možno uvolnit i samotnou stránku. Pokud je taková stránka uložena ve vyrovnávací paměti stránek, odstraní se z ní a uvolní se.

Pokud se při tomto postupu podaří uvolnit dostatek fyzických stránek, čeká odkládací démon na další pravidelné probuzení. Protože žádná z uvolněných stránek nebyla součástí virtuálního adresového prostoru nějakého procesu (jednalo se o stránky vyrovnávacích pamětí), není nutné modifikovat tabulky stránek. Pokud se nepodaří uvolnit dostatek stránek vyrovnávacích pamětí, pokusí se odkládací démon odložit nějaké sdílené stránky.

3.8.2 Odkládání sdílených stránek Systemu V

Sdílená paměť Systemu V je meziprocesový komunikační mechanismus, kdy dva nebo více procesů sdílejí část své virtuální paměti, aby si mohly vzájemně předávat informace. Způsob, jakým se paměť mezi procesy sdílí, je podrobněji popsán v kapitole o meziprocesové komunikaci. Nám v této chvíli stačí vědět, že sdílená paměť Systemu V je popsána datovou strukturou `shmid_ds`. Tato struktura obsahuje ukazatel na seznam datových struktur `vm_area_struct`, jednu pro každý proces sdílející danou oblast virtuální paměti. Datové struktury `vm_area_struct` říkají, kam ve virtuálním adresovém prostoru daného procesu se má mapovat daná sdílená oblast paměti. Jednotlivé datové struktury `vm_area_struct` jsou

navzájem provázány ukazateli `vm_next_shared` a `vm_prev_shared`. Každá datová struktura `shmid_ds` navíc obsahuje seznam položek tabulek stránek, které popisují fyzické stránky, na něž se virtuální sdílené stránky mapují.

15 Odkládací démon jádra používá při odkládání sdílených stránek Systemu V hodinový algoritmus. Vždy, když je spuštěn, pamatuje si, kterou oblast sdílené virtuální paměti naposledy odložil. K tomu používá dva údaje, první je index do množiny datových struktur `schmid_ds`, druhý je index do seznamu položek tabulek stránek dané oblasti sdílené paměti. Tím je zajištěno, že stránky sdílené paměti jsou „obětovány“ spravedlivě.

Protože číslo rámce fyzické stránky sdílené virtuální stránky je uloženo v tabulkách stránek všech procesů, které danou oblast virtuální paměti sdílejí, musí odkládací démon jádra modifikovat všechny tyto tabulky aby se indikovalo, že stránka už není v paměti a je držena v odkládacím souboru. Při odkládání každé sdílené stránky musí odkládací démon nalézt záznam o této stránce v každé tabulce stránek všech procesů, které stránku sdílejí (záznam se nalezne podle ukazatele v datové struktuře `vm_area_struct`). Pokud je záznam o této sdílené stránce v tabulce stránek platný, démon jej označí jako neplatný a odložený a sníží počitadlo použití této sdílené stránky o jedničku. Záznam v tabulce stránek o odložené sdílené stránce obsahuje index do množiny struktur `shmid_ds` a index do položek tabulky stránek pro tuto oblast sdílené paměti.

Pokud počitadlo použití stránky dosáhne po modifikaci tabulek všech sdílejících procesů nulu, je možno sdílenou stránku zapsat do odkládacího souboru. Položka tabulky stránek v seznamu, na nějž ukazuje datová struktura `shmid_ds`, se pro tuto oblast sdílené paměti nahradí položkou odložené stránky. Položka odložené stránky je neplatná položka tabulky stránek, obsahuje ale index do množiny otevřených odkládacích souborů a offset toho souboru, v němž je možno odloženou stránku nalézt. Tyto informace se použijí jakmile bude nutné vrátit stránku zpět do fyzické paměti.

3.8.3 Odkládání a rušení stránek

16 Odkládací démon zkontroluje všechny procesy v systému a zjistí, zda některý je vhodným kandidátem na odložení.

Vhodným kandidátem jsou procesy, které je možno odložit (u některých to nejde) a které vlastní jednu či více stránek, jež je možno z paměti odložit nebo zrušit. Stránky se z fyzické paměti odloží do systémového odkládacího souboru pouze v tom případě, že je není možno obnovit nějakou jinou metodou.

Většina stránek spustitelného obrazu pochází ze souboru tohoto obrazu a je možno je jednoduše obnovit novým přečtením tohoto souboru. Například spustitelné instrukce se nikdy nemodifikují, takže se ani nikdy nezapisují do odkládacího souboru. Takovéto stránky je možno přímo zrušit - až se na ně proces bude příště odkazovat, načtou se do paměti zpátky přímo ze spustitelného obrazu.

Jakmile je nalezen proces k odložení, odkládací démon prohlédne všechny oblasti jeho virtuální paměti a hledá oblasti, které nejsou sdíleny ani uzamčeny.

Linux neodkládá všechny odložitelné stránky vybraného procesu, zruší pouze několik málo stránek.

Pokud jsou stránky v paměti uzamčeny, není možno je odložit ani zrušit.

Odkládací algoritmus Linuxu používá stárnutí stránek. Každá stránka má počitadlo (uložené v datové struktuře `mem_map_t`), které dává odkládacímu démonu jádra jakousi představu o tom, zda stránka stojí či nestojí za odložení. Stránky stárnou, když nejsou používány, a mládnou při každém přístupu. Odkládací démon odkládá pouze staré stránky. Implicitní operace při alokování nové stránky je, že stránka získá výchozí věk 3. Při každém přístupu se její věk inkrementuje o 3 až do maxima 20. Vždy když je odkládací démon spuštěn, nechá stránky zestárnout tím, že jejich věk dekrementuje o jednu. Toto implicitní chování je možno měnit, a proto se všechny tyto (a další odkládací informace) udržují ve struktuře `swap_control`.

Pokud je stránka stará (věk = 0), odkládací démon ji dále zpracuje. Modifikované stránky je nutné odložit. K popisu těchto vlastností stránek používá Linux strojově závislé příznaky ve strukturách PTE (viz obrázek 3.2). Ne všechny modifikované stránky je však nezbytně nutné uložit do odkládacího souboru. Každá oblast virtuální paměti procesu může mít svou vlastní odkládací operaci (na kterou ukazuje ukazatel `vm_ops` struktury `vm_area_struct`) a pak se použije tato operace. Pokud operace není definována, odkládací démon implicitně alokuje stránku v odkládacím souboru a zapíše rušenou stránku na odkládací zařízení.

Položka tabulky stránek pro odloženou stránku se přepíše údajem, který říká, že položka není platná, a obsahuje informace o tom, kde se stránka v odkládacím souboru nachází. Tento údaj obsahuje jednak offset odkládacího souboru, jednak určení, který odkládací soubor byl použit. Ať už byla použita jakákoliv metoda odkládání, původní fyzická stránka je uvolněna a přidá se zpět do seznamu `free_area`. Čisté (přesněji řečeno nemodifikované) stránky je možno zrušit a uložit zpět do struktury `free_area` okamžitě.

Pokud bylo odloženo a zrušeno dostatečné množství stránek, odkládací démon usíná. Při příštím probuzení posoudí jiný proces. Tímto způsobem odkládací démon „uzobne“ pár stránek každému procesu, dokud nebude mít systém opět dostatek paměti. Je to podstatně spravedlivější řešení než úplné odložení celého procesu.

3.9 Odkládací vyrovnávací paměť

Při odkládání stránek do odkládacích souborů se Linux snaží vyhnout zápisu, pokud to není nezbytné. Může nastat situace, kdy je stránka jak v odkládacím souboru, tak ve fyzické paměti. Dochází k tomu v případě, že stránka odložená z paměti byla později do paměti vrácena, když ji její proces opět potřeboval. Pokud nedošlo k zápisu do této stránky v paměti, její kopie v odkládacím souboru je stále platná.

Ke sledování těchto stránek používá Linux vyrovnávací odkládací paměť. Tato paměť představuje seznam položek tabulek stránek, jednu pro každou fyzickou stránku v systému. V položkách odložených stránek je zapsáno, ve kterém souboru jsou uloženy a jejich pozice v tomto souboru. Pokud je údaj ve vyrovnávací paměti nenulový, znamená to, že stránka uložená v odkládacím souboru nebyla modifikována. Pokud dojde k modifikaci stránky (zápisem do této stránky), údaj o ní se z odkládací vyrovnávací paměti odstraní.

Když Linux potřebuje fyzickou stránku odložil do odkládacího souboru, podívá se nejprve do odkládací vyrovnávací paměti, a pokud v ní najde platný záznam o stránce, nemusí ji zapisovat do odkládacího souboru. Platný záznam totiž indikuje, že stránka nebyla od posledního načtení z odkládacího souboru modifikována.

Položky v odkládací vyrovnávací paměti představují položky tabulky stránek pro odložené stránky. Jsou označeny jako neplatné, obsahují však informace, podle kterých Linux nalezne správný odkládací soubor a správnou stránku v tomto souboru.

3.10 Opětovné vkládání stránek

Modifikované stránky uložené v odkládacím souboru mohou být později znovu zapotřebí, například v okamžiku, kdy aplikace potřebuje zapisovat do té oblasti virtuální paměti, jejíž obsah je uložen v odložené fyzické stránce. Pokus o přístup k virtuální stránce, která není přítomna ve fyzické paměti, vyvolá výpadek stránky. Výpadkem stránky signalizuje procesor operačnímu systému, že není schopen přeložit virtuální adresu na fyzickou. V našem případě je to způsobeno tím, že při odložení stránky z paměti byl její záznam v tabulce stránek označen jako neplatný. Procesor není schopen obsloužit překlad virtuální adresy na fyzickou, takže předává řízení zpět operačnímu systému a říká mu, jaká virtuální adresa vypadla a jaký byl důvod výpadku. Formát těchto informací a způsob, jakým procesor předává řízení operačnímu systému, je závislý na procesoru.

19 Procesorově závislý kód obsluhy výpadku stránky musí nalézt datovou strukturu `vm_area_struct` popisující oblast virtuální paměti, do níž spadá vypadnuvší adresa. Dosáhne toho prohledáváním všech datových struktur `vm_area_struct` daného procesu tak dlouho,

dokud nenajde tu, která obsahuje adresu, jež způsobila výpadek. Jedná se o časově velmi kritický kód, a proto jsou datové struktury `vm_area_struct` procesu organizovány tak, aby vyhledávání zabralo co nejméně času.

Jakmile se obslouží všechny procesorově závislé akce a nalezne se oblast, do níž spadá vypadlá adresa, je už obsluha výpadku dále obecná a na procesoru nezávislá. 20

Obecný kód obsluhy výpadku se podívá na záznam v tabulce stránek pro vypadlou stránku. Pokud položka indikuje, že stránka je odložena, musí ji Linux opětovně vložit zpět do fyzické paměti. Formát položky tabulky stránek pro odložené stránky je procesorově závislý, všechny procesory však položku označují jako neplatnou a ukládají v ní nějakou informaci o tom, kde odloženou stránku nalézt. Tyto informace Linux potřebuje, aby mohl stránku vrátit zpět do fyzické paměti.

V tomto okamžiku Linux zná adresu, která způsobila výpadek, a má položku tabulky obsahující informaci o tom, kam byla stránka odložena. Datová struktura `vm_area_struct` může obsahovat ukazatel na rutinu, která opětovně vloží jakoukoliv stránku dané oblasti virtuální paměti zpět do fyzické paměti. Jedná se o operaci *swapin*. Pokud daná oblast paměti má definovánu operaci *swapin*, Linux ji použije. To je mimo jiné i způsob, jakým se obsluhuje odkládání stránek sdílených mechanismem System V, protože jejich odkládání vyžaduje speciální obsluhu danou tím, že formát odložených sdílených stránek je poněkud odlišný od normálních odložených stránek. Operace *swapin* nemusí být definována a v takovém případě Linux předpokládá, že se jedná o běžnou stránku, která nevyžaduje speciální obsluhu. 21 22 23

Nealokuje volnou fyzickou stránku a načte stránku zpět z odkládacího souboru. Informace o tom, kde se stránka v odkládacím souboru nachází (a ve kterém odkládacím souboru), se zjistí z položky tabulky stránek pro tuto stránku.

Pokud přístup, který způsobil výpadek stránky, nebyl pokusem o zápis, zůstává stránka ve vyrovnávací odkládací paměti a její položka v tabulce stránek je označena jako nezapisovatelná. Pokud následně dojde k zápisu do stránky, generuje se nový výpadek stránky a v tom okamžiku systém označí stránku jako modifikovanou a odstraní ji z vyrovnávací odkládací paměti. Pokud do stránky nebylo zapsáno a je zapotřebí ji znovu načíst, Linux si ušetří zápis stránky do odkládacího souboru, protože soubor už obsahuje platnou kopii stránky.

Pokud byl výpadek způsobem zápisem, stránka se odstraní z vyrovnávací paměti rovnou a označí se jako stránka modifikovaná a s povoleným zápisem.

Odkazy na zdrojové texty jádra

1 – Viz fs/buffer.c
2 – Viz mm/filemap.c
3 – Viz swap.h, mm/swap_state.c, mm/swapfile.c
4 – Viz include/asm/pgtable.h
5 – Viz include/linux/mm.h
6 – Viz __get_free_pages() v souboru mm/page_alloc.c
7 – Viz free_pages () v souboru mm/page_alloc.c
8 – Viz handle_mm_fault() v souboru mm/filemap.c
9 – Viz do_no_page() v souboru mm/memory.c
10 – Viz filemap_nopage() v souboru mm/filemap.c
11 – Viz include/linux/pagemap.h
12 – Viz kswapd() v souboru mm/vmscan.c
13 – Viz shrink_mmap() v souboru mm/filemap.c
14 – Viz try_to_free_buffer() v souboru fs/buffer.c
15 – Viz shm_swap() v souboru ipc/shm.c
16 – Viz swap_out() v souboru mm/vmscan.c
17 – Aby to mohl udělat, sleduje ukazatel vm_next na struktury vm area struct seřazené v seznamu mm_struct daného procesu.
18 – Viz swap_out() v souboru mm/vmscan.c
19 – Viz do_page_fault() v souboru arch/i386/mm/fault.c
20 – Viz do_no_page() v souboru mm/memory.c
21 – Viz do_swap_page() v soubrou mm/memory.c
22 – Viz shm_swap_in() v souboru ipc/shm.c
23 – Viz swap_in() v souboru mm/page_alloc.c

Kapitola

4

Procesy

V této kapitole je popsáno co to jsou procesy a jak jádro Linuxu procesy vytváří, spravuje a ruší.

Procesy provádějí v operačním systému úlohy. Program je souhrn strojových instrukcí a dat uložených ve spustitelném obrazu na disku a je to v zásadě pasivní entita. Naproti tomu proces můžeme chápat jako program v akci.

Jedná se o dynamickou entitu, která se trvale mění tak jak procesor vykonává jednotlivé strojové instrukce. Kromě instrukcí a dat programu jsou součástí procesu také čítač instrukcí a všechny registry procesoru a dále zásobník, který obsahuje dočasně odložená data jako parametry rutin, návratové adresy a uložené proměnné. Momentálně spuštěný program nebo proces zahrnuje všechny aktivity procesoru. Linux je multiprocesový operační systém. Procesy jsou oddělené úlohy, každý má vlastní práva a vlastní zodpovědnost. Pokud havaruje jeden proces, nezpůsobí to havárii jiného procesu v systému. Každý jeden proces běží ve svém vlastním virtuálním adresovém prostoru a není schopen komunikovat s ostatními procesy jinak než pomocí bezpečných, jádrem řízených mechanismů.

V době svého života proces používá řadu systémových prostředků. Používá procesor k vykonávání svých instrukcí a fyzickou paměť systému k uložení sebe sama a svých dat. Otevírá a používá soubory v souborovém systému a může přímo či nepřímo používat fyzická zařízení systému. Linux musí sledovat jednak proces a jednak prostředky, které proces využívá, aby mohl zajistit spravedlivé rozdělení prostředků mezi všemi procesy v systému. Nebylo by vůči ostatním procesům spravedlivé, kdyby si jeden proces monopolizoval většinu fyzické paměti nebo procesorového času.

Nejcennějším prostředkem systému je procesor, protože je v něm obvykle pouze jeden. Linux je multiprocesový operační systém a jeho cílem je zajistit, aby v každém okamžiku běžel na každém procesoru nějaký proces, aby se procesorů využilo co nejvíce. Pokud je procesů více než procesorů (což obvykle bývá), musejí ostatní procesy s během počkat než bude procesor volný. Multiprocessing je jednoduchá myšlenka - proces běží dokud nemusí čekat, obvykle na nějaký systémový prostředek, pokud prostředek má, může běžet dále. V jednoprocesovém systému, jako je například DOS, by procesor po dobu čekání na prostředek zůstal nečinný a mrhal by časem. V multiprocesovém systému je v paměti současně přítomno více procesů. Kdykoliv musí proces čekat, operační systém mu odebere procesor a předá jej jinému procesu, který jej potřebuje více. Rozhodování o tom, který proces má příště běžet, provádí plánovač úloh. Linux používá řadu různých plánovacích strategií, aby se zajistilo spravedlivé rozdělení procesoru.

Linux podporuje různé formáty spustitelných souborů, jedním je ELF, dalším může být Java a všechny tyto formáty musejí být spravovány transparentně tak, jak procesy využívají sdílené knihovny systému.

4.1 Procesy na Linuxu

1 Aby mohl Linux jednotlivé procesy v systému spravovat, je každý proces reprezentován datovou strukturou `task_struct` (termíny „úloha“ (task) a „proces“ jsou v Linuxu rovnocenné). Vektor `task` je pole ukazatelů na všechny struktury `task_struct` v systému.

Znamená to, že maximální počet procesů v systému je omezen velikostí vektoru `task`, který má implicitně 512 položek. Při vzniku procesu se v paměti alokuje nová struktura `task_struct` a přidá se vektoru `task`. Aby se usnadnilo vyhledávání, na aktuální proces je možno odkazovat se ukazatelem `current`.

Kromě normálních procesů Linux dále podporuje takzvané procesy reálného času. Tyto procesy musejí velmi rychle reagovat na externí události (proto procesy „reálného času“) a plánovač s nimi zachází odlišně od normálních uživatelských procesů. Přestože struktura `task_struct` je poměrně rozsáhlá a složitá, jednotlivé položky je možno rozdělit do několika funkčních oblastí:

Status

Při běhu procesu se podle okolností mění jeho *status*. Procesy v Linuxu mohou mít následující status[1]:

1 Neuvádím status SWAPPING, protože ten se zřejmě nepoužívá.

Running (běžící) Proces buď právě běží (aktuální proces), nebo je připraven k běhu (čeká na přidělení nějakého procesoru).

Waiting (čeká) Proces čeká na událost nebo prostředek. Linux rozlišuje dva typy čekajících procesů - *přerušitelné* a *nepřerušitelné*. Přerušitelné procesy je možno přerušit zasláním signálu, zatímco nepřerušitelné procesy čekají přímo na nějakou hardwarovou událost a není možno je přerušit za žádných okolností.

Stopped (zastaven) Proces byl zastaven, obvykle zasláním nějakého signálu. V zastaveném stavu se může nacházet například laděný proces.

Zombie Ukončený proces, který má ve vektoru `task` stále z nějakého důvodu zachovánu strukturu `task_struct`. Je to přesně to, co napovídá název, tedy mrtvý proces.

Plánovací informace

Tyto informace potřebuje plánovač k rozhodování, který proces si nejvíce zaslouží spustit.

Identifikátory

Každý proces má svůj identifikátor procesu. Identifikátor není indexem do vektoru `task`, je to prostě číslo. Každý proces má dále identifikátory uživatele a skupiny, které slouží k řízení přístupových práv procesu k souborům a zařízením v systému.

Meziprocesová komunikace

Linux podporuje klasické unixovské komunikační mechanismy jako signály, roury a semafory a dále mechanismy Systemu V pro sdílení paměti, semafory a fronty zpráv. Mechanismy pro meziprocesovou komunikaci jsou popsány v kapitole Meziprocesová komunikace.

Odkazy

V Linuxu není žádný proces nezávislý na ostatních procesech. Všechny procesy vyjma inicializačního procesu mají svůj rodičovský proces. Nové procesy nevznikají, kopírují se, nebo přesněji *klonují*, z předchozích procesů. Každá struktura `task_struct` každého procesu obsahuje ukazatele na jeho rodičovský proces a na jeho sourozence (ostatní procesy se stejným rodičovským procesem) a dále ukazatele na jeho synovské procesy. Rodinné vztahy mezi procesy v systému můžete zjistit pomocí příkazu `pstree`:

```
init(1)-+-crond(98)
        |-emacs(387)
        |-gpm(146)
        |-inetd(110)
        |-kerneld(18)
        |-kflushd(2)
        |-klogd(87)
        |-kswapd(3)
        |-login(160)---bash(192)---emacs(225)
        |-lpd(121)
        |-mingetty(161)
        |-mingetty(162)
        |-mingetty(163)
        |-mingetty(164)
        |-login(403)---bash(404)---pstree(594)
        |-sendmail(134)
        |-syslogd(78)
        `-update(166)
```

Navíc jsou všechny procesy v systému svázány obousměrně propojeným seznamem, jehož kořenem je datová struktura `task_struct` procesu `init`. Tento seznam jádru umožňuje dohled nad všemi procesy v systému. Navíc je potřebný pro podporu příkazů jako jsou `ps` nebo `kill`.

Časy a časovače

Jádro udržuje údaj o čase spuštění procesu i o celkovém času procesoru, který proces doposud spotřeboval. S každým tikem hodin jádro inkrementuje časový údaj o tom, kolik času proces strávil v systému a v uživatelském režimu. Linux navíc podporuje *intervalové* časovače procesů; proces může pomocí systémových volání tyto časovače nastavit tak, aby po uplynutí určité doby poslaly procesu signál. Tyto časovače mohou být jednorázové nebo periodické.

Souborový systém

Procesy mohou podle potřeby otevírat a zavírat soubory a struktura `task_struct` procesu obsahuje ukazatele na deskriptory otevřených souborů a dále ukazatele na dva VFS inody. Inody jednoznačně popisují soubor nebo adresář v souborovém systému a představují také uniformní rozhraní k nižším vrstvám souborového systému. Podpora souborových systémů v Linuxu je vysvětlena v kapitole Souborový systém. První ukazatel je na inode kořene procesu (jeho domovského adresáře), druhý ukazuje na jeho aktuální či pracovní (*pwd*) adresář.

Označení *pwd* je odvozeno od příkazu `pwd` systému Unix, který vypisuje pracovní adresář procesu. Pro inody je vedeno počitadlo `count`, které říká, že se na ně jeden či více procesů odkazují.

Virtuální paměť

Většina procesů má nějakou virtuální paměť (nemají ji pouze démony a vlákna jádra) a jádro Linuxu musí sledovat, jak se virtuální paměť mapuje na fyzickou paměť systému.

Procesorově specifický kontext

Proces můžeme chápat jako souhrn celého aktuálního stavu systému. Vždy, když proces běží, používá registry procesoru, zásobník a podobně. To všechno je kontext procesu a když dojde k pozastavení procesu, musí se celý kontext procesu uložit do struktury `task_struct`. Když plánovač proces opět spustí, kontext procesu se odtud obnoví.

4.2 Identifikátory

Linux, stejně jako systém Unix, používá ke kontrole přístupových práv k souborům identifikátory uživatelů a skupin. Každý soubor a adresář v Linuxu má své vlastníky a svá oprávnění, která popisují práva uživatelů systému k tomuto souboru či adresáři. Základními oprávněními jsou práva *čtení*, *zápisu* a *spuštění*, která se přiřazují třem třídám uživatelů: vlastníkovi souboru, procesům patřícím do nějaké skupiny a všem procesům v systému. Každá třída uživatelů může mít jiná oprávnění, takže je třeba možné nastavit oprávnění tak, že vlastník bude moci soubor číst i zapisovat, skupina souboru jej bude moci jenom číst a všechny ostatní procesy v systému nebudou mít žádná přístupová práva.

Skupiny představují způsob přiřazení privilegií k souborům a adresářům skupinám uživatelů a ne jen jednomu uživateli nebo všem uživatelům v systému. Můžete například vytvořit skupinu pro všechny uživatele podílející se na nějakém projektu a nastavit ji tak, že bude moci číst i modifikovat zdrojové kódy projektu. Proces může patřit do více skupin (maximální počet je implicitně 32) a tyto skupiny se ukládají ve vektoru `groups` struktury `task_struct` každého procesu. Pokud má nějaká skupina přístupová práva k nějakému souboru a proces do této skupiny patří, má k danému souboru práva této skupiny.

Ve struktuře `task_struct` procesu se udržují čtyři dvojice uživatelských a skupinových identifikátorů procesu:

uid, gid — Identifikátor uživatele a skupiny toho uživatele, jehož jménem proces běží.

efektivní uid a gid Existují některé programy, které mění uid a gid spouštějícího procesu na své vlastní (uložené jako atributy v inodu spustitelného obrazu). Tyto programy se označují jako *setuid* programy a jsou velmi užitečné, protože představují způsob jak omezit přístup ke službám, zejména službám spouštěným jménem někoho jiného, například síťových démonů. Efektivní uid a gid jsou nastavena podle atributů setuid programu, hodnoty uid a gid zůstávají nezměněny. Při kontrole přístupových práv používá jádro hodnoty efektivních uid a gid.

uid a gid souborového systému Jsou normálně stejné jako efektivní uid a gid a slouží při kontrole přístupových práv k souborovému systému. Jsou nutné pro souborové systémy připojené prostřednictvím NFS, kdy server NFS v uživatelském režimu potřebuje přístup k souborům tak, jako kdyby byl určitým procesem. V takovém případě se změní uid a gid pouze souborového systému, ne efektivní uid a gid. Tím se zabrání situaci, kdy by zlomyslný uživatel mohl serveru NFS poslat signál *kill*. Signály *kill* se doručují procesům s určitým efektivním uid a gid.

uložené uid a gid Tyto hodnoty jsou vyžadovány normou POSIX a používají se v programech, které mění uid a gid procesu prostřednictvím systémových volání. Slouží k uložení skutečného uid a gid v době, kdy jsou původní hodnoty uid a gid změněny.

4.3 Plánování

Všechny procesy běží částečně v uživatelském režimu a částečně v systémovém režimu. Nízkoúrovňová hardwarová podpora těchto režimů může být různá, obecně ale platí, že existují nějaké bezpečnostní mechanismy při přechodu z uživatelského režimu do systémového a zpět. V uživatelském režimu má proces výrazně menší privilegia než v režimu systémovém. Vždy při použití systémového volání se proces přepíná z uživatelského režimu do systémového režimu a pokračuje v práci. V té době pracuje jménem procesu jádro. V Linuxu procesy nemohou vynuceně přerušit aktuálně běžící proces, nemohou jeho běh pozastavit, aby mohly běžet samy. Každý proces se dobrovolně vzdává procesoru, na němž běží v době, kdy musí čekat na nějakou systémovou událost. Proces řekněme čeká například na načtení znaku ze souboru. Toto čekání se provádí využitím systémového volání, v systémovém režimu. Proces použije knihovních funkcí k otevření a čtení souboru a poté následně používá systémových volání ke čtení bajtů z otevřeného souboru. V takovém případě je čekající proces pozastaven a umožní se běh jinému, potřebnějšímu procesu.

Procesy používají systémových volání často, takže mohou být také často přinuceny čekat. Ale i v této situaci může nějaký proces bez čekání běžet příliš dlouho a spotřebovávat nepřiměřené množství procesorového času, proto Linux používá preemptivní plánování úloh. Při využití tohoto schématu je každému procesu umožněno běžet malý objem času, 200 ms, a jakmile tento čas vyprší, naplánuje se běh jiného procesu a původní proces musí čekat na další příležitost ke běhu. Tento krátký přidělovaný časový úsek se označuje jako *časové kvantum (time-slice)*.

Rozhodování o tom, který proces právě nechat běžet, je úkolem *plánovače*.

Spustitelný proces je takový, který čeká pouze na přidělení procesoru. Linux používá rozumně jednoduchý na prioritě založený plánovací algoritmus, který volí aktuální procesy. Když zvolí nový aktuální proces, uloží status momentálně aktuálního procesu, procesorově specifické registry a další kontextové informace do datové struktury `task_struct`. Poté obnoví status nově naplánovaného procesu (což je opět procesorově závislá operace) a předá mu řízení systému. Aby mohl plánovač spravedlivě rozdělovat procesorový čas mezi spustitelné procesy v systému, ukládá si do struktury `task_struct` každého procesu následující informace:

policy Plánovací strategie používaná pro tento proces. Linux má dva typy procesů, normální a procesy reálného času. Procesy reálného času mají vyšší prioritu než všechny ostatní procesy. Pokud je k běhu připraven proces reálného času, bude vždy spuštěn. Procesy reálného času mohou používat dva typy strategie, `policy`, a to buď *round robin*, nebo *first in first out*. V plánování typu *round robin* se používá cyklická obsluha procesů a jednotlivé procesy se spouštějí v řadě za sebou. Při strategii *first in first out* se spustitelné procesy spouštějí v pořadí uvedeném ve spouštěcí frontě, které se nikdy nemění.

priority Priorita, kterou plánovač dává procesu. Je to zároveň množství času (v jednotkách zvaných `jiffies` *), které proces poběží, když bude naplánován. Prioritu procesů je možno měnit pomocí systémových volání a příkazem *renice*.

rt_priority Linux podporuje procesy reálného času, které se plánují tak, aby měly větší prioritu než všechny ostatní procesy v systému. Tato položka umožňuje plánovači přiřadit každému procesu reálného času jeho relativní prioritu. Prioritu procesů reálného času je možno měnit pomocí systémových volání.

counter Počet časových okamžiků (v `jiffies`), které proces může běžet. Při naplánování procesu se nastaví na hodnotu `priority` a s každým hodinovým tikem se dekrementuje.

* Poznámka překladatele: „vteřinka“, „momentík“.

Plánovač je spouštěn z několika míst v jádře. Spouští se po převedení aktuálního procesu do fronty čekajících procesů, může být zavolán po ukončení systémového volání, těsně před přepnutím procesu ze systémového režimu do uživatelského režimu. Dalším důvodem spuštění může být, že systémový časovač dodekrementoval hodnotu `counter` procesu na nulu. Při každém spuštění provede plánovač následující úkoly:

Úlohy jádra Plánovač spustí bottom-half obsluhu a zpracuje frontu úloh plánovače. Tato „odlehčená“ vlákna jádra jsou podrobně popsána v kapitole Mechanismy jádra.

Aktuální proces Před výběrem dalšího procesu musí být spuštěn aktuální proces.

Pokud se používá plánovací strategie *round robin*, uloží se proces na konec fronty spouštěných procesů.

Pokud je úloha v přerušitelném (`INTERRUPTIBLE`) stavu a od posledního naplánování obdržela signál, přepne ji plánovač do stavu `RUNNING`.

Pokud vypršel aktuální proces, dostává se do stavu `RUNNING`.

Pokud je aktuální proces ve stavu `RUNNING`, zůstává v něm.

Procesy, které nejsou ani `RUNNING`, ani `INTERRUPTIBLE` se odstraní z fronty spouštěných procesů. Znamená to, že plánovač nebude tyto procesy posuzovat při výběru nejvhodnějšího procesu pro spuštění.

Výběr procesu Plánovač prohlédne procesy ve frontě spouštěných procesů a vybere nejlepšího kandidáta na spuštění. Pokud je ve frontě nějaký proces reálného času (proces s plánovací strategií reálného času), bude ohodnocen více než normální procesy. Váha normálního procesu odpovídá jeho hodnotě `counter`, u procesů reálného času to je `counter` plus 1000. Znamená to, že pokud je v systému nějaký spustitelný proces reálného času, bude spuštěn vždy dříve než normální spustitelné procesy. Aktuální proces, který vypotřeboval nějakou část svého časového kvanta (jeho hodnota `counter` byla snížena), je v nevýhodě před ostatními procesy se stejnou prioritou, což je v pořádku. Pokud má stejnou prioritu několik procesů, bude vybrán první proces z fronty. Aktuální proces bude zařazen na konec fronty. Ve vyváženém systému s mnoha procesy se stejnou prioritou se budou tyto procesy spouštět jeden po druhém. Takovéto plánování se označuje jako *round robin* – plánování cyklickou obsluhou. Jak ale procesy čekají na různé prostředky, pořadí jejich provádění se různě prohazuje.

Přepnutí procesů

Pokud je nejvhodnějším kandidátem na spuštění jiný proces než aktuální, musí být aktuální proces pozastaven a připraví se ke spuštění jiný proces. Když proces běží, používá registry a fyzickou paměť procesoru a systému. Při každém volání rutin jim v registrech předává parametry a může využívat hodnoty na zásobníku, například k uložení návratové adresy do volající rutiny. Když tedy plánovač běží, běží v kontextu aktuálního procesu. Jedná se sice o privilegovaný režim, režim jádra, ale stále běží jako aktuální proces. Když dojde k pozastavení tohoto procesu, je nutné uložit celý jeho strojový stav včetně čítače instrukcí (PC) a všech registrů procesoru do jeho struktury `task_struct`. Pak je nutné obnovit strojový stav nově naplánovaného procesu. Jedná se o strojově závislé operace, každý procesor to provádí trochu odlišným způsobem, obvykle však pro tuto operaci existuje nějaká hardwarová podpora.

Přepnutí procesů je poslední operace prováděná plánovačem. Uložený kontext předchozího procesu je tedy snímek hardwarového kontextu systému v okamžiku, kdy byl tento proces na konci plánovače. Obdobně tedy když dojde k nahrání nového procesu, v jeho snímku je rovněž zachycena situace, kdy se proces nacházel na konci plánovače, včetně obsahu instrukčního čítače a registrů.

Pokud předchozí proces nebo nový aktuální proces používají virtuální paměť, může být zapotřebí změnit údaje ve stránkovacích tabulkách. I tato akce je závislá na architektuře. Procesory jako Alpha AXP, které používají překladové tabulky a uložení položek tabulky stránek ve vyrovnávací paměti, musejí zneplatnit ty údaje ve vyrovnávací paměti, které patřily předchozímu procesu.

4.3.1 Plánování ve víceprocesorových systémech

Víceprocesorové systémy jsou ve světě Linuxu poměrně zřídkavé, bylo však vynaloženo dost úsilí při budování Linuxu jako operačního systému typu SMP (Symmetric Multi-Processing). Symetrický multiprocesorový systém je takový, který je schopen stejnoměrně rozdělovat práci mezi všechny procesory v systému. Toto vyvažování je nejzřetelnější právě v plánovači.

Ve víceprocesorovém systému doufáme, že každý procesor provádí nějaký proces. Každý z procesorů spouští plánovač separátně podle toho, kdy jeho proces vyčerpá své časové kvantum nebo když musí čekat na systémové prostředky. První zajímavá věc v SMP systému je ta, že v něm není pouze jediný nečinný proces. V jednoprocesorovém systému je nečinným procesem první úloha ve vektoru `task`, v SMP systému existuje pro každý procesor jeden

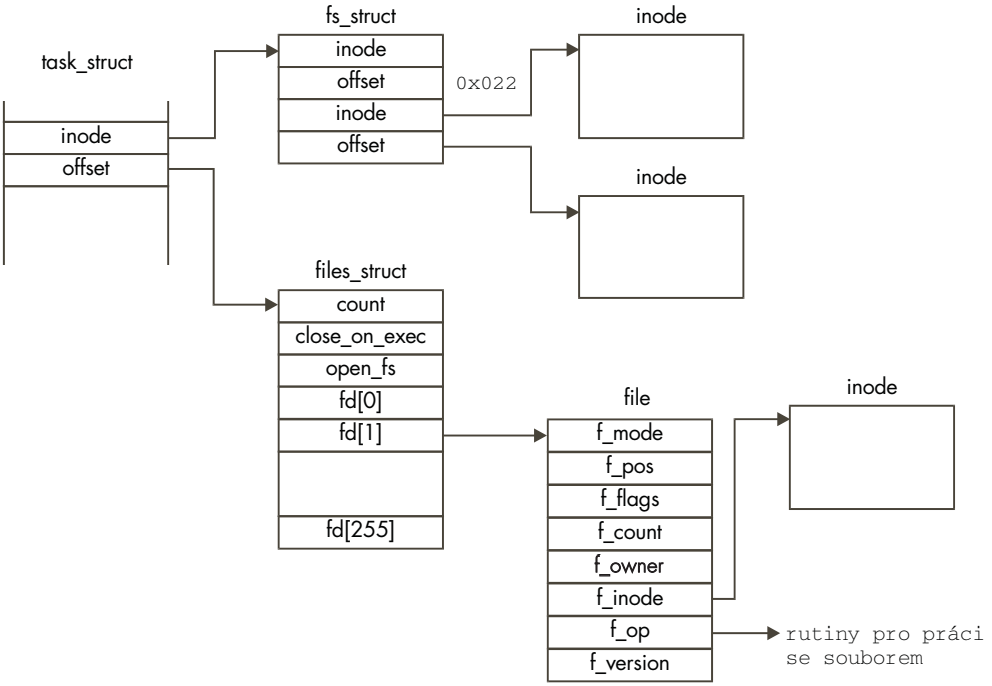
nečinný proces, navíc můžete mít několik nečinných procesorů. Navíc existuje jeden aktuální proces pro každý procesor, takže SMP systém musí vést záznamy o aktuálních a nečinných procesech pro každý procesor.

V SMP systému obsahuje struktura `task_struct` každého procesu číslo procesoru, na němž proces právě běží (`processor`), a rovněž číslo procesoru, na němž běžel naposledy (`last_processor`). Neexistuje žádný důvod, proč by proces nemohl při každém naplánování běžet na jiném procesoru, Linux však může provádění procesu omezit pouze na jeden nebo několik procesorů v systému pomocí masky `processor_mask`. Pokud je v ní nastaven N-tý bit, může proces běžet na N-tém procesoru. Když plánovač rozhoduje o naplánování nového procesu pro určitý procesor, nebude brát v úvahu ty, které nemají pro tento procesor nastaven příslušný bit v masce `processor_mask`. Plánovač navíc vždy částečně upřednostňuje procesy, které naposledy běžely na stejném procesoru, protože převedení procesu na jiný procesor s sebou nese jistou režii navíc.

4.4 Soubory

Na obrázku 4.1 vidíme, že v systému jsou pro každý proces dvě datové struktury, které popisují procesově specifické informace souborového systému. První z nich, struktura `fs_struct`, obsahuje ukazatele na VFS inody procesu a jeho hodnotu `umask`. Hodnota `umask` je implicitní hodnota pro vytváření nových souborů procesem a je možno ji změnit pomocí systémových volání.

Druhá datová struktura, struktura `files_struct`, obsahuje informace o všech souborech, které proces momentálně používá. Program čte ze *standardního vstupu* a zapisuje na *standardní výstup*. Všechna chybová hlášení se posílají na *standardní chybový výstup*. Fyzicky to mohou být soubory, terminálové linky nebo reálná zařízení, z pohledu programu jsou to ale vždy soubory. Každý soubor má svůj vlastní deskriptor a struktura `files_struct` obsahuje ukazatele na maximálně 256 datových struktur `file`, z nichž každá popisuje jeden soubor používaný procesem. Položka `f_mode` obsahuje režim vytvoření souboru: pro čtení, pro zápis i čtení nebo pouze pro zápis. `f_pos` obsahuje pozici v souboru, kde se provede následující operace čtení nebo zápisu. `f_inode` ukazuje na inode souboru a `f_ops` je ukazatel na vektor adres rutin, jedna pro každou funkci, kterou je možno se souborem provádět. Může to být například funkce pro zápis dat. Tato abstraktnost rozhraní je velmi mocná a umožňuje Linuxu podporovat široké spektrum různých typů souborů. Například roury se v Linuxu podporují, jak uvidíme později, právě tímto mechanismem.



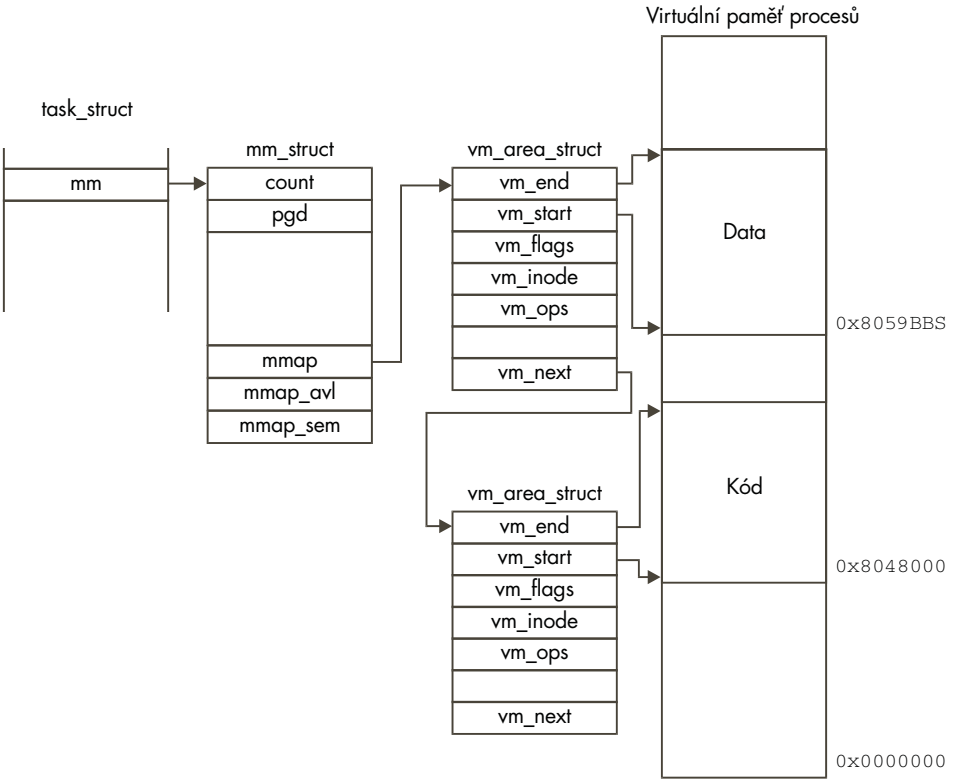
Obrázek 4.1
Soubory procesu

Vždy při otevření souboru se jeden z volných ukazatelů struktury `files_struct` použije jako ukazatel na novou strukturu `file`. Procesy v Linuxu očekávají při svém spuštění tři otevřené deskriptory souborů. Označují se jako *standardní vstup*, *standardní výstup* a *chybový výstup* a obvykle se dědí z vytvářejícího rodičovského procesu. Všechny přístupy k souborům se provádějí pomocí standardních systémových volání, která přebírají nebo vracejí deskriptory souborů. Tyto deskriptory jsou indexy do vektoru `fd` procesu, takže standardní vstup a výstup a chybový výstup mají deskriptory 0, 1 a 2. Všechny přístupy k souborům používají ke splnění svých požadavků operační rutiny datové struktury `file` spolu s inody VFS.

4.5 Virtuální paměť

Virtuální paměť procesu obsahuje spustitelný kód a data z mnoha zdrojů. Prvním zdrojem je nahraný obraz spustitelného programu, řekněme příkaz jako `ls`. Tento příkaz se, stejně jako všechny spustitelné obrazy, skládá jak ze spustitelného kódu, tak i z dat. Obraz obsahuje všechny informace potřebné k zavedení spustitelného kódu a s ním souvisejících dat do virtuální paměti procesu. Proces dále může alokovat (virtuální) paměť pro potřeby své práce, řek-

něme k uložení obsahu souborů, které načítá. Tato nově alokovaná virtuální paměť se musí navázat do stávající virtuální paměti procesu tak, aby ji bylo možno využít. Dále procesy používají knihovny obecně užitečných rutin, například rutin pro manipulaci se soubory. Nemá smysl, aby měl každý proces svou vlastní kopii knihovny, a proto Linux používá sdílené knihovny, které může používat více běžících procesů najednou. Kód a data těchto sdílených knihoven musí být rovněž přítomny ve virtuální paměti procesu a také všech ostatních procesů, které knihovnu používají.



Obrázek 4.2
Virtuální paměť procesu

V určitém časovém úseku proces nemusí používat veškerý kód a data obsažená ve své virtuální paměti. Může obsahovat části kódu, které se využívají jenom v určitých situacích, například v době inicializace nebo při zpracování určité události. Může využívat pouze některé rutiny ze sdílených knihoven. Bylo by plýtváním nahrávat do fyzické paměti veškerý kód a data, aby tam potom ležely nevyužity. Vynásobme takovéto plýtvání počtem procesů v systému a měli bychom systém, který by pracoval velmi neefektivně. Proto Linux používá techniku zvanou *stránkování na žádost*, kdy se virtuální paměť procesu zavádí do fyzické paměti pou-

ze v okamžiku, kdy ji proces potřebuje. Namísto přímého nahrání veškerého kódu a dat do paměti tedy Linux pouze modifikuje tabulku stránek procesu tak, že jednotlivé oblasti virtuální paměti označí jako používané, ale v paměti neexistující. Když se pak proces pokusí o přístup k těmto částem kódu či dat, hardware systému detekuje výpadek stránky a předá řízení jádru Linuxu, které situaci napraví. Proto musí Linux pro každou oblast virtuální paměti v adresovém prostoru procesu vědět, odkud kód pochází a jak jej do paměti dostat, aby mohl takovéto výpadky stránek ošetřit.

Jádro Linuxu tedy potřebuje spravovat všechny tyto oblasti virtuální paměti. Obsah virtuální paměti každého procesu je popsán datovou strukturou `mm_struct`, na kterou se ukazuje ze struktury `task_struct` procesu. Datová struktura `mm_struct` každého procesu obsahuje také informace o nahraném spustitelném obrazu a ukazatel na tabulku stránek procesu. Dále obsahuje ukazatele na seznam datových struktur `vm_area_struct`, z nichž každá reprezentuje jednu oblast virtuální paměti procesu.

Tento seznam je uspořádán vzestupně podle pořadí oblastí ve virtuální paměti. Na obrázku 4.2 vidíme rozvržení virtuální paměti jednoduchého procesu včetně datových struktur jádra, které virtuální paměť spravují. Protože jednotlivé oblasti virtuální paměti pocházejí z různých zdrojů, používá Linux abstraktní rozhraní, kdy struktura `vm_area_struct` obsahuje sadu ukazatelů (`vm_ops`) na rutiny obsluhy virtuální paměti. Díky tomu je možno celou virtuální paměť procesu obsluhovat konzistentním způsobem bez ohledu na to, že služby nižší úrovně se mohou pro jednotlivé oblasti lišit. Existuje zde například rutina, která se bude volat v případě, že se proces pokusí o přístup k oblasti paměti, která neexistuje; tímto mechanismem se obsluhují výpadky stránek.

Se seznamem struktur `vm_area_struct` jádro trvale pracuje, když vytváří nové oblasti virtuální paměti procesu nebo když opravuje odkazy na virtuální oblasti, které nejsou přítomny ve fyzické paměti procesu. Díky tomu se doba, strávená hledáním správné struktury `vm_area_struct`, stává kritickou pro celý výkon systému. Aby se přístup zrychlil, organizuje Linux struktury `vm_area_struct` také do takzvaného AVL (Adelson-Velskii a Landis) stromu. Strom je organizován tak, že každá struktura `vm_area_struct` (nebo též *uzel*) obsahuje levý a pravý ukazatel na své sousedy ve stromu. Levý ukazatel ukazuje na uzel s nižší počáteční virtuální adresou, pravý ukazatel ukazuje na uzel s vyšší počáteční virtuální adresou. Při hledání správného uzlu začne Linux od kořene stromu a pohybuje se z každého uzlu buď vlevo, nebo vpravo, až najde správný uzel. Pochopitelně nic není zadarmo a v tomto případě platíme za efektivní hledání vyšší náročností vložení nové struktury `vm_area_struct` do stromu.

Při alokování nové oblasti virtuální paměti Linux neprovádí skutečnou alokaci fyzické paměti. Pouze popíše nově vzniklou virtuální oblast vytvořením nové struktury `vm_area_struct`. Tato struktura se zapojí do seznamu oblastí virtuální paměti procesu. Když se proces pokusí o zá-

pis na virtuální adresu v nově vytvořené oblasti, dojde k výpadku stránky. Procesor se pokusí o dekódování virtuální adresy, protože však pro danou adresu neexistuje platná položka tabulky stránek, generuje výpadek stránky a přenechá řízení jádru systému. Linux se podívá, zda se požadovaná adresa nachází v existující oblasti virtuální paměti procesu. Pokud ano, vytvoří Linux patřičnou položku tabulky stránek a alokuje fyzickou paměťovou stránku. Pak může být nutné přenést stránku do fyzické paměti z diskového souboru nebo z odkládacího souboru. Poté je možno proces znovu spustit na stejné instrukci, která vyvolala výpadek stránky, a protože stránka už nyní existuje, proces může dále pokračovat.

4.6 Vytvoření procesu

Když se systém spustí, běží v režimu jádra a existuje pouze jediný, iniciální proces. Stejně jako všechny ostatní procesy, i iniciální proces má svůj strojový stav (kontext) reprezentovaný hodnotami registrů, zásobníkem a podobně. Když se vytvoří a spustí další procesy, bude tento stav uložen v datové struktuře `task_struct` iniciálního procesu. Na konci inicializace systému spustí iniciální proces vlákno jádra (zvané `init`) a pak nečinně čeká a nic nedělá. Kdykoliv není nic jiného na práci, spustí plánovač tento pozastavený proces. Struktura `task_struct` tohoto procesu jako jediná není alokována dynamicky, je staticky definována přímo v jádře a poněkud matoucně se jmenuje `init_task`.

Vlákno jádra či proces `init` má identifikátor procesu 1, protože se jedná o první faktický proces v systému. Provede nějaké počáteční nastavení systému (například otevření systémové konzoly a připojení kořenového souborového systému) a pak spustí inicializační program systému. Podle konkrétního systému se jedná o program `/etc/init`, `/bin/init` nebo `/sbin/init`. Při vytváření nových procesů v systému používá program `init` jako skriptový soubor `/etc/inittab`. Tyto nové procesy pak mohou samy vytvářet další nové procesy. Například proces `getty` může při pokusu o přihlášení vytvořit proces `login`. Všechny procesy v systému jsou (přímo či nepřímo) potomky procesu `init`.

Nové procesy se vytvářejí klonováním starých procesů, přesněji řečeno klonováním aktuálního procesu. Nová úloha se vytvoří systémovým voláním (*fork* nebo *clone*) a samotné klonování se odehraje v jádře v režimu jádra. Ve fyzické paměti systému se alokuje nová datová struktura `task_struct` a jedna nebo více fyzických stránek pro zásobník klonovaného procesu. Může se vytvořit identifikátor nového procesu, který musí být odlišný od identifikátorů všech existujících procesů. Je však možné, že nově vytvořený proces si ponechá identifikátor svého rodičovského procesu. Do vektoru `task` se vloží nová struktura `task_struct` a obsah struktury `task_struct` starého (aktuálního) procesu se zkopíruje do nové struktury.

Klonováním procesů umožňuje Linux dvěma procesům sdílet prostředky. Týká se to souborů, obsluhy signálů a virtuální paměti. Když se prostředky sdílejí, hodnota jejich počitadla `count` se zvyšuje, takže Linux příslušný prostředek neuvolní do té doby, dokud jej nepřestanou používat oba procesy. Pokud například klonovaný proces sdílí virtuální paměť se svým rodičem, jeho struktura `task_struct` bude obsahovat ukazatel na strukturu `mm_struct` rodičovského procesu a počitadlo `count` v této struktuře bude inkrementováno, aby se ukázalo, kolik procesů strukturu momentálně sdílí.

Klonování virtuální paměti je poněkud více rafinované. Musí se vytvořit nová sada datových struktur `vm_area_struct`, dále na ně ukazující struktura `mm_struct` a tabulka stránek nového procesu. V tomto okamžiku se ještě žádná virtuální paměť nekopíruje. Bylo by to dost obtížné a zdlouhavé vzhledem k tomu, že část virtuální paměti může ležet ve fyzické paměti, část ve spustitelných obrazech na disku a část třeba v odkládacím souboru. Namísto toho používá Linux techniku zvanou „kopírování při zápisu“, což znamená, že ke kopírování virtuální paměti dojde pouze v případě, že jeden z procesů do ní bude zapisovat. Virtuální paměť, do níž se nezapisuje (i když by k tomu mohlo dojít), se může mezi oběma procesy sdílet bez jakéhokoliv nebezpečí. Aby technika „kopírování při zápisu“ fungovala, jsou v tabulce stránek sdílené zapisovatelné stránky označeny jako „pouze pro čtení“ a ve struktuře `vm_area_struct` je uvedeno, že se jedná o stránce chráněné kopírováním při zápisu. Pokud se jeden z procesů pokusí o zápis do této oblasti virtuální paměti, dojde k výpadku stránky. Teprve v této fázi Linux vytvoří kopii paměti a upraví tabulky stránek a datové struktury virtuální paměti obou procesů.

4.7 Časy a časovače

Jádro si pamatuje čas vytvoření procesu a také objem času procesu, který proces po dobu svého života spotřeboval. S každým tikem hodin aktualizuje jádro čas, který proces existuje, a čas, který strávil v uživatelském režimu. Tyto časy se měří v jednotkách zvaných `jiffies`.

Kromě těchto účetních časovačů podporuje Linux ještě procesově závislé *intervalové* časovače.

Proces si může pomocí těchto časovačů nechat posílat různé signály. Linux podporuje tři typy časovačů:

reálné Tyto časovače běží v reálném čase a když doběhnou, proces dostane signál `SIGALARM`.

virtuální Tyto časovače běží pouze v době, kdy běží proces, a když doběhnou, dostane proces signál `SIGVTALARM`.

profilové Tyto časovače běží, pokud běží samotný proces nebo pokud systém jménem procesu vykonává nějakou činnost. Když časovače doběhnou, proces obdrží signál `SIGPROF`.

V dané chvíli může běžet jeden nebo více časovačů a Linux má všechny potřebné informace o časovačích uloženy v datové struktuře `task_struct` procesu. Pomocí různých systémových volání je možno časovače nastavovat, spouštět, zastavovat a přečíst jejich momentální hodnoty. Virtuální a profilové časovače se obsluhují stejným způsobem. 7

S každým tikem hodin se dekrementují časovače procesu a když doběhnou, pošle se příslušný signál.

Reálné časovače fungují poněkud jinak a při jejich obsluze používá Linux časovací mechanismy popsané v kapitole Mechanismy jádra. Každý proces má svou vlastní datovou strukturu `timer_list` a pokud používá reálné časovače, přidávají se do fronty systémových časovačů. Když časovač doběhne, odstraní jej bottom-half handler z fronty a zavolá obsluhu intervalového časovače. 8

Tím se generuje signál `SIGALARM`, časovač se restartuje a znovu se přidává do fronty systémových časovačů. 9

4.8 Spouštění programů

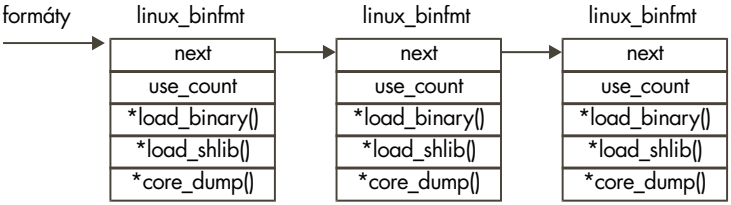
V Linuxu se, stejně jako v Unixu, programy a příkazy normálně spouštějí pomocí interpretu příkazů. Příkazový interpret je uživatelský proces jako každý jiný a označuje se termínem *shell* [3].

V Linuxu existuje mnoho příkazových interpretů, nejpopulárnějšími jsou `sh`, `bash` a `tcsh`. S výjimkou několika vestavěných příkazů jako `cd` nebo `pwd` jsou ostatní příkazy spustitelné binární soubory. Při zadání příkazu prohledává příkazový interpret adresáře ve *vyhledávací cestě* procesu, uložené v proměnné prostředí `PATH`, a hledá spustitelný obraz zadaného jména. Pokud soubor najde, nahraje jej a spustí. Příkazový interpret vytvoří svůj klon pomocí výše popsaného mechanismu *fork* a nový proces nahradí binární obraz příkazového interpretu obsahem právě nahraného spustitelného obrazu. Za normálních okolností příkazový interpret čeká na dokončení příkazu, přesněji řečeno na skončení synovského procesu. Příkazový interpret můžete znovu vyvolat tím, že synovský proces přesunete do pozadí stiskem `CONTROL-Z`, čímž se synovskému procesu pošle signál `SIGSTOP` a proces se zastaví. Příkazem `bg` pří-

[3] Představme si jádro jako základ a kolem něj je obalena ulita (shell), která představuje uživatelské rozhraní.

kazového interpretu pak můžete přesunout do pozadí příkazový interpret, který synovskému procesu pošle signál `SIGCONT` a tak obnoví jeho běh. Příkazový interpret pak zůstává v pozadí, dokud synovský proces neskončí nebo dokud nepotřebuje terminálový vstup či výstup.

Spustitelný soubor může mít mnoho formátů nebo se může jednat o skript. Skriptové soubory je nutné rozeznat a nechat je provést příslušným interpretem, například skripty příkazového interpretu se provádějí programem `/bin/sh`. Spustitelné objektové soubory obsahují spustitelný kód a data doplněná o další informace, které umožňují operačnímu systému nahrát soubor do paměti a spustit jej. Nejběžnějším formátem spustitelného souboru v Linuxu je formát ELF, teoreticky je však Linux natolik pružný, že může obsloužit prakticky jakýkoliv objektový formát.



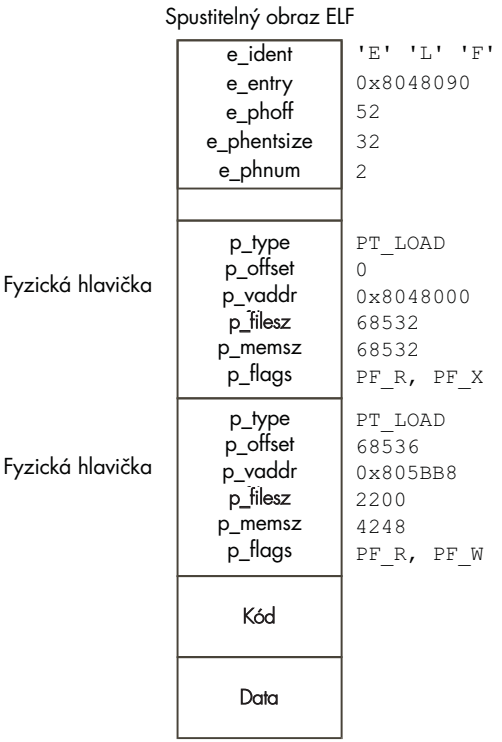
Obrázek 4.3
Registrované binární formáty

Stejně jako u souborových systémů jsou i binární formáty buď součástí jádra Linuxu nebo se dají dohrát jako moduly. Jádro udržuje seznam podporovaných binárních formátů (viz obrázek 4.3) a když dojde k pokusu o spuštění souboru, vyzkouší se každý binární formát dokud některý z nich nebude fungovat.

Linux běžně podporuje formáty `a.out` a `ELF`. Spustitelné soubory nemusejí být nahrány v paměti celé, používá se metoda zvaná vynucené nahrávání. Jednotlivé části spustitelného obrazu se zavádějí do paměti podle toho, jak je proces potřebuje. Nepoužívané části je možno z paměti uvolnit.

4.8.1 ELF

Objektový souborový formát ELF (Executable and Linkable Format) navržený v Unix System Laboratories je dnes zaveden jako nejčastěji používaný formát v Linuxu. Přestože v porovnání s jinými objektovými formáty jako `ESCOFF` a `a.out` má tento formát poněkud vyšší výkonnostní režii, je daleko pružnější. Spustitelné soubory ve formátu ELF obsahují spustitelný kód, někdy označovaný jako *text*, a *data*. Tabulky ve spustitelném obraze říkají, jak má být proces umístěn do virtuální paměti procesu. Staticky linkované obrazy jsou sestaveny linkerem (`ld`) do jediného obrazu, který obsahuje veškerý kód a data potřebná ke spuštění tohoto obrazu. Obraz dále specifikuje své rozvržení v paměti a adresu, od níž se má kód začít vykonávat.



Obrázek 4.4
Souborový formát ELF.

11 Na obrázku 4.4 vidíme uspořádání staticky linkovaného spustitelného obrazu ve formátu ELF.

Jedná se o jednoduchý program v jazyce C, který vytiskne text „hello world“ a ukončí se. Hlavička souboru říká, že se jedná o obraz ELF se dvěma fyzickými hlavičkami (hodnota `e_phnum` je 2), které začínají 52 bajtů (`e_phoff`) od počátku souboru. První fyzická hlavička popisuje spustitelný kód obrazu. Patří na virtuální adresu *0x8048000* a je celkem 65 532 bajtů dlouhý. Velikost je dána tím, že jde o staticky linkovaný obraz, který obsahuje celý knihovní kód funkce `printf()`. Vstupní bod obrazu, tedy adresa první instrukce programu, není na počátku obrazu, ale na virtuální adrese *0x8048090* (položka `e_entry`). Kód začíná ihned za druhou fyzickou hlavičkou. Tato druhá hlavička popisuje data programu a nahrává se do virtuální paměti na adresu *0x8059BB8*. Data je možno číst i zapisovat. Můžete si všimnout, že velikost dat je 2 200 bajtů (`p_filesz`), zatímco velikost paměti dat je 4 248 bajtů. Je to dáno tím, že prvních 2 200 bajtů obsahuje předinicializovaná data, zatímco zbývajících 2 048 bajtů obsahuje data, která se budou inicializovat až při běhu kódu.

Když Linux nahrává spustitelný obraz ve formátu ELF do virtuálního paměťového prostoru procesu, neprovádí skutečné nahrávání obrazu.

Nastaví datové struktury virtuální paměti, strom struktur `vm_area_struct` a tabulky stránek. Když se program začne provádět, dojde k výpadku stránky, což způsobí, že se kód a data programu načtou do fyzické paměti. Nepoužité části programu se do paměti nikdy nenahrají. Jakmile zavaděč binárního formátu ELF zjistí, že nahrávaný obraz je platným obrazem ve formátu ELF, odstraní z virtuální paměti procesu obraz stávajícího prováděného programu. Protože proces je klonem obrazu (všechny procesy jsou klonem), starý obraz je obraz programu prováděného rodičovským procesem, například tedy obrazem interpretu příkazů. Odstranění starého spustitelného obrazu zruší staré struktury virtuální paměti a tabulky stránek. Dojde rovněž k vymazání všech handlerů signálů a k zavření všech otevřených souborů. Na konci rušení je proces připraven přijmout nový spustitelný obraz. Bez ohledu na formát spustitelného obrazu se struktura `mm_struct` procesu nastavuje vždy stejnými informacemi. Musí totiž obsahovat ukazatele na začátek a konec kódu a dat obrazu. Tyto hodnoty se načtou z fyzických hlaviček formátu ELF a jimi určené oblasti programu se mapují do virtuálního adresového prostoru procesu. Zde se rovněž nastaví datové struktury `vm_area_struct` a provede se inicializace tabulky stránek procesu. Datová struktura `mm_struct` dále obsahuje ukazatele na parametry předávané programu a na proměnné prostředí procesu.

Sdílené knihovny ve formátu ELF

Dynamicky linkované obrazy neobsahují veškerý kód a data, která ke své činnosti potřebují. Některé části jsou umístěny ve sdílených knihovnách, které se zavádějí až v okamžiku spuštění programu. Tabulky sdílených knihoven formátu ELF dále slouží *dynamickému linkeru* při linkování sdílených knihoven do běžícího programu. Linux používá několik dynamických linkerů, `ld.so.1`, `libc.so.1` a `ld-linux.so.1`, všechny jsou umístěny v adresáři `/lib`. Knihovny obsahují běžně používaný kód, například internacionální podporu. Bez použití dynamického linkování by každý program musel obsahovat vlastní kopii těchto knihoven a bylo by tak zapotřebí podstatně více diskového prostoru a virtuální paměti. Při dynamickém linkování jsou v tabulkách obrazu ELF uloženy informace pro každou knihovní funkci, na níž se program odkazuje. Tyto informace dynamickému linkeru říkají jak nalézt knihovní funkci a jak ji vlinkovat do adresového prostoru programu.

4.8.2 Skriptové soubory

Skriptové soubory jsou spustitelné soubory, které ke svému běhu potřebují interpret. V Linuxu existuje celá řada interpretů, například `wish`, `perl` a příkazové interprety jako `tcsh`. Linux používá standardní konvenci systému Unix, kdy první řádek skriptu obsahuje jméno interpretu. Typický skript bude tedy začínat třeba takto:

```
#!/usr/bin/wish
```

13 Zavaděč skriptu se pokusí nalézt interpret skriptu.

Provádí to tak, že se pokusí otevřít spustitelný soubor, uvedený v prvním řádku skriptu. Pokud se mu jej podaří otevřít, má ukazatel na jeho VFS inode a může přejít k dalšímu kroku a nechat interpret vykonat skriptový soubor. Jméno skriptového souboru je parametrem číslo nula (tedy prvním parametrem) interpretu a všechny ostatní parametry se posouvají o jednu pozici vzad (původně první parametr se stává druhým a tak dále). Nahrání interpretu se provádí stejným mechanismem, jakým Linux nahrává všechny spustitelné soubory. Linux opět zkouší všechny registrované binární formáty dokud nenalezne vyhovující formát. Znamená to, že teoreticky můžete na sebe stavět několik interpretů a binárních formátů. Obsluha spustitelných souborů v Linuxu je tedy velice pružná.

Odkazy na zdrojové texty jádra

1 – Viz include/-linux/sched.h
2 – Viz schedule()in kernel/sched.c
3 – Viz schedule()in kernel/sched.c
4 – Viz include/-linux/sched.h
5 – Viz do_fork() in kernel/fork.c
6 – Viz kernel /itimer.c
7 – Viz do_it_virtual() in kernel/sched.c
8 – Viz do_it_prof() in kernel /sched.c
9 – Viz it_real_fn() in kernel/itimer.c
10 – Viz do_execve() in fs/exec.c
11 – Viz include/-linux/elf.h
12 – Viz do_load_elf_binary() in fs/binfmt_elf.c
13 – Viz do_load_script() in fs/-binfmt_script.c

Kapitola

5

Meziprocesorová komunikace

Při vzájemné koordinaci svých aktivit komunikují procesy navzájem mezi sebou a jádrem. Linux podporuje řadu mechanismů pro meziprocesovou komunikaci (IPC). Dvěma z nich jsou signály a roury, Linux však dále podporuje IPC mechanismy Systemu V, pojmenované podle verze Unixu, kde byly poprvé zavedeny.

5.1 Signály

Signály jsou jednou z nejstarších metod meziprocesové komunikace zavedených v systémech Unix. Slouží k signalizaci asynchronních událostí jednomu nebo více procesům. Signál může být generován přerušením klávesnice nebo chybovými okolnostmi, jako například pokusem o přístup k neexistující oblasti virtuální paměti. Signály jsou rovněž využívány v příkazových interpretech k signalizaci řídících příkazů v synovských procesech.

Je definována jakási množina signálů, které může generovat jádro nebo jiné procesy za předpokladu, že mají dostatečná privilegia. Seznam všech existujících signálů můžete získat pomocí příkazu `kill` (`kill -l`), na mém systému (s procesorem Intel) dostávám následující seznam:

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
```

Pro platformu Alpha se signály mohou lišit. Proces se může rozhodnout, že většinu generovaných signálů bude ignorovat. Existují ale dvě výjimky: signál SIGSTOP, který způsobení pozastavení činnosti procesu, a signál SIGKILL, který způsobí ukončení procesu. Ve všech ostatních případech může proces pro signály zvolit libovolnou obsluhu. Procesy mohou signály zablokovat nebo, pokud je neblokují, mohou zvolit vlastní obsluhu nebo mohou obsluhu signálu ponechat na jádru. Pokud je signál obsluhován jádrem, používá jádro implicitní obslužné mechanismy příslušného signálu. Například implicitní akce při obsluze signálu SIGFPE (výjimka operace v pohyblivé řádové čárce) je vytvoření souboru core a ukončení procesu. Signály nemají žádnou vzájemnou prioritu. Pokud dojde ve stejném okamžiku k vygenerování dvou signálů, mohou se v procesu objevit v jakémkoliv pořadí. Obdobně neexistuje žádný mechanismus pro obsluhu více stejných signálů. Neexistuje způsob jak může proces říci, zda obdržel jeden signál SIGCONT nebo 42 těchto signálů.

Linux implementuje signály pomocí informací uložených ve struktuře task_struct procesu. Počet podporovaných signálů je omezen velikostí slova procesoru. Procesory se slovem o velikosti 32 bitů mohou mít maximálně 32 různých signálů, zatímco 64bitové procesory, jako například Alpha AXP, mohou mít až 64 signálů. Momentálně aktivní signály jsou uloženy v položce signal s maskou blokovaných signálů uloženou v položce blocked. S výjimkou signálů SIGSTOP a SIGKILL je možno blokovat všechny signály. Pokud je generován blokovaný signál, zůstává platný až do doby, než dojde k jeho odblokování. Linux dále udržuje informace o obsluze všech možných signálů, které jsou uloženy v datové struktuře sigaction, na níž ukazuje struktura task_struct. Kromě dalšího obsahuje buď adresu obslužné rutiny signálu, nebo příznak, který Linuxu říká, zda si proces přeje signál ignorovat, nebo zda jej má obsluhovat jádro. Proces může implicitně nastavenou obsluhu signálů modifikovat pomocí různých systémových volání, která následně modifikují strukturu sigaction a také hodnotu masky blocked.

Ne každý proces v systému může posílat signály všem ostatním procesům, to může pouze jádro a superuživatel. Normální procesy mohou posílat signály pouze procesům se stejným *uid* a *gid* nebo procesům ve stejné skupině procesů. Signály jsou generovány nastavením příslušného bitu v položce signal struktury task_struct. Pokud proces signál nezablokoval a čeká v přerušitelném stavu, dojde k jeho probuzení přepnutím stavu na spuštěný a zajištěním, že se bude nacházet ve frontě spouštěných procesů. Díky tomu jej bude plánovač v dalším kole plánovaní považovat za kandidáta na spuštění. Pokud je požadována implicitní obsluha, je Linux schopen obsluhu signálu optimalizovat. Pokud se například objeví signál SIGWINCH (změna fokusu X okna) a požaduje se implicitní obsluha, pak není potřeba udělat vůbec nic.

Signály se procesům nepředávají bezprostředně po jejich výskytu, musejí počkat až do příštího spuštění procesu. Vždy když proces opouští systémové volání, kontrolují se jeho položky signal a blocked a pokud se v nich nachází nějaký neblokovaný signál, je možno jej ny-

ní doručit. Může se to jevit jako velmi nespolehlivá metoda, avšak každý proces v systému trvale používá systémová volání, například při zápisu znaku na terminál. Proces se může v případě potřeby rozhodnout čekat na signál, pak je uveden do pozastaveného, nicméně přerušitelného stavu až do doby, než se signál objeví. Kód obsluhy signálů v Linuxu pak pro každý neblokovaný přítomný signál vyhodnocuje údaje ve struktuře `sigaction`.

Pokud je obsluha signálu nastavena na implicitní obsluhu, obslouží signál jádro. Implicitní obsluha signálu `SIGSTOP` převede proces do zastaveného stavu a nechá plánovač spustit další proces. Implicitní obsluha signálu `SIGFPE` provede výpis jádra a ukončí proces. Proces však může zvolit vlastní obsluhu signálu. Jedná se o rutinu, jejíž adresa je uložena ve struktuře `sigaction` a která se bude volat v okamžiku výskytu signálu. Jádro musí zavolat obslužnou rutinu procesu, což je akce závislá na konkrétní platformě, systém se ale vždy musí vyrovnávat s faktem, že proces se momentálně nachází v režimu jádra a následujícím krokem je návrat do uživatelského režimu, z nějž byla volána nějaká systémová rutina nebo rutina jádra. Tento problém se řeší pomocí manipulace se zásobníkem a registry procesu. Ukazatel instrukcí programu se nastaví na adresu rutiny obsluhy signálu a parametry předávané rutině se uloží na zásobník nebo zapíší do příslušných registrů. Když proces obnoví svou činnost, vypadá to, jako kdyby byla obslužná rutina signálu volána běžnými mechanismy.

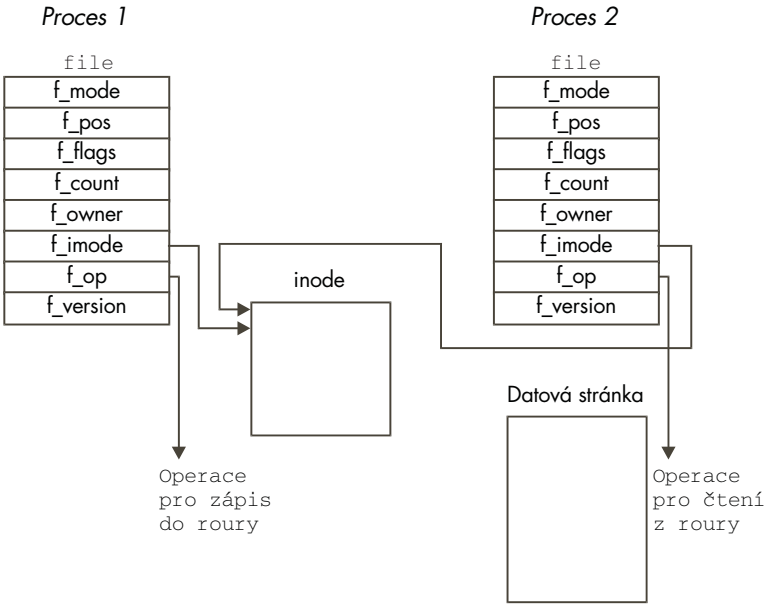
Linux je kompatibilní s normou POSIX, takže procesy mohou při volání obslužné rutiny signálu měnit blokování signálů. Obnáší to změnu masky `blocked` při volání obsluhy signálu. Maska `blocked` musí být po skončení obslužné rutiny nastavena zpět na původní hodnotu. Linux proto na zásobník přidává adresu úklidové rutiny, která zajistí obnovu původní hodnota masky. Linux rovněž provádí optimalizaci v případě, kdy se musí volat více obslužných rutin, a to tak, že jejich adresy najednou naskládá na zásobník, takže když jedna obsluha skončí, vrací se přímo do další obslužné rutiny a až poslední skáče do úklidové rutiny.

5.2 Roury

Všechny klasické příkazové interprety Linuxu podporují přesměrování. Například:

```
$ ls | pr | lpr
```

Tento příkaz předává rourou výpis obsahu adresáře pořízený příkazem **`ls`** na standardní vstup příkazu **`pr`**, který jej rozděluje na stránky. Nakonec se standardní výstup příkazu **`pr`** předává na standardní vstup příkazu **`lpr`**, který výsledek vytiskne na tiskárně. Roury jsou jednosměrné bajtové proudy, které propojují standardní výstup jednoho procesu se standardním vstupem druhého procesu. Žádný z procesů si tohoto přesměrování není vědom a všem funguje jako normálně. Dočasné roury mezi procesy obsluhuje příkazový interpret.



Obrázek 5.1
Roury

1 V Linuxu se roury implementují pomocí dvou datových struktur `file`, které obě ukazují na stejný dočasný inode VFS, který sám o sobě ukazuje na fyzickou stránku v paměti. Na obrázku 5.1 vidíme, že každá ze struktur `file` obsahuje ukazatele na rozdílné vektory souborových rutin; jedna obsahuje ukazatel na operaci zápisu do roury, druhá ukazatel na operaci čtení z roury.

Tím se před obecnými systémovými voláními zajišťujícími čtení a zápis do souborů skrývají implementační detaily. Když první proces zapisuje do roury, kopírují se bajty na sdílenou datovou stránku, když druhý proces čte z roury, bajty se kopírují ze sdílené stránky. Linux musí synchronizovat přístup k rouře. Musí zajistit, aby čtenář i písař roury byli v koordinaci, k zajištění jejich zamykání používá fronty a signály.

2 Když písař zapisuje do roury, používá standardní knihovní funkce zápisu. Všem těmto funkcím se předávají deskriptory, které jsou indexy do množiny datových struktur `file` procesu, každý deskriptor reprezentuje jeden otevřený soubor nebo, jako v tomto případě, otevřenou rouru. Zápisová rutina pak používá k zajištění požadavků na zápis informace uložené v inodu VFS, který reprezentuje danou rouru.

Dokud je dostatek místa k zápisu bajtů do roury a dokud není roura zamčena čtenářem, Linux ji zamyká pro písaře a kopíruje bajty zapisované z adresového prostoru písaře do sdílené datové stránky. Když si rouru zamkne čtenář nebo když v ní není dostatek místa pro data, aktuální proces se uspí a čeká v čekací frontě inodu roury. Plánovač pak naplánuje spuštění jiného procesu. Proces je přerušitelný, takže může přijímat signály a bude čtenářem probuzen až bude dostatek místa pro zápis dat nebo když dojde k odemčení roury. Po zapsání všech dat se inode roury odemkne a bude probuzen čtenář, čekající ve frontě tohoto inodu.

Čtení dat z roury je velmi podobné jejich zápisu. Procesům je povoleno neblokovací čtení (závisí to na režimu otevření souboru či roury) a v takovém případě pokud nejsou žádná data k načtení nebo pokud je roura uzamčená, vzniká chyba. Znamená to, že proces může pokračovat. Druhá možnost je čekat ve frontě inodu roury tak dlouho, dokud zápisová operace neskončí. Když oba procesy skončí s používáním roury, zruší se její inode i sdílená datová stránka.

Linux rovněž podporuje *pojmenované* roury, zvané také FIFO, protože zřetězují operace na principu First In, First Out. První data zapsaná do roury budou také jako první přečtena. Na rozdíl od normálních rour nejsou FIFO dočasné objekty, představují trvalé entity v souborovém systému a vytvářejí se příkazem `mkfifo`. Procesy mohou FIFO používat, pokud k nim mají příslušná přístupová práva. Způsob otevření FIFO se poněkud liší od rour. Roura (tedy její dvě datové struktury `file`, její inode a sdílená datová stránka) se vytvářejí jednorázově za běhu, zatímco FIFO existuje trvale a uživatelé si je otevírají a zavírají. Linux musí ošetřovat situace, kdy čtenář otevře FIFO dříve než písař nebo kdy se čtenář pokouší číst, přestože písař ještě nic nezapsal. Až na tyto výjimky se FIFO obsluhují prakticky úplně stejně jako roury a používají stejné datové struktury a operace.

5.3 Sokety

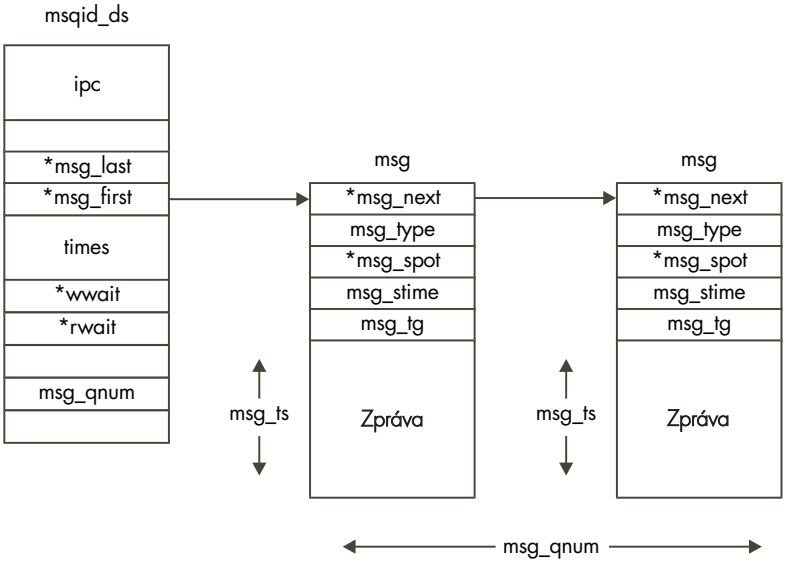
5.3.1 IPC mechanismy Systemu V

Linux podporuje tři mechanismy pro meziprocesovou komunikaci, které se poprvé objevily v Unixu System V v roce 1983. Jedná se o fronty zpráv, semafory a sdílenou paměť. Tyto IPC mechanismy Systemu V všechny používají shodné autentifikační metody. Procesy mohou k těmto prostředkům přistupovat pouze když jádru pomocí systémových volání předají jedinečný referenční identifikátor. Přístup k IPC objektům Systemu V se řídí pomocí přístupových oprávnění velmi podobně, jako se řídí přístup k souborům. Přístupová práva k IPC objektům Systemu V nastavuje tvůrce objektu pomocí systémových volání. Referenční identifikátor objektu používají všechny tyto mechanismy jako index do tabulky prostředků. Nejde ovšem přímo o index, k vygenerování indexu jsou nutné ještě pomocné výpočty.

4 Všechny datové struktury reprezentující IPC objekty Systemu V se v Linuxu ukládají ve struktuře `ipc_perm`, která obsahuje uživatelské a skupinové identifikátory procesů, které objekt vytvořily a vlastní, dále přístupový režim tohoto objektu (pro vlastníka, skupinu a ostatní) a konečně klíč IPC objektu. Tento klíč slouží k určení referenčního identifikátoru IPC objektu. Podporují se dva typy klíčů: veřejný a privátní. Pokud je klíč veřejný, může referenční identifikátor objektu zjistit kterýkoliv proces v systému, pokud k tomu má dostatečná práva. S IPC objekty se nikdy nemanipuluje pomocí klíče, vždy pouze prostřednictvím referenčního identifikátoru.

5.3.2 Fronty zpráv

Fronty zpráv umožňují jednomu nebo více procesům posílat zprávy, které jeden nebo více procesů bude číst. Linux udržuje seznam front zpráv, vektor `msgque`, jehož každý prvek ukazuje na jednu strukturu `msqid_ds`, jež plně popisuje jednu frontu zpráv. Když se vytváří fronta zpráv, alokuje se v systémové paměti nová datová struktura `msgid_ds` a přidá se do vektoru.



Obrázek 5.2
Fronty zpráv

5 Každá datová struktura `msgid_ds` obsahuje datovou strukturu `pic_perm` a ukazatele na zprávy v této frontě. Kromě toho Linux ukládá časy modifikace fronty jako například čas posledního zápisu do fronty a podobně. Struktura `msgid_ds` obsahuje dále dvě čekací fronty, jednu pro písaře fronty zpráv a druhou pro její čtenáře.

Vždy když se proces pokusí o zápis zprávy do fronty, porovnají se jeho efektivní uživatelské a skupinové ID s hodnotami v datové struktuře `ipc_perm` fronty. Pokud proces má právo zápisu do fronty, může se zpráva zkopírovat z adresového prostoru procesu do datové struktury `msg`, která se přidá na konec fronty zpráv. Může se nicméně stát, že pro zprávu nebude dostatek volného místa, protože Linux omezuje počet a délku zpráv, které je možno zapsat. V takovém případě se proces umístí do čekací fronty písařů fronty a plánovač naplánuje spuštění dalšího procesu. K probuzení písaře dojde po přečtení jedné nebo více zpráv z fronty zpráv.

Čtení z fronty je podobný proces. Nejprve se opět kontrolují přístupová práva k frontě. Čtenář si může zvolit přečtení první zprávy ve frontě bez ohledu na její typ, nebo může vybrat zprávu určitého typu. Pokud zadaným kritériím nevyhovuje žádná zpráva, čtenář bude přesunut do čekací fronty čtenářů a plánovač spustí jiný proces. Když dojde k zápisu nové zprávy do fronty, proces bude probuzen a znovu spuštěn.

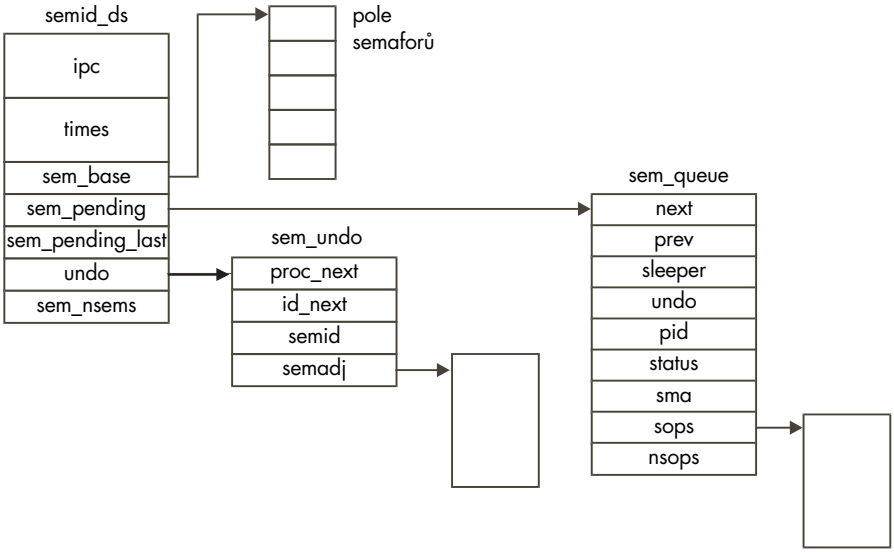
5.3.3 Semafory

Ve své nejjednodušší podobě je semafor místo v paměti, které může více procesů testovat a nastavovat. Operace testování a nastavování je z hlediska procesu nepřerušitelná, atomická - jakmile je jednou vyvolána, nic ji nemůže zastavit. Výsledkem operace testování a nastavení je součet aktuální hodnoty semaforu a nastavované hodnoty, která může být kladná nebo záporná. Podle výsledku operace může být proces uspán dokud hodnota semaforu nebude změněna jiným procesem. Semafory se dají použít k implementaci *kritických sekcí*, kritických oblastí kódu, které může v jednom okamžiku vykonávat pouze jediný proces.

Řekněme, že máte mnoho spolupracujících procesů, které všechny čtou a zapisují záznamy v jednom datovém souboru. Budete zřejmě potřebovat přísně omezit přístup k souboru. Můžete použít semafor s počáteční hodnotou jedna a kolem kódu pro operaci se souborem umístit dvě semaforové operace: první bude testovat a dekrementovat hodnotu semaforu, druhá ji bude testovat a inkrementovat. První proces, který se pokusí o přístup k souboru, bude dekrementovat semafor a zdaří-li se mu to, bude mít semafor nově hodnotu 0. Tento proces nyní může pokračovat dále a používat datový soubor, pokud se však nyní pokusí o dekrementaci semaforu jiný proces, operace se nezdaří protože nová hodnota semaforu by byla -1. Druhý proces bude pozastaven do doby, než první proces inkrementuje hodnotu semaforu zpět na 1. Nyní je možno pozastavený proces probudit a tentokrát pokus o dekrementaci semaforu uspěje.

Každý objekt semaforu je popsán polem, Linux k tomuto účelu používá datovou strukturu `semid_ds`. Na všechny datové struktury `semid_ds` v systému se odkazuje pole `sema-ry`, vektor ukazatelů. Všechny procesy, které mohou manipulovat s polem určitého objektu semafor, s polem manipulují pomocí systémových volání. Systémové volání může specifikovat mnoho operací, přičemž každá je popsána třemi vstupními hodnotami: indexem semaforu, operační hodnotou a množinou příznaků. Index semaforu je index do pole semaforů, ope-

rační hodnota je číslo, které se přičte k aktuální hodnotě semaforu. Nejprve Linux testuje, zda operace může proběhnout úspěšně. Operace proběhne úspěšně v případě, že po přičtení operační hodnoty k aktuální hodnotě semaforu bude výsledek větší než nula, nebo pokud jsou nulové jak operační hodnota, tak aktuální hodnota. Pokud by operace proběhla neúspěšně, Linux pozastaví proces, ovšem pouze v případě, že nebyl aktivní příznak neblokující operace. Pokud proces bude pozastaven, Linux musí uložit stav požadované operace a poté proces přemístit do čekací fronty. Dosáhne toho vytvořením datové struktury `sem_queue` na zásobníku a jejím naplněním hodnotami předávanými při volání operace. Nová datová struktura `sem_queue` se umístí na konec čekací fronty semaforu (pomocí ukazatelů `sem_pending` a `sem_pending_last`). Aktuální proces se umístí do čekací fronty v datové struktuře `sem_queue` (položka `sleeper`) a volá se plánovač, který spustí další proces.



Obrázek 5.3
Semafory

Pokud by všechny požadované semaforové operace uspěly a proces není nutno pozastavit, Linux pokračuje a provede požadované operace nad požadovanými položkami pole semaforů. Dále musí Linux otestovat, zda některý z čekajících procesů může nyní uspět ve své požadované operaci. Projde všechny položky fronty čekajících procesů (`sem_turn`) a testuje, zda jejich operace tentokrát uspěje. Pokud ano, odstraní datovou strukturu `sem_queue` z fronty čekajících procesů a provede nad polem semaforů požadovanou operaci. Probudí čekající proces, takže jej bude možno spustit při příštím chodu plánovače. Linux zopakuje průchod frontou čekajících procesů znovu od začátku a hledá, zda není možno obsloužit další požadavek až do chvíle, kdy už není možno žádnou operaci provést a není možno probudit žádný proces.

U semaforu se objevuje nebezpečí *zablokování - deadlock*. Dojde k tomu, pokud by nějaký proces při vstupu do kritické sekce změnil stav semaforu, avšak poté by kritickou sekci korektně neopustil například protože havaruje nebo je explicitně zrušen. Linux se proti zablokování chrání udržováním seznamu změn pole semaforů. Smyslem je, aby se podle seznamu změn dal semafor uvést do stejného stavu, v jakém byl před provedením operací požadovaných zaniklým procesem. Změny se ukládají v datových strukturách `sem_undo` ukládaných ve frontě ve strukturách `semid_ds` i `task_struct` těch procesů, které používají semafory.

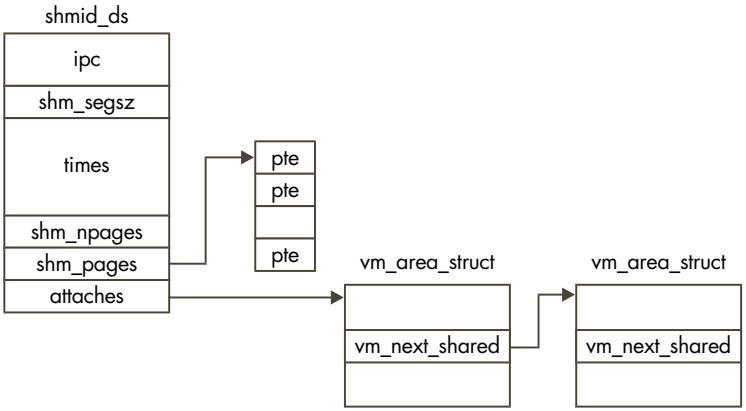
Každá jednotlivá semaforová operace může žádat o zaznamenání změny. Linux pro každý proces a každé semaforové pole spravuje minimálně jednu strukturu `sem_undo`. Pokud proces žádnou nemá, Linux ji v okamžiku potřeby vytvoří. Nová struktura `sem_undo` se zařadí do front jak ve struktuře `task_struct` procesu, tak ve struktuře `semid_ds` semaforu. Operace prováděné nad polem semaforů se negují a zapisují do příslušných položek pole změn v této struktuře `sem_undo`. Pokud je tedy operační hodnota požadavku 2, do pole změn se přičte hodnota -2.

Když dojde k zániku procesu, Linux při jeho ukončení prochází datové struktury `sem_undo` a provádí změny na příslušných semaforových polích. Pokud daný semafor neexistuje, zůstává struktura `sem_undo` zařazena ve frontě struktury `task_struct`, identifikátor semaforu však není platný. V takovém případě úklidový kód semaforu datovou strukturu `sem_undo` jednoduše zruší.

5.3.4 Sdílená paměť

Sdílená paměť umožňuje jednomu nebo více procesům komunikovat prostřednictvím oblasti paměti, která se objevuje ve virtuálním adresovém prostoru každého z nich. Na stránky virtuální paměti je uveden odkaz prostřednictvím položek v tabulce stránek každého ze sdílejících procesů. Sdílená paměť se u jednotlivých procesů nemusí objevovat na stejném místě virtuální paměti. Stejně jako u všech IPC objektů Systemu V je i přístup k oblastem sdílené paměti řízen pomocí klíčů a kontroly přístupových práv. Jakmile je sdílení paměti povoleno, nijak už se nekontroluje způsob využití sdílené paměti jednotlivými sdílejícími procesy. Ty musí při synchronizaci přístupu do sdílené paměti používat jiné mechanismy, například semafory.

Každá nově vytvořená oblast sdílené paměti je reprezentována datovou strukturou `shmid_ds`. Tyto struktury se ukládají ve vektoru `shm_segs`. Datová struktura `shmid_ds` popisuje jak velká je oblast sdílené paměti, kolik procesů ji používá a jak se sdílená paměť mapuje do jejich adresových prostorů. Přístupová práva k paměti a typ klíče je určen tím, kdo sdílenou paměť vytvořil. Pokud má tvůrce sdílené oblasti paměti dostatečná práva, může dokonce nařídit trvalé uzamčení sdílené oblasti ve fyzické paměti.



Obrázek 5.4
Sdílená paměť

Každý proces, který si přeje paměť sdílet, se k ní musí připojit pomocí systémového volání. To vytvoří novou datovou strukturu `vm_area_struct` popisující sdílenou paměť v tomto procesu. Proces může sám rozhodnout, kam v jeho adresovém prostoru se má sdílená paměť mapovat, nebo může toto rozhodnutí ponechat na operačním systému. Nová struktura `vm_area_struct` se připojí do seznamu těchto struktur, na něž ukazuje struktura `shmid_ds`. K propojení struktur týkajících se oblastí sdílené paměti slouží ukazatele `vm_next_shared` a `vm_prev_shared`. V průběhu připojení se virtuální paměť fakticky nevytváří, k tomu dojde až v okamžiku prvního přístupu ke sdílené oblasti.

Když se proces poprvé pokusí o přístup do některé ze stránek sdílené paměti, dojde k výpadku stránky. Při obsluze výpadku Linux nejprve hledá strukturu `vm_area_struct`, která popisuje vypadnuvší stránku. Ta obsahuje ukazatele na obslužné rutiny příslušného typu virtuální paměti. Obslužný kód výpadku sdílené stránky hledá v položkách tabulky stránek strukturu `shmid_ds` a zjišťuje, zda pro danou stránku struktura existuje. Pokud neexistuje, provede alokaci fyzické stránky a vytvoří pro ni položku v tabulce stránek. Kromě tabulky stránek aktuálního procesu se položka umístí i ve struktuře `shmid_ds`. Znamená to, že když se o přístup ke stejné oblasti sdílené paměti pokusí další proces a dojde k výpadku stránky, použije obslužný kód výpadku pro tento proces stejnou již vytvořenou fyzickou stránku. Tedy první proces přistupující ke sdílené oblasti paměti způsobí vytvoření sdílené paměti a u všech dalších procesů už dochází pouze k namapování fyzicky existující sdílené paměti do jejich virtuálního adresového prostoru.

Jakmile proces nebude sdílenou paměť dále potřebovat, odpojí se od ní. Dokud existují jiné procesy, které stejnou sdílenou oblast ještě využívají, je odpojení záležitostí pouze aktuálního procesu. Z datové struktury `shmid_ds` se odstraní jeho položka `vm_area_struct` a zruší se. Aktualizuje se tabulka stránek aktuálního procesu a oblast paměti po-

užívaná pro sdílení se označí za neplatnou. Když se od sdílené paměti odpojí poslední proces, dojde k uvolnění stránek sdílené paměti z fyzické paměti a zruší se také struktura `shmid_ds` dané oblasti sdílené paměti.

Další komplikace při práci se sdílenou pamětí se objevují v okamžiku, pokud stránky sdílené paměti nejsou uzamčeny ve fyzické paměti. V takovém případě může dojít k odložení sdílené paměti na disk v době, kdy je nedostatek fyzické paměti. Mechanismus odkládání sdílených paměťových stránek je popsán v kapitole Správa paměti.

Odkazy na zdrojové texty jádra

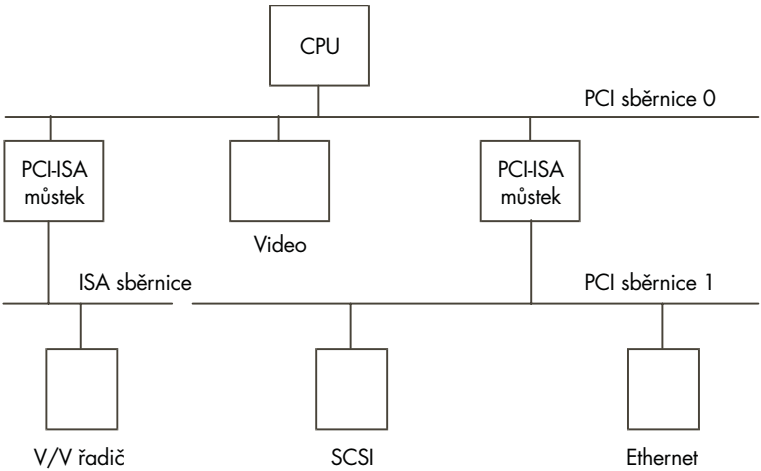
1 – `Viz include/linux/-inode_fs_i.h`
2 – `Viz pipe_write()in fs/pipe.c`
3 – `Viz pipe_read()in fs/pipe.c`
4 – `Viz include/-linux/ipc.h`
5 – `Viz include/-linux/msg.h`
6 – `Viz include/-linux/sem.h`
7 – `Viz include/-linux/sem.h`

Kapitola

6

PCI

Peripheral Component Interconnect (PCI)* je, jak už jeho jméno napovídá, standard popisují způsob propojování periferních zařízení počítačového systému strukturovaným a řízeným způsobem. Standard popisuje jednak způsoby elektrického připojení jednotlivých komponent a také způsoby, jakými se mají zařízení chovat. V této kapitole hovoříme o metodách, jimiž jádro Linuxu inicializuje sběrnice a zařízení PCI.



Obrázek 6.1
Příklad PCI systému

* Poznámka korektora: Celá kniha je zaměřena na jádra verze 2.0.x, v jádrech 2.1.x je celý systém PCI díky Martinu Marešovi kompletně přeprogramován.

Na obrázku 6.1 vidíme logický diagram systému PCI. sběrnice PCI a můstky PCI-PCI představují pojivo, které vzájemně propojuje komponenty systému. Procesor je připojen na sběrnici PCI 0, primární sběrnici, PCI stejně jako videokarta. Speciální zařízení PCI, můstek PCI-PCI, propojuje primární sběrnici PCI se sekundární sběrnicí PCI, sběrnicí PCI 1. V terminologii specifikace PCI se sběrnice PCI 1 označuje jako *downstream* můstku PCI-PCI, sběrnice 0 se označuje jako *upstream* můstku. K sekundární sběrnici PCI jsou připojeny SCSI řadič a ethernetová karta. Fyzicky mohou být jak můstek, sekundární sběrnice i obě zařízení součástí jedné karty PCI. můstek PCI-ISA slouží pro podporu starších zařízení ISA, na obrázku máme jako příklad zařízení ISA uveden kombinovaný řadič, sloužící k připojení řekněme myši a diskety.

6.1 Adresové prostory PCI

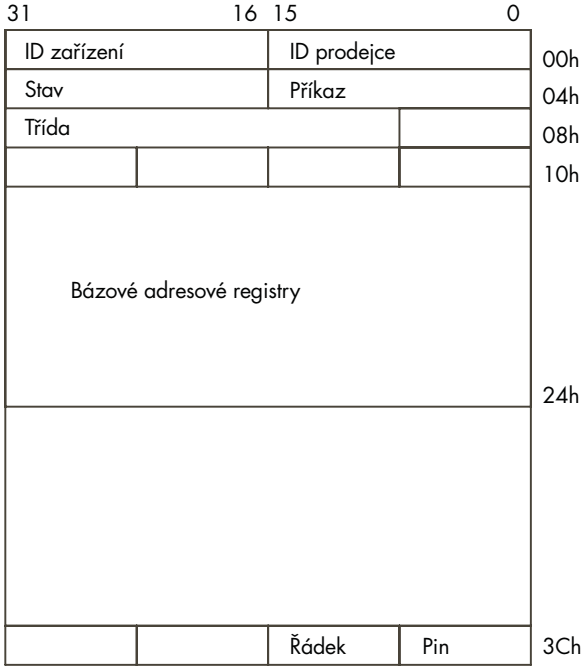
Procesor a zařízení PCI potřebují přístup ke vzájemně sdílené oblasti paměti. Tuto paměť používají ovladače zařízení k řízení zařízení PCI a k výměně informací mezi sebou. Typicky sdílená paměť obsahuje řídicí a stavové registry zařízení. Tyto registry slouží k řízení zařízení a ke čtení jeho stavu. Například ovladač PCI řadiče SCSI bude číst obsah stavového registru při zjišťování, zda je zařízení SCSI připraveno zapsat blok informací na disk SCSI. Zápisem do řídícího registru může provést aktivaci zařízení po jeho zapnutí.

Jako sdílenou paměť by bylo možno využít systémovou paměť procesoru, pokud by tomu ale tak bylo, vždy když by zařízení přistupovalo k paměti, musel by být procesor pozastaven a čekat na dokončení přístupu zařízením. Přístup k paměti je obecně omezen vždy jen na jednu komponentu v systému. Tímto způsobem by tedy došlo ke zpomalení systému. Rovněž není příliš vhodné, aby periferní zařízení mohla libovolným způsobem manipulovat s hlavní operační pamětí. Bylo by to velmi nebezpečné, protože nevhodnými manipulacemi by systém mohl být uveden do zcela nestabilního stavu.

Periferní zařízení mají svůj vlastní adresový prostor. Procesor k němu může přistupovat, avšak přístup zařízení k systémové paměti je přísně řízen prostřednictvím kanálů DMA. Zařízení ISA mají přístup ke dvěma adresovým prostorům: ISA V/V a paměti ISA. Zařízení PCI používají tři oblasti: PCI V/V, paměť PCI a konfigurační prostor PCI. Všechny tyto oblasti jsou přístupné také procesoru s tím, že PCI V/V a paměť PCI používají ovladače zařízení, konfigurační prostor PCI je využíván jádrem při inicializaci PCI.

Procesor Alpha AXP neumí přímo přistupovat k jiným adresovým oblastem než je systémový adresový prostor. Používá proto podpůrné čipy, které zajišťují přístup do jiných adresových prostorů, jako je například konfigurační prostor PCI. Používá rozptýlené mapovací schéma, které „ukradne“ část obrovského virtuálního adresového prostoru a mapuje do ní adresový prostor zařízení PCI.

6.2 Konfigurační hlavičky PCI



Obrázek 6.2
Konfigurační hlavička PCI

Každé zařízení PCI v systému včetně můstků PCI-PCI používá konfigurační datovou strukturu, která je umístěna někde v konfiguračním adresovém prostoru PCI. Konfigurační hlavička slouží systému k identifikaci a řízení zařízení. Přesné umístění hlavičky v konfiguračním prostoru záleží na umístění zařízení v topologii PCI. Například videokarta PCI zapojená v určitém slotu PCI na základní desce bude mít konfigurační hlavičku na určitém místě prostoru, zapojíme-li ji do jiného slotu, pak se hlavička objeví v jiném místě konfiguračního prostoru. To ovšem nevadí, protože ať je zařízení nebo můstek umístěno kdekoliv, systém je vždy najde a nakonfiguruje pomocí stavových a konfiguračních registrů v hlavičce.

Typicky bývají systémy navrženy tak, aby každý slot PCI měl svou konfigurační hlavičku na offsetu, který odpovídá pozici slotu na desce. Takže například první slot na desce bude mít svou hlavičku na offsetu 0 konfiguračního prostoru, druhý slot na offsetu 256 (všechny hlavičky mají pevnou délku 256 bajtů) a tak dále. Musí existovat systémově závislý hardwarový mechanismus, který umožní konfiguračnímu kódu prozkoumat všechny možné konfigurační hlavičky sběrnice PCI a zjistit, která zařízení jsou připojena a která ne, a to prostým čtením jednoho pole každé hlavičky (typicky položky *Identifikátor výrobce*) a generováním nějakých

chyb. Standard definuje jedno možné chybové hlášení jako vrácení hodnoty *0xFFFFFFFF* při pokusu o čtení *Identifikátoru výrobce* nebo *Identifikátoru zařízení*, čímž se indikuje nevyužitý slot PCI.

1 Na obrázku 6.2 je znázorněna struktura konfigurační hlavičky. Skládá se z následujících polí:

Identifikátor výrobce Jednoznačné číslo popisující výrobce zařízení PCI. Identifikátor zařízení PCI firmy Digital je *0x1011*, Intel používá identifikátor *0x8086*.

Identifikátor zařízení Jednoznačné číslo popisující samotné zařízení. Například rychlá etherentová karta 21141 firmy Digital má identifikátor *0x0009*.

Status Toto pole obsahuje status zařízení s tím, že význam jednotlivých bitů pole je definován standardem.

Příkaz Zápisem do tohoto pole systém řídí zařízení, například povoluje zařízení přístup do PCI V/V adresového prostoru.

Kód třídy Identifikátor typu zařízení. Pro každý typ zařízení, například video, SCSI a podobně, existuje standardní třída zařízení. Zařízení SCSI mají přidělenu třídu *0x0100*.

Registry bázové adresy Tyto registry slouží k určení a alokaci typu, velikosti a umístění adresového prostoru PCI V/V a paměti PCI, které zařízení může používat.

Přerušovací pin Přerušení od karty se na sběrnici PCI předávají prostřednictvím čtyř fyzických pinů. Standard se označují jako piny A, B, C a D. Položka *přerušovací pin* udává, který z těchto pinů zařízení PCI používá. Obecně je tento údaj pevně dán při výrobě zařízení. Znamená to, že při každém spuštění používá zařízení stejný přerušovací pin. Tato informace slouží subsystému obsluhy přerušení k obsluze přerušení od zařízení.

Přerušovací linka Pole *přerušovací linka* slouží k předání handlu přerušení mezi inicializačním kódem PCI, ovladačem zařízení a subsystémem obsluhy přerušení. Zde zapsaná hodnota nemá pro ovladač zařízení význam, umožňuje však obsluze přerušení správně směrovat přerušení od zařízení PCI na obslužný kód přerušení správného ovladače zařízení v systému Linux. Podrobnosti o obsluze přerušení v Linuxu jsou uvedeny v kapitole „Přerušení a jejich obsluha“.

6.3 Adresy PCI V/V a paměti PCI

Tyto dva adresové prostory slouží zařízením pro komunikaci s jejich ovladačem, který běží na procesoru v jádře Linuxu. Například rychlá ethernetová karta DEC 21141 mapuje své interní registry právě do prostoru PCI V/V. Její ovladač v jádře Linuxu může prostřednictvím čtení a zápisu do těchto registrů kartu řídit. Videokarty typicky používají velké oblasti paměti PCI k ukládání videoinformací.

Dokud nedojde k nastavení systému PCI a není povolen přístup zařízení do těchto adresových prostorů pomocí pole *Příkaz* v konfigurační hlavičce, žádné zařízení nesmí těchto adresových prostorů využívat. Je třeba zdůraznit, že konfigurační prostor PCI je přístupný pouze konfiguračnímu kódu v jádře, ovladače zařízení pak pracují pouze s prostorem V/V a paměťovým prostorem.

6.4 Můstky PCI-ISA

Tyto můstky slouží k podpoře starších zařízení ISA. Překládají přístupy do adresových prostorů PCI na přístup do adresových prostorů zařízení ISA. V dnešní době je většina systémů vybavena několika sloty ISA a několika sloty PCI. Postupem času potřeba podpory starších zařízení poklesne a budou k dispozici už pouze systémy PCI. Umístění jednotlivých zařízení v adresovém prostoru ISA (ISA V/V a ISA paměti) bylo zavedeno v temném dávnověku prvních PC s procesorem Intel 8080. Dokonce i nejmodernější počítač s procesorem Alpha AXP za pět tisíc dolarů má řadič disketových mechanik ISA mapován na stejných adresách ISA jako první počítače IBM PC. Specifikace PCI se s tímto problémem vyrovnává tak, že nižší oblasti adresových prostorů PCI rezervuje pro periferie ISA a používá jednoduchý můstek PCI-ISA, který překládá přístupy do paměti PCI na přístup do odpovídajících oblastí paměti ISA.

6.5 Můstky PCI-PCI

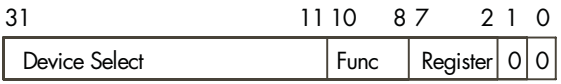
Můstky PCI-PCI jsou speciální zařízení PCI, která sdružují dohromady sběrnice PCI v systému. Jednoduché systémy mají jedinou sběrnici PCI, ovšem počet zařízení, které může jedna sběrnice PCI podporovat, je omezen jejími elektrickými vlastnostmi. Když se pomocí můstů PCI-PCI přidají další sběrnice PCI, může systém podporovat více zařízení PCI. To je důležité zejména u vysoce výkonných serverů. Linux samozřejmě plně podporuje funkci můstků PCI-PCI.

6.5.1 Můstky PCI-PCI: okna PCI V/V a paměti PCI

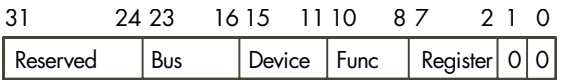
Můstky PCI-PCI předávají směrem dolů pouze vybranou podmnožinu požadavků na PCI V/V a paměťový prostor PCI. Například na obrázku 6.1 bude můstek PCI-PCI předávat ze sběrnice PCI 0 na sběrnici PCI 1 pouze ty požadavky na čtení a zápis, které se vztahují na oblasti V/V a paměti přiřazené buď SCSI nebo ethernetovému zařízení, ostatní požadavky bude ignorovat. Tato filtrace zamezuje zbytečnému šíření adres v systému. Aby mohl můstek takovouto filtrační funkci plnit, musí mít naprogramovány bázovou adresu a limit V/V prostoru a paměťového prostoru, které má předávat ze své primární sběrnice na svou sekundární sběrnici. Jakmile je jednou můstek naprogramován, stává se neviditelným, protože ovladače zařízení přistupují ke svým zařízením pouze prostřednictvím přidělených oken adresových prostorů. To je důležitá funkce, která usnadňuje práci autorům ovladačů zařízení PCI. Na druhé straně se tím ale pro jádro poněkud komplikuje způsob konfigurace můstku, jak uvidíme dále.

6.5.2 Můstky PCI-PCI: konfigurační cykly a číslování PCI sběrnic

Aby mohl inicializační kód PCI adresovat zařízení, která nejsou připojena na hlavní sběrnici PCI, musí existovat mechanismus, který můstkům umožní rozhodnout, zda předat konfigurační cykly z primárního na sekundární rozhraní. Cyklus je vlastně adresa, která se objevuje na sběrnici PCI. Specifikace PCI definuje dva formáty konfiguračních adres PCI, typ 0 a typ 1, které jsou znázorněny na obrázcích 6.3 a 6.4. Konfigurační cyklus typu 0 neobsahuje číslo sběrnice a všechna zařízení jej chápou jako konfigurační cyklus zařízení na této sběrnici. Bity 31 až 11 konfiguračního cyklu 0 se chápou jako pole výběru zařízení. Jedna možnost při návrhu systému je každým bitem vybírat jedno zařízení. V takovém případě by se bitem 11 vybíralo zařízení na PCI slotu 0, bitem 12 zařízení na slotu 1 a tak dále. Druhá možnost je na bity 31 až 11 zapisovat přímo čísla slotů PCI. Použitá metoda závisí na řadiči PCI paměti.



Obrázek 6.3
Konfigurační cyklus 0



Obrázek 6.4
Konfigurační cyklus 1

Konfigurační cyklus typu 1 obsahuje číslo sběrnice PCI a tento typ cyklu je ignorován všemi zařízeními s výjimkou můstků PCI-PCI. Když můstek PCI-PCI uvidí cyklus typu 1, rozhodne se, zda jej předat sekundární sběrnici. Zda můstek bude cyklus typu 1 ignorovat nebo zda jej předá sekundární sběrnici závisí na jeho konfiguraci. Každý můstek PCI-PCI má přiděleno číslo primární sběrnice a číslo sekundární sběrnice. Primární sběrnice je sběrnice blíže procesoru, sekundární sběrnice je ta dále od procesoru. Každý PCI-PCI můstek dále zná číslo podřízené PCI sběrnice, což je nejvyšší číslo ze všech sběrnic PCI, které jsou připojeny dalšími můstky k sekundární sběrnici tohoto můstku. Jinak řečeno, číslo podřízené sběrnice je nejvyšší číslo PCI sběrnice směrem dolů od můstku. Když můstek PCI-PCI zaregistruje konfigurační cyklus typu 1, provede jednu ze tří následujících operací:

- Ignoruje cyklus pokud číslo sběrnice uvedené v cyklu neleží mezi číslem sekundární sběrnice můstku a číslem podřízené sběrnice (včetně).
- Konvertuje cyklus na typ 0 pokud číslo sběrnice v cyklu odpovídá číslu sekundární sběrnice můstku.
- Předává nezměněný cyklus na sekundární rozhraní, pokud číslo sběrnice je větší než číslo sekundární sběrnice, ale menší nebo rovno číslu podřízené sběrnice.

Pokud tedy budeme chtít adresovat zařízení 1 na sběrnici 3 podle topologie uvedené na obrázku 6.9, bude muset procesor generovat konfigurační příkaz typu 1. Můstek 1 jej nezměněný předá můstku 2. Můstek 2 jej bude ignorovat, ale můstek 3 jej zkonvertuje na konfigurační příkaz typu 0 a předá jej na sběrnici 3, kde na něj zareaguje zařízení 1.

Mechanismus alokace čísel jednotlivým sběrnicím PCI při inicializaci systému je plně v moci příslušného operačního systému, pro všechny můstky PCI-PCI však musí platit následující pravidlo:

Čísla všech sběrnic pod můstkem PCI-PCI musí být větší než číslo sekundární sběrnice a menší nebo rovna číslu podřízené sběrnice.

Pokud by toto pravidlo bylo porušeno, můstky PCI-PCI by neprováděly správně překlad a předávání konfiguračních cyklů typu 1 a systému by se nepodařilo nalézt a inicializovat zařízení PCI v systému. Kvůli dodržení číslovacího schématu konfiguruje Linux tato speciální zařízení v určitém pořadí. V dále uvedené části „Přiřazení čísel sběrnic“ je podrobněji popsáno schéma číslování sběrnic PCI.

6.6 Inicializace PCI v Linuxu

Inicializační kód PCI je v Linuxu rozdělen na tři logické části:

Ovladač zařízení PCI Tento pseudoovladač zařízení prohledává systém PCI počínaje sběrnicí 0 a nalezne všechna zařízení PCI a můstky v systému. Vytvoří seznam datových struktur popisujících topologii systému. Navíc očísluje všechny nalezené můstky.

PCI BIOS Tato softwarová vrstva zajišťuje služby popsané ve specifikaci PCI BIOSu. Přestože systémy Alpha přímo neposkytují služby BIOSu, v jádře Linuxu je obsažen ekvivalentní kód, zajišťující stejné funkce.

PCI fixup Systémově specifický kód zajišťuje systémově závislé dokončení inicializace PCI.

6.6.1 Datové struktury PCI v jádře

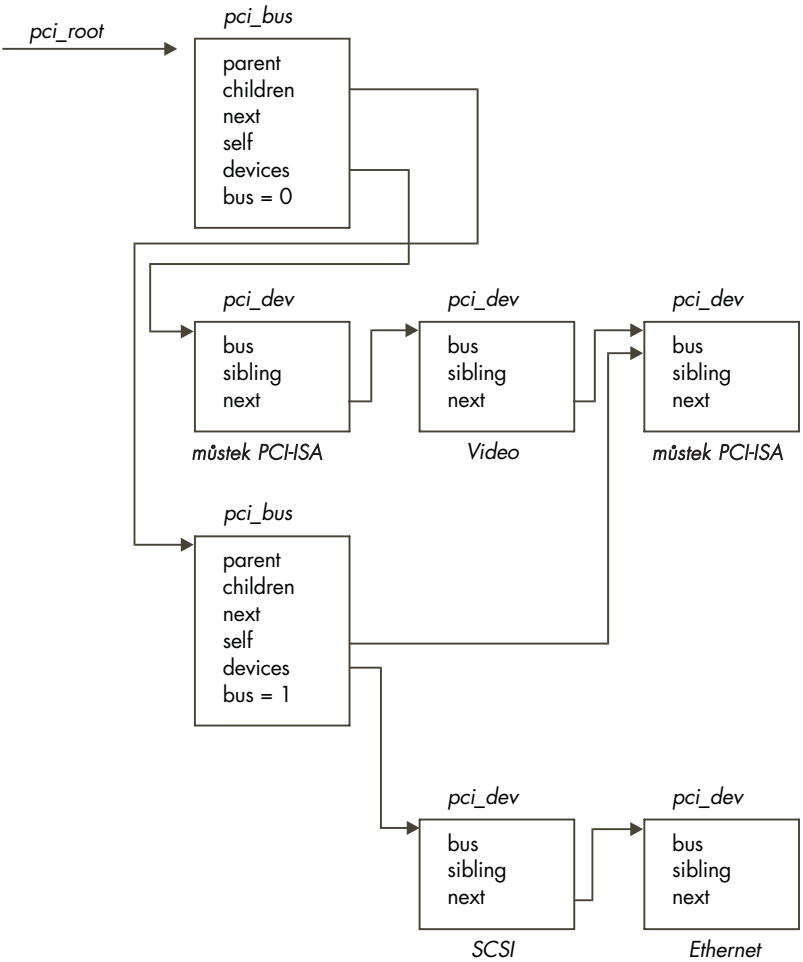
Když jádro Linuxu inicializuje systém PCI, buduje si datové struktury, které zrcadlí skutečnou topologii systému PCI. Na obrázku 6.5 je vidět vztah těchto datových struktur, které by odpovídaly struktuře PCI na obrázku 6.1.

Každé zařízení PCI (včetně můstků PCI-PCI) je popsáno datovou strukturou `pci_dev`. Každá sběrnice PCI je popsána strukturou `pci_bus`. Výsledkem je stromová struktura PCI sběrnic, která má každá k sobě připojeno několik zařízení PCI. Protože sběrnice PCI je dosažitelná pouze prostřednictvím můstku PCI-PCI (s výjimkou primární sběrnice, PCI 0), obsahuje každá struktura `pci_bus` ukazatel na zařízení PCI (můstek PCI-PCI), které ji zpřístupňuje. Toto zařízení je synovským zařízením rodičovské sběrnice té sběrnice, k níž je uvažovaná sekundární sběrnice připojena.

Na obrázku 6.5 není znázorněn ukazatel na všechna zařízení PCI v systému, ukazatel `pci_devices`. Každé zařízení PCI v systému má svou strukturu `pci_dev` zařazenu v seznamu uvedeným tímto ukazatelem. Tento seznam slouží jádru k rychlému nalezení všech zařízení PCI v systému.

6.6.2 Ovladač zařízení PCI

Ovladač zařízení PCI není ve skutečnosti vůbec ovladačem zařízení, jedná se o funkci operačního systému, která se volá v době inicializace systému. Inicializační kód PCI musí prohlédnout všechny sběrnice PCI v systému a nalézt všechna zařízení PCI včetně můstků PCI-PCI.



Obrázek 6.5
Datové struktury PCI v jádře

Používá kód PCI BIOSu k nalezení všech možných slotů na právě prohledávané sběrnici PCI. Pokud je slot obsazen, vytvoří strukturu `pci_dev` popisující zařízení a připojí ji k seznamu známých zařízení PCI (na který ukazuje `pci_devices`).

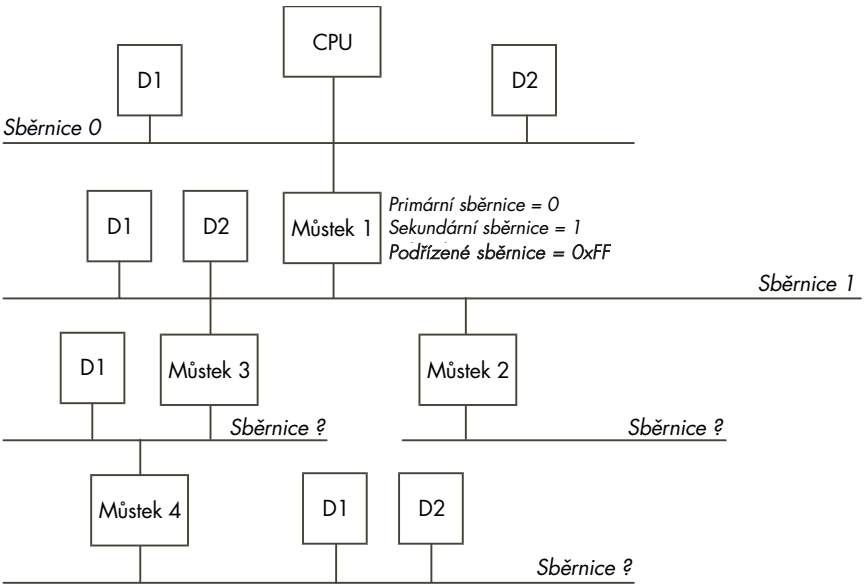
Inicializační kód PCI začíná od sběrnice PCI 0. Pokouší se načíst *identifikátor výrobce* a *identifikátor zařízení* každého možného zařízení ve všech slotech. Když nalezne obsazený slot, vytvoří strukturu `pci_dev` popisující zařízení. Všechny struktury `pci_dev` vytvořené inicializačním kódem PCI (včetně struktur pro můstky PCI) jsou zařazeny do jednosměrně propojeného seznamu `pci_devices`.

Pokud je nalezené zařízení PCI můstkem, vytvoří se struktura `pci_bus` a připojí se ke stromu struktur `pci_bus` a `pci_dev`, na kterou ukazuje `pci_root`. Inicializační kód je schopen rozpoznat můstky, protože všechny můstky mají přidělen kód třídy *0x060400*. Poté jádro Linuxu nakonfiguruje sběrnici PCI na druhé (sekundární) straně právě nalezeného můstku. Pokud je nalezeno více můstků PCI-PCI, nakonfigurují se všechny. Tento proces se označuje jako algoritmus „do hloubky“, protože sběrnice je nejprve zmapována ve směru „do hloubky“ a teprve pak se mapuje „do šířky“. Když se budeme držet obrázku 6.1, nakonfiguruje Linux sběrnici 1 a její SCSI a etherentové zařízení dříve, než provede konfiguraci videozařízení na sběrnici 0.

Když Linux vyhledává sběrnice PCI, musí rovněž konfigurovat mezilehlé můstky PCI-PCI a přidělit jim čísla sekundární a podřízené sběrnice. Tento postup je podrobně popsán v následující části.

Konfigurace můstků PCI-PCI – Přidělování čísel sběrnic

Aby můstek mohl propouštět zápisy a čtení ve V/V prostoru, paměťovém prostoru a konfiguračním prostoru, potřebuje znát následující informace:



Obrázek 6.6
Konfigurace systému PCI, část 1

číslo primární sběrnice Číslo sběrnice bezprostředně nadřazené můstku.

číslo sekundární sběrnice Číslo sběrnice bezprostředně podřízené můstku.

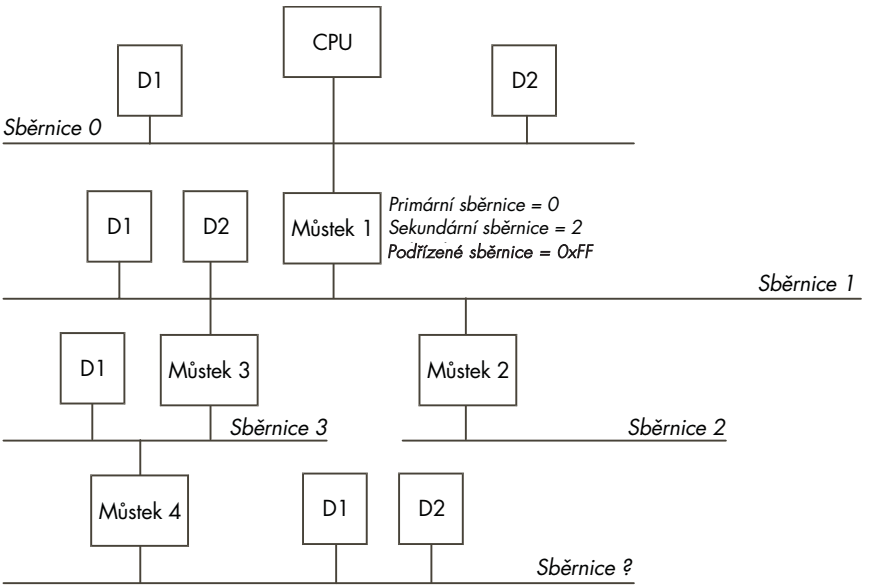
číslo podřízené sběrnice Nejvyšší ze všech čísel sběrnic, které jsou přes můstek směrem dolů dosažitelné.

okna V/V a paměťové oblasti Bázová adresa a velikost V/V prostoru a paměťového prostoru pro všechny směrem dolů adresovatelné sběrnice.

Problém je v tom, že v době kdy chcete nakonfigurovat nějaký můstek PCI-PCI, neznáte číslo jeho podřízené sběrnice. Nevíte, zda je pod ním další můstek a i kdybyste to věděli, nevíte, jaká čísla budou přiřazena pod ním. Odpovědí je použití hloubkového rekurzivního algoritmu, který hledá můstky na všech sběrnicích a jak je nachází, přiřazuje jim čísla. Když je nalezen můstek a očísluje se jeho sekundární sběrnice, jako číslo podřízené sběrnice se dočasně přiřadí číslo *0xFF* a prohledávají se a číslují všechny můstky a sběrnice pod ním. Celé to vypadá složitě, avšak následující příklad by měl vše vyjasnit.

Číslování můstků PCI-PCI – krok 1

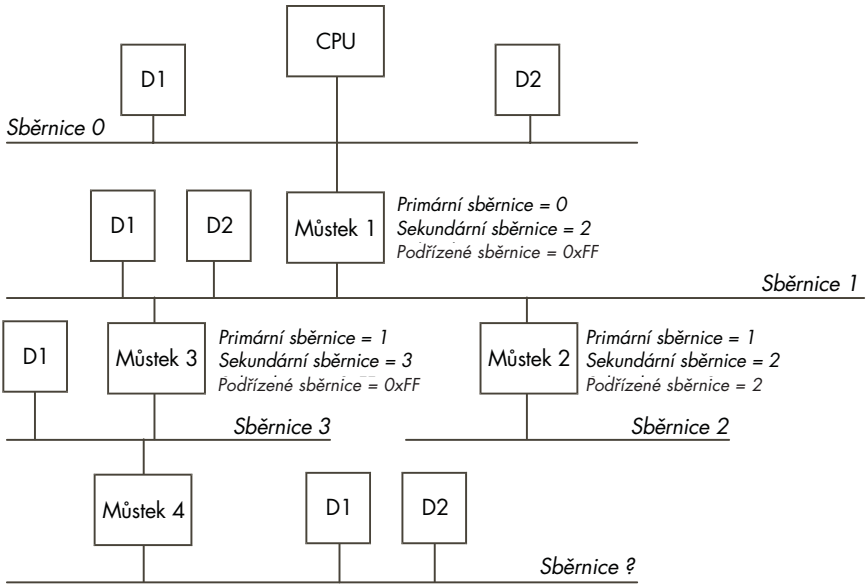
Když vezmeme topologii podle obrázku 6.6, nalezne se jako první můstek 1. Sběrnice PCI pod tímto můstkem bude očíslována jako 1 a můstku 1 bude přiřazeno číslo sekundární sběrnice rovno jedné a číslo podřízené sběrnice rovno dočasně *0xFF*. Znamená to, že všechny konfigurační cykly typu 1 určující sběrnici PCI číslo 1 a vyšší projdou můstkem 1 na sběrnici 1. Pokud mají číslo sběrnice rovno 1, budou tímto můstkem přeloženy na cykly typu 0, pokud mají vyšší číslo sběrnice, zůstanou nezměněny. A to je přesně to, co inicializační kód potřebuje, aby mohl pokračovat a prohlédnout sběrnici 1.



Obrázek 6.7
Konfigurace systému PCI – část 2

Číslování PCI-PCI můstků - krok 2

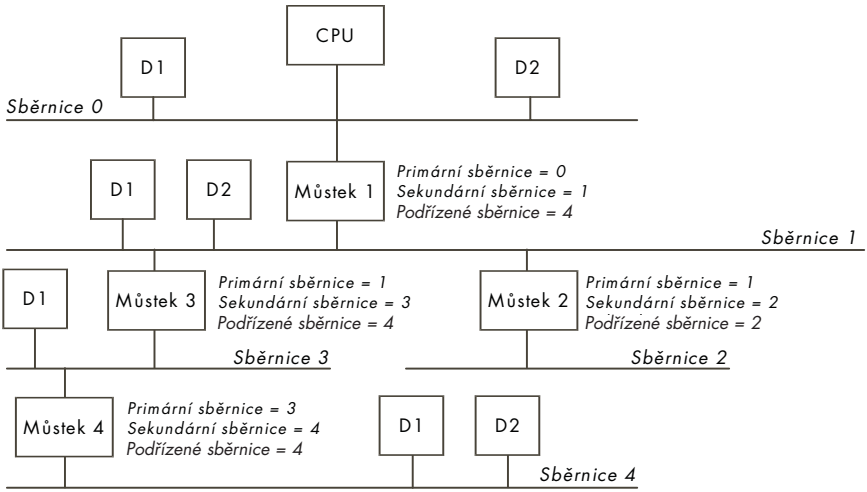
Linux používá hloubkový algoritmus, takže inicializační kód nyní začíná prohlížet můstek 1. Pod ním nalezne můstek 2. Pod můstkem 2 už nejsou žádné další můstky, takže se mu jako číslo podřízené sběrnice přiřadí 2, což odpovídá číslu jeho sekundární sběrnice. Na obrázku 6.7 je vidět přiřazení čísel sběrnic a konfigurace můstků v této fázi.



Obrázek 6.8
Konfigurace systému PCI – část 3

Číslování můstků PCI-PCI – krok 3

Inicializační kód se vrátí ke sběrnici 1 a na ní nalezne další můstek, můstek 3. Jako číslo primární sběrnice bude mít přiřazeno 1, jako číslo sekundární sběrnice 3 a jako číslo podřízené sběrnice *0xFF*. Na obrázku 6.8 vidíme, jak je systém nakonfigurován v tomto okamžiku. Konfigurační cykly typu 1 s čísly sběrnice 1, 2 a 3 se budou správně doručovat na příslušné sběrnice.



Obrázek 6.9
Konfigurace systému PCI – část 4

Číslování můstků PCI-PCI – krok 4

Linux začne prohlížet sběrnici PCI 3 pod můstkem 3. Na této sběrnici je další můstek (můstek 4), kterému se jako primární sběrnice přiřadí 3, jako sekundární sběrnice 4. Jedná se o poslední můstek v této větvi, takže jako číslo podřízené sběrnice bude mít rovněž přiřazeno 4. Inicializační kód se pak vrátí k můstku 3 a jako podřízenou sběrnici nastaví sběrnici 4. Nakonec inicializační kód přiřadí podřízenou sběrnici 4 i můstku 1. Na obrázku 6.9 vidíme finální přiřazení čísel sběrnic.

6.6.3 Funkce PCI BIOSu

Funkce PCI BIOSu jsou standardní množinou operací, které jsou společné pro všechny platformy. Jsou například stejné jak na systémech Intel, tak i Alpha AXP. Umožňují procesoru řídit přístup ke všem adresovým prostorům sběrnice PCI.

Tyto funkce může používat pouze kód jádra a ovladače zařízení.

6.6.4 PCI Fixup

Fixup kód procesoru Alpha AXP toho dělá podstatně více než stejný kód pro procesor Intel (který v zásadě nedělá vůbec nic).

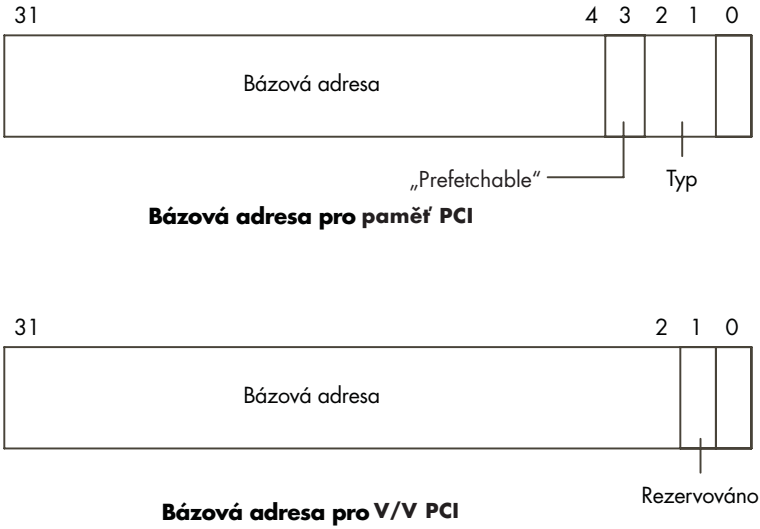
7 U systémů na platformě Intel provede kompletní inicializaci PCI systému BIOS, spouštěný v době startu počítače. Linuxu už toho kromě mapování konfigurace mnoho nezbývá. U jiných systémů je v další fázi konfigurace nutné provést následující operace:

- Přiřazení V/V prostoru a paměťového prostoru jednotlivým zařízením.
- Konfigurace adresových oken jednotlivých můstků.
- Generování hodnot přerušovací linky jednotlivých zařízení, které slouží k obsluze přerušení od zařízení.

V následujícím textu je popsáno, jak příslušný kód funguje.

Zjištění kolik V/V a paměťového prostoru zařízení potřebuje

Každého nalezeného zařízení PCI se systém zeptá, kolik vyžaduje V/V prostoru a paměťového prostoru. Provede se to tak, že do bázových registrů adres se zapíší samé jedničky a poté se registry přečtou. Zařízení vrátí nuly na nevýznamných adresových bitech, čímž fakticky oznámí velikost potřebného adresového prostoru.



Obrázek 6.10
Konfigurační hlavička PCI: Bázové registry adresy

Existují dvě základní podoby bázového registru. První udává v rámci jakého adresového rozsahu musejí být umístěny registry zařízení, ať už ve V/V prostoru nebo paměťovém prostoru - to se určuje podle 0. bitu registru. Na obrázku 6.10 jsou znázorněny dvě podoby bázového registru pro paměť a V/V prostor.

Kolik adresového prostoru je zapotřebí se zjistí tak, že do bázového registru se zapíší samé jedničky a poté se jeho hodnota přečte zpět. Zařízení zapíše nuly na ty bity adresy, které je nezajímají, čímž sdělí velikost potřebného adresového prostoru. Tímto mechanismem se zajišťuje, že velikost každého adresového prostoru je vždy mocnina dvou a prostor je tak přirozeně zarovnán.

Když například inicializujete ethernetovou kartu DEC 21141, sdělí vám, že potřebuje *0x100* bajtů adresového prostoru jak ve V/V prostoru, tak v paměťovém prostoru. Inicializační kód provede alokaci požadovaného prostoru. Jakmile je prostor alokován, jsou v něm vidět řídící a stavové registry zařízení.

Přiřazení V/V a paměti můstkům a zařízením

Stejně jako veškerá paměť, i paměťový prostor PCI V/V a paměti PCI je konečný a omezený. Fixup kód na systémech jiných než Intel (a kód BIOSu na systémech Intel) musí efektivním způsobem přidělit každému zařízení jím požadovaný objem paměti. Jak V/V, tak i paměťový prostor musejí být alokovány v přirozeně zarovnaných úsecích. Pokud zařízení například požaduje V/V prostor o velikosti *0xB0*, musí být prostor zarovnán na adresu, která je násobkem *0xB0*. Kromě toho musejí být přidělované úseky V/V a paměti zarovnávány na hranice 4 KB a 1 MB. Když si k tomu přidáme, že u každého zařízení na sekundární sběrnici můstku musejí jeho adresy ležet uvnitř prostoru přidělenému primární sběrnici tohoto můstku, může být efektivní alokace prostoru obtížným úkolem.

Algoritmus používaný Linuxem počítá s tím, že každé zařízení uvedené ve stromu zařízení a sběrnic, který byl sestaven inicializačním kódem PCI, bude mít paměť přidělovánu v pořadí rostoucích adres. K průchodu datových struktur `pci_bus` a `pci_dev` se opět používá rekurzivní algoritmus. Vychází z kořene struktury PCI (ukazatel `pci_root`) a zajišťuje následující operace:

- Zarovná globální ukazatele PCI V/V prostoru a paměťového prostoru na hranice 4 KB, respektive 1 MB.
- Pro každé zařízení na aktuální sběrnici (v pořadí podle rostoucích požadavků na V/V oblast)
  - alokuje prostor ve V/V a paměti
  - o příslušný objem posune globální ukazatele V/V a paměti
  - povolí zařízení používat V/V a paměť
- Rekurzivně alokuje prostor pro všechny sběrnice pod aktuální sběrnicí. I zde dochází ke změně globálních bázových adres.

- Zarovná globální ukazatele V/V a paměti na 4 KB a 1 MB a při té příležitosti zjistí velikost a bázi V/V a paměťového okna požadovaného daným můstkem.
- Naprogramuje můstek tak, že mu oznámí bázi a velikost V/V a paměťového prostoru podřízené sběrnice.
- Aktivuje přenášení V/V a paměťových přístupů přes můstek. Znamená to, že pokud se na primární sběrnici můstku objeví požadavek na nějakou V/V nebo paměťovou adresu, která patří do okna přiřazeného sekundární sběrnici můstku, přenese můstek tento požadavek na sekundární sběrnici.

Vezměme si jako příklad PCI systém znázorněný na obrázku 6.1. Pak Fixup kód nakonfiguruje systém následujícím způsobem:

Zarovnání bází PCI Počáteční báze PCI V/V je *0x4000*, paměti *0x100000*. Díky tomu mohou můstky PCI-ISA překládat všechny adresy pod těmito hranicemi na adresové cykly sběrnice ISA.

Videozařízení Požaduje *0x200000* PCI paměti, takže tento objem naalokujeme počínaje bázovou adresou *0x200000*, protože paměťové bloky musejí být přirozeně zarovnány. Adresa paměťové báze se posouvá na *0x400000*, V/V báze zůstává na *0x4000*.

Můstek PCI-PCI Projdeme můstkem a alokujeme paměť pod ním, v této chvíli nemusíme zarovnávat báze adres, protože jsou zarovnány správně.

Ethernetové zařízení Požaduje *0xB0* bajtů V/V prostoru i paměťového prostoru. Alokujeme mu V/V prostor od báze *0x4000* a paměťový prostor od báze *0x400000*. Báze paměti se posouvá na *0x4000B0*, báze V/V se posouvá na *0x40B0*.

Zařízení SCSI Požaduje *0x1000* paměti, takže dostává (po přirozeném zarovnání) přidělenu paměť od báze *0x401000*. Báze V/V prostoru zůstává *0x40B0*, báze paměťového prostoru je *0x402000*.

Okna můstku Nyní jsme se vrátili nad můstek a nastavujeme V/V okno na interval *0x4000* až *0x40B0* a paměťové okno na *0x400000* až *0x402000*. Znamená to, že můstek bude ignorovat například přístupu k videozařízení a naopak bude předávat přístupy k SCSI nebo síťovému zařízení.

Odkazy na zdrojové texty jádra

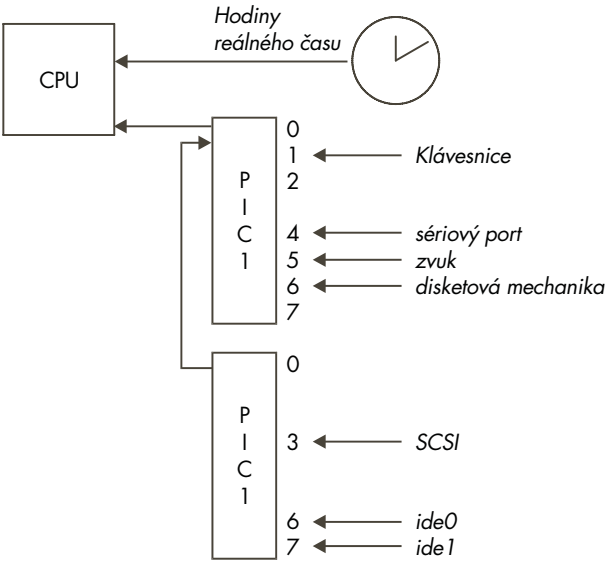
1 – Viz include/-linux pci.h
2 – Viz drivers/-pci/pci.c and include/linux/-pci.h
3 – Viz arch/*/-kernel bios32.c
4 – Viz arch/*/-kernel bios32.c
5 – Viz Scan_bus() in drivers/pci-pci.c
6 – Viz arch/*/-kernel bios32.c
7 – Viz arch/*/-kernel/bios32.c

Kapitola

7

Přerušení a jejich obsluha

V této kapitole se zaměříme na mechanismy, jimiž jádro Linuxu obsluhuje přerušení. I když jádro k obsluze přerušení používá obecné mechanismy a rozhraní, většina detailů v obsluze přerušení závisí na architektuře.



Obrázek 7.1
Logický diagram cesty přerušení

K vykonávání různých úkolů používá Linux různá hardwarová zařízení. Videokarta ovládá monitor, řadič IDE připojuje disk a podobně. Všechna tato zařízení je možno ovládat synchronně, což znamená, že na zařízení pošlete požadavek na nějakou operaci (řekněme na zá-

pis bloku dat z paměti na disk) a poté čekáte, až operace skončí. Tato metoda, i když by teoreticky mohla být funkční, by byla velice neefektivní a operační systém by strávil velké množství času nicneděláním, kdy by čekal na dokončení různých operací. Daleko lepší a efektivnější metoda je vznést požadavek a poté dělat něco dalšího, užitečného a později být přerušen v okamžiku, kdy zařízení požadavek splnilo. Při použití tohoto mechanismu se v systému může v jednom okamžiku provádět řada různých požadavků na různých zařízeních.

Pro přerušení činnosti procesoru musí existovat nějaká hardwarová podpora. Většina, ne-li všechny univerzální procesory, jako například Alpha AXP, používají podobnou metodu. Některé z fyzických vývodů procesoru jsou zapojeny tak, že změna napětí na těchto vývodech (řekněme z +5V na -5V) způsobí, že procesor přestane provádět to, co právě dělá, a začne provádět speciální kód pro obsluhu přerušení. Jeden z těchto vývodů může být připojen k intervalovému časovači, který bude generovat přerušení každou tisícinu sekundy, další mohou být připojeny k jiným zařízením, například k řadiči SCSI.

Většina systémů používá speciální řadič přerušení, který seskupuje dohromady přerušení od různých zařízení a předává je na jediný vývod procesoru. Tím se ušetří počet vývodů na procesoru a zároveň se zvyšuje celková flexibilita při návrhu systému. Řadič přerušení používá při své činnosti maskovací a řídicí registr. Nastavením bitů v maskovacím registru je možno zapínat a vypínat přerušení od různých zdrojů, ve stavovém registru můžeme zjistit, která přerušení jsou právě aktivní.

Některá přerušení mohou být pevně zapojena, například přerušení od časovače může být natvrdo připojeno na třetí vývod řadiče přerušení. Připojení dalších pinů může záviset na tom, jaké karty jsou zapojeny v určitých slotech ISA a PCI. Například 4. pin řadiče přerušení může být připojen k slotu PCI 0, který může jeden den sloužit k připojení síťové karty, druhý den v něm však může být zapojen řadič SCSI. Plyne z toho, že každý systém má své vlastní mechanismy předávání přerušení a operační systém musí být dostatečně pružný, aby se s tím dokázal vyrovnat.

Většina moderních univerzálních procesorů obsluhuje přerušení stejným způsobem. Když dojde k hardwarovému přerušení, CPU přeruší provádění právě aktivní instrukce a skočí na určité místo v paměti, kde je obsažen buď přímo kód obsluhy přerušení, nebo instrukce, která zajišťuje větvení obsluhy různých přerušení. Obslužný kód obvykle pracuje ve zvláštním režimu procesoru, v takzvaném *přerušovacím* režimu, a v té době nemůže za normálních okolností dojít k žádnému jinému přerušení. I zde však jsou výjimky – některé procesory například přidělují přerušením priority a povolují výskyt přerušení s vyšší prioritou. Znamená to ale, že primární kód obsluhy přerušení musí být napsán velmi pozorně a velmi často používá vlastní zásobník, který používá k uložení prováděcího stavu procesoru (tedy všech registrů a kontextu procesoru) předtím, než přejde na samotnou obsluhu události, jež přeru-

šení vyvolala. Některé procesory mají speciální skupinu registrů, které jsou přístupné pouze v přerušovacím režimu a kód obsluhy přerušení může těchto registrů využít k uložení většiny kontextu procesoru.

7.1 Programovatelné řadiče přerušení

Návrháři systémů mohou použít libovolnou architekturu přerušení podle své volby, počítače IBM PC používají obvod Intel 82C59A-2 nebo jeho deriváty - programovatelný řadič přerušení. Tento řadič se používá už od úsvitu počítačů PC a jeho registry se adresují na pevně zavedené lokace adresového prostoru sběrnice ISA. Dokonce i nejmodernější chipsety stále podporují stejné typy registrů na stejných místech v paměti. Systémy založené na jiném procesoru než Intel, například Alpha AXP, nejsou těmito architektonickými omezeními svázány, a tak často používají jiné řadiče přerušení.

Na obrázku 7.1 vidíme dva 8bitové řadiče spřažené dohromady, každý má vlastní maskovací a stavový registr, PIC1 a PIC2. Maskovací registry se mapují na adresy *0x21* a *0xA1*, stavové registry na adresy *0x20* a *0xA0*. Zapsáním jedničky na určitý bit maskovacího registru se přerušení povoluje, zapsáním nuly se vypíná. Tedy zápis jedničky na bit 3 povolí přerušení 3, nula na stejném bitu toto přerušení deaktivuje. Je bohužel nepříjemné, že maskovací registry jsou navrženy pouze pro zápis, nemůžete z nich zpět načíst hodnotu, která je v nich zapsána. Znamená to, že Linux si musí vést lokální kopii nastavení maskovacích registrů. V rutinách pro aktivaci a deaktivaci přerušení modifikuje tyto uložené hodnoty a při každém zápisu do registru zapisuje celou hodnotu masky.

Když dojde k výskytu přerušení, přečte obslužný kód přerušení dva stavové registry přerušení (ISR). Registr na adrese *0x20* je chápán jako nižších osm bitů 16bitového přerušovacího registru, registr na adrese *0xA0* jako vyšších osm bitů. Takže přerušení na prvním bitu registru na adrese *0xA0* bude chápáno jako přerušení 9. Bit 2 řadiče PIC1 není k dispozici, protože slouží k řetězení přerušení od řadiče PIC2. Každé přerušení na řadiči PIC2 se projeví jako přerušení na druhém bitu řadiče PIC1.

7.2 Inicializace datových struktur obsluhy přerušení

Datové struktury jádra pro obsluhu přerušení inicializují ovladače zařízení podle svých požadavků na řízení systémových přerušení. K tomu účelu používají ovladače skupinu služeb jádra, které slouží k žádosti o přerušení, k jejich aktivaci a deaktivaci.

Jednotlivé ovladače zařízení prostřednictvím těchto rutin registrují adresy svých obslužných rutin přerušení.

Některá přerušení jsou pevně dána konvencemi architektury PC, takže ovladače při své inicializaci prostě jen požádají o toto přerušení. To se týká například ovladače disketových mechanik, které vždy používají přerušení IRQ 6. Za jiných okolností nemusí ovladač zařízení vědět, jaké přerušení jeho zařízení používá. Tento problém se neobjevuje u ovladačů zařízení PCI, které vždy znají číslo přerušení používané jejich zařízením. Bohužel však neexistuje jednoduchý způsob, jak mohou číslo přerušení svého zařízení zjistit ovladače zařízení ISA. Linux tento problém řeší tak, že umožňuje ovladačům vyhledat číslo přerušení svého zařízení.

Nejprve ovladač provede nějakou akci, která vyvolá přerušení od zařízení. Poté se povolí všechna nepřiřazená přerušení. Znamená to, že přerušení od našeho zařízení nyní bude prostřednictvím programovatelného řadiče přerušení doručeno. Linux přečte obsah stavového registru přerušení a předá jej ovladači zařízení. Nenulový výsledek znamená, že v průběhu testu se objevilo jedno nebo více přerušení. Ovladač zruší režim hledání a nepřiřazená přerušení se opět zakáží.

2

Pokud se ovladači zařízení ISA podařilo nalézt číslo přerušení jeho zařízení, může nyní normálním postupem požádat o řízení tohoto přerušení.

Systémy PCI jsou podstatně dynamičtější než systémy ISA. Číslo přerušení, které bude zařízení ISA používat, se velmi často nastavuje jumpery přímo na řadiči a ovladač zařízení musí toto číslo znát. Naproti tomu zařízení PCI dostávají čísla přerušení přidělena PCI BIOSem nebo subsystémem PCI při inicializaci sběrnice PCI při zavádění systému. Každé zařízení PCI může použít jednu ze čtyř pozic přerušení, A, B, C nebo D. Toto nastavení je dáno výrobcem zařízení a většina zařízení PCI implicitně používá přerušovací pozici A. Pak je možno směrovat pozici A slotu PCI 4 na šestý pin řadiče přerušení, pozici B slotu 4 na sedmý pin řadiče a tak dále.

Směrování přerušení PCI je dáno výhradně architekturou systému, a proto musí být k dispozici nějaký kód, který rozumí topologii směrování přerušení PCI. Na systémech Intel to zajišťuje kód BIOSu, který se provádí při zavádění systému, na systémech bez BIOSu (například na platformě Alpha AXP) však tuto funkci plní jádro Linuxu.

3

Inicializační kód PCI zapisuje číslo pinu řadiče přerušení do konfigurační hlavičky každého zařízení PCI. Čísla přerušení stanovuje na základě znalosti směrovací topologie přerušení PCI, na znalosti čísel slotů PCI a jimi používaných přerušení PCI. Číslo přerušení používané zařízením je pak pevně dáno a je uloženo v konfigurační hlavičce tohoto zařízení. Tyto informace se zapisují do položky *interrupt line*. Když se spustí ovladač zařízení, přečte si tuto informaci a může jádro Linuxu požádat o předání kontroly nad daným přerušením.

V systému se může nacházet více zdrojů přerušení PCI, například pokud se používají můstky PCI-PCI. Počet zdrojů přerušení může přesáhnout počet dostupných pinů programovatelného řadiče přerušení. V takovém případě mohou zařízení PCI sdílet přerušení, tedy jeden pin řadiče přerušení přijímá přerušení od více zařízení. Tento mechanismus Linux podporuje tak, že první zařízení žádající o přidělení přerušovacího pinu oznamuje, zda se jeho pin může sdílet. Sdílení přerušení vede k více datovým strukturám `irqaction`, na něž všechny ukazuje jeden vektor `irq_action`. Když se objeví sdílené přerušení, Linux bude volat všechny obslužné kódy tohoto přerušení. Každý ovladač zařízení, který může sdílet přerušení (což by měly být všechny ovladače zařízení PCI) musí být připraven na to, že se bude volat obslužná rutina přerušení i v případě, že není co obsluhovat.

7.3 Obsluha přerušení

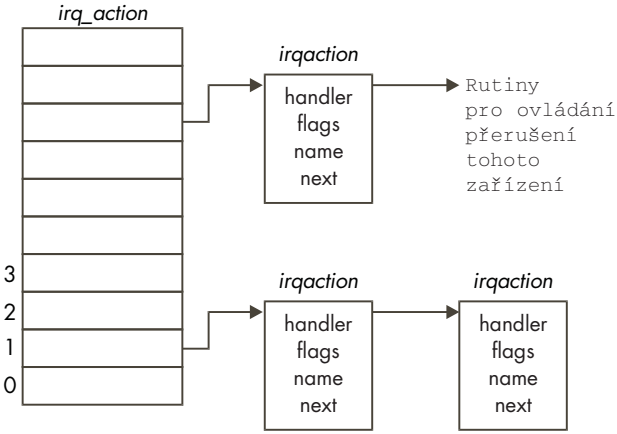
Jedním ze základních úkolů subsystému obsluhy přerušení v Linuxu je nasměrování přerušení na správnou obslužnou rutinu. Tento kód musí rozumět topologii přerušení v systému. Pokud například řadič disketové mechaniky používá šestý pin řadiče přerušení,[1] pak řadič přerušení musí toto přerušení detekovat jako přerušení od disketového řadiče a musí je předat kódu obsluhy přerušení od diskety. Linux používá sadu ukazatelů na datové struktury obsahující adresy obslužných rutin přerušení. Tyto rutiny jsou součástí ovladačů zařízení připojených k systému a každý ovladač musí v době své inicializace požádat o obsluhu svého přerušení. Na obrázku 7.2 je vidět strukturu `irq_action`, která je vektorem ukazatelů na struktury `irqaction`. Každá datová struktura `irqaction` obsahuje informace o obsluze svého přerušení včetně adres obslužné rutiny tohoto přerušení. Protože se počty přerušení a jejich obsluha mohou na různých architekturách nebo i na různých systémech se stejnou architekturou lišit, je obslužný kód přerušení Linuxu závislý na architektuře. Znamená to, že velikost vektoru `irq_action` se liší podle počtu možných zdrojů přerušení.

Když se objeví přerušení, Linux musí nejprve zjistit jeho zdroj tím, že přečte stavový registr programovatelného řadiče přerušení systému. Zdroj přerušení pak přeloží na offset ve vektoru `irq_action`. Takže například přerušení na šestém pinu řadiče přerušení (od ovladače disket) bude přeloženo na sedmý ukazatel ve vektoru obsluhy přerušení. Pokud pro dané přerušení není zaveden žádný obslužný kód, ohlásí jádro Linuxu chybu, v opačném případě postupně zavolá všechny obslužné kódy daného přerušení.

Když jádro Linuxu zavolá obslužnou rutinu přerušení nějakého ovladače zařízení, musí rutina rychle zjistit proč k přerušení došlo a vhodně na to zareagovat. K zjištění příčiny přerušení čte ovladač zařízení stavový registr svého zařízení. Zařízení může hlásit chybu nebo může oznamovat, že operace skončila. Například řadič disketové mechaniky může hlásit, že čtecí

[1] Zrovna řadič disketových mechanik v systémech PC používá fixně přidělené přerušení a podle zavedené konvence je vždy připojen na přerušení 6.

hlavička je vystavena nad požadovaným sektorem diskety. Jakmile je zjištěn důvod přerušení, může ovladač zařízení potřebovat provést další obsluhu. Pokud to je nutné, má jádro Linuxu mechanismy, které umožňují odložit tuto práci na později. Tím se zabrání, aby procesor strávil příliš mnoho času v přerušovacím režimu. Další podrobnosti naleznete v kapitole o ovladačích zařízení.



Obrázek 7.2
Datové struktury obsluhy přerušení

Odkazy na zdrojové texty jádra

1 – `Viz request_irq(), enable_irq() and disable_irq() in arch/*/kernel irq.c`

2 – `Viz irq_probe_*() in arch/*/kernel/-irq.c`

3 – `Viz arch/alpha/-kernel/biow32.c`

Kapitola

8

Ovladače zařízení

Jedním z úkolů operačního systému je izolovat uživatele od specifik hardwarových zařízení. Například virtuální souborový systém nabízí uniformní pohled na všechny připojené souborové systémy bez ohledu na příslušná fyzická zařízení. V této kapitole popisujeme, jak Linux spravuje fyzická zařízení v systému.

Procesor není jediným inteligentním zařízením v systému, každé fyzické zařízení má svůj vlastní hardwarový řadič. Klávesnice, myš a sériové porty jsou řízeny vstupně/výstupním čipem, disky IDE řadičem IDE, disky SCSI řadičem SCSI a tak dále. Každý hardwarový řadič má své řídící a stavové registry, které se pro různá zařízení liší. Řídící registry řadiče SCSI Adaptec 2940 jsou úplně jiné než registry řadiče SCSI NCR 810. Řídící registry slouží ke spouštění a zastavení zařízení, k jeho inicializaci a diagnostice problémů. Namísto toho, aby obslužný kód jednotlivých zařízení v systému obsahovala každá aplikace, je tento kód přítomen pouze v jádře systému. Program, který obsluhuje hardwarový řadič, se označuje jako ovladač zařízení. Ovladače zařízení v Linuxu jsou v zásadě sdílené knihovny privilegovaných, paměťově rezidentních nízkoúrovňových rutin pro obsluhu hardwaru. Právě ovladače zařízení skrývají odlišnosti mezi různými zařízeními.

Jedna ze základních funkcí je abstrakce obsluhy zařízení. Všechna hardwarová zařízení vypadají jako normální soubory, dají se otevírat, zavírat, číst a zapisovat pomocí stejných standardních systémových volání, jaká se používají pro soubory. Každé zařízení v systému je reprezentováno *souborem zařízení*, například první disk IDE je reprezentován souborem `/dev/hda`. Pro bloková zařízení (disky) a znaková zařízení se tyto soubory vytvářejí příkazem `mknod` a popisují zařízení pomocí hlavního a vedlejšího čísla zařízení. Síťová zařízení jsou rovněž reprezentována soubory zařízení, ty však vytváří přímo Linux, když síťové zařízení nalezne. Všechna zařízení ovládaná společným ovladačem zařízení mají stejné hlavní číslo. Vedlejší čísla pak slouží k odlišení těchto zařízení a jejich řadičů, například každá oblast

primárního disku IDE má jiné vedlejší číslo zařízení. Takže například `/dev/hda2`, druhá oblast primárního disku IDE, má hlavní číslo 3 a vedlejší číslo 2. Linux mapuje soubor zařízení předávaný v systémovém volání (například při připojování souborového systému na blokovém zařízení) na ovladač zařízení pomocí hlavního čísla zařízení a řady systémových tabulek, například tabulky znakových zařízení, `chrdevs`.

Linux podporuje tři typy hardwarových zařízení: znakové, blokové a síťové. Znaková zařízení se čtou a zapisují přímo bez použití bufferů, jsou to například sériové porty `/dev/cua0` a `/dev/cua1`. Bloková zařízení je možno číst a zapisovat pouze v násobcích velikosti bloku, který typicky bývá 512 nebo 1 024 bajtů. K blokovým zařízením se přistupuje přes buffery a je možno k nim přistupovat náhodně, tedy je možno přečíst nebo zapsat libovolný blok bez ohledu na jeho polohu na zařízení. K blokovým zařízením je možno přistupovat přes příslušný soubor zařízení, daleko častěji se k nim však přistupuje přes souborový systém. Pouze bloková zařízení podporují připojení souborových systémů. K síťovým zařízením se přistupuje přes soketové rozhraní BSD a síťový subsystém popsaný v kapitole „Sítě“.

V jádře Linuxu existuje řada různých ovladačů zařízení (mimo jiné i v tom je síla Linuxu), všechny však mají určité společné rysy:

kód jádra Ovladače zařízení jsou částí jádra a stejně jako ostatní kód v jádře, pokud by fungovaly chybně, mohly by vážně poškodit celý systém. Špatně napsaný ovladač zařízení může zhroutit celý systém, poškodit souborový systém a zničit data.

rozhraní jádra Ovladače zařízení musejí jádru Linuxu nebo subsystému, jehož jsou součástí, poskytovat standardní rozhraní. Například terminálové zařízení poskytuje jádru souborové vstupně/výstupní rozhraní, zařízení SCSI poskytuje rozhraní zařízení SCSI pro subsystém SCSI, který pak jádru dále poskytuje jednak souborové vstupně/výstupní rozhraní a jednak bufferové rozhraní.

mechanismy a služby jádra Ovladače zařízení používají ke své činnosti standardní služby jádra, jako je alokace paměti, doručování přerušení a čekací fronty.

modularita Většina ovladačů zařízení v Linuxu může být nahrána podle potřeby, když je nějaký modul jádra potřebuje, a odstraněna, když nejsou více zapotřebí. Díky tomu je jádro velmi přizpůsobivé a efektivně využívá systémových prostředků.

konfigurovatelnost Ovladače zařízení mohou být vestavěny přímo do jádra. Které z ovladačů vestavět, se konfiguruje při překladu jádra.

dynamičnost Když se systém zavádí a inicializují se jednotlivé ovladače zařízení, každý ovladač hledá hardwarové zařízení, které má ovládat. Nevadí, pokud zařízení ovládané nějakým určitým ovladačem neexistuje. V takovém případě je ovladač prostě nadbytečný a nezpůsobí žádnou škodu kromě toho, že zabírá část systémové paměti.

8.1 Dotazování a přerušení

Vždy, když ovladač zařízení vydá nějaký povel, například „*přesuň hlavičku na 42. stopu diskety*“, může si ovladač zvolit metodu jak zjistí, že operace byla dokončena. Ovladač se může buď zařízení dotazovat, nebo může použít přerušení.

Dotazování typicky znamená, že ovladač opakovaně čte řídicí registr zařízení tak dlouho, až se status zařízení změní a indikuje dokončení operace. Protože ovladač zařízení je součástí jádra, bylo by nevhodné, kdyby se delší dobu dotazoval, protože nic jiného v jádře by nemohlo pracovat až do doby, než by byl požadavek splněn. Namísto cyklického dotazování používají ovladače zařízení časovač, který způsobí, že jádro periodicky volá nějakou rutinu ovladače. Obsluha časovače bude tak periodicky testovat status zařízení. Tento mechanismus používá Linux pro obsluhu disketových mechanik. Dotazování pomocí časovačů je efektivnější řešení, nejefektivnější je však použití přerušení.

Přerušením řízený ovladač zařízení se používá pro zařízení, která v době, kdy potřebují nějakou obsluhu, vyvolají přerušení. Například ovladač ethernetové karty vyvolá přerušení, když karta obdrží paket ze sítě. Jádro Linuxu musí být schopno doručit přerušení od hardwarového zařízení správnému ovladači zařízení. Dosahuje se toho tím, že ovladače zařízení v jádře registrují používaná přerušení. Registruje se adresa obslužné rutiny přerušení a číslo přerušení, které chce ovladač obsluhovat. V souboru `/proc/interrupts` můžete zjistit počet přerušení v systému a jejich obsluhu ovladači zařízení:

```
 0:          727432    timer
 1:           20534    keyboard
 2:               0    cascade
 3:           79691  + serial
 4:           28258  + serial
 5:               1    sound blaster
11:           20868  + aic7xxx
13:               1    math error
14:             247  + ide0
15:             170  + ide1
```

Žádosti o přerušení ovladač registruje v době své inicializace. Některá přerušení v systému jsou pevně dána konvencemi architektury IBM PC. Například ovladače disket vždy používají přerušení 6. Jiná přerušení, například přerušení od zařízení PCI, se přidělují dynamicky při startu systému. V takovém případě musí ovladač nejprve zjistit číslo přerušení (IRQ) svého zařízení a teprve potom může žádat o jeho obsluhu. Pro přerušení Linux podporuje standardní volání PCI BIOSu, která umožňují zjistit informace o zařízeních v systému včetně jim přidělených čísel přerušení.

Mechanismus doručení přerušení procesoru je závislý na architektuře, nicméně na většině architektur se přerušení doručují ve speciálním režimu, který zabrání výskytu jiných přerušení v systému. Ovladač zařízení by měl při obsluze přerušení pracovat co nejrychleji, aby jádro mohlo záhy prohlásit přerušení za obsloužené a mohlo se vrátit k přerušené práci. Ovladače zařízení, které při výskytu přerušení potřebují provést větší objem práce, mohou použít obslužný mechanismus bottom-half nebo frontu úloh, čímž zajistí, aby se potřebná obsluha volala později v normálním režimu.

8.2 Přímý přístup do paměti (DMA)

Použití přerušením řízených ovladačů zařízení funguje dobře, pokud se přenášejí rozumně nízké objemy dat. Například modem o rychlosti 9 600 Baudů přenese přibližně jeden znak za jednu milisekundu. Pokud je latentní doba přerušení, tedy čas mezi tím než zařízení vyvolá přerušení a než se předá řízení přerušovací obsluze, malá (řekněme 2 milisekundy), je celkový dopad datového přenosu na výkon systému velmi nízký. Přenos dat prostřednictvím modemu o rychlosti 9 600 Baudů spotřebuje pouhých 0,002% procesorového času. U vysokorychlostních zařízení, jako jsou například pevné disky nebo síťové karty, je zatížení přenosem dat podstatně vyšší. Rozhraní SCSI je schopno za sekundu přenést až 40 MB dat.

K vyřešení tohoto problému byl vyvinut mechanismus přímého přístupu do paměti, DMA. řadič DMA umožňuje zařízením přenášet data do nebo ze systémové paměti bez zásahu procesoru. Řadič ISA DMA na počítačích PC má 8 kanálů DMA, z nichž 7 je přístupných pro potřeby ovladačů zařízení. Každému kanálu DMA je přiřazen 16bitový adresový registr a 16bitové počitadlo. K zahájení datového přenosu musí ovladač zařízení nejprve nastavit adresový registr a čítač a dále musí určit směr přenosu, zápis či čtení. Pak oznámí zařízení, že jakmile bude chtít, může zahájit přenos DMA. Po skončení přenosu vyvolá zařízení přerušení. Zatímco přenos probíhá, procesor není zatěžován a může se věnovat jiné činnosti.

Ovladače zařízení musejí s DMA spolupracovat velmi opatrně. Řadič DMA v první řadě neví nic o virtuální paměti, má přístup přímo k fyzické paměti systému. Data přenášená pomocí DMA tedy musí být uložena jako souvislý blok fyzické paměti. Znamená to, že DMA nemůžete použít přímo ve virtuálním adresovém prostoru procesu. Můžete nicméně zamknout fyzic-

ké stránky procesu v paměti, takže po dobu přenosu DMA nemůže dojít k jejich odložení. Dále řadič DMA nemůže přistupovat k celé fyzické paměti. Adresový registr řadiče DMA reprezentuje prvních 16 bitů adresy přenosu, dalších 8 bitů je dáno stránkovým registrem. Znamená to, že přenosy DMA mohou probíhat pouze v prvních 16 MB fyzické paměti.

Kanály DMA jsou vzácná zařízení, protože je jich pouze sedm a nedají se mezi zařízeními sdílet. Stejně jako s přerušeními, musí být ovladač zařízení schopen určit, který kanál DMA může použít. Obdobně jako u přerušení, některá zařízení mají kanál DMA pevně dán. Například disketový řadič používá vždy kanál DMA 2. Někdy je možno kanály DMA zařízení nastavovat pomocí přepínačů, tento způsob používá řada síťových karet. Modernějším zařízením je možno (prostřednictvím jejich řídicích registrů) říci, jaký kanál DMA mají použít a v takovém případě si ovladač zařízení prostě zvolí jeden z volných kanálů DMA.

Linux sleduje využití kanálů DMA pomocí vektoru datových struktur `dma_chan` (jedna pro každý kanál DMA). Datová struktura `dma_chan` obsahuje pouze dvě položky, ukazatel na řetězec popisující vlastníka kanálu DMA a příznak, zda kanál DMA je či není přidělen. Když vypisujete soubor `/proc/dma`, vypisuje se obsah právě tohoto vektoru.

8.3 Paměť

Ovladače zařízení musejí být při práci s pamětí velmi opatrné. Protože jsou součástí jádra, nemohou používat virtuální paměť. Vždy, když je ovladač zařízení spuštěn, ať už po přijetí přerušení nebo mechanismem bottom-half nebo obsluhou fronty úloh, může být aktuální proces jiný. Ovladač zařízení se nemůže spoléhat na to, že zrovna poběží určitý proces, i když pracuje jeho jménem. Stejně jako zbytek jádra, i ovladače zařízení používají různé datové struktury k uložení stavu zařízení, které ovládají. Tyto datové struktury mohou být alokovány staticky jako součást ovladače, to by však bylo plýtvání, protože jádro by bylo větší, než je nezbytně nutné. Většina ovladačů zařízení si pro uložení svých dat alokuje nestránkovanou paměť jádra.

Linux poskytuje rutiny pro alokaci a dealokaci paměti jádra a tyto rutiny využívají právě ovladače zařízení. Paměť jádra se alokuje v úsecích, jejichž velikost je vždy mocninou dvou. Například 128 nebo 512 bajtů, dokonce i v případě, že ovladač zařízení požaduje méně. Velikost paměti požadovaná ovladačem se zaokrouhlí nahoru na nejbližší hranici. Tím se usnadňuje dealokace paměti jádra, protože menší volné bloky je možno snáze spojovat do větších.

Může se stát, že při požadavku na paměť jádra bude zapotřebí provést větší množství práce. Pokud je málo volné paměti, může být nutné zrušit nebo odložit nějaké fyzické stránky. Za normálních okolností Linux žadatele pozastaví a umístí jej do čekací fronty, dokud nebude dostatek paměti. Ne všechny ovladače paměti (nebo přímo samotný kód jádra) si to vždy mohou dovolit, takže alokační rutiny paměti jádra je možno volat také tak, aby v případě, že nebudou schopny paměť poskytnout okamžitě, vyvolaly chybu. Pokud ovladač zařízení žádá

o paměť pro potřeby přenosu DMA, může specifikovat, že paměť bude využita pro DMA. Díky tomu potřebuje znát podrobnosti o přidělování paměti pro potřeby DMA pouze jádro a ne samotný ovladač.

8.4 Komunikace ovladačů s jádrem

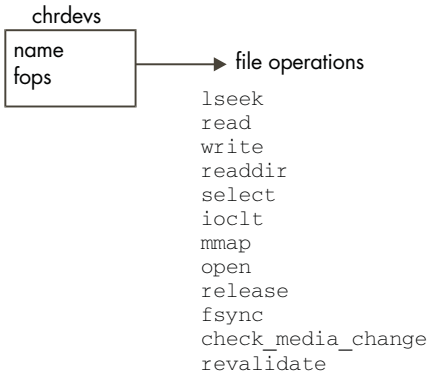
Jádro Linuxu musí být schopno komunikovat s ovladači standardními způsoby. Každá třída ovladačů, znakových, blokových a síťových, poskytuje jednotné rozhraní, které jádro používá při požadavcích na služby. Toto společné rozhraní zajišťuje, že jádro může ve většině případů komunikovat s ovladači zcela rozdílných zařízení úplně stejně. Například disky SCSI a IDE se chovají velmi rozdílně, jádro Linuxu však při komunikaci s oběma z nich používá stejné rozhraní.

Linux je značně dynamický, vždy když se jádro Linuxu zavádí, může detekovat různá fyzická zařízení a může tedy potřebovat různé ovladače zařízení. Linux umožňuje zahrnout ovladače zařízení přímo do jádra prostřednictvím konfiguračních skriptů jádra. Když se takové ovladače při zavádění inicializují, mohou zjistit, že nemají žádné zařízení, které budou ovládat. Další ovladače je možno nahrávat do jádra podle potřeby. Aby se dalo s touto dynamickou podstatou ovladačů zařízení pracovat, musejí se ovladače vždy při inicializaci zaregistrovat. Linux udržuje tabulky registrovaných ovladačů zařízení jako součást rozhraní pro komunikaci s nimi. Tyto tabulky obsahují ukazatele na rutiny a informace, které slouží pro podporu rozhraní jednotlivých tříd ovladačů.

8.4.1 Znaková zařízení

Ke znakovým zařízením, nejjednodušším zařízením Linuxu, se přistupuje stejně jako k souborům - aplikace k jejich otevření, čtení a zápisu používají stejná standardní systémová volání jako kdyby šlo o soubory. Platí to i v případech, kdy se jedná například o modem, který prostřednictvím démona PPP slouží k připojení systému na síť. Když se znakové zařízení inicializuje, jeho ovladač se v jádře Linuxu registruje přidáním položky do vektoru `chrdevs` datových struktur `device_struct`. Hlavní identifikátor zařízení (například 4 pro zařízení `tty`) slouží jako index tohoto vektoru. Hlavní identifikátor je pro zařízení pevně dán.

2 Každá položka vektoru `chrdevs`, datová struktura `device_struct`, obsahuje dva prvky: ukazatel na jméno registrovaného ovladače zařízení a ukazatel na blok souborových operací. Blok souborových operací představuje adresy rutin v ovladači zařízení, které obsluhují jednotlivé souborové operace jako otevření, čtení, zápis a zavření. Obsah souboru `/proc/devices` se pro znaková zařízení pořizuje právě z vektoru `chrdevs`.



Obrázek 8.1
Znakové zařízení

Když dojde k otevření speciálního znakového souboru reprezentujícího znakové zařízení (například souboru `/dev/cua0`), musí jádro vše zařídit tak, aby se volaly souborové rutiny správného ovladače znakového zařízení. Stejně jako u normálních souborů nebo adresářů, každý soubor zařízení má rovněž svůj VFS inode. Inode pro soubor znakového zařízení, respektive pro soubor každého zařízení, obsahuje jak hlavní, tak vedlejší identifikátor zařízení. VFS inode byl vytvořen nějakým souborovým systémem na nižší úrovni, například systémem `ext2`, z informací v reálném souborovém systému, když byl speciální soubor zařízení nalezen. 3

Každý VFS inode má přiřazenu množinu souborových operací, které se liší podle toho, jaký objekt souborového systému příslušný inode reprezentuje. Když se vytváří VFS inode reprezentující soubor znakového zařízení, nastaví se souborové operace na implicitní operace znakového zařízení.

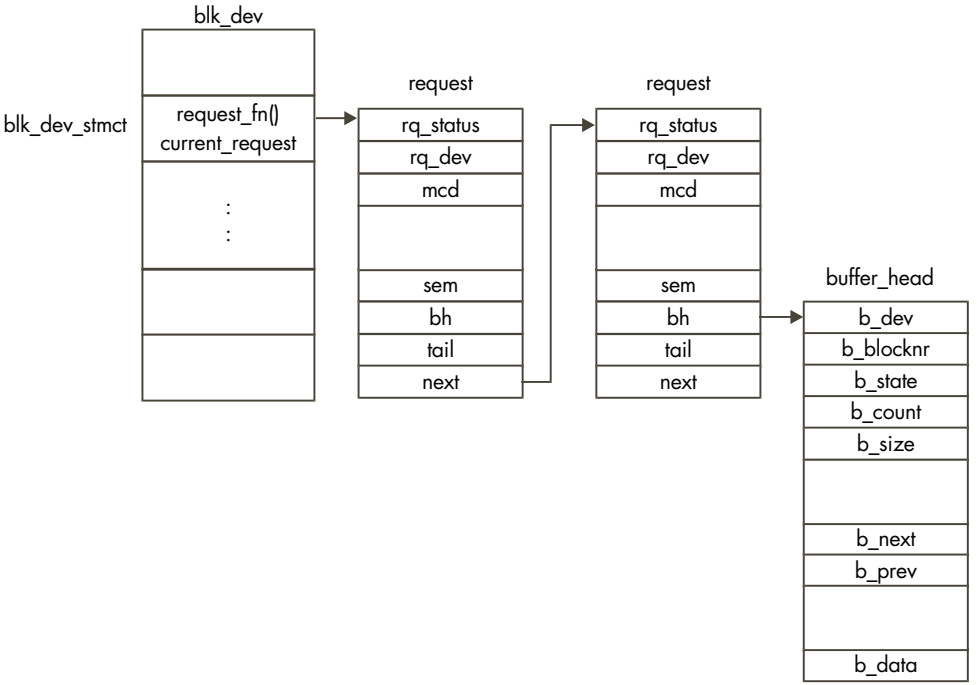
Z implicitních operací je definována pouze jedna - operace otevření. Když aplikace otevře 4
soubor znakového zařízení, obecná obslužná rutina otevření v jádře použije hlavní identifikátor zařízení jako index do vektoru `chrdevs` a získá blok souborových operací pro toto zařízení. Dále vytvoří datovou strukturu `file`, která popisuje soubor znakového zařízení a její ukazatel souborových operací nasměruje na ovladač zařízení. Následně se všechny souborové operace v aplikaci budou mapovat na volání rutin v ovladači zařízení. 5

8.4.2 Bloková zařízení

Bloková zařízení rovněž podporují stejné metody přístupu jako k souborům. Mechanismus používaný k přiřazení správné skupiny souborových operací otevřenému souboru zařízení funguje velmi podobně jako u znakových zařízení. Registrovaná bloková zařízení Linux ukládá ve vektoru `blkdevs`. Tento vektor, stejně jako vektor `chrdevs`, je indexován pomocí 6
hlavního čísla zařízení. Jeho položkami jsou rovněž datové struktury `device_struct`. Na

rozdíl od znakových zařízení jsou bloková zařízení rozdělena do tříd. Například zařízení SCSI patří do jedné třídy, zařízení IDE do jiné. V jádře Linuxu se registrují právě třídy, které pak zajišťují souborové operace. Ovladač zařízení jedné třídy blokových zařízení poskytuje rozhraní specifické pro danou třídu. Takže například ovladač konkrétního zařízení SCSI poskytuje rozhraní subsystému SCSI, které pak subsystém SCSI používá k poskytnutí rozhraní souborových operací jádru.

Každé blokové zařízení musí poskytovat rozhraní k bufferovým operacím a také normální rozhraní souborových operací. Každý ovladač blokového zařízení vyplňuje své údaje v datové struktuře `blk_dev_struct` vektoru `blk_dev`. Tento vektor se opět indexuje hlavním číslem zařízení. Datová struktura `blk_dev_struct` se skládá z adresy rutiny žádostí a ze seznamu datových struktur `request`, z nichž každá reprezentuje jednu žádost o čtení nebo zapsání bloku dat na zařízení.



Obrázek 8.2
Žádosti bufferů blokových zařízení

Vždy, když vyrovnávací paměť bufferů chce přečíst nebo zapsat blok dat z nebo na registrované zařízení, přidává do příslušné struktury `blk_dev_struct` další strukturu `request`. Na obrázku 8.2 vidíme, že každá žádost má ukazatele na jeden nebo více datových struktur `buffer_head`, z nichž každá znamená jednu žádost o čtení nebo zápis blo-

ku dat. Datové struktury `buffer_head` jsou uzamčeny (vyrovnávací pamětí bufferů) a může existovat proces, čekající na dokončení blokové operace v tomto bufferu. Každá struktura `request` se alokuje ze statického seznamu `all_requests`. Když se přidává žádost do prázdného seznamu žádostí, volá se funkce ovladače, která zahajuje zpracování fronty žádostí. Ovladač pak jednoduše zpracuje všechny žádosti ve frontě.

Když ovladač zařízení dokončí zpracování žádosti, musí odstranit všechny struktury `buffer_head` ze struktury `request`, označí je jako aktuální a odemkne je. Tímto odemknutím struktur `buffer_head` dojde k probuzení procesu, který čekal na dokončení blokové operace. Příkladem může být situace, kdy se hledá jméno souboru a souborový systém `ext2` musí z blokového zařízení, na němž je souborový systém uložen, načíst blok dat, který obsahuje další adresářovou položku souborového systému. Proces spí na datové struktuře `buffer_head` až do doby, než jej ovladač zařízení probudí. Datová struktura `request` se označí jako volná, takže ji bude možno použít pro uložení dalšího požadavku na blokové zařízení.

8.5 Pevné disky

Pevné disky představují trvalou metodu uložení dat, která ukládají na rotující diskové povrchy. Při zápisu dat malinká hlavička zmagnetizuje nepatrnou část magnetického povrchu. Při čtení dat hlavička detekuje, zda je příslušný kousíček povrchu zmagnetován nebo ne.

Disková jednotka se skládá z jednoho nebo více kotoučů, které jsou vyrobeny z leštěného skleněného nebo keramického materiálu a jsou pokryty tenkou vrstvičkou oxidu železa. Jednotlivé kotouče jsou připevněny ke společné ose a rotují konstantní rychlostí, která může být od 3 000 do 10 000 ot./min. podle typu disku. Srovnejte tuto rychlost s disketou, která se otáčí rychlostí 360 ot./min. Čtecí/zápisová hlavička zodpovídá za čtení a zápis dat na povrch kotouče, pro každý kotouč jsou dvě hlavičky, každá pracuje na jednom povrchu. Hlavičky se fyzicky nedotýkají povrchu kotouče, vznášejí se na vzduchovém polštáři ve vzdálenosti kolem několika nanometrů. Vystavovací rameno zajišťuje pohyb hlaviček nad povrchem kotouče. Všechny hlavičky jsou vystavovány společně, pohybují se nad kotouči vždy stejně.

Každý povrch je rozdělen na soustředné kruhové prstence zvané *stopy*. Stopa 0 je nejblíže vnějšímu okraji povrchu, stopa s nejvyšším číslem je nejblíže ke středu povrchu. *Cylindr* je skupina stop se stejným číslem. Takže například všechny páté stopy na obou stranách všech kotoučů se označují jako pátý cylindr. Protože počet cylindrů je stejný jako počet stop, geometrie disku se často popisuje v cylindrech. Každá stopa je rozdělena na *sektory*. Sektor je nejmenší datová jednotka, kterou je možno z disku najednou přečíst nebo zapsat a odpovídá tak velikosti bloku. Sektory jsou obvykle velké 512 bajtů a jejich velikost zpravidla nastavuje při fyzickém formátování disku jeho výrobce.

Disky se většinou popisují svou geometrií; počtem cylindrů, hlaviček a sektorů. Například v době zavádění může Linux popisovat jeden z disků IDE takto:

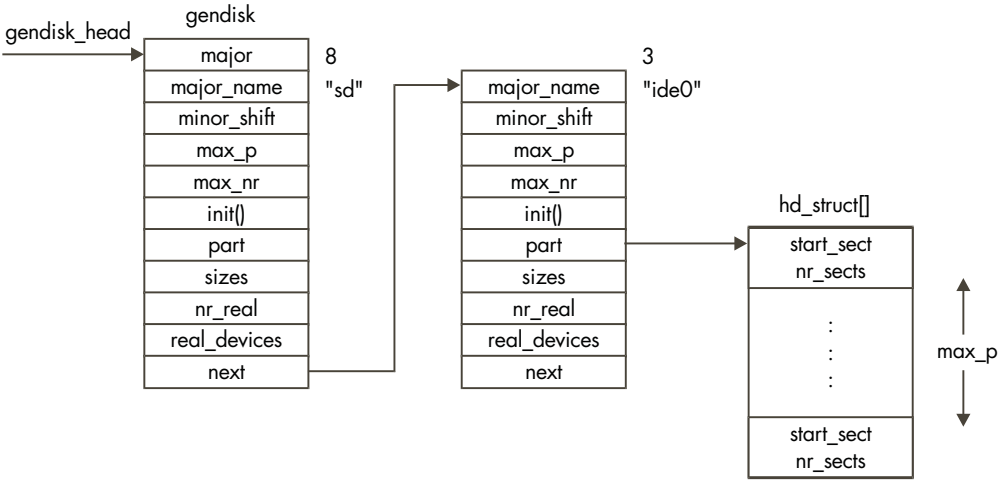
```
hdb: Conner Peripherals 540MB - CFS540A, 516MB w/64kB Cache,
  CHS=1050/16/63
```

Znamená to, že disk má 1 050 cylindrů (stop), 16 hlaviček (8 kotoučů) a 63 sektorů na každé stopě. Při velikosti sektoru 512 bajtů nám to dává celkovou kapacitu disku 529 200 bajtů. To ovšem neodpovídá hlášené velikosti 516 MB, protože některé sektory se používají k uložení rozdělovacích informací. Některé disky automaticky nalézají vadné sektory a reindexují disk tak, aby se vyhnul jejich použití.

Pevné disky se dále dělí na *oblasti*. Oblast je velká oblast sektorů, přidělená pro určitý účel. Rozdělením operačního disku na oblasti je možno tento disk používat několika operačními systémy pro různé účely. Řada linuxových systémů používá jeden disk se třemi oblastmi: na jedné je souborový systém DOS, na druhé systém ext2 a třetí slouží jako odkládací oblasti. Rozdělení disku na oblasti je popsáno rozdělovací tabulkou (partition table), každá položka této tabulky popisuje začátek a konec jedné oblasti v číslech hlaviček, sektorů a cylindrů. U disků formátovaných systémem DOS, tedy rozdělených příkazem `fdisk`, existují čtyři primární diskové oblasti. Ne všechny čtyři položky rozdělovací tabulky musejí být použity. `fdisk` podporuje tři typy partic: primární, rozšířené a logické. Rozšířené oblasti nejsou oblastmi v pravém slova smyslu, obsahují pouze několik logických oblastí. Rozšířené a logické oblasti se používají jako metoda obejití omezení pouhých čtyř primárních oblastí. Následující výpis příkazu `fdisk` pochází z disku, který je rozdělen na dvě oblasti:

```
Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders
Units = cylinders of 2048 * 512 bytes

   Device Boot    Begin    Start    End   Blocks   Id   System
/dev/sda1             1        1    478   489456   83   Linux native
/dev/sda2           479      479    510    32768   82   Linux swap

Expert command (m for help): p

Disk /dev/sda: 64 heads, 32 sectors, 510 cylinders

Nr   AF  Hd  Sec  Cyl   Hd   Sec   Cyl   Start    Size   ID
 1   00   1    1    0   63    32   477      32  978912   83
 2   00   0    1  478   63    32   509  978944   65536   82
 3   00   0    0    0    0     0     0       0       0   00
 4   00   0    0    0    0     0     0       0       0   00
```

Vidíme, že první oblast začíná na cylindru 0, hlavičce 1 a sektoru 1 a končí na cylindru 477, sektoru 32 a hlavičce 63. Protože na stopě je 32 sektorů a disk má 64 hlaviček, tato oblast leží na celých prvních 478 cylindrech disku[1]. Program `fdisk` implicitně zarovnává oblasti na hranice celých cylindrů. Začíná na nejvnějším cylindru (0) a pokračuje směrem ke středu disku až na cylindr 477. Druhá oblast, odkládací oblast, začíná na následujícím cylindru (478) a pokračuje až k nejvnitřnímu cylindru disku.



Obrázek 8.3
Seznam disků

V době inicializace Linux zmapuje topologii pevných disků v systému. Zjistí, kolik disků a jakého typu systém obsahuje. Dále Linux zjistí, jak jsou jednotlivé disky rozděleny na oblasti. Všechny tyto údaje se ukládají jako seznam datových struktur `gendisk`, na nějž ukazuje ukazatel `gendisk_head`. Když se inicializují jednotlivé diskové subsystémy, například IDE, generují se datové struktury `gendisk` reprezentující nalezené disky. Děje se to ve stejné době, kdy ovladač registruje souborové operace a přidává záznam o sobě do struktury `blk_dev`. Každá datová struktura `gendisk` má jednoznačné hlavní číslo zařízení, které odpovídá hlavnímu číslu blokového zařízení. Například subsystém SCSI vytvoří jednu položku `gendisk` („sd“) s hlavním číslem 8, hlavním číslem všech diskových zařízení SCSI. Na obrázku 8.3 vidíme dvě položky `gendisk`, první pro subsystém SCSI a druhou pro disk IDE. To je položka `ide0`, primární disk IDE.

[1] Pozn. překl.: Součástí první oblasti není pouze sektor 1 na 0. cylindru a 0. povrchu, tento sektor (úplně první sektor disku) je takzvaným *master zaváděcím sektorem* a je na něm (mimo jiné) uložena právě rozdělovací tabulka disku.

I když diskové subsystémy při své inicializaci budují struktury `gendisk`, ke hledání oblastí je používá pouze Linux. Každý diskový subsystém si navíc vytváří své vlastní struktury, které mu slouží k mapování hlavního a vedlejšího čísla zařízení na příslušné oblasti fyzických disků. Vždy, když se zapisuje nebo čte na nebo z blokového zařízení, ať už přes buffery nebo souborovou operací, jádro směruje operaci na příslušný ovladač pomocí hlavního čísla zařízení, které nalezne v souboru ovladače zařízení (například */dev/sda2*). Až samotný ovladač disku nebo diskový subsystém pak mapuje vedlejší číslo zařízení na konkrétní fyzický disk a oblastí.

8.5.1 Disky IDE

Na linuxových systémech se dnes nejčastěji používají disky IDE (Integrated Disk Electronic). IDE je diskové rozhraní, v podstatě vstupně/výstupní sběrnice podobně jako SCSI. Každý řadič IDE může ovládat dva disky, jeden v režimu *master*, druhý v režimu *slave*. Rozlišení disku master a slave se obvykle provádí přepínači přímo na disku. První řadič IDE v systému se označuje jako primární, druhý jako sekundární a tak dále. IDE zvládá datový přenos rychlostí kolem 3,3 MB za sekundu, maximální velikost disku IDE může být 538 MB. Extended IDE, (EIDE) povoluje disky o velikosti až 8,6 GB a přenosová rychlost se zvýšila na 16,6 MB za sekundu. Disky IDE a EIDE jsou levnější než disky SCSI a většina moderních PC má integrován jeden nebo více řadičů IDE.

Linux disky IDE pojmenovává v tom pořadí, v jakém nalezne jejich řadiče. Disk master na primárním řadiči se označuje jako `/dev/hda`, disk slave bude `/dev/hdb`. `/dev/hdc` je disk master na sekundárním řadiči IDE. Subsystém IDE registruje v jádře řadiče IDE, *ne* disky. Hlavní číslo primárního řadiče IDE je 3, hlavní číslo sekundárního řadiče je 22. Znamená to, že pokud má systém dva řadiče IDE, budou ve vektorech `blk_dev` a `blkdevs` na indexech 3 a 22 položky subsystému IDE. Soubory zařízení disků IDE toto označení reflektují, takže soubory `/dev/hda` i `/dev/hdb` budou mít jako hlavní číslo zařízení uvedeno 3. Všechny souborové nebo bufferové operace nad těmito blokovými zařízeními se předají subsystému IDE, protože jádro používá jako index hlavní číslo zařízení. Po vznesení požadavku už je starostí subsystému IDE aby zjistil, kterého disku se požadavek týká. K tomu používá IDE subsystém vedlejší číslo zařízení, které obsahuje informace potřebné ke směrování požadavku na správnou partici správného disku. Identifikátory zařízení pro `/dev/hdb`, disk slave na primárním řadiči IDE, jsou *(3,64)*. Identifikátor první oblast tohoto disku (`/dev/hdb1`) je *(3,65)*.

8.5.2 Inicializace subsystému IDE

Disky IDE existují prakticky po celou dobu existence počítačů IBM PC. V průběhu této doby se jejich rozhraní měnilo, takže inicializace subsystému IDE je složitější, než to vypadá na první pohled.

Linux je schopen podporovat maximálně čtyři řadiče IDE. Každý řadič je reprezentován strukturou `ide_hwif_t` ve vektoru `ide_hwifs`. Každá struktura `ide_hwif_t` obsahuje dvě struktury `ide_drive_t`, pro případný disk master a slave u řadiče. V době inicializace subsystému IDE Linux nejprve zjišťuje, zda informace o discích nejsou přístupny v paměti CMOS. Tato baterií zálohovaná paměť zachovává svůj obsah i po vypnutí počítače. Paměť se fyzicky nachází v hodinách reálného času, které běží bez ohledu na to, zda počítač je či není zapnutý. Obsah paměti CMOS je nastavován BIOSem a může Linuxu říci, jaké disky a řadiče byly nalezeny. Linux převezme od BIOSu údaje o geometrii disků a uloží je do datové struktury `ide_hwif_t`. Většina moderních PC používá chipset PCI, například Intel 82430 VX, který obsahuje i řadič PCI EIDE. Subsystém IDE používá volání PCI BIOSu k nalezení řadičů PCI (E)IDE v systému. Pak volá specifické rutiny daného chipsetu.

Jakmile je nalezeno rozhraní IDE či řadič, nastaví se jeho struktura `ide_hwif_t` tak, aby obsahovala informace o discích, připojených k tomuto řadiči. V době práce zapisuje ovladač příkazy IDE do příkazového registru řadiče IDE, který je k dispozici ve V/V adresovém prostoru. Implicitní V/V adresa pro řídící a stavové registry primárního řadiče IDE je v rozsahu *0x1F0 - 0x1F7*. Tyto adresy jsou dány konvencí už od dob prvních PC. Ovladač IDE registruje všechny řadiče ve vyrovnávací paměti bufferů a ve VFS zápisem do vektorů `blk_dev` a `blkdevs`. Ovladač IDE musí rovněž požádat o převzetí příslušného přerušení. Hodnoty přerušení jsou rovněž dány konvencí a je to 14 pro primární řadič IDE a 15 pro sekundární řadič. Stejně jako další konfigurační podrobnosti je však možno tyto hodnoty změnit. Pro každý řadič IDE nalezený v době startu systému přidává dále ovladač záznam IDE `gendisk` do seznamu těchto struktur. Tento seznam se později používá k nalezení rozdělovacích tabulek všech nalezených pevných disků. Kód pro kontrolu partic ví, že každý řadič IDE může mít připojeny dva disky.

8.5.3 SCSI disky

Sběrnice SCSI (Small Computer System Interface) je výkonná datová sběrnice, která podporuje až osm zařízení na jedné sběrnici včetně jednoho nebo více hostitelů. Každé zařízení musí mít jednoznačné identifikační číslo, které se obvykle nastavuje přepínači přímo na zařízení. Data je možno přenášet synchronně nebo asynchronně mezi libovolnými dvěma zařízeními na sběrnici s šířkou přenosu 32 bitů a maximální přenosovou rychlostí 40 MB za sekundu. Sběrnice SCSI přenáší mezi zařízeními jak data, tak stavové informace. Každá transakce mezi *iniciátorem* a *cílem* se může dělit na osm oddělených fází. Aktuální fázi sběrnice SCSI je možno určit z pěti řídících signálů na sběrnici. Uveďme si přehled jednotlivých fází:

BUS FREE Žádné zařízení nemá řízení nad sběrnicí a momentálně neprobíhají žádné transakce.

ARBITRATION Zařízení SCSI se pokouší získat řízení nad sběrnicí. Na adresové piny vysílá své identifikační číslo. Řízení získá zařízení s nejvyšším identifikačním číslem.

SELECTION Když zařízení získá řízení nad sběrnicí, signalizuje nyní cílové zařízení, kterému chce vyslat příkaz. Na adresové piny vysílá identifikátor cílového zařízení.

RESELECTION V průběhu zpracování žádosti se mohou zařízení odpojit. Cíl požadavku pak opětně vybírá iniciátora. Tuto fázi nepodporují všechna zařízení SCSI.

COMMAND Iniciátor cíli předává 6, 10 nebo 12 bajtový příkaz.

DATA IN, DATA OUT V průběhu těchto fází dochází k výměně dat mezi iniciátorem a cílem.

STATUS Do této fáze se přechází po dokončení všech příkazů a cíl nyní může iniciátorovi poslat stavový bajt indikující úspěch či neúspěch operace.

MESSAGE IN, MESSAGE OUT
Doplňující informace posílané mezi iniciátorem a cílem.

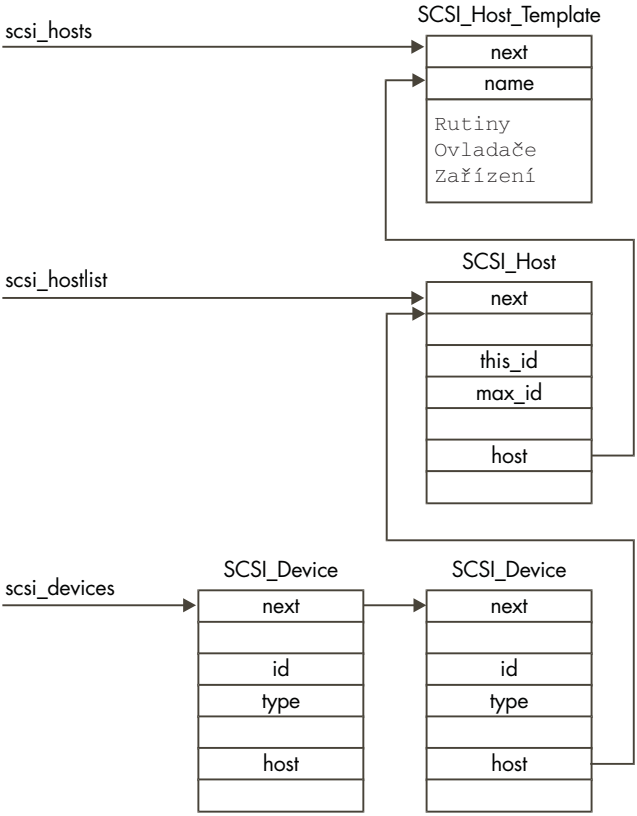
Subsystém SCSI v Linuxu se skládá ze dvou základních prvků, každý z nich je reprezentován svou datovou strukturou.

Hostitel Hostitel SCSI je fyzické hardwarové zařízení, řadič SCSI. Příkladem může být řadič NCR810 PCI SCSI. Pokud by v systému bylo více stejných řadičů SCSI, každý by byl reprezentován jako samostatný hostitel. Znamená to, že ovladač zařízení SCSI může řídit více než jeden řadič. Hostitelé SCSI ve většině případů fungují v roli iniciátora příkazu.

Zařízení Nejběžnějším zařízením SCSI je disk SCSI, nicméně standard SCSI podporuje i jiná zařízení, například pásky, CD-ROM a také obecná zařízení SCSI. Zařízení SCSI ve většině případů fungují jako cíle příkazů SCSI. K různým zařízením je nutno se chovat různě, například u zařízení s výměnnými médii jako jsou pásky a CD-ROM je nutno detekovat, zda je médium vloženo. Různé typy zařízení mají různá hlavní čísla zařízení, takže Linux může správně obsluhovat blokové požadavky.

Inicializace subsystému SCSI

Inicializace subsystému SCSI je poměrně složitá vzhledem k dynamické povaze sběrnic SCSI a jejich zařízení. Linux inicializuje zařízení SCSI v době zavádění, nejprve nalezne všechny řadiče SCSI a pak prohlíží jejich sběrnice a zjišťuje všechna připojená zařízení. Poté provede inicializaci jednotlivých zařízení a zpřístupní je zbytku jádra prostřednictvím normálních souborových a bufferových operací. Inicializace probíhá ve čtyřech fázích:



Obrázek 8.4
Datové struktury SCSI

Nejprve Linux zjistí, které hostitelské adaptéry SCSI, řadiče, jejichž ovladače byly při sestavování jádra zahrnuty v jádře, jsou hardwarově přítomny. Každý řadič SCSI má svou položku `Scsi_Host_Template` ve vektoru `builtin_scsi_hosts`. Datová struktura `Scsi_Host_Template` obsahuje ukazatele na rutiny poskytující služby specifické pro daný řadič, například detekci zařízení připojených ke sběrnici. Tyto rutiny volá subsystém SCSI v době své inicializace a jsou součástí ovladače zařízení SCSI daného typu. Každému detekovanému řadiči SCSI, tedy tomu, který je k systému skutečně připojen, se struktura

Scsi_Host_Template přidá do seznamu aktivních řadičů SCSI scsi_hosts. Každý detekovaný řadič je dále reprezentován strukturou Scsi_Host v seznamu scsi_hostlist. Například systém se dvěma řadiči NCR810 PCI SCSI vytvoří dvě položky Scsi_Host, jednu pro každý řadič. Každá struktura Scsi_Host ukazuje na strukturu Scsi_Host_Template, která reprezentuje ovladač zařízení.

V této fázi jsou tedy známy všechny řadiče SCSI. Subsystém SCSI musí dále zjistit, jaká zařízení jsou připojena ke sběrnicím jednotlivých řadičů. Zařízení SCSI se číslují čísly 0 až 7 včetně, číslo každého zařízení (identifikátor SCSI) musí být na sběrnici jedinečné. Identifikátor se obvykle nastavuje přepínači přímo na zařízení. Inicializační kód nalezne zařízení SCSI na sběrnici tím, že mu posílá příkaz TEST_UNIT_READY. Když zařízení odpoví, přečte se jeho identifikace zasláním příkazu ENQUIRY. Takto Linux získá jméno výrobce, označení modelu a verze. Příkazy SCSI jsou reprezentovány datovou strukturou Scsi_Cmnd a předávají se ovladači příslušného řadiče SCSI voláním rutin ovladače, jejichž adresy jsou uvedeny v jeho struktuře Scsi_Host_Template. Každé nalezené zařízení SCSI se reprezentuje strukturou Scsi_Device, která vždy ukazuje na svůj rodičovský Scsi_Host. Všechny struktury Scsi_Device se přidávají do seznamu scsi_devices. Na obrázku 8.4 vidíme, jak spolu hlavní datové struktury souvisejí.

Existují čtyři typy zařízení SCSI: disky, pásky, CD a obecná zařízení. Každý z těchto typů se v jádře registruje zvlášť jako blokové zařízení s rozdílným hlavním číslem. Registrují se ovšem pouze v případě, že je alespoň jedno zařízení daného typu připojeno. U každého typu, řekněme u disků, se udržuje samostatná tabulka zařízení. Tato tabulka slouží ke směrování blokových operací jádra na správný ovladač zařízení a SCSI řadič. Každý typ zařízení SCSI je reprezentován datovou strukturou Scsi_Device_Template. Ta obsahuje informace o typu zařízení a adresy rutin, které zajišťují různé operace. Subsystém SCSI používá tyto struktury k volání rutin příslušného typu zařízení pro každé zařízení SCSI. Jinak řečeno, pokud chce SCSI subsystém pracovat s diskem SCSI, bude volat rutiny pro typ diskové zařízení. Datové struktury Scsi_Type_Template se přidávají do seznamu scsi_devicelist pokud bylo detekováno jedno nebo více zařízení daného typu.

V závěrečné fázi inicializace subsystému SCSI se volají dokončovací rutiny pro každou registrovanou strukturu Scsi_Device_Template. U diskových typů dojde k roztočení všech disků a k zjištění jejich geometrie. Dále se přidává datová struktura gendisk reprezentující všechny disky SCSI do seznamu disků, který jsme viděli na obrázku 8.3.

Doručení požadavků blokovým zařízením

Jakmile proběhne inicializace subsystému SCSI, je možno zařízení SCSI používat. Každý aktivní typ zařízení SCSI se v jádře registruje, takže Linux na něj může směrovat požadavky na bloková zařízení. Může se jednat o bufferové operace přes struktury blk_dev nebo o soubo-

rové operace přes struktury `blkdevs`. Vezměme si jako příklad ovladač disku SCSI, na němž jsou jedna nebo více oblastí se souborovým systémem ext2. Jakým způsobem jádro zajistí předání požadavků na správný disk SCSI, na němž je připojená příslušná oblast ext2?

Každý požadavek na zápis nebo čtení bloku dat na nebo z diskové oblasti SCSI vede k vytvoření nové struktury `request`, která se přidává do seznamu `current_request` disků SCSI ve vektoru `blk_dev`. Pokud se seznam žádostí zpracovává, nemusí vyrovnávací paměť bufferů udělat nic dalšího, v opačném případě musí „postrčit“ subsystém SCSI, aby začal požadavky ve frontě zpracovávat. Každý disk SCSI v systému je reprezentován strukturou `Scsi_Disk`. Ty se udržují ve vektoru `rscsi_disk`, který je indexován pomocí vedlejšího čísla zařízení oblastí disků SCSI. Například zařízení `/dev/sdb1` má hlavní číslo 8 a vedlejší číslo 17, z něhož se generuje index 1. Každá datová struktura `Scsi_Disk` obsahuje ukazatel na strukturu `Scsi_Device`, která reprezentuje toto zařízení. Ta dále ukazuje na strukturu `Scsi_Host`, která je „vlastníkem“ zařízení. Datová struktura `request` vytvořená vyrovnávací pamětí bufferů se přeloží na strukturu `Scsi_Cmd`, která udává, jaký příkaz se musí zařízení SCSI poslat a tento příkaz se uloží do fronty ve struktuře `Scsi_Host`, reprezentující příslušné zařízení. Tuto frontu zpracovává ovladač zařízení SCSI, který zajistí splnění požadavku na čtení nebo zápis.

8.6 Síťová zařízení

Síťová zařízení jsou, z pohledu síťového subsystému Linuxu, entity, které posílají a přijímají datové pakety. Obvykle se jedná o fyzická zařízení, například o síťovou kartu. Některá síťová zařízení jsou nicméně pouze logická zařízení, například zpětné zařízení, které slouží k posílání paketů sobě sama. Každé síťové zařízení je reprezentováno strukturou `device`. Ovladače síťových zařízení registrují svá zařízení v Linuxu v době inicializace sítě při zavádění jádra. Datová struktura `device` obsahuje informace o zařízení a adresy funkcí, které umožňují různým podporovaným síťovým protokolům použít služeb zařízení. Tyto funkce se většinou týkají odesílání dat prostřednictvím síťového zařízení. Zařízení používá standardní podpůrné mechanismy sítě k předání doručených dat příslušné protokolové vrstvě. Všechna síťová data (pakety), odeslaná i přijatá, jsou reprezentována strukturami `sk_buff`, což jsou pružné datové struktury, které umožňují snadno přidávat a odebírat hlavičky síťových protokolů. Mechanismy, jimiž protokoly různých síťových vrstev používají síťová zařízení a mechanismy, jimiž se data prostřednictvím struktur `sk_buff` předávají tam a zpět, jsou podrobně popsány v kapitole „Sítě“. V této kapitole se soustředíme na datovou strukturu `device` a na detekci a inicializaci síťových zařízení.

Datová struktura `device` obsahuje následující informace o síťovém zařízení:

jméno

Na rozdíl od znakových a blokových zařízení, jejichž soubory zařízení se vytvářejí příkazem `mknod`, soubory síťových zařízení se vytvářejí samy při detekci a inicializaci síťových zařízení. Jména těchto souborů jsou standardní, každé z nich reprezentuje své zařízení. Více zařízení stejného typu se čísluje od nuly nahoru. Takže oblasti IDE disku se prezentují jako `/dev/hda1`, `/dev/hda2`, `/dev/hda3` a tak dále. Uveďme si jména některých běžných síťových zařízení:

`/dev/slN`	Zařízení SLIP
`/dev/pppN`	Zařízení PPP
`/dev/lo`	zpětná zařízení

sběrnicové informace

Tyto informace ovladač zařízení potřebuje, aby mohl zařízení řídit. *irq* je číslo přerušení, které zařízení používá. *base address* je bázová adresa řídících a stavových registrů zařízení ve V/V adresovém prostoru. *DMA channel* je číslo kanálu DMA, který zařízení používá. Všechny tyto informace se nastavují při zavádění, kdy dochází k inicializaci zařízení.

příznaky rozhraní

Tyto příznaky popisují vlastnosti a možnosti síťových zařízení:

IFF_UP	Rozhraní je aktivní a běží.
IFF_BROADCAST	Ve struktuře `device` je platná adresa broadcast.
IFF_DEBUG	Aktivován ladicí režim zařízení.
IFF_LOOPBACK	Jedná se o zpětné zařízení.
IFF_POINTTOPOINT	Jedná se o dvoubodové zařízení (PPP nebo SLIP).
IFF_NOTRAILERS	Zařízení nepodporuje trailery paketů.
IFF_RUNNING	Byly alokovány prostředky.
IFF_NOARP	Zařízení nepodporuje protokol ARP.
IFF_PROMISC	Zařízení je v promiskuitním režimu, přijímá všechny pakety bez ohledu na to, komu byly adresovány.
IFF_ALLMULTI	Zařízení přijímá všechny hromadně vysílané IP-rámce.
IFF_MULTICAST	Zařízení je schopno přijímat hromadně vysílané IP-rámce.

protokolové informace Popisují, jak může být toto zařízení použito různými protokolovými vrstvami:

mtu Maximální velikost paketu, který toto zařízení dokáže odeslat, v této hodnotě není započtena velikost hlavičky linkové vrstvy, kterou zařízení bude muset přidat. Podle této hodnoty volí síťové vrstvy, například IP, velikost odesílaných paketů.

family Tato hodnota udává, jakou protokolovou rodinu zařízení podporuje. U všech síťových zařízení v Linuxu je podporována rodina AF_INET, rodina internetových protokolů.

type Typ hardwarového rozhraní říká, k jakému typu média je zařízení připojeno. Síťová zařízení Linuxu mohou podporovat řadu různých fyzických médií, například Ethernet, X.25, Token Ring, SLIP, PPP a Apple Localtalk.

adresy Ve struktuře `device` jsou uložena čísla adres, které se vztahují k tomuto zařízení, včetně například IP adres.

fronta paketů Fronta struktur `sk_buff`, tedy paketů, které čekají na odeslání tímto zařízením.

podpůrné funkce Každé zařízení poskytuje standardní skupinu funkcí, které mohou protokolové vrstvy používat jako rozhraní k linkové vrstvě daného zařízení. Patří sem rutiny pro přípravu a odeslání rámce, rutiny pro připojení standardní hlavičky rámce a pro pořizování statistik. Statistiky zařízení je možno zjistit příkazem `ifconfig`.

8.6.1 Inicializace síťových zařízení

Ovladače síťových zařízení mohou být, stejně jako ovladače jiných zařízení, vestavěny přímo do jádra. Každé potenciální síťové zařízení je reprezentováno datovou strukturou `device` v seznamu síťových zařízení, na nějž ukazuje ukazatel `dev_base`. Pokud síťové vrstvy potřebují provést nějakou pro zařízení specifickou operaci, volají některou z řady síťových služebních rutin, jejichž adresy jsou uloženy ve struktuře `device`. Na počátku obsahuje každá datová struktura `device` pouze adresu inicializační rutiny.

S ovladači síťových zařízení je nutné řešit dva problémy. První problém je v tom, že ne pro každý ovladač vestavěný do jádra musí být skutečně přítomno síťové zařízení. Druhým problémem je, že ethernetová zařízení se vždy označují jako `eth0`, `eth1` a tak dále, bez ohledu na to, jaký ovladač fyzického zařízení se pro ně používá. Problém „chybějících“ síťových zařízení se řeší velmi jednoduše. Když se volá inicializační rutina každého zařízení, vrátí

příznak, který říká, zda zařízení je či není přítomno. Pokud ovladač nenalezne žádné „své“ zařízení, odstraní se jeho položka `device` ze seznamu `dev_base`. Pokud ovladač nalezne zařízení, vyplní zbytek datové struktury `device` informacemi o zařízení a adresami podpůrných funkcí ovladače.

Druhý problém, problém dynamického mapování ethernetových zařízení `ethN`, se řeší elegantněji. V seznamu zařízení je standardně osm položek: `eth0`, `eth1` a tak dále až `eth7`. Inicializační rutina všech těchto zařízení je stejná, zkouší postupně všechny ovladače ethernetových karet v systému tak dlouho, dokud jeden z nich nenalezne své zařízení. Když ovladač nalezne zařízení, vyplní si svou strukturu `ethN`, kterou nyní vlastní. V této fázi ovladač zařízení rovněž provede fyzickou inicializaci zařízení a zjistí informace o použitém přerušení, DMA a podobně. Ovladač může nalézt několik instancí svého zařízení a v takovém případě převezme několik struktur `ethN`. Pokud se naplní všech osm standardních zařízení `ethN`, další ethernetová zařízení se nehledají.

Odkazy na zdrojové texty jádra

1 – `Viz fs/devices.c`

2 – `Viz include/-linux/major.h`

3 – `Viz ext2_read_inode() in fs/ext2/inode.c`

4 – `def_chr_fops`

5 – `Vizchrdev_open() in fs/devices.c`

6 – `Viz fs/devices.c`

7 – `Vizdrivers/block/-ll_rw_blk.c`

8 – `Viz include/-linux/blkdev.h`

9 – `Viz include/linux/-netdevice.h`

Kapitola

9

Souborový systém

V této kapitole popisujeme, jak Linux spravuje soubory na podporovaných souborových systémech. Popisujeme zde virtuální souborový systém (VFS) a vysvětlujeme dále podporu reálných souborových systémů.

Jednou z velkých výhod Linuxu je, že dokáže podporovat různé souborové systémy. Díky tomu je velmi pružný a dokáže spolupracovat s mnoha jinými operačními systémy. V době vzniku tohoto textu podporoval Linux 15 souborových systémů: `ext`, `ext2`, `xia`, `minix`, `umsdos`, `msdos`, `vfat`, `proc`, `smb`, `ncp`, `iso9660`, `sysv`, `hpfs`, `affs` a `ufs` a tento počet bezpochyby časem poroste.

V Linuxu, stejně jako v Unixu, se k jednotlivým samostatným souborovým systémům nepřistupuje prostřednictvím identifikátorů zařízení (jako jsou třeba čísla nebo označení diskových jednotek), namísto toho jsou všechny systémy složeny do jedné hierarchické stromové struktury, která reprezentuje celý souborový systém jako jedinou entitu. Při připojení každého souborového systému jej Linux do tohoto stromu přidá. Všechny souborové systémy libovolného typu se připojují do nějakého adresáře a soubory připojeného souborového systému překryjí původní obsah adresáře. Tento adresář se označuje jako připojovací adresář nebo také připojovací bod. Když se souborový systém odpojí, znovu se objeví původní obsah připojovacího adresáře.

Při inicializaci disku (například příkazem `fdisk`) se na něm vytváří struktura oblastí, které rozdělují jeden fyzický disk na několik logických částí. Každá oblast může nést jeden souborový systém, například `ext2`. Souborové systémy organizují soubory do logických hierarchických struktur pomocí adresářů, odkazů a podobně, všechno je uloženo v blocích na fyzickém zařízení. Souborový systém: Disková oblast `/dev/hda1` disku IDE (první oblast prvního disku IDE) je blokové zařízení. Zařízení, která mohou nést souborové systémy, se označují jako bloková zařízení. Souborové systémy v Linuxu požadují bloky uspořádané v prosté lineární ko-

lekci, nestarají se o fyzickou geometrii disku. Když souborový systém požaduje přečtení nějakého bloku, je úkolem ovladače zařízení, aby číslo bloku převedl na údaje, jimž zařízení rozumí: na stopy, sektory a povrchy. Souborový systém musí vypadat, chovat se a fungovat stejně bez ohledu na to, jaké fyzické zařízení jej hostí. Souborový systém nemusí dokonce být ani na lokálním disku, může se jednat o disk, vzdáleně připojený po síti. Podívejme se na následující příklad, kdy je kořenový souborový systém Linuxu umístěn na disku SCSI:

```
A       E       boot    etc     lib     opt             tmp     usr
C       F       cdrom   fd      proc    root            var     sbin
D       bin     dev     home    mnt     lost+found
```

Ani uživatelé, ani programy, které se soubory pracují, nepotřebují vědět, že adresář `/C` je ve skutečnosti připojený souborový systém VFAT na prvním disku IDE v systému. V tomto příkladu (je to souborový systém mého počítače) je `/E` master disk IDE na sekundárním řadiči IDE. Není vůbec důležité ani to, že primární řadič IDE je připojen sběrnicí PCI, sekundární je připojen sběrnicí ISA a ovládá také mechaniku CD-ROM. Pomocí modemu a síťového protokolu PPP se mohu připojit k síti v práci a v takovém případě si mohu souborový systém mého Linuxu v práci připojit do adresáře `/mnt/remote`.

Soubory v souborovém systému jsou kolekce dat. Text této kapitoly je například uložen v ASCII souboru pojmenovaném `filesystems.tex`. Souborový systém obsahuje jednak data uložená v souborech, ukládá si však i svou vlastní strukturu. Udržuje všechny informace, které uživatelé a procesy vnímají jako soubory, adresáře, odkazy, informace o přístupových právech a další. Navíc musí všechny tyto informace ukládat bezpečně, protože základní integrita celého operačního systému závisí právě na souborovém systému. Nikdo nebude používat souborový systém, kde se data a soubory náhodně ztrácejí.

`Minix`, první souborový systém používaný Linuxem, byl dost omezující a málo výkonný.

Názvy souborů mohly mít maximálně 14 znaků (což je ovšem pořád lepší než konvence 8.3) a velikost souborů byla omezena na 64 MB. Limit 64 MB vypadá sice na první pohled jako dostatečný, i skromné databáze však bývají větší. První souborový systém, vyvinutý speciálně pro Linux, byl Extended File System, `ext`, byl uveden v dubnu 1992 a i když řadu problémů vyřešil, jeho výkon byl stále nedostačující.

V roce 1993 se pak objevil Second Extended File System, `ext2`. Právě tento souborový systém si nyní podrobně popíšeme.

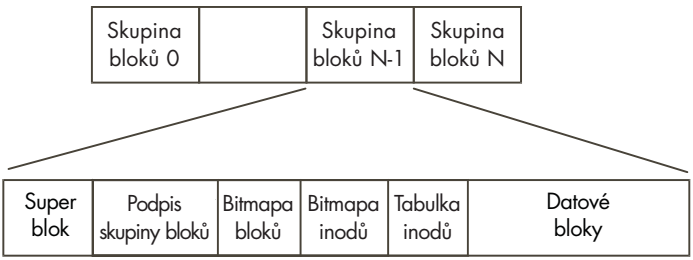
Když byl Linux doplněn o souborový systém `ext`, došlo ještě k další významné změně. Reálné souborové systémy byly odděleny od operačního systému a systémových služeb vrstvou, zvanou virtuální souborový systém, VFS.

VFS umožňuje podporu řady často velmi rozdílných souborových systémů, z nichž každý poskytuje na stranu VFS jednotné programové rozhraní. Všechny podrobnosti jednotlivých souborových systémů jsou softwarově zakryty, takže zbytku jádra Linuxu a všem programům v systému se všechny souborové systémy jeví stejně. Vrstva VFS umožňuje transparentně připojit mnoho různých souborových systémů najednou.

Virtuální souborový systém Linuxu je navržen tak, aby přístup k souborům byl co nejrychlejší a nejefektivnější. Je třeba dále zajistit, aby soubory a data byly uloženy správně. Tyto dva požadavky se mohou vzájemně vylučovat. VFS ukládá v paměti informace o každém připojeném a používaném souborovém systému. Je nutné věnovat velkou pozornost správné aktualizaci souborového systému, když dojde k modifikaci dat ve vyrovnávacích pamětech při vytváření, zápisu a rušení souborů a adresářů. Pokud byste mohli vidět datové struktury souborového systému v běžícím jádře, viděli byste zapisované a čtené bloky dat. Když ovladače zařízení pracují a načítají a zapisují data, stále dochází k vytváření a rušení datových struktur, reprezentujících adresáře a soubory. Nejdůležitější ze všech těchto vyrovnávacích pamětí je vyrovnávací paměť bufferů, která je integrována do mechanismů, jimiž jednotlivé souborové systémy přistupují ke svým blokovým zařízením. Když se přistupuje k jednotlivým blokům, ukládají se ve vyrovnávací paměti bufferů do různých front, které reprezentují jejich stav. Vyrovnávací paměť bufferů neuchovává pouze datové buffery, usnadňuje rovněž správu asynchronních rozhraní ovladačů blokových zařízení.

9.1 Souborový systém ext2

Souborový systém `ext2` byl navržen (Rémy Cardem) jako rozšiřitelný a výkonný souborový systém speciálně po Linux. Doposud je to nejoblíbenější souborový systém celé komunity Linuxu a je součástí všech dnešních distribucí Linuxu.



Obrázek 9.1
Fyzické rozvržení souborového systému ext2

Souborový systém `ext2`, stejně jako řada jiných souborových systémů, je vystavěn na předpokladu, že data ukládaná v souborech jsou uložena v datových blocích. Tyto datové bloky jsou stejně dlouhé a i když v různých souborových systémech ext2 mohou být délky odlišné,

nastavují se jednou provždy při vytváření souborového systému (příkazem `mke2fs`). Velikost každého souboru je zaokrouhlena nahoru na celý násobek velikosti bloku. Pokud má blok velikost 1 024 bajtů, pak bude soubor o délce 1 025 bajtů zabírat dva tyto bloky. Bohužel to znamená, že průměrně s každým souborem přicházíme o půl datového bloku. Linux ovšem v tomto případě, stejně jako většina jiných operačních systémů, dává přednost relativně neefektivnímu využití disku před větším zatížením CPU. Ne všechny bloky v souborovém systému slouží k ukládání dat, některé musejí obsahovat informace, popisující strukturu souborového systému. Systém `ext2` definuje topologii souborového systému popisem každého souboru prostřednictvím datové struktury *inode*. Inode udává, ve kterých blocích je soubor uložen, stejně jako přístupová práva k souboru, časy modifikace souboru a typ souboru. Každý soubor v souborovém systému `ext2` je popsán právě jedním inodem a každý inode má své jednoznačné identifikační číslo. Inody celého souborového systému jsou uloženy pohromadě v tabulce inodů. Adresáře v systému `ext2` jsou jen speciální soubory (rovněž popsané inodem), které obsahují ukazatele na inody popisující položky v daném adresáři.

Na obrázku 9.1 vidíme strukturu, jakou systém `ext2` obsazuje bloky na blokově organizovaném zařízení. Z pohledu souborového systému představují bloková zařízení pouze lineární posloupnost bloků, které je možno číst a zapisovat. Souborový systém se nepotřebuje zabývat umístěním jednotlivých bloků na fyzickém médiu, to je starost ovladače zařízení. Kdykoliv souborový systém potřebuje přečíst informace nebo data z nějakého blokového zařízení, požádá příslušný ovladač zařízení o načtení celočíselného počtu bloků. Souborový systém ext2 rozděluje logickou oblast na níž je vytvořen na *skupiny bloků*.

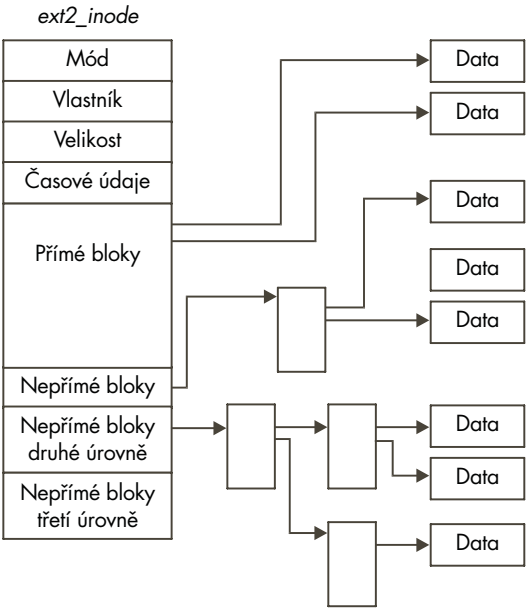
Každá skupina duplikuje informace kritické pro integritu operačního systému a dále obsahuje skutečné soubory a adresáře jako bloky informací a dat. Tato duplikace je nezbytná pro případ havárie systému, kdy je nutná jeho obnova. V následujících částech je podrobněji vysvětlen obsah skupin bloků.

9.1.1 Inode

V souborovém systému `ext2` představuje inode základní stavební kámen: každý soubor a adresář v systému je popsán právě jedním inodem. Inody souborového systému `ext2` jsou pro každou skupinu bloků udržovány v tabulce inodů společné s bitovou mapou, která nese informace o alokovaných a nealokovaných inodech. Na obrázku 9.2 vidíme strukturu inode systému `ext2`. Kromě dalšího obsahuje inode následující informace:

2

mód Toto pole obsahuje dvě informace: co daný inode popisuje a přístupová práva, která uživatelé k danému objektu mají. V systému ext2 může inode popisovat soubor, adresář, symbolický odkaz, blokové zařízení, znakové zařízení nebo FIFO.



Obrázek 9.2
Inode systému ext2

informace o vlastníkovi Uživatelský a skupinový identifikátor vlastníka souboru nebo adresáře. Díky těmto údajům může souborový systém správně řídit přístup k objektu.

velikost Velikost souboru v bajtech.

časové značky Čas, kdy byl inode vytvořen a kdy byl naposledy modifikován.

datové bloky Ukazatel na blok dat obsahující data, která tento inode popisuje. Prvních dvanáct jsou ukazatelé na fyzické bloky obsahující data objektu, popsaného tímto inodem, poslední tři ukazatele zavádějí stále větší a větší nepřímo adresované objemy dat. Například ukazatel na dvojitý nepřímý ukazatel ukazuje na blok ukazatelů na bloky ukazatelů na datové bloky. Znamená to, že k souborům o délce dvanáct bloků a méně se přistupuje snáze a rychleji, než k souborům delším.

Je nutné si uvědomit, že inody systému `ext2` mohou popisovat rovněž speciální soubory zařízení. To nejsou skutečné soubory, ale odkazy, které mohou programy používat při přístupu

k zařízením. Všechny soubory zařízení v adresáři `/dev` slouží právě k tomu, aby programy mohly používat různá zařízení. Například program `mount` používá jako parametr jméno souboru zařízení, které se má připojit.

9.1.2 Superblok

Superblok obsahuje základní popis souborového systému. Informace uložené v superbloku používají mechanismy pro správu souborového systému při používání a údržbě systému. Obvykle se při připojení disku čte pouze superblok ve skupině bloků 0, nicméně každá skupina bloků obsahuje kopii superbloku pro případ poškození souborového systému. Mimo jiné obsahuje superblok následující informace.

kouzelné číslo (magic number)
: Podle této hodnoty připojovací program pozná, že se jedná o superblok souborového systému `ext2`. V současné verzi se používá číslo *0xEF53*.

číslo revize
: Podle hlavního a vedlejšího revizního čísla připojovací software pozná, zda se jedná o verzi, která podporuje či nepodporuje určité funkce, dostupné pouze v určitých revizích systému. Součástí těchto údajů je i údaj o kompatibilitě, říkající, které funkce je možno v této verzi bezpečně používat.

připojovací počitadlo a maximální připojovací počet
: Tyto dvě hodnoty slouží ke zjištění, zda se má provést úplná kontrola souborového systému. Připojovací počitadlo se inkrementuje při každém připojení systému a když dosáhne maximálního připojovacího počtu, objeví se hlášení „byl dosažen maximální počet připojení, spusťte `e2fsck`“.

číslo skupiny bloků
: Číslo skupiny bloků, v němž je tato kopie superbloku uložena.

velikost bloku
: Velikost bloku v této konfiguraci systému, například 1 024 bajtů.

počet bloků ve skupině
: Počet bloků ve skupině. Stejně jako u velikosti bloku, i tato hodnota se pevně nastavuje při vytváření systému.

počet volných bloků
: Počet volných bloků v souborovém systému.

počet volných inodů
: Počet volných inodů v souborovém systému.

první inode
: Číslo prvního inode souborového systému. První inode v systému `ext2` je inode kořenového adresáře systému.

9.1.3 Deskriptor skupiny

Každá skupina bloků má svůj deskriptor, který ji popisuje. Stejně jako superblok, všechny deskriptory skupin jsou duplikovány ve všech skupinách pro případ poškození souborového systému.

Každý deskriptor skupiny obsahuje následující údaje:

bloková bitmapa Číslo bloku, v němž je uložena bloková bitová mapa této skupiny. Používá se při alokaci a dealokaci bloků.

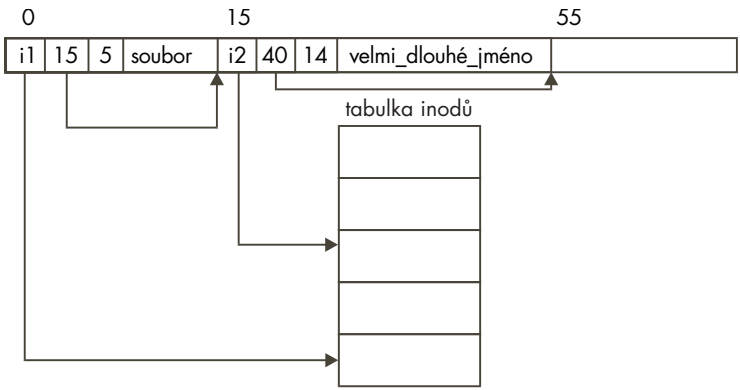
inodová bitmapa Číslo bloku, v němž je uložena bitová mapa pro alokování inodů. Používá se při alokaci a dealokaci inodů.

tabulka inodů Číslo počátečního bloku tabulky inodů této skupiny. Každý inode je reprezentován dříve popsanou datovou strukturou.

počet volných bloků, počet volných inode, počet vytvořených adresářů
Deskriptory skupin jsou uloženy jeden za druhým a dohromady vytvářejí tabulku deskriptorů skupin. Každá skupina obsahuje celou tabulku deskriptorů uloženou hned za kopií superbloku. Souborový systém `ext2` fakticky používá pouze první kopii (ve skupině 0). Ostatní kopie, stejně jako kopie superbloku, slouží pouze v případě porušení hlavní kopie.

9.1.4 Adresáře

V souborovém systému `ext2` jsou adresáře reprezentovány jako speciální soubory, které slouží k vytváření a ukládání přístupových cest k souborům v systému. Na obrázku 9.3 je vidět struktura adresářového záznamu v paměti.



Obrázek 9.3
Adresáře systému ext2

Adresářový soubor je seznam položek adresáře, z nichž každá obsahuje následující informace:

inode Inode této adresářové položky. Slouží jako index do pole inodů udržovaných v tabulce inodů skupiny bloků. Na obrázku 9.3 se adresářová položka pro soubor pojmenovaný `soubor` odkazuje na inode číslo `i1`.

délka jména Délka jména této adresářové položky v bajtech.

jméno Jméno této adresářové položky.

První dvě položky každého adresáře jsou vždy standardně „.“ a „..“ s významem „tento adresář“ a „rodičovský adresář“.

9.1.5 Nalezení souboru v systému ext2

Jména souborů v Linuxu používají stejný formát jako jména v systémech Unix. Jedná se o sérii jmen adresářů oddělených normálním lomítkem („/“) ukončená jménem souboru. Příkladem jména může být `/home/rusling/.cshrc`, kde `/home` a `/rusling` jsou jména adresářů a soubor se jmenuje `.cshrc`. Stejně jako ostatní systémy Unix se ani Linux nezajímá o formát samotného jména souboru, které může mít libovolnou délku a může se skládat z jakýchkoliv tisknutelných znaků. Když se v systému `ext2` hledá inode reprezentující určitý soubor, musí systém rozebrat jméno souboru na jednotlivé adresáře až dojde k samotnému souboru.

První potřebný inode je inode kořenového adresáře souborového systému a jeho číslo najdeme v superbloku souborového systému. Samotný inode pak nalezneme v tabulce inodů příslušné skupiny bloků. Pokud například kořen systému má inode číslo 42, musíme nalézt 42. inode z tabulky inodů nulté skupiny. Kořenový inode je inode adresáře, tedy mód tohoto inodu říká, že se jedná o adresář, a jeho datové bloky pak obsahují adresářové položky.

`home` je jen jedna z mnoha adresářových položek a tato adresářová položka nám řekne číslo inode, který popisuje adresář `/home`. Pak musíme načíst tento adresář (nejprve načteme jeho inode a poté adresářové položky z datových bloků, určených inodem) a hledáme položku `rusling`, z níž zjistíme číslo inodu popisujícího adresář `/home/rusling`. Nakonec přečteme adresářové položky určené inodem popisujícím adresář `/home/rusling` a budeme hledat číslo inode pro soubor `.cshrc` odkud se už dostaneme k datovým blokům, obsahujícím informace uložené v tomto souboru.

9.1.6 Změna velikosti souboru v systému ext2

Běžným problémem souborových systémů je jejich sklon k fragmentaci. Bloky obsahující data souboru jsou rozmístěny v celém souborovém systému a sekvenční přístup k souboru se stává tím více neefektivní, čím je rozptyl bloků větší. Souborový systém `ext2` se snaží tomuto problému předcházet tak, že nové bloky pro soubor alokuje fyzicky blízko ke stávajícím blokům, přinejmenším však ve stejné skupině bloků. Pouze pokud se mu to nezdaří, alokuje bloky z jiné skupiny bloků.

Vždy když se proces pokouší o zápis dat do souboru, souborový systém kontroluje, zda data nepřesahují za poslední blok alokovaný danému souboru. Pokud ano, pak je nutné pro soubor alokovat další blok. Dokud alokace neskončí, proces nemůže pokračovat - před pokračováním musí počkat, dokud souborový systém neprovede alokaci bloku a nezapíše do něj data. První věc, kterou alokační rutina bloku udělá, je, že zamkne superblok souborového systému. Alokace a dealokace bloků vede ke změnám údajů v superbloku a souborový systém nemůže dopustit, aby tyto změny provádělo více procesů najednou. Pokud alokaci dalšího bloku požaduje i jiný proces, musí počkat, dokud neskončí alokace pro předchozí proces. Procesy čekající na superblok jsou pozastaveny a nemohou pracovat dokud superblok nebude opět uvolněn. Přístup k superbloku se přiděluje na principu „kdo dřív přijde, ten dřív mele“ a jakmile proces jednou získá superblok, vlastní jej, dokud alokace neskončí. Po získání superbloku proces nejprve kontroluje, zda je v systému dostatek volných bloků. Pokud není dost volných bloků, požadavek na alokaci nemůže být splněn a proces se vzdává vlády nad superblokem.

Pokud v souborovém systému je dostatek volných bloků, proces se je pokusí alokovat.

6

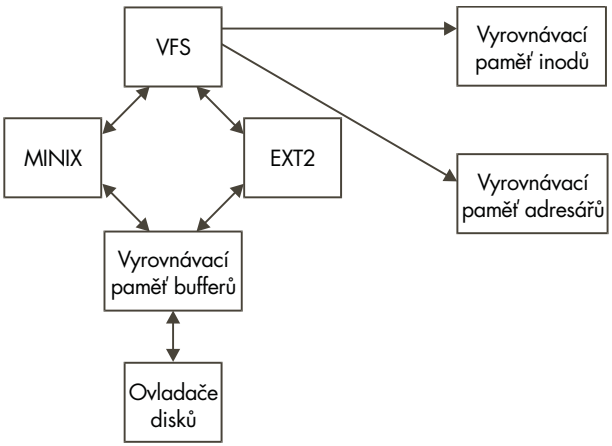
Pokud je souborový systém `ext2` nastaven tak, aby prováděl prealokaci bloků, můžeme použít jeden z prealokovaných bloků. Prealokovaný blok ve skutečnosti neexistuje, je pouze rezervován v bitové mapě alokovaných bloků. VFS inode, reprezentující soubor pro nějž se pokoušíme nový blok alokovat, obsahuje dvě položky specifické pro souborový systém `ext2`, `prealloc_block` a `prealloc_count`, což jsou čísla prvního prealokovaného bloku a jejich počet. Pokud není prealokován žádný blok nebo není prealokace bloků aktivována, musí souborový systém `ext2` provést alokaci nového bloku. Systém se nejprve pokusí alokovat blok bezprostředně za posledním alokovaným blokem souboru. Logicky je to to nejefektivnější místo, protože se tak výrazně urychluje sekvenční přístup. Pokud blok není volný, pak se hledá volný blok někde v rámci 64 následujících bloků. Takovýto blok, i když není ideální, je přinejmenším dostatečně blízko a ve stejné skupině bloků, jako ostatní bloky souboru.

Pokud není ani takovýto blok volný, pak se hledá ve všech ostatních skupinách bloků dokud se nepodaří nějakou volnou sekvenci bloků nalézt. Kód alokace bloků se pokouší ve skupinách bloků nalézt sérii osmi volných bloků. Pokud nenalezne osm volných bloků pohromadě, spokojí se i s méně. Pokud byla aktivována a použita prealokace bloků, provede se aktualizace údajů `prealloc_block` a `prealloc_count`.

Ať už bude volný blok nalezen kdekoliv, provede alokační kód aktualizaci bitové mapy alokovaných bloků příslušné skupiny bloků a příslušného datového bufferu ve vyrovnávací paměti bufferů. Datový buffer je jednoznačně identifikován identifikátorem příslušného zařízení souborového systému a číslem alokovaného bloku. Datový buffer se vynuluje a označí jako modifikovaný, aby bylo zřejmé, že jeho obsah nebyl doposud zapsán na fyzický disk. Nakonec se jako modifikovaný označí superblok, aby se vědělo, že byl změněn, a odemkne se. Pokud existují další procesy čekající na superblok, spustí se první z nich ve frontě a získá výhradní právo manipulace se superblokem pro své souborové operace. Data procesu se zapíší do nového datového bloku a až dojde k zaplnění i tohoto bloku, celý postup se opakuje a alokuje se nový datový blok.

9.2 Virtuální souborový systém (VFS)

Obrázek 9.4 znázorňuje vztah mezi virtuálním souborovým systémem jádra Linuxu a reálnými souborovými systémy. Virtuální souborový systém musí spravovat všechny souborové systémy připojené v daném okamžiku. K tomu účelu používá datové struktury popisující celý (virtuální) souborový systém a také reálné, připojené souborové systémy.



Obrázek 9.4
Logický diagram virtuálního souborového systému

7 Je poněkud zavádějící, že VFS popisuje souborové systémy pomocí superbloků a inodů prakticky stejným způsobem, jako je používá souborový systém `ext2`. Stejně jako inody v `ext2`, i inody systému VFS popisují soubory a adresáře v tomto systému, obsah a topologii virtuálního souborového systému. Od této chvíle budu, aby se předešlo zmatení pojmů, používat termíny inode VFS a superblok VFS k odlišení od jejich jmenovců v systému `ext2`.

Při inicializaci se každý souborový systém registruje ve VFS. Dochází k tomu v době, kdy probíhá inicializace samotného operačního systému při zavádění. Reálné souborové systémy jsou buď vestavěny přímo v jádře, nebo se dohrávají jako samostatné moduly. Moduly souborových systémů se nahrávají podle potřeby, takže například pokud je jako modul jádra implementován souborový systém `vfat`, nahraje se pouze v případě, že se připojí souborový systém `vfat`. Když se připojí souborový systém na blokovém zařízení, včetně kořenového souborového systému, musí VFS přečíst jeho superblok. Rutina pro načtení superbloku každého typu souborového systému musí zjistit topologii systému a namapovat tyto informace do superbloku VFS. VFS udržuje seznam připojených souborových systémů společně s jejich superbloky VFS. Každý superblok VFS obsahuje informace a ukazatele na rutiny, provádějící jednotlivé funkce. Takže například superblok reprezentující připojený systém `ext2` obsahuje ukazatel na rutinu načtení inode specifickou pro systém `ext2`. Rutina pro načtení `ext2` inodu, stejně jako rutiny pro načtení inodu jiných souborových systémů, vyplňuje údaje v inodu VFS. Každý superblok VFS obsahuje ukazatel na kořenový inode VFS příslušného souborového systému. Pro kořenový souborový systém je to inode reprezentující adresář „/“. Takovéto mapování informací je velmi efektivní pro souborový systém `ext2`, je však méně výkonné pro jiné souborové systémy.

Když procesy v systému přistupují k souborům a adresářům, volají se systémové rutiny, které procházejí inody VFS.

Například zadání příkazu `ls` pro adresář nebo `cat` pro soubor způsobí, že VFS prohledá své inody VFS, reprezentující celý souborový systém. Protože každý soubor a adresář v systému je reprezentován inodem VFS, často se opakuje přístup k určitým inodům. Tyto inody jsou uloženy ve vyrovnávací paměti inodů, takže se přístup k nim urychluje. Pokud nějaký inode není ve vyrovnávací paměti, pak se musí zavolat rutina pro konkrétní souborový systém, která zajistí načtení příslušného inode. Operace načtení inode způsobí, že inode se umístí ve vyrovnávací paměti a další přístupy k tomuto inodu už končí jen ve vyrovnávací paměti. Nejméně používané VFS inody se z vyrovnávací paměti odstraňují.

Všechny souborové systémy v Linuxu používají společnou vyrovnávací paměť bufferů, ve které se ukládají datové buffery fyzických zařízení kvůli zrychlení přístupu k fyzickým zařízením jednotlivých souborových systémů.

Tato vyrovnávací paměť bufferů je na souborovém systému nezávislá a je integrována v mechanismu, který jádro Linuxu používá k alokaci, čtení a zápisu datových bufferů. Je velmi výhodné, že souborové systémy Linuxu jsou nezávislé na příslušném fyzickém zařízení a na ovladači, který s tímto zařízením pracuje. Všechna bloková zařízení se registrují v jádře Linuxu a nabízejí uniformní, blokové, obvykle asynchronní rozhraní. Dělají to i poměrně komplikovaná bloková zařízení, jako jsou například disky SCSI. Protože reálné souborové systémy načítají data z příslušných fyzických zařízení, vede to ke generování požadavků na ovladač

blokového zařízení, který musí zajistit načtení bloku ze svého zařízení. Do rozhraní blokového zařízení je integrována vyrovnávací paměť bloků. Když se různými souborovými systémy načítají bloky, ukládají se v globální vyrovnávací paměti bufferů sdílené všemi souborovými systémy a jádrem Linuxu. Buffery v této paměti jsou identifikovány číslem bloku a jednoznačným identifikátorem zařízení, z něhož byly načteny. Takže pokud se nějaká data vyžadují častěji, získají se přímo z vyrovnávací paměti a ne z disku, což by trvalo déle. Některá zařízení podporují dopředné čtení, kdy se datové bloky načítají spekulativně čistě pro případ, že by byly zapotřebí.

10 VFS dále udržuje vyrovnávací paměť prohledávaných adresářů, takže se rychle naleznou inody často používaných adresářů.

Můžete jako malý pokus vyzkoušet vypsat obsah adresáře, který jste si doposud neprohlíželi. Při prvním výpisu zaregistrujete určité zpoždění, druhý výpis však proběhne okamžitě. Vyrovnávací paměť adresářů neobsahuje přímo inody adresářů, ty jsou uloženy ve vyrovnávací paměti inodů, obsahuje pouze mapování mezi plným jménem adresáře a číslem jeho inodu.

9.2.1 Superblok VFS

Každý připojený souborový systém je reprezentován superblokem VFS. Kromě jiného obsa-
11 huje superblok VFS následující informace:

zařízení Identifikátor blokového zařízení, na němž je souborový systém uložen. Například `/dev/hda1`, první disk IDE v systému, má identifikátor *0x301*.

ukazatele inodů Ukazatel `mounted` ukazuje na kořenový inode tohoto souborového systému. Ukazatel `covered` ukazuje na inode reprezentující adresář, na němž je tento souborový systém připojen. Superblok kořenového souborového systému neobsahuje ukazatel `covered`.

velikost bloku Velikost bloku v daném souborovém systému, například 1 024 bajtů.

operace superbloku Ukazatel na množinu rutin superbloku daného souborového systému. Tyto rutiny mimo jiné slouží ke čtení a zápisu inodů a superbloků.

typ souborového systému Ukazatel na datovou strukturu `file_system_type` připojeného systému.

specifické pro souborový systém Ukazatel na informace, používané konkrétním souborovým systémem.

9.2.2 Inode VFS

Stejně jako v systému `ext2`, i ve VFS je každý soubor, adresář a podobně reprezentován právě jedním inodem VFS.

Informace v jednotlivých inodech VFS se získávají z informací reálných souborových systémů prostřednictvím systémově závislých rutin. inody VFS existují pouze v paměti jádra a ukládají se ve vyrovnávací paměti inodů VFS tak dlouho, dokud jsou potřebné. Kromě jiného obsahují inody VFS následující informace:

zařízení Identifikátor zařízení, na němž je uložen soubor nebo jiný objekt, který tento inode VFS reprezentuje.

číslo inode Číslo inodu souboru, je jedinečné v rámci souborového systému. Kombinace hodnot `device` a `inode number` je jedinečná v celém VFS.

mód Stejně jako u `ext2` i zde tento údaj popisuje typ objektu reprezentovaného tímto inodem a přístupová práva k němu.

uživatelská id Identifikátory vlastníka.

časy Časy vytvoření, modifikace a zápisu.

velikost bloku Velikost bloku tohoto souboru v bajtech, například 1 024 bajtů.

operace Ukazatel na blok adres rutin. Tyto rutiny jsou závislé na souborovém systému a zajišťují operace nad tímto inodem, například zkrácení souboru reprezentovaného tímto inodem.

počitadlo Počet systémových komponent, které inode momentálně používají. Nulová hodnota znamená, že inode je možno zrušit nebo nově obsadit.

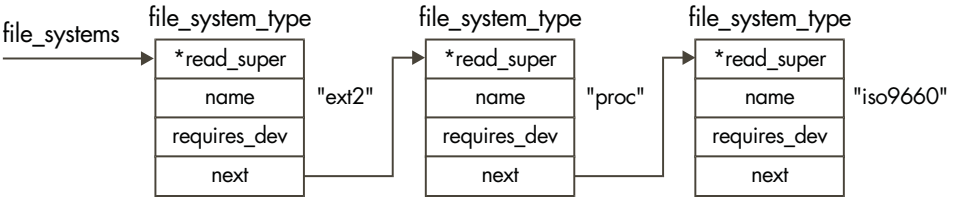
zámek Toto pole slouží k uzamčení inodu VFS, například v době, kdy je načítán ze souborového systému.

modifikován Příznak modifikace inode, pokud byl modifikován, bude nutná modifikace i ve fyzickém souborovém systému.

informace specifické pro souborový systém

9.2.3 Registrace souborových systémů

Když sestavujete jádro Linuxu, určujete, které z podporovaných souborových systémů budete chtít vestavět do jádra. Po sestavení jádra obsahuje inicializační kód souborového systému volání inicializačních rutin všech vestavěných souborových systémů.



Obrázek 9.5
Registrované souborové systémy

Souborové systémy mohou být v Linuxu sestaveny rovněž jako samostatné moduly a v takovém případě se provede jejich nahrání až v okamžiku, kdy budou zapotřebí, nebo když jsou nahrány ručně příkazem `insmod`. Vždy, když je nahrán modul souborového systému, registruje se v jádře a když se odstraňuje, ruší svou registraci. Inicializační rutina každého souborového systému provede registraci ve virtuálním souborovém systému, kde je systém reprezentován datovou strukturou `file_system_type`, která obsahuje jméno souborového systému a ukazatel na rutinu načtení jeho superbloku VFS. Na obrázku 9.5 vidíme, že struktury `file_system_type` se ukládají do seznamu, na jehož začátek ukazuje ukazatel `file_systems`. Každá datová struktura `file_system_type` obsahuje následující informace:

rutina načtení superbloku — Tuto rutinu VFS volá, když je připojena nějaká instance souborového systému.

jméno souborového systému — Jméno souborového systému, například `ext2`.

příznak potřebnosti zařízení — Potřebuje tento souborový systém podpůrné zařízení? Ne všechny souborové systémy potřebují zařízení, na němž jsou uloženy. Například souborový systém `/proc` nepotřebuje blokové zařízení.

Registrované souborové systémy je možno zjistit v souboru `/proc/filesystems`. Např:

```
       ext2
nodev proc
       iso9660
```

9.2.4 Připojení souborového systému

Když superuživatel připojuje souborový systém, musí jádro Linuxu nejprve ověřit parametry předané systémovému volání. Přestože příkaz `mount` provádí nějaké základní kontroly, nemůže vědět, jaké souborové systémy jádro podporuje a zda požadovaný přípojný bod opravdu existuje. Vezměme například následující příkaz:

```
$ mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
```

Tento příkaz předává jádru tři informace: jméno souborového systému, fyzické blokové zařízení, které souborový systém obsahuje, a kam v topologii stávajícího souborového systému má být nový souborový systém připojen.

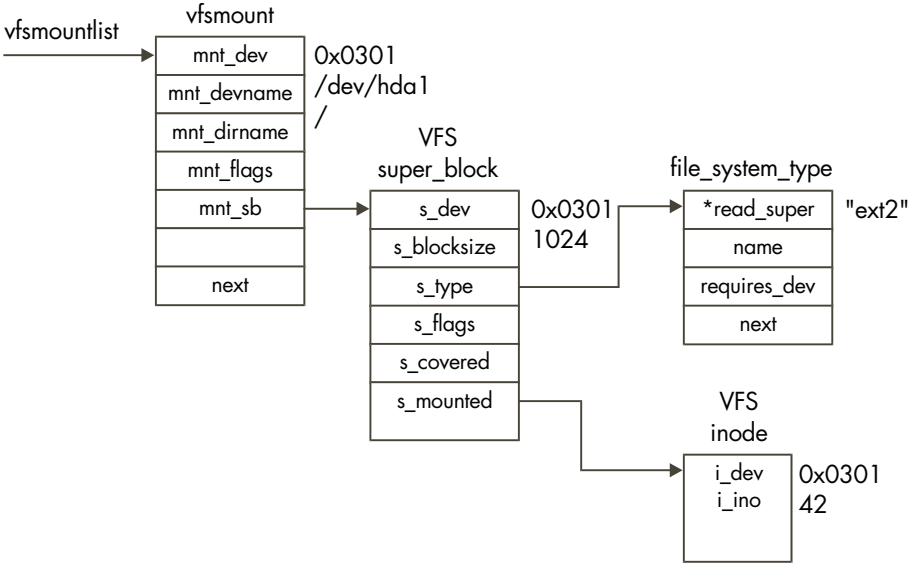
První věc, kterou musí VFS udělat, je, že musí nalézt příslušný souborový systém. Provede to prohlížením známých souborových systémů ve strukturách `file_system_type`, na něž ukazuje ukazatel `file_systems`.

Pokud najde souborový systém požadovaného jména, ví, že daný typ souborového systému je jádrem podporován a má adresu systémově specifické rutiny pro načtení superbloku souborového systému. Pokud nenalezne souborový systém požadovaného jména, není nic ztraceno v případě, že jádro dokáže nahrávat moduly podle potřeby (viz kapitola „Moduly“). V takovém případě jádro požádá démona jádra o nahrání modulu příslušného souborového systému a poté může pokračovat.

Pokud požadované fyzické zařízení není dosud připojeno, musí se nalézt inode VFS pro adresář, který má být připojovacím bodem zařízení. Tento VFS inode může být ve vyrovnávací paměti inodů nebo jej bude nutné načíst z blokového zařízení, na němž je uložen souborový systém připojovacího bodu. Když se inode nalezne, zkontroluje se, zda je to opravdu adresář a zda na něj není připojen jiný souborový systém. Jeden adresář není možno použít jako připojovací bod pro více než jeden souborový systém.

V této fázi musí připojovací kód VFS alokovat superblok VFS a předat mu informace o rutině pro načtení superbloku připojovaného systému. Všechny superbloky VFS jsou udržovány ve vektoru `super_blocks` datových struktur `super_block` a při připojování se jedna z nich musí alokovat. Rutina načtení superbloku vyplní údaje superbloku VFS informacemi, které načte z fyzického zařízení. Pro souborový systém `ext2` je toto mapování nebo překlad informací velmi jednoduché, protože se jednoduše přečte superblok systému `ext2` a naplní se jím superblok VFS. U jiných souborových systémů, například systému MS-DOS, to není tak jednoduché. Ať už je souborový systém jakýkoliv, naplnění superbloku VFS předpokládá, že souborový systém musí z fyzického blokového zařízení, na němž je uložen, přečíst své charakteristiky. Pokud není možno ze zařízení číst nebo pokud neobsahuje požadovaný typ souborového systému, připojování skončí chybou.

Každý připojený souborový systém je popsán datovou strukturou `vfsmount`, viz obrázek 9.6. Tyto struktury jsou seřazeny v seznamu, na nějž ukazuje ukazatel `vfsmntlist`.



Obrázek 9.6
Připojený souborový systém

Další ukazatel (`vfsmnttail`) ukazuje na poslední položku tohoto seznamu a ukazatel `mru_vfsmnt` ukazuje na naposledy použitý souborový systém. Každá struktura `vfsmount` obsahuje číslo zařízení blokového zařízení, na němž je souborový systém uložen, adresář, do nějž je systém připojen, a ukazatel na superblok VFS alokovaný při připojení tohoto souborového systému. Superblok VFS pak ukazuje na datovou strukturu `file_system_type` tohoto souborového systému a na jeho kořenový inode. Tento inode zůstává rezidentně ve vyrovnávací paměti inodů VFS po celou dobu, kdy je souborový systém připojen.

9.2.5 Nalezení souboru ve virtuálním souborovém systému

Při hledání inodu VFS ve virtuálním souborovém systému musí VFS rozložit jméno na jednotlivé adresáře a postupně nalézt inody všech adresářů ve jméně. Hledání každého adresáře obnáší volání rutin konkrétního souborového systému, jejichž adresy jsou uloženy v inodu VFS reprezentujícím rodičovský adresář. Celé to funguje díky tomu, že kořenový adresář každého souborového systému máme vždy k dispozici a ukazuje na něj superblok VFS daného souborového systému. Vždy, když reálný souborový systém hledá adresář, pokouší se jej nejprve nalézt ve vyrovnávací paměti adresářů. Pokud v této paměti adresář není, získává reálný souborový systém VFS inode buď přímo ze souborového systému nebo z vyrovnávací paměti inodů.

9.2.6 Odpojení souborového systému

V příručce k mému autu je postup montáže obvykle popsán jako „opak demontáže“ a tato moudrost v zásadě platí i pro odpojení souborového systému.

Souborový systém není možno odpojit, pokud někdo v systému používá nějaký z jeho souborů. Nemůžete tedy například odpojit `/mnt/cdrom`, pokud nějaký proces používá tento adresář nebo některý z jemu podřízených adresářů. Pokud cokoliv používá souborový systém, který má být odpojen, mohou být ve vyrovnávací paměti inodů VFS nějaké inody VFS tohoto souborového systému. Kód odpojení tyto inody hledá tak, že prochází seznam inodů a hledá inody vlastněné zařízením, na němž je souborový systém umístěn. Pokud byl modifikován superblok VFS systému, musí se zapsat zpět na disk. Po jeho zapsání na disk se paměť zabraná superblokem vrátí zpět jádru. Nakonec se ze seznamu `vfsmntlist` odstraní struktura `vfsmount` a uvolní se z paměti.

9.2.7 Vyrovnávací paměť inodů VFS

Když se pracuje s připojenými systémy, průběžně dochází k načítání a případnému zápisu jejich inodů VFS. Virtuální souborový systém udržuje vyrovnávací paměť inodů, která slouží ke zrychlení přístupu na všechny připojené souborové systémy. Vždy, když se podaří načíst inode VFS z vyrovnávací paměti, ušetří se přístup na fyzické zařízení.

Vyrovnávací paměť inodů VFS je implementována jako hashovací tabulka, jejíž položky jsou ukazatelé na seznamy inodů VFS se stejnou hashovací hodnotou. Hashovací hodnota inodu se počítá z čísla inodu a z identifikátoru příslušného fyzického zařízení, který obsahuje daný souborový systém. Vždy, když VFS potřebuje přistupovat k inodu, nejprve se podívá do vyrovnávací paměti inodů. Při hledání inodu ve vyrovnávací paměti vypočítá systém nejprve jeho hashovací hodnotu a pak ji použije jako index do hashovací tabulky inodů. Tím získá ukazatel na seznam inodů se stejnou hashovací hodnotou. Z tohoto seznamu načítá každý inode dokud nenalezne ten, jehož číslo a zařízení odpovídá požadovanému inodu.

Pokud se podaří nalézt inode ve vyrovnávací paměti, inkrementuje se jeho počitadlo, aby se detekovalo, že inode má dalšího uživatele, a systém pokračuje v práci. Pokud inode nebyl ve vyrovnávací paměti, musí se nalézt volný inode VFS, aby souborový systém mohl načíst požadovaný inode do paměti. VFS má řadu možností jak získat volný inode. Pokud systém může alokovat další inode VFS, pak se to provede - alokuje se stránka jádra, vytvoří se v ní volné inody a připojí se k seznamu inodů VFS. Všechny inody VFS vytvářejí seznam, na nějž

ukazuje ukazatel `first_inode` a hashovací tabulka inodů. Pokud už má systém alokován maximální povolený počet inodů, musí nalézt inode, který bude vhodným kandidátem na nové použití. Dobrými kandidáty jsou inody s nulovým počitadlem uživatelů, což znamená, že je momentálně nikdo v systému nevyužívá. Významné inody VFS, například kořenové inody jednotlivých souborových systémů, mají počitadlo použití vždy nenulové, a tak nemohou být nikdy nahrazeny. Když se podaří nalézt vhodný inode pro nové použití, vyčistí se. Inode mohl být modifikován a v takovém případě je nutné zapsat jej zpět do souborového systému, nebo může být zamčený a systém musí před pokračováním práce počkat na jeho odemčení. Před novým použitím musí být inode vyčištěn.

Ať už je nový inode VFS nalezen jakkoliv, zavolá se rutina specifického souborového systému, která jej naplní informacemi ze skutečného souborového systému. V době plnění má inode počitadlo použití rovno jedné a je uzamčen, takže jej nikdo nemůže použít dříve, než bude inode obsahovat platné informace.

Aby se získal požadovaný inode VFS, bude systém možná muset postupně přistupovat k několika jiným inodům. K tomu dochází, když čtete adresář: je sice zapotřebí pouze inode požadovaného adresáře, musí se však načíst inody i mezilehlých adresářů. Jak se vyrovnávací paměť inodů plní a rozrůstá, méně používané inody se ruší a v paměti zůstávají pouze ty častěji používané.

9.2.8 Vyrovnávací paměť adresářů

Kvůli zrychlení přístupu k často používaným adresářům vytváří VFS vyrovnávací paměť adresářových položek.

21

Když reálný souborový systém prohledává adresáře, zjištěné informace se přidávají do vyrovnávací paměti adresářů. Při příštím prohlížení stejného adresáře, například při vypisování jeho obsahu nebo při otevírání souboru v tomto adresáři, se informace naleznou ve vyrovnávací paměti. Ve vyrovnávací paměti se ukládají pouze krátké adresářové položky (do délky 15 znaků), což je ale rozumné řešení, protože nejčastěji se používají právě nejkratší jména. Například adresář `/usr/X11R6/bin` je velmi často používán spuštěným X serverem.

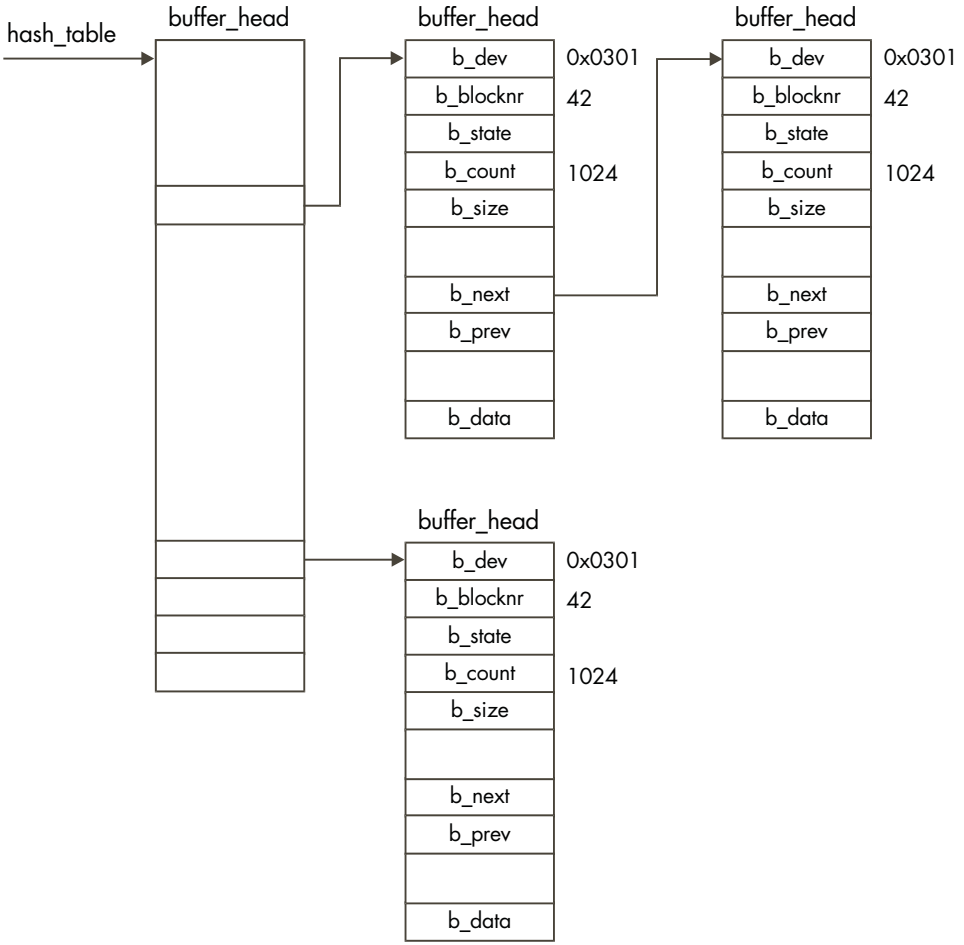
Vyrovnávací paměť adresářů se skládá z hashovací tabulky, jejíž každá položka ukazuje na seznam adresářových položek, které mají stejnou hashovací hodnotu. Hashovací funkce vypočítává offset v tabulce z čísla zařízení, na němž je příslušný souborový systém uložen, a ze jména adresáře. Díky tomu se jednotlivé položky dají rychle nalézt. Bylo by k ničemu mít vyrovnávací paměť, pokud by hledání (a případné nenalezení) položky v této paměti trvalo dlouho.

Aby byl obsah paměti platný a aktuální, udržuje VFS seznamy položek podle toho, kdy byly naposledy použity - takzvané LRU (Least Recently Used) seznamy. Vždy když se do vyrovnávací paměti přidává adresář, přidává se na konec LRU seznamu první úrovně. Pokud je vyrovnávací paměť plná, odstraní se tím zároveň položka na začátku tohoto LRU seznamu. Jakmile je položka použita znovu, přemístí se na konec LRU seznamu druhé úrovně. I zde může případně dojít ke zrušení položky ze začátku tohoto seznamu. Rušení položek ze začátků obou úrovní seznamů je zcela v pořádku. Jediný důvod, jak se mohla položka na začátku seznamu ocitnout, je ten, že už k ní delší dobu nebylo přistupováno. Pokud by se používala, byla by více ke konci seznamu. Položky v LRU seznamu druhé úrovně jsou uloženy „bezpečněji“ než v seznamu první úrovně. To je pochopitelně záměrné, protože tyto položky byly nejen nalezeny, ale také se s nimi opakovaně pracovalo.

9.3 Vyrovnávací paměť bufferů

Při používání připojených souborových systémů se objevuje řada požadavků na načtení nebo zápis bloků na bloková zařízení. Všechny požadavky na čtení a zápis se ovladači zařízení předávají jako datová struktura `buffer_head` prostřednictvím volání standardních rutin jádra. Tato struktura obsahuje všechny informace, které ovladač blokového zařízení potřebuje - identifikátor blokového zařízení jednoznačně identifikuje zařízení a číslo bloku oznamuje ovladači, který blok má přečíst. Všechna bloková zařízení jsou chápána jako lineární posloupnost bloků stejné velikosti. Kvůli zrychlení přístupu k fyzickým blokovým zařízením používá Linux vyrovnávací paměť blokových bufferů. Všechny blokové buffery v systému jsou drženy někde ve vyrovnávací paměti bufferů včetně nových, nepoužitých bloků. Tuto paměť sdílejí všechna fyzická bloková zařízení. V kterémkoliv okamžiku je ve vyrovnávací paměti řada blokových bufferů často v různých stavech, které patří vždy nějakému fyzickému zařízení. Každý blokový buffer použitý ke čtení dat z blokového zařízení nebo k jejich zápisu se ukládá ve vyrovnávací paměti bufferů. Časem může být z vyrovnávací paměti odstraněn aby se uvolnilo místo pro potřebnější buffer, nebo, pokud je používán často, může v paměti zůstávat dlouhou dobu.

Blokové buffery ve vyrovnávací paměti jsou jednoznačně identifikovány identifikátorem zařízení a číslem bloku, který je v bufferu uložen. Vyrovnávací paměť bufferů se skládá ze dvou částí. První část je seznam volných blokových bufferů. Existuje vždy jeden seznam pro jednu podporovanou velikost bufferů a volné blokové buffery v systému se v okamžiku svého vytvoření nebo vyprázdnění zařadí do příslušné fronty volných bufferů. Momentálně jsou podporovány buffery o velikosti 512, 1 024, 2 048, 4 096 a 8 192 bajtů. Druhou částí je samotná vyrovnávací paměť. Je to hashovací tabulka, která slouží jako vektor ukazatelů na řetězce bufferů se stejným hashovacím indexem. Hashovací index se generuje podle identifikátoru zařízení a čísla bloku. Na obrázku 9.7 vidíme hashovací tabulku a několik položek vyrovnávací paměti. Blokový buffer je buď v jednom ze seznamu volných bufferů, nebo ve vyrovnávací paměti.



Obrázek 9.7
Vyrovnávací paměť bufferů

Když je umístěn ve vyrovnávací paměti, je také zařazen v LRU seznamech. Pro každý typ bufferu existuje jeden LRU seznam, který systému umožňuje snadno realizovat potřebné operace nad daným typem bufferů, například zápis bufferů s novými daty na disk. Typ bufferu odráží jeho stav. Linux v současné době podporuje následující typy:

clean Nepoužité, nové buffery.

locked Buffer je uzamčen, čeká na zapsání.

dirty Buffer je modifikován. Obsahuje nová platná data a bude zapsán na disk, zatím ale zápis ještě nebyl naplánován.

shared Sdílené buffery.

unshared Buffery, které byly sdíleny, nyní však už sdíleny nejsou.

Vždy, když systém potřebuje z fyzického zařízení načíst buffer, zkouší jej nejprve najít ve vyrovnávací paměti bufferů. Pokud v ní buffer nenajde, vezme buffer potřebné délky ze seznamu volných bufferů a přemístí jej do vyrovnávací paměti. Pokud je buffer v paměti, může ale nemusí být aktuální. Pokud není aktuální nebo pokud je to nový blokový buffer, musí systém požádat ovladač zařízení o načtení příslušného bloku dat z disku.

Stejně jako všechny ostatní vyrovnávací paměti, i vyrovnávací paměť bufferů musí být spravována tak, aby pracovala efektivně a aby zajišťovala spravedlivou alokaci položek paměti mezi všemi blokovými zařízeními. K provádění řady potřebných úklidových operací ve vyrovnávací paměti používá Linux démona `bdflush`, některé operace nicméně probíhají automaticky jako přímý důsledek používání paměti.

9.3.1 Démon bdflush

Démon `bdflush` je démon jádra, který zajišťuje dynamickou reakci v případě, kdy je v systému příliš mnoho modifikovaných bufferů, tedy bufferů obsahujících data, která musejí být časem zapsána na disk. Spouští se jako vlákno jádra při startu systému a sám sebe nazývá „kflushd“, což je jméno, které uvidíte, když si příkazem `ps` zobrazíte procesy v systému. Většinu času démon spí a čeká, až počet modifikovaných bufferů dosáhne určité hranice. Když se alokují a ruší buffery, kontroluje se počet modifikovaných bufferů. Pokud dosáhne počet modifikovaných bufferů určitého procenta ze všech bufferů v systému, probouzí se démon `bdflush`. Implicitní hranice je 60 %, pokud je ale v systému nedostatek bufferů, bude démon probuzen dříve. Nastavená hodnota se dá zobrazit a měnit příkazem `update`:

```
# update -d

bdflush version 1.4
0: 60 Max fraction of LRU list to examine for dirty blocks
1: 500 Max number of dirty blocks to write each time bdflush activated
2: 64 Num of clean buffers to be loaded onto free list by refill_freelist
3: 256 Dirty block threshold for activating bdflush in refill_freelist
4: 15 Percentage of cache to scan for free clusters
5: 3000 Time for data buffers to age before flushing
6: 500 Time for non-data (dir, bitmap, etc) buffers to age before flushing
7: 1884 Time buffer cache load average constant
8: 2 LAV ratio (used to determine threshold for buffer fratricide).
```

Všechny modifikované buffery jsou umístěny v LRU seznamu `BUF_DIRTY` a démon `bdflush` se snaží vždy nějaký rozumný počet zapsat na disk. I tuto hodnotu je možno zobrazit a nastavit příkazem `update` (viz výše).

9.3.2 Proces update

Proces `update` není jen příkaz, je to zároveň i démon. Běží jako superuživatelský (v době inicializace systému) a periodicky vyprazdňuje všechny staré modifikované buffery na disk. Dosahuje toho voláním systémových rutin, které dělají více méně to samé, co démon `bdflush`. Vždy, když se dokončí modifikace bufferu, přidělí se mu čas, kdy má být zapsán na disk. Vždy, když `update` běží, prozkoumá všechny modifikované buffery v systému a hledá ty, které už mají být zapsány. Všechny takové buffery se zapíší na disk.

9.4 Souborový systém /proc

Souborový systém `/proc` ukazuje skutečnou sílu virtuálního souborového systému Linuxu. Fyzicky ve skutečnosti neexistují (zase jeden kouzelnický trik Linuxu) ani adresář `/proc`, ani jeho podadresáře a soubory. Jak tedy můžete pořídit výpis souboru `/proc/devices`? Souborový systém `/proc` se, stejně jako reálné souborové systémy, registruje ve virtuálním souborovém systému. Když však VFS tento systém volá a požaduje jeho inody při otevírání jeho souborů a adresářů, vytváří systém `/proc` tyto soubory a adresáře podle informací v jádře. Například soubor `/proc/devices` je generován z datových struktur jádra, popisujících jeho zařízení.

Souborový systém `/proc` poskytuje uživateli okno do interní činnosti jádra. V souborovém systému `/proc` vytváří své položky několik subsystémů Linuxu, například moduly jádra popsané v kapitole „Moduly“.

9.5 Speciální soubory zařízení.

Linux, stejně jako všechny verze systému Unix, představuje svá hardwarová zařízení jako soubory. Například `/dev/null` je nulové zařízení. Soubory zařízení nezabírají v souborovém systému žádný datový prostor, představují pouze přístupový bod k ovladači zařízení. Souborový systém `ext2` i virtuální souborový systém implementují soubory zařízení jako speciální typy inodů. Existují dva typy souborů zařízení: znakové a blokové soubory. I v samotném jádře implementuje ovladač zařízení souborové operace: můžete jej otevírat, zavírat a podobně. Znaková zařízení umožňují vstupně/výstupní operace ve znakovém režimu, bloková zařízení vyžadují veškerý přístup prostřednictvím vyrovnávací paměti bufferů. Velmi často pro některé subsystémy neexistují ovladače reálných zařízení, ale takzvané ovladače pseudozařízení, napří-

klad vrstva ovladače zařízení SCSI. Na soubory zařízení se odkazuje pomocí hlavního čísla, které identifikuje typ zařízení, a pomocí vedlejšího čísla, které identifikuje jednotku, instanci daného hlavního typu. Například disky IDE primárního řadiče IDE mají hlavní číslo 3 a první oblast 24
každého disku IDE má vedlejší číslo 1. Tak nám výpis `ls -l` souboru `/dev/hda1` dává:

```
$ brw-rw---- 1 root     disk        3,     1  Nov 24   15:09 /dev/hda1
```

V jádře je každé zařízení jednoznačně popsáno datovým typem `kdev_t`, který je dlouhý dva 25
bajty, první bajt obsahuje vedlejší číslo zařízení, druhý obsahuje hlavní číslo zařízení.

Výše uvedené zařízení IDE je v jádře označeno hodnotou *0x0301*. Inode systému ext2 reprezentující blokové nebo znakové zařízení má v ukazateli na první datový blok uloženo hlavní a vedlejší číslo zařízení. Když VFS takovýto inode načte, nastaví se identifikátor zařízení do položky `i_rdev` inodu VFS.

Odkazy na zdrojové texty jádra

1 – `Viz fs/ext2/*`
2 – `Viz include/linux/-ext2_fs_i.h`
3 – `Viz include/linux/-ext2-fs_sb.h`
4 – `Viz ext2_group_desc in include/-linux/ext2:fs.h`
5 – `Viz ext2_dir_entry in include/-linux/ext2_fs.h`
6 – `Viz ext2_new_block() in fs/ext2/-balloc.c`
7 – `Viz fs/*`
8 – `Viz fs/inode.c`
9 – `Viz fs/buffer.c`
10 – `Viz fs/dcache.c`
11 – `Viz include/-linux/fs.h`
12 – `Viz include/-linux/fs.h`
13 – `Viz sys_setup() in fs/-filesystems.c`
14 – `Viz file_system_type in include/-linux/fs.h`
15 – `Viz do_mount() in fs/super.c`
16 – `Viz get_fs_type() in fs/super.c`
17 – `Viz add_vfsmnt() in fs/super.c`
18 – `Viz do_umount() in fs/super.c`
19 – `Viz remove_vfsmnt() in fs/super.c`
20 – `Viz fs/inode.c`

21 – Viz fs/dcache.c

22 – Viz bdflush() in fs/buffer.c

23 – Viz sys bdflush() in fs/buffer.c

24 – Viz /include/linux/major.h for all of Linux's major device numbers

25 – Viz include/-linux/kdev_t.h

Kapitola

10

Sítě

Termíny sítě a Linux jsou prakticky synonyma. Linux je fakticky produktem Internetu nebo WWW. Jeho tvůrci a uživatelé používají Internet k výměně nápadů a samotný Linux se často používá k zajištění síťových potřeb různých organizací. V této kapitole se popisuje jak Linux podporuje síťové protokoly souhrnně označované jako TPC/IP.

Protokoly TCP/IP byly navrženy pro komunikaci počítačů připojených k síti ARPANET, americké výzkumné sítě financované vládou USA. ARPANET vedl ke vzniku základních technik počítačových sítí, jako jsou přepínání paketů a vrstvení protokolů, kdy jeden protokol poskytuje služby dalšímu. ARPANET byl oficiálně zrušen v roce 1988, jeho následníci (NSFNET[1] a Internet) se však stále rozrůstají. To co dnes označujeme jako World Wide Web je vybudováno právě na ARPANETu a protokolech TCP/IP. Unix byl na ARPANETu velmi používán a jeho první verze s podporou sítí byla verze BSD 4.3. Implementace síťových služeb v Linuxu je postavena na BSD verzi 4.3 v tom, že (s určitými výjimkami) podporuje BSD sokety a plně implementuje podporu TCP/IP. Toto programové rozhraní bylo zvoleno jednak kvůli jeho oblibě a jednak aby se napomohlo přenositelnosti aplikací mezi Linuxem a jinými platformami Unixu.

10.1 Přehled TCP/IP

V této části popisujeme základní principy sítí na bázi TCP/IP. Nejedná se o vyčerpávající přehled, doporučuji vám ale přečíst si jej. V IP síti má každý počítač přiřazenu IP adresu, což je 32bitové číslo jednoznačně popisující daný počítač. Web je velmi rozsáhlý a stále roste. Každá IP síť a každý k ní připojený počítač musí mít přiřazenu jedinečnou IP adresu. IP adresy

[1] National Science Foundation Network

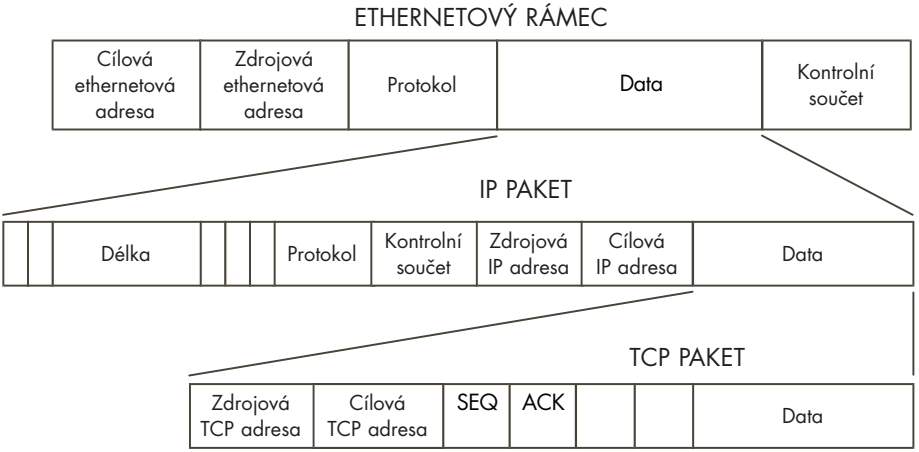
se reprezentují jako čtyři čísla oddělená tečkami, například `16.42.0.9`. IP adresa se skládá ze dvou částí, z *adresy sítě* a *adresy počítače*. Velikosti těchto částí mohou být různé (existuje několik tříd IP adres), pokud ale jako příklad vezmeme adresu `16.42.0.9`, pak adresa sítě je `16.42` a adresa počítače `0.9`. Adresa počítače se dále může dělit na *podsíť* a adresu počítače. Pokud znovu použijeme jako příklad adresu `16.42.0.9`, pak podsíť může být `16.42.0` a na ní je počítač `9`. Toto dělení IP adres umožňuje organizacím strukturovat své sítě. Například `16.42` může být adresa sítě společnosti ACME Computer Company, `16.42.0` bude jejich podsíť `0`, `16.42.1` podsíť `1`. Tyto podsítě mohou být v různých budovách, propojených pevnou linkou nebo radiovým spojem. IP adresy přiřazuje administrátor sítě a použití podsítí je dobrá metoda, jak administraci distribuovat. Administrátor podsítě pak může libovolně přiřazovat adresy v rámci své podsítě.

Obecně vzato se IP adresy špatně pamatují, daleko příjemnější jsou jména. Jméno `linux.acme.com` se pamatuje daleko snáze než `16.42.0.9`, potřebujeme ovšem nějaký mechanismus pro konverzi síťových jmen na IP adresy.

Jména mohou být staticky uvedena v souboru `/etc/hosts`, nebo Linux může o překlad jména na adresu požádat server DNS. V takovém případě musí lokální počítač znát adresu jednoho nebo více serverů DNS, které se udávají v souboru `/etc/resolv.conf`.

Vždy, když se připojíte k jinému počítači, řekněme při prohlížení webovské stránky, komunikujete s ním prostřednictvím jeho IP adresy. Tato data jsou uložena v IP paketech, z nichž každý má hlavičku, která obsahuje IP adresy zdrojového a cílového počítače, kontrolní součet a další užitečné údaje. Kontrolní součet se odvozuje z dat v IP paketu a umožňuje příjemci paketu rozhodnout, zda paket nebyl při přenosu poškozen například rušením na telefonní lince. Data odesílaná aplikací se mohou dělit na menší pakety, s nimiž se snáze manipuluje. Velikost datových paketů závisí na použitém fyzickém síťovém médiu, například ethernetové pakety jsou obecně větší než pakety PPP. Cílový počítač musí zpětně sestavit pakety dohromady a poté je teprve předá přijímající aplikaci. Tuto fragmentaci a zpětné skládání dat můžete vidět graficky, když přistupujete k webovské stránce s řadou obrázků prostřednictvím poměrně pomalé sériové linky.

Počítače připojené ke stejné podsíti si mohou mezi sebou vyměňovat IP pakety přímo, všechny ostatní pakety budou odesílány na speciální počítač, *bránu*. Brána (nebo router) je připojena k více než jedné podsíti a předává pakety zachycené na jedné podsíti patřící na jinou. Pokud budeme mít například podsítě `16.42.1.0` a `16.42.0.0` propojeny branou, musí se všechny pakety z podsítě `0` určené na podsíť `1` předávat bráně, která zajistí jejich směrování. Lokální počítač si sestavuje směrovací tabulku, která mu umožňuje směrovat IP pakety na správné brány. Pro každou cílovou IP adresu obsahuje směrovací tabulka údaje, které Linuxu říkají, kterému počítači má paket poslat, aby se dostal na místo určení. Tyto směrovací tabulky jsou dynamické a mění se podle toho, jak aplikace používají síť a jak se mění topologie sítě.



Obrázek 10.1
Protokolové vrstvy TCP/IP

Protokol IP je protokol transportní vrstvy, který ostatním protokolům zajišťuje přenos jejich dat. Protokol TCP (Transmission Control Protocol) je protokol zajišťující spolehlivé dvoubodové spojení, který pro příjem a vysílaní svých paketů používá protokol IP. Tak jako má svou hlavičku IP paket, má svou hlavičku i TCP paket. TCP je protokol s navazováním spojení, kdy jsou dvě síťové aplikace spojeny jedním virtuálním spojem, přestože ten může vést přes řadu podsítí, bran a routerů. TCP spolehlivě přijímá a odesílá data mezi těmito dvěma aplikacemi a zaručuje, že nedojde k žádné ztrátě ani zdvojení paketů. Když TCP odesílá svůj paket prostřednictvím IP, data obsažená v IP paketu jsou celý TCP paket. IP vrstvy obou komunikujících počítačů zodpovídají za odesílání a příjem IP paketů. Protokol UDP (User Datagram Protocol) rovněž používá jako transportní protokol IP, na rozdíl od TCP však UDP není spolehlivý protokol a zajišťuje datagramové služby. Toto využití protokolu IP jako nosiče jiných protokolů předpokládá, že přijímající IP vrstva musí vědět, jaké vyšší protokolové vrstvě musí předat data IP paketu. Proto je v hlavičce každého IP paketu hodnota, obsahující identifikátor protokolu. Když TCP požádá IP vrstvu o přenos TCP paketu, bude hlavička IP paketu obsahovat údaj o tom, že se přenáší TCP paket. Přijímající IP vrstva se podle tohoto identifikátoru rozhoduje, jaké vrstvě data předat, v tomto případě je předá TCP vrstvě. Když aplikace komunikují pomocí TCP/IP, musejí udávat nejen cílovou IP adresu, ale také adresu *portu* aplikace. Adresa portu jednoznačně identifikuje aplikaci a standardní síťové aplikace používají standardní porty, například webový server používá port 80. Registrované porty je možno vidět v souboru `/etc/services`.

Vrstvení protokolů nekončí u TCP, UDP a IP. Samotná vrstva IP používá řadu různých fyzických médií k přenosu IP paketů na jiné počítače. Tato média sama mohou přidávat vlastní protokolové hlavičky. Jedním příkladem může být ethernetová vrstva, vrstvy PPP a SLIP fungu-

jí zase jinak. Ethernetová síť umožňuje propojit současně řadu počítačů fyzicky jedním kabelem. Každý vysílaný etherentový rámec se tak dostane ke všem připojeným počítačům, a proto má každé ethernetové zařízení jedinečnou adresu. Každý ethernetový rámec vyslaný na nějakou adresu bude přijat adresovaným počítačem a všechny ostatní připojené počítače jej budou ignorovat. Tato jedinečná adresa je nastavena každému ethernetovému zařízení při výrobě a obvykle je uložena v paměti SROM[2] na ethernetové kartě. Ethernetové adresy mají délku 6 bajtů, například `08-00-2b-00-49-A4`. Některé ethernetové adresy jsou rezervovány pro potřeby hromadného vysílání a rámce poslané na takovéto adresy budou zachyceny všemi připojenými počítači. Protože ethernetový frame může (jako data) nést řadu rozdílných protokolů, podobně jako IP paket obsahuje v hlavičce identifikátor protokolu. Díky tomu může ethernetová vrstva správně postoupit přijaté rámce IP vrstvě.

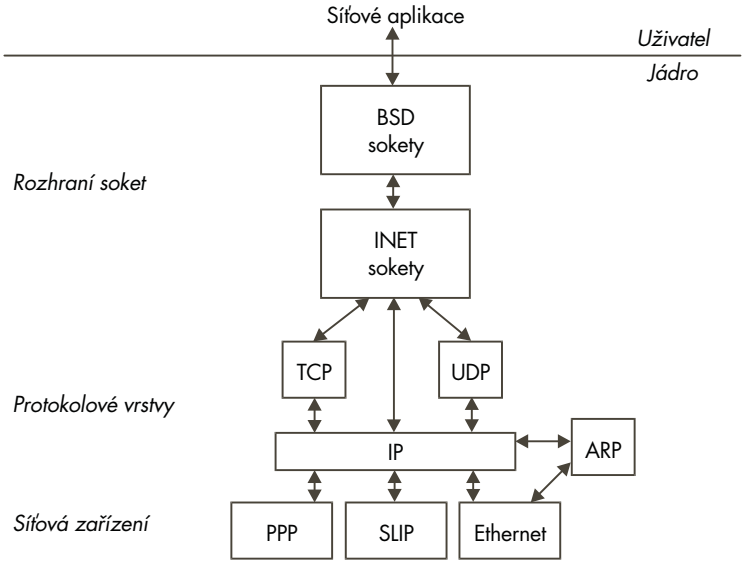
Aby mohla IP vrstva poslat IP paket protokolem jako je ethernet, musí zjistit ethernetovou adresu cílového počítače. IP adresy představují totiž pouze adresační koncept, ethernetová zařízení mají své vlastní fyzické adresy. IP adresy může administrátor sítě přiřazovat a měnit podle potřeb, ovšem síťový hardware reaguje pouze na svou vlastní ethernetovou adresu nebo na speciální hromadně vysílané pakety, které přijímají všechna zařízení. Linux používá k překladu IP adres na reálné hardwarové adresy, jako je například ethernetová adresa, protokol ARP (Address Resolution Protocol). Počítač, který potřebuje zjistit hardwarovou adresu odpovídající určité IP adrese, pošle požadavek ARP obsahující IP adresu, která se má přeložit hromadným vysílacím mechanismem, takže tento paket zachytí všechny počítače na sítí. Počítač, kterému patří požadovaná IP adresa, odpoví paketem ARP, v němž je uvedena jeho fyzická adresa. ARP není omezen pouze na ethernetová zařízení, dokáže zajistit překlad IP adres i na jiných médiích, například na sítích FDDI. Zařízení, která neumějí reagovat na protokol ARP, jsou označena tak, že se s nimi Linux nepokouší tímto protokolem komunikovat. Existuje také obrácená funkce, reverzní ARP nebo RARP, která překládá fyzické hardwarové adresy na IP adresy. Tuto funkci používají brány, které reagují na žádosti ARP z IP adres, které jsou na vzdálené síti.

10.2 Síťové vrstvy TCP/IP v Linuxu

Stejně jako samotné síťové protokoly, i Linux implementuje internetovou rodinu protokolů pomocí několika vzájemně propojených vrstev softwaru, jak můžeme vidět na obrázku 10.2. Sokety BSD poskytují programům služby obecné správy soketů. Tato vrstva je podporována soketovou vrstvou INET, která řídí komunikaci koncových bodů protokolů TCP a UDP. UDP (User Datagram Protocol) je protokol bez navazování spojení, zatímco TCP (Transmission Control Protocol) je spolehlivý dvoubodový protokol. Když se vysílají UDP pakety, Linux neví a nestará se, zda pakety správně dojdou k cíli. TCP pakety jsou číslovány a oba konce spo-

[2] Synchronous Read Only Memory

jení TCP zajišťují, že vysílaná data budou správně přijata. IP vrstva obsahuje kód implementující IP protokol. Tento kód připojuje k vysílaným datům IP hlavičky a ví, jak má příchozí IP pakety předávat vrstvám TCP nebo UDP. Pod IP vrstvou, která slouží jako podpora veškerého síťového software v Linuxu, jsou síťová zařízení, například PPP a ethernet. Síťová zařízení nemusejí vždy reprezentovat nějaké fyzické zařízení. Některá, například zpětné zařízení, jsou záležitosti čistě softwarové. Na rozdíl od standardních zařízení v Linuxu, která se vytvářejí příkazem `mknod`, se síťová zařízení objevují pouze v případě, že je příslušný síťový software nalezne a zinicializuje. Zařízení `eth0` uvidíte pouze v případě, že jádro má vestavěn příslušný ovladač ethernetového zařízení. Protokol ARP je umístěn mezi IP vrstvou a protokoly, které podporují adresaci pomocí ARP.



Obrázek 10.2
Síťové vrstvy v Linuxu

10.3 Soketové rozhraní BSD

Jedná se o obecné rozhraní, které podporuje nejenom různé formy síťové komunikace, ale slouží také jako meziprocesový komunikační mechanismus. Soket popisuje jeden konec komunikační linky, dva komunikující procesy budou mít každý svůj soket, popisující jejich konce společné komunikační linky. Soket je možno chápat jako zvláštní případ roury, na rozdíl od rour, ale nemá soket žádné omezení objemu dat, která může obsahovat. Linux podporuje několik tříd soketů, které jsou známé jako *adresové rodiny*. Je to dáno tím, že každá třída má své vlastní adresační metody. Linux podporuje následující soketové adresové rodiny či domény:

UNIX	Sokety unixové domény
INET	Adresová rodina Internetu podporuje komunikaci protokoly TCP/IP
AX25	Amateur radio X25
IPX	Novell IPX
APPLETALK	Appletalk DDP
X25	X25

Existuje několik typů soketů, které reprezentují typ služby, podporující spojení. Ne všechny adresové rodiny podporují všechny typy služeb. Sokety BSD podporují řadu soketových typů:

stream Tyto sokety poskytují spolehlivé obousměrné sekvenční datové proudy se zajištěním, že při přenosu nemůže dojít ke ztrátě, porušení nebo duplikaci dat. Streamy jsou podporovány TCP protokolem v adresové rodině INET.

datagram Tyto sokety rovněž zajišťují obousměrný datový přenos, na rozdíl od streamů však nezaručují doručení zpráv. I když zpráva dorazí, není zaručeno, že dorazí ve správném pořadí nebo neduplikovaně a neporušeně. Tento typ soketů je podporován UDP protokolem z adresové rodiny INET.

raw Umožňuje procesům přímý přístup k nízkoúrovňovým protokolům. Je například možné otevřít raw soket na ethernetové zařízení nebo sledovat nízkoúrovňový IP datový provoz.

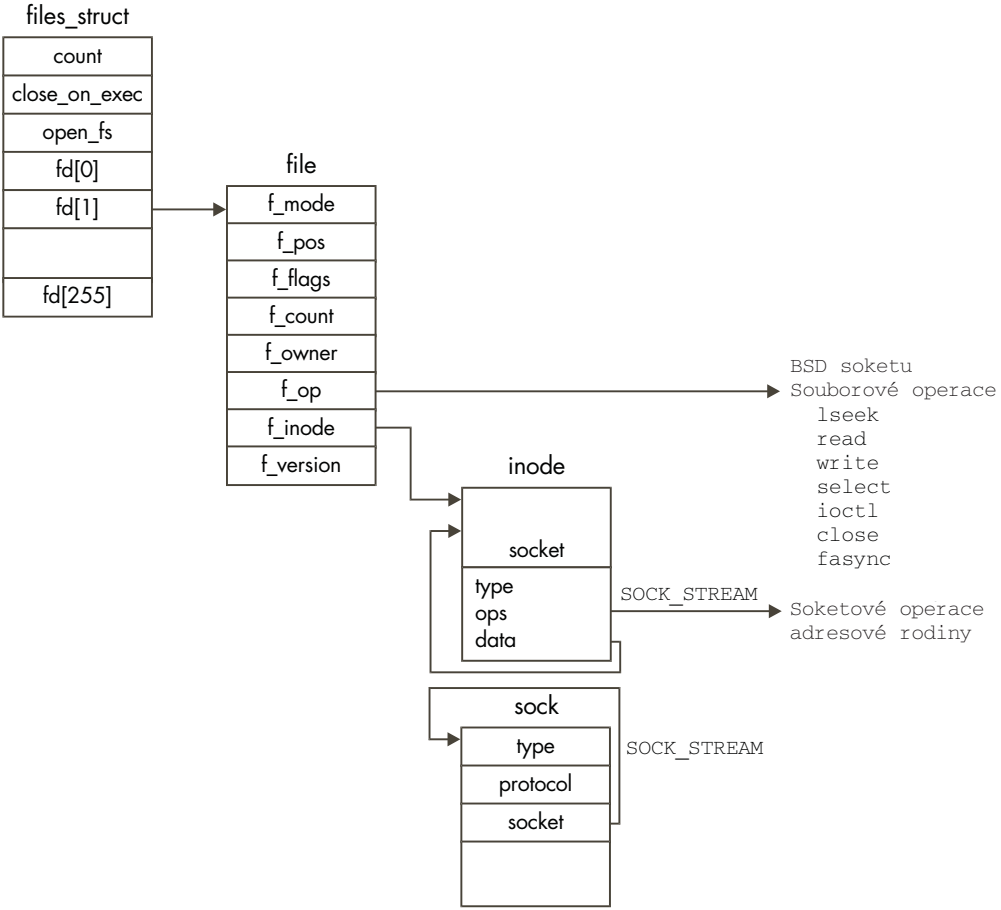
reliable delivered messages
Velmi se podobají datagramům, je však zajištěno doručení dat.

sequenced packet Velmi se podobají streamům, velikost datových paketů je ale pevná.

packet Toto není standardní BSD soketový typ, jedná se o rozšíření Linuxu, které umožňuje procesům přistupovat přímo k paketům na úrovni zařízení.

Procesy komunikující prostřednictvím soketů používají model klient/server. Server nabízí službu a klient tuto službu využívá. Jedním příkladem může být webový server, který nabízí webovské stránky, a webový klient, prohlížeč, který tyto stránky čte. Server, který používá sokety, musí soket nejprve vytvořit a poté mu přidělit jméno. Formát jména závisí na adresové rodině soketu a ve svém důsledku představuje jméno lokální adresu serveru. Jméno nebo adresa soketu se udává pomocí datové struktury `sockaddr`. Soket rodiny INET bude mít přiřazenu IP adresu. Čísla registrovaných portů můžeme zjistit v souboru `/etc/services`, například webový server používá port 80. Po přiřazení adresy soketu pak server naslouchá příchozím spojením a čeká na požadavky na přidělenou adresu. Původce požadavku, klient, vy-

tvoří soket a předává jím žádost o spojení, přičemž jako cílovou adresu udává adresu serveru. Pro soket INET je adresou serveru jeho IP adresa a číslo portu. Žádost musí projít různými protokolovými vrstvami a nakonec skončí v soketu, na němž server poslouchá. Když server žádost přijme, může ji buď akceptovat, nebo odmítnout. Pokud ji akceptuje, musí server vytvořit nový soket, na němž žádost zpracuje. Soket používaný k příjmu požadavků se totiž nedá zároveň použít k jejich plnění. Po navázání spojení mohou obě strany volně posílat data. Po ukončení přenosů přestává být spojení zapotřebí a může se zrušit. Po celou dobu se zajišťuje, aby bylo s datovými pakety správně naloženo.



Obrázek 10.3
Datové struktury BSD soketu

Přesný význam operací nad BSD sokety záleží na adresové rodině, kde jsou sokety implementovány. Vytvoření TCP/IP spojení je velmi odlišné od navázání spojení pomocí X.25. Stejně jako virtuální souborový systém, i BSD soketovou vrstvu používá Linux jako abstrakci od konkrétních síťových mechanismů. Aplikace pracuje s rozhraním BSD vrstvy, která zajišťuje provedení všech operací příslušnými programy příslušných adresových rodin. V době inicializace jádra se adresové rodiny vestavěné v jádře registrují u BSD rozhraní. Později, když aplikace vytváří a používá BSD sokety, se vytvoří asociace mezi BSD soketem a příslušnou adresovou rodinou. Tyto asociace se vytvářejí pomocí křížových datových struktur a tabulek adres rutin, kterými adresové rodiny nabízejí své specifické služby. Existuje například pro adresovou rodinu specifická rutina pro vytvoření soketu, kterou BSD rozhraní používá, když aplikace chce vytvořit soket.

Při konfiguraci jádra se do vektoru `protocols` vkládá řada adresových rodin a protokolů. Každá položka je reprezentována svým jménem, například „INET“, a adresou inicializační rutiny. Při inicializaci soketového rozhraní v době zavádění systému se postupně volají jednotlivé inicializační rutiny. U adresových rodin vede toto volání k registraci sady protokolových operací. Jedná se o sadu rutin, z nichž každá provádí určitou operaci specificky pro danou adresovou rodinu. Registrované protokolové operace se ukládají ve vektoru `pops`, vektoru ukazatelů na datové struktury `proto_ops`. 1

Datová struktura `proto_ops` se skládá z typu adresové rodiny a z ukazatelů na rutiny soketových operací, specifické pro danou adresovou rodinu. Vektor `pops` je indexován identifikátorem adresové rodiny, například rodiny INET (AF_INET je 2).

10.4 Soketová vrstva INET

Soketová vrstva INET podporuje internetovou adresovou rodinu, která obsahuje protokoly TCP/IP. Jak už bylo popsáno dříve, tyto protokoly jsou vrstveny, jeden protokol využívá služby jiného. TCP/IP kód a datové struktury v Linuxu odpovídají tomuto vrstvení. Prostřednictvím skupiny soketových operací internetové rodiny, které se registrují v době inicializace sítě, se poskytuje rozhraní soketové vrstvě BSD. Soketová vrstva BSD pak volá podpůrné rutiny internetové vrstvy z datové struktury `proto_ops` této registrované vrstvy. Například žádost o vytvoření BSD soketu s uvedením adresy ve vrstvě INET povede k volání rutiny pro vytvoření soketu v internetové vrstvě. U všech těchto operací předává soketová vrstva BSD internetové vrstvě datovou strukturu `socket` reprezentující BSD soket. Aby nevznikaly zmatky tím, že se struktura `socket` naplní informacemi specifickými pro protokoly TCP/IP, používá internetová vrstva vlastní strukturu `sock`, kterou propojuje se strukturou `socket` BSD vrstvy. Toto propojení je znázorněno na obrázku 10.3. Datová struktura `sock` je propojena s datovou strukturou `socket` pomocí ukazatele `data` v BSD soketu. Soketová volání do vrstvy INET tak mohou snadno získat datovou strukturu `sock`. Při vytvoření datové 2

struktury sock se nastaví také ukazatel protokolových operací, který závisí na používaném protokolu. Pokud byl požadován protokol TCP, bude ukazatel protokolových operací ukazovat na sadu TCP operací, potřebných pro práci s TCP spojením.

10.4.1 Vytvoření BSD soketu

Systémové volání pro vytvoření nového soketu přebírá identifikátory adresové rodiny, typu soketu a protokolu.

Podle požadované adresové rodiny se nejprve ve vektoru pops nalezne požadovaná rodina. Může se stát, že určitá adresová rodina je implementována jako modul jádra a v takovém případě bude muset démon kerneld příslušný modul nejprve nahrát. Dále se alokuje nová datová struktura socket reprezentující vytvářený BSD soket. Datová struktura socket je ve skutečnosti fyzicky součástí datové struktury inode VFS a alokace soketu fakticky představuje alokaci inodu VFS. Může to vypadat na první pohled podivně, je ale třeba vzít v úvahu, že se sokety se pracuje stejným způsobem jako s běžnými soubory. Tak jak jsou všechny soubory reprezentovány strukturou VFS inode kvůli zajištění souborových operací, i BSD sokety je nutné ze stejných důvodů reprezentovat jako inody VFS.

Nově vytvořená datová struktura socket obsahuje ukazatel na specifické soketové operace požadované adresové rodiny, který ukazuje na datovou strukturu proto_ops zjištěnou z vektoru pops. Typ se nastaví podle požadovaného typu soketu na jednu z hodnot SOCK_STREAM, SOCK_DGRAM a podobně. Pak se volá rutina vytvoření soketu specifická pro danou adresovou rodinu, jejíž adresa je uložena v datové struktuře proto_ops.

Z vektoru fd aktuálního procesu se alokuje volný deskriptor souboru a inicializuje se datová struktura file, na níž deskriptor ukazuje. Tato inicializace zahrnuje nastavení ukazatele souborových operací na souborové operace BSD soketu, které jsou zajišťovány BSD soketovým rozhraním. Všechny operace se dále směrují na soketové rozhraní, které je předává příslušné adresové rodině prostřednictvím volání rutin specifické adresové rodiny.

10.4.2 Přiřazení adresy BSD soketu INET

Aby mohl server poslouchat a čekat na příchozí žádosti o spojení, musí si vytvořit soket INET BSD a přiřadit mu adresu. Operace přiřazení je z větší části obsluhována soketovou vrstvou INET s částečnou podporou nižších protokolových vrstev TCP a UDP. Soket s přiřazenou adresou se nedá použít k žádné jiné komunikaci. Znamená to, že status ve struktuře socket musí být TCP_CLOSE. Adresa sockaddr předávaná operaci přiřazení obsahuje IP adresu a nepovinně číslo portu. Obvykle se přiřazuje IP adresa přidělená některému síťovému zařízení, které podporuje adresovou rodinu INET, je aktivní a dá se použít. Která síťová rozhraní jsou v systému momentálně aktivní, můžete zjistit příkazem ifconfig. Může se také

přiřadit vysílací IP adresa se samými nulami nebo samými jedničkami. Jedná se o speciální adresy s významem „poslat všem“. IP adresa může být také volena libovolně pokud počítač pracuje jako transparentní proxy-server nebo firewall, přiřazení adresy ovšem může provést pouze proces s oprávněními superuživatele. IP adresa přiřazená soketu se ukládá v datové struktuře `sock` v položkách `recv_addr` a `saddr`. Používají se při hashovacím vyhledávání a jako adresa odesílatele. Číslo portu je nepovinné a pokud není uvedeno, požádá se příslušná podpůrná síťová vrstva o přiřazení volného portu. Konvencí je dáno, že porty s čísly menšími než 1 024 mohou používat pouze superuživatelské procesy. Pokud se port přiřazuje automaticky podpůrnou síťovou vrstvou, je vždy přiřazeno číslo větší než 1 024.

Paket přijatý nějakým síťovým zařízením se musí předat správným soketům INET a BSD, které jej zpracují. Z toho důvodu udržují UDP a TCP hashovací tabulky, které se používají k nalezení adresy při zachycení IP paketu a jeho předávání správné dvojici `socket`/`sock`. TCP je protokol s navazováním spojení, takže zpracování TCP paketu je podstatně složitější než zpracování UDP paketu.

UDP udržuje hashovací tabulku alokovaných UDP portů, tabulku `udp_hash`. Skládá se z ukazatelů na datové struktury `sock` a indexuje se hashovací funkcí založenou na čísle portu. Protože hashovací tabulka je podstatně menší než rozsah dostupných čísel portů (`udp_hash` má pouze 128 položek, respektive `UDP_HTABLE_SIZE` položek), některé položky z tabulky ukazují na řetězce datových struktur `sock`, které jsou propojeny pomocí ukazatele `next` ve struktuře `sock`.

TCP je podstatně složitější a udržuje několik hashovacích tabulek. TCP ovšem nepřidává datovou strukturu `sock` při přiřazování adresy, v té době pouze kontroluje, zda požadovaný port už nebyl přidělen. Datová struktura `sock` se přidává do hashovacích tabulek TCP až při vyvolání operace *listen*.

10.4.3 Vytvoření spojení na BSD soketu

Jakmile soket vytvoříme a nepoužíváme jej k poslouchání příchozích žádostí o spojení, můžeme jej použít k vytvoření odchozí žádosti o navázání spojení. U protokolů bez navazování spojení, jako je například UDP, tato operace nic moc neobnáší, ovšem u protokolů s navazováním spojení (jako je TCP) to obnáší vytvoření virtuálního spoje mezi dvěma aplikacemi.

Odchozí spojení je možno navázat pouze na BSD soket INET, který je ve vhodném stavu - tedy takový, který nemá doposud žádné spojení navázáno a nepoužívá se k zachycování příchozích požadavků. Znamená to, že status ve struktuře `socket` musí být `SS_UNCONNECTED`. UDP protokol nenavazuje virtuální spoj mezi dvěma aplikacemi, všechny jím odesílané zprávy jsou datagramy, tedy zprávy, které mohou, ale nemusejí dorazit ke svému cíli. I tento protokol ale podporuje BSD soketovou operaci *connect*. Operace navázání spojení na UDP soketu jednodu-

še nastaví adresy vzdálené aplikace, tedy její IP adresu a číslo IP portu. Dále se nastavuje vyrovnávací paměť položek směrovací tabulky, takže UDP pakety posílané na daný soket nebudou muset opakovaně načítat směrovací databázi (dokud původní trasa zůstane platná). Na směrovací informace ve vyrovnávací paměti ukazuje položka `ip_route_cache` v datové struktuře `sock`. Pokud nejsou zadány žádné adresovací informace, použijí se při odesílání zpráv daných BSD soketem právě směrovací informace a IP adresační informace uložené ve vyrovnávací paměti. Nakonec převede UDP strukturu `sock` do stavu `TCP_ESTABLISHED`.

Operace navázání spojení na TCP soketu musí nejprve vytvořit TCP zprávu s informacemi o požadavku na spojení a odeslat ji na zadanou cílovou IP adresu. TCP zpráva obsahuje informace o spojení, počáteční sekvenční číslo, maximální velikost zprávy, kterou je schopen žadatel zpracovat, velikost příjmového a odesílacího okna a další. V rámci protokolu TCP jsou všechny zprávy číslovány a počáteční sekvenční číslo se použije jako číslo první zprávy. Linux volí rozumně náhodné hodnoty, aby se předešlo některým typům protokolových útoků. Každá zpráva odeslaná jedním koncem TCP spojení a úspěšně přijatá druhým koncem spojení je potvrzena, takže odesílatel ví, že zpráva dorazila v pořádku a nepoškozená. Nepotvrzené zprávy se posílají znovu. Velikost vysílacího a příjmového okna udává počet zpráv, které je možno odeslat najednou bez potvrzení. Maximální velikost zprávy vychází ze síťového zařízení používaného iniciátorem spojení. Pokud druhá strana spojení podporuje jinou maximální velikost zprávy, použije se vždy minimum z obou hodnot. Aplikace navazující TCP spojení musí počkat na odpověď od vzdálené aplikace, která požadavek buď přijme, nebo odmítne. Protože TCP `sock` nyní čeká na příchozí zprávu, zařadí se do struktury `tcp_listening_hash`, která zajišťuje předávání došlých TCP zpráv správné struktuře `sock`. TCP rovněž spustí časovač, takže pokud na požadavek nepřijde do nějaké doby odpověď, požadavek propadne.

10.4.4 Poslouchání na BSD soketu INET

Když má soket přiřazenu adresu, může poslouchat a čekat na příchozí požadavky na spojení na přiřazené adrese. Síťová aplikace může poslouchat soketem bez předchozího přiřazení adresy, v takovém případě soketová vrstva INET přiřadí soketu automaticky nepoužité číslo portu (v daném protokolu). Soketová funkce poslechu převede soket do stavu `TCP_LISTEN` a provede všechny síťově závislé operace potřebné k příjmu žádostí.

U UDP soketů stačí pouze změnit stav soketu, TCP však přidává aktivovanou strukturu `sock` do dvou hashovacích tabulek. Jde o tabulky `tcp_bound_hash` a `tcp_listening_hash`. Obě se indexují prostřednictvím hashovací funkce založené na čísle IP portu.

Kdykoliv aktivní naslouchající soket přijme příchozí požadavek na navázání TCP spojení, vytvoří pro něj TCP novou datovou strukturu `sock`. Tato datová struktura `sock` se stává základem TCP spojení, pokud bude skutečně akceptováno. Dále se naklonuje příchozí

sk_buff obsahující žádost o spojení a zařadí se do fronty receive_queue naslouchající datové struktury sock. Klon bufferu sk_buff obsahuje ukazatel na nově vytvořenou datovou strukturu sock.

10.4.5 Příjem požadavku na spojení

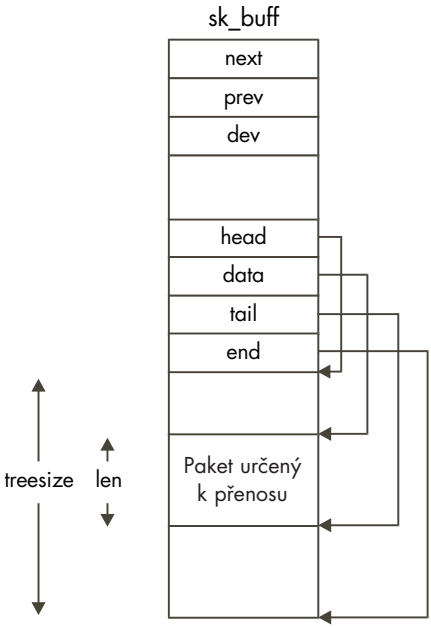
Protokol UDP nepodporuje koncept spojení, takže příjem požadavku na navázání spojení se týká pouze protokolu TCP, u něhož operace přijetí požadavku vede k vytvoření nové datové struktury socket z původního naslouchajícího soketu. Operace přijetí požadavku se pak předá příslušné podpůrné protokolové vrstvě, v tomto případě INET, aby se zajistilo přijetí příchozích požadavků. Protokolová vrstva INET nenechá operaci *accept* proběhnout, pokud podpůrný protokol, řekněme UDP, nepodporuje navazování spojení. V opačném případě se operace *accept* předá protokolu, řekněme TCP. Operace *accept* může být buď blokující, nebo neblokující. U neblokující operace končí operace neúspěšně, pokud není žádné příchozí spojení, které by mělo být přijato, a datová struktura socket se zruší. U blokující operace se síťová aplikace provádějící tuto operaci přidá do čekací fronty a je pozastavena, dokud nebude přijat TCP požadavek. Jakmile dojde požadavek, buffer sk_buff s žádostí se zruší a datová struktura sock je vrácena vrstvě INET, kde je přiřazena dříve vytvořené datové struktuře socket. Síťové aplikaci se vrací deskriptor souboru (fd) nového soketu a aplikace pak může tento deskriptor používat k soketovým operacím nad nově vytvořeným BSD soketem.

10.5 Vrstva IP

10.5.1 Soketové buffery

Jedním z problémů použití mnoha vrstev síťových protokolů, kdy jeden využívá služeb dalšího, spočívá v tom, že každý protokol potřebuje k odesílaným datům přidávat své hlavičky a patičky a při přijetí dat je musí naopak odstraňovat. Tím se komplikuje předávání datových bufferů mezi protokoly, protože každá vrstva musí zjišťovat, kde jsou její hlavičky a patičky. Jedním řešením by bylo kopírování části bufferu v každé vrstvě, to by ovšem bylo neefektivní. Namísto toho používá Linux k předávání dat mezi protokolovými vrstvami a ovladači síťových zařízení datovou strukturu sk_buff. Struktura sk_buff obsahuje ukazatele a délky, takže jednotlivé protokolové vrstvy mohou s daty aplikace pracovat pomocí standardních funkcí či metod.

Na obrázku 10.4 vidíme datovou strukturu sk_buff, ke každé takové struktuře je dále přiřazen blok dat. Struktura sk_buff má čtyři datové ukazatele, které slouží k manipulaci a úpravám nad daty soketového bufferu:



Obrázek 10.4
Soketový buffer (`sk_buff`)

head Ukazuje na začátek datové oblasti v paměti. Tato hodnota je dána v okamžiku, kdy se alokuje struktura `sk_buff` a její datová oblast.

data Ukazuje na momentální začátek protokolových dat. Tento ukazatel se mění podle toho, která protokolová vrstva momentálně strukturu `sk_buff` vlastní.

tail Ukazuje na momentální konec protokolových dat. Tato hodnota opět závisí na protokolu, který strukturu vlastní.

end Ukazuje na konec datové oblasti v paměti. Hodnota je pevně dána při alokaci bufferu.

Dále struktura obsahuje dvě délkové položky, `len` a `truesize`, které obsahují délku paketu v aktuálním protokolu a celkovou velikost datového bufferu. Obslužný kód struktury `sk_buff` poskytuje standardní mechanismy pro přidávání a rušení hlaviček a patiček jednotlivých protokolů. Tyto operace zajišťují koherentní manipulace s položkami `data`, `tail` a `len`:

push Přesune ukazatel `data` směrem k začátku datové oblasti a inkrementuje údaj `len`. Používá se při přidávání dat nebo protokolových hlaviček na začátek odesílaných dat.

6 **pull** Přesouvá ukazatel `data` dále od začátku, směrem ke konci datové oblasti, a dekrementuje údaj `len`. Používá se při rušení dat nebo protokolových hlaviček ze začátku oblasti.

7 **put** Přesouvá ukazatel `tail` směrem od začátku a inkrementuje údaj `len`. Používá se při přidávání dat nebo protokolových informací na konec stávajících dat.

8 **trim** Přesouvá ukazatel `tail` směrem k začátku oblasti a dekrementuje údaj `len`. Používá se při rušení dat nebo protokolových patiček z konce stávajících dat.

Datová struktura `sk_buff` dále obsahuje ukazatele používané pro ukládání struktur do obousměrně propojeného kruhového seznamu při jejich zpracovávání. Existují obecné funkce nad strukturou `sk_buff`, které zajišťují přidávání a rušení struktury na začátek a konec těchto seznamů.

10.5.2 Příjem IP paketů

V kapitole „Ovladače zařízení“ je popsáno, jak se ovladače síťových zařízení přidávají do jádra a inicializují. Výsledkem je několik datových struktur `device` propojených v seznamu `dev_base`. Každá datová struktura `device` popisuje své zařízení a poskytuje množinu operací, které volají síťové protokoly, když potřebují, aby ovladač síťového zařízení provedl nějakou operaci. Tyto funkce se vesměs týkají odesílání dat a práce s adresou síťového zařízení. Když síťové zařízení zachytí paket ze sítě, musí přijatá data zkonvertovat na strukturu
9 `sk_buff`. Tyto struktury `sk_buff` se pak přidávají do fronty `backlog` síťových zařízení.

Když se fronta `backlog` příliš rozroste, přijaté buffery `sk_buff` se ruší. Nastaví se příznak spuštění síťového bottom-half ovladače, který bude muset zajistit zpracování přijatých paketů.

Když plánovač spustí síťový bottom-half ovladač, zpracují se nejprve všechny síťové pakety
10 čekající na odeslání a poté se zpracovávají buffery `sk_buff` ve frontě `backlog` a rozhoduje se, které protokolové vrstvě se mají přijaté pakety předat.

Při inicializaci síťového systému se každý protokol registruje přidáním datové struktury `packet_type` buď do seznamu `ptype_all`, nebo do hashovací tabulky `ptype_base`. Datová struktura `packet_type` obsahuje typ protokolu, ukazatel na síťové zařízení, ukazatel na rutinu zpracování přijatých dat a konečně ukazatel na další strukturu `packet_type` v seznamu nebo v hashovacích řetězcích. Seznam `ptype_all` může sloužit k identifikaci všech paketů přijatých ze sítě, normálně se však nepoužívá. Hashovací tabulka `ptype_base` je hashována identifikátorem protokolu a slouží k rozhodování, kterému protokolu se má předat došlý síťový paket. Síťový bottom-half ovladač porovnává typ protokolu v došlém bufferu `sk_buff` s jednou nebo více položkami `packet_type` v tabulce. Protokolu může odpoví-

dat více než jedna položka, například při odposlechu veškerého provozu na síti, a v takovém případě se buffer `sk_buff` klonuje. Pak se buffer `sk_buff` předá obslužné rutině odpovídajícího protokolu.

10.5.3 Odesílání IP paketů

Pakety jsou buď odesílány síťovými aplikacemi při výměně dat, nebo je generují samotné síťové protokoly při navazování spojení a podpoře navázaného spojení. Ať už paket vznikne jakkoliv, vytvoří se pro uložení jeho dat a hlaviček přidávaných protokoly různých vrstev struktura `sk_buff`.

Strukturu `sk_buff` je třeba předat síťovému zařízení, které ji odešle. Nejprve se protokol, řekněme IP, musí rozhodnout, které síťové zařízení má použít. Záleží to na optimální trase. Pokud je počítač připojen k síti jediným modemem, například protokolem PPP, je volba trasy jednoduchá. Paket se bude buď zpětným zařízením posílat na lokální počítač, nebo se bude posílat na bránu na druhém konci modemového spojení. U počítačů připojených k síti ethernet je rozhodování složitější, protože v jednom okamžiku mají přístupných mnoho počítačů.

U každého odesílaného paketu používá protokol IP směrovací tabulky, v nichž zjišťuje trasu k cílové adrese. Každá IP adresa úspěšně nalezená ve směrovací tabulce vrací strukturu `rtable`, která popisuje trasu, jež se má použít. Sem spadá jednak zdrojová IP adresa, která se má použít, adresa datové struktury `device` síťového zařízení a případně nezbytné hardwarově závislé údaje. Tyto údaje se týkají přímo síťového zařízení a obsahují například fyzickou zdrojovou a cílovou adresu a další informace specifické pro konkrétní médium. Pokud je síťovým zařízením ethernet, pak budou hardwarové údaje vypadat stejně jako na obrázku 10.1 a zdrojová a cílová adresa budou fyzické ethernetové adresy. Hardwarová hlavička se trvale uchovává u každé trasy, protože je nutné ji připojit ke každému odesílanému paketu a její opakované sestavování by zbytečně zdržovalo. Hardwarová hlavička může obsahovat fyzické adresy, které se musejí nejprve přeložit protokolem ARP. V takovém případě bude odeslání paketu pozastaveno, dokud se nepodaří potřebné adresy zjistit. Jakmile jsou adresy jednou určeny a sestaví se hardwarová hlavička, uloží se, takže další odesílané pakety už nebudou muset protokolem ARP nic zjišťovat.

10.5.4 Fragmentace dat

Každé síťové zařízení má dánu maximální velikost paketu a není schopno odeslat nebo přimout paket větší. S tím IP protokol počítá a je schopen fragmentovat data na menší jednotky, aby se vyhovělo maximální velikosti paketu toho zařízení, které bude paket přenášet. Hlavička IP paketu obsahuje i fragmentační pole, v němž je uložen příznak fragmentace a fragmentační offset.

13 Když je IP paket připraven k odeslání, IP nalezne síťové zařízení, přes nějž bude paket odesílat. Toto zařízení se nalezne podle směrovacích tabulek IP protokolu. Každá položka `device` obsahuje mimo jiné údaj o maximální přenosové jednotce (v bajtech), údaj `mtu`. Pokud je *mtu* zařízení menší než velikost IP paketu, jež se má odeslat, je nutné rozdělit IP paket na menší části (o velikosti *mtu*). Každý fragment je reprezentován vlastní strukturou `sk_buff`, v IP hlavičce je označeno, že se jedná o fragment a je uveden jeho offset v původním paketu. Poslední fragment je označen jako poslední. Pokud v průběhu fragmentace nebude IP protokol schopen alokovat další strukturu `sk_buff`, odeslání se nezdaří.

Příjem IP fragmentů je poněkud složitější než jejich odesílání, protože jednotlivé fragmenty mohou být obecně přijaty v libovolném pořadí a před složením je nutné přijmout všechny.
14 Vždy, když se přijme IP paket, kontroluje se, zda se nejedná o IP fragment. Když se přijme první fragment zprávy, IP vytvoří novou datovou strukturu `ipq`, která se připojí k seznamu `ipqueue` IP fragmentů, čekajících na rekombinaci. Při příjmu dalších fragmentů se vždy nalezne správná struktura `ipq` a přidá se k ní nová struktura `ipfrag` popisující nově přijatý fragment. Každá datová struktura `ipq` jednoznačně určuje fragmentovanou zprávu podle zdrojové a cílové IP adresy, identifikátoru protokolu a identifikátoru IP rámce. Jakmile se přijmou všechny fragmenty, zkombinují se do jediné struktury `sk_buff` a postoupí se ke zpracování vyšší protokolové vrstvě. Každá struktura `ipq` obsahuje časovač znovu spouštěný po přijetí každého nového fragmentu. Pokud tento časovač vyprší, datová struktura `ipq` a její struktury `ipfrag` budou zrušeny a předpokládá se, že se části paketu cestou ztratily. Pak záleží na nadřízené protokolové vrstvě, aby zajistila opakované zaslání zprávy.

10.6 Protokol ARP (Address Resolution Protocol)

Protokol ARP je nástroj pro překlad IP adres na fyzické hardwarové adresy, například ethernetové adresy. Protokol IP potřebuje tento překlad předtím, než může zařízení předat paket
15 (ve formátu `sk_buff`) k odeslání.

Protokol IP provádí různé kontroly, aby zjistil, zda příslušné zařízení potřebuje hardwarovou hlavičku, a pokud ano, zda není nutné vybudovat hlavičku znovu. Linux ukládá hardwarové
16 hlavičky ve vyrovnávací paměti, aby je nemusel neustále opakovaně vytvářet. Pokud je nutné sestavit hardwarovou hlavičku znovu, zavolá se rutina sestavení hlavičky v ovladači příslušného síťového zařízení. Všechna ethernetová zařízení používají stejnou obecnou rutinu pro sestavení hlavičky, která používá protokol ARP k překladu cílové IP adresy na fyzickou adresu.

Protokol ARP je sám o sobě velmi jednoduchý a skládá se ze dvou typů zpráv: ARP žádosti a ARP odpovědi. ARP žádost obsahuje IP adresu kterou je třeba přeložit, a odpověď obsahuje její překlad, hardwarovou adresu. ARP žádost se hromadně zašle na všechny počítače při-

pojené k síti, takže například na ethernetové síti zachytí ARP žádost všechny počítače na stejném segmentu. Počítač s požadovanou IP adresou reaguje na žádost a pošle odpověď, v níž uvádí svou hardwarovou adresu.

Protokolová vrstva ARP je v Linuxu postavena na tabulce datových struktur `arp_table`, které každá popisují překlad jedné IP adresy na fyzickou adresu. Tyto položky se vytvářejí, když je potřeba přeložit IP adresu, a odstraňují se po určité době. Každá datová struktura `arp_table` má následující položky:

poslední použití	Čas, kdy byla tato položka naposledy použita.
poslední aktualizace	Čas, kdy byla položka naposledy aktualizována.
příznaky	Popisují stav položky, například zda je úplná a podobně.
IP adresa	IP adresa, kterou tato položka popisuje.
hardwarová adresa	Přeložená hardwarová adresa.
hardwarová hlavička	Ukazatel na uloženou hardwarovou hlavičku.
časovač	Položka `timer_lost` používaná k rušení ARP žádostí, na něž nepřišla odpověď.
opakování	Počet pokusů, kolikrát se vznášela žádost.
fronta `sk_buff`	Seznam struktur `sk_buff`, které čekají na překlad této IP adresy.

ARP tabulka se skládá z tabulky ukazatelů (vektor `arp_tables`) na řetězce položek `arp_table`. Položky se kvůli zrychlení přístupu ukládají tak, že poslední dva bajty IP adresy generují index do tabulky a pak se prohlíží řetězec položek, až se najde ta správná. Linux ukládá připravené hardwarové hlavičky rovněž ve struktuře `hh_cache`, do níž struktura `arp_table` ukazuje.

Když je požadován překlad IP adresy a v ARP tabulce není požadovaná položka nalezena, musí ARP poslat ARP žádost. Vytvoří se nová položka `arp_table` a do její fronty se zařadí `sk_buff` obsahující paket, který překlad potřebuje. Odešle se ARP žádost a spustí se časovač. Pokud se do zadaného intervalu neobjeví odpověď, ARP několikrát žádost zopakuje a pokud odpověď stále nepřichází, odstraní se položka `arp_table`. Na tuto skutečnost budou upozorněny všechny struktury `sk_buff`, které čekaly ve frontě zrušené položky, a záleží na protokolové vrstvě, která je chtěla odeslat, jak se s tím vyrovná. UDP se o ztracené pakety nestará, TCP se pokusí o opakované odeslání. Pokud vlastník požadované IP adresy odpoví svou hardwarovou adresou, položka `arp_table` tabulky se označí jako úplná a všechny struktury `sk_buff` v její frontě budou předány k odeslání. Hardwarová adresa se zapíše do hardwarové hlavičky každého bufferu `sk_buff`.

Protokolová vrstva ARP musí také odpovídat na ARP žádosti na IP adresu lokálního počítače. Registruje si svůj protokol (`ETH_P_ARP`), čímž se generuje datová struktura `packet_type`. Znamená to, že se jí budou předávat všechny přijaté ARP pakety. Kromě ARP odpovědí tedy vrstva obdrží i všechny ARP žádosti. Sama generuje ARP odpovědi podle hardwarové adresy uložené ve struktuře `device` toho zařízení, které žádost přijalo.

S postupem času se topologie sítě může měnit a IP adresy mohou být přiřazovány jiným hardwarovým adresám. Například některé vytáčené služby přiřazují IP adresy vždy při navázání spojení. Aby ARP tabulka obsahovala aktuální informace, spouští ARP periodický časovač, který prohlédne všechny struktury `arp_table` a hledá a odstraňuje prošlé. Zároveň ovšem hlídá, aby neodstranil položky, kterým je přiřazena hardwarová hlavička. Odstranění takovýchto položek by mohlo být nebezpečné, protože na nich závisí další datové struktury. Některé položky ARP tabulky jsou trvalé a jsou označeny tak, že nebudou nikdy zrušeny. Vždy, když se alokuje nová položka a ARP tabulka by dosáhla své maximální velikosti, tabulka se redukuje vyhledáním a zrušením nejstarších položek.

10.7 Směrování

Směrování rozhoduje o tom, kam poslat IP pakety určené konkrétní IP adrese. Při odesílání IP paketu je mnoho rozhodování. Dá se cíle vůbec dosáhnout? Pokud ano, kterým síťovým zařízením paket odeslat? Pokud je možno použít více síťových zařízení, které je nejlepší? Odpovědi na tyto otázky poskytuje směrovací tabulka. Jedná se o dvě databáze, důležitější je *databáze směrovacích informací*. Jedná se o úplný seznam známých IP cílů a nejlepších tras k nim. Druhá, menší a rychlejší databáze, *směrovací cache*, slouží k rychlému nalezení tras na různé cíle. Stejně jako všechny vyrovnávací paměti, i ona obsahuje pouze nejčastěji používané trasy, její obsah se odvozuje z databáze směrovacích informací.

Trasy se přidávají a ruší na základě IOCTL žádostí soketového rozhraní BSD. Tyto žádosti se předávají ke zpracování příslušnému protokolu. Protokolová vrstva INET povoluje rušit a přidávat IP trasy pouze procesům s oprávněními superuživatele. Trasy mohou být pevné nebo se mohou v čase dynamicky měnit. Většina systémů používá pevné trasy, dynamické trasy obvykle používají pouze routery. Router používá směrovací protokoly, jimiž trvale ověřuje dostupnost tras ke všem známým IP cílům. Systémy, které nepracují jako routery, se označují jako koncové systémy. Směrovací protokoly jsou implementovány jako démony, například *gated*, a trasy přidávají a ruší rovněž přes funkci IOCTL soketového rozhraní BSD.

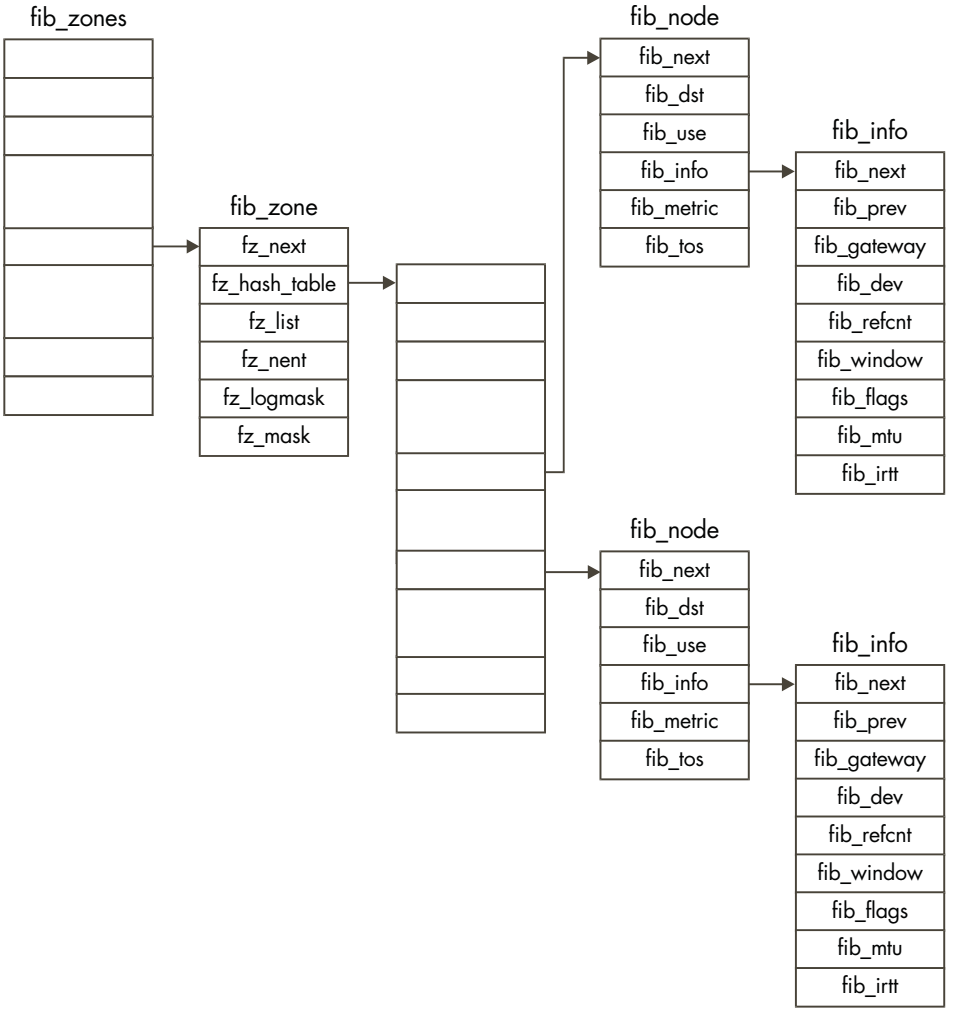
10.7.1 Směrovací cache

Vždy, když se hledá trasa, prohlíží se nejprve směrovací cache. Pokud v ní není požadovaná trasa nalezena, hledá se trasa v databázi směrovacích informací. Pokud trasa není ani tam, není možno paket odeslat a aplikace na to bude upozorněna. Pokud je trasa v databázi směrovacích informací a není ve směrovací cache, vytvoří se pro ni nová položka a přidá se do cache. Směrovací cache je tabulka (`ip_rt_hash_table`), která obsahuje ukazatele na řetězce struktur `rtable`. Index do směrovací tabulky je hashovací funkce založená na dvou nejméně významných bajtech IP adresy. U těchto dvou bajtů je nejpravděpodobnější, že se budou pro různé cíle lišit, a tak se zajišťuje nejlepší rozptyl hodnot v hashovací tabulce. Každá struktura `rtable` obsahuje informace o trase: cílovou IP adresu, zařízení používané pro směrování k danému cíli, maximální povolenou velikost paketu tohoto zařízení a další. Obsahuje také referenční počitadlo, počitadlo použití a časovou značku doby posledního použití. Referenční počitadlo se inkrementuje vždy, když je trasa používána, aby se zaznamenával počet síťových spojení, které danou trasu používají. Hodnota se dekrementuje, když zařízení přestane trasu používat. Počitadlo použití se inkrementuje při každém hledání trasy a používá se k seřazování položek `rtable` v jednotlivých řetězcích hashovací tabulky. Periodicky se kontroluje časová značka jednotlivých položek a zjišťuje se, zda položka není příliš stará. Pokud trasa nebyla delší dobu použita, odstraní se z vyrovnávací paměti. Trasy jsou ve vyrovnávací paměti uloženy tak, že nejčastěji používané trasy stojí na počátku hashovacích řetězců. Díky tomu je jejich nalezení nejrychlejší.

10.7.2 Databáze směrovacích informací

Databáze směrovacích informací (viz obr. 10.5) obsahuje přehled všech tras, které systém v daném okamžiku zná. Jedná se o poměrně komplikovanou datovou strukturu a i když je uspořádána poměrně efektivně, rozhodně se v ní nehledá příliš rychle. Bylo by každopádně velmi pomalé hledat v této databázi trasu pro každý odesílaný IP paket. Z toho důvodu se používá směrovací cache, která zrychluje odesílání IP paketu na často používané adresy. Směrovací cache se odvozuje z databáze směrovacích informací a obsahuje její nejčastěji používané položky.

Každá IP podsíť je reprezentována datovou strukturou `fib_zone`. Na tyto struktury se ukazuje z hashovací tabulky `fib_zones`. Hashovací index se odvozuje z masky IP podsítě. Všechny trasy na stejné podsítě jsou popsány párem struktur `fib_node` a `fib_info` uložených ve frontě `fz_list` každé datové struktury `fib_zone`. Pokud v jedné podsíti bude velké množství tras, vygeneruje se hashovací tabulka, která usnadní hledání struktur `fib_node`.



Obrázek 10.5
Databáze směrovacích informací

Na stejnou podsíť může existovat několik tras, které vedou přes různé brány. Směrovací vrstva IP protokolu nepovoluje více tras na jednu podsíť přes stejnou bránu. Jinak řečeno, pokud na jednu podsíť vede více tras, každá vede přes jinou bránu. Každá trasa má přiřazenu svou *metriku*. Metrika udává míru výhodnosti trasy. V zásadě představuje metrika trasy počet podsítí, přes které se musí projít, než se dojde k požadovanému cíli. Čím vyšší metrika, tím horší trasa.

Odkazy na zdrojové texty jádra

1 – Viz include/-linux/net.h
2 – Viz include/-net/sock.h
3 – Viz sys_socket() in net/socket.c
4 – Viz include/-linux skbuff.h
5 – Viz skb_push() in include/-linux/skbuff.h
6 – Viz skb_pull() in include/-linux skbuff.h
7 – Viz skb_put() in include/linux/-skbuff.h
8 – Viz skb_trim() in include/-linux/skbuff.h
9 – Viz netif_rx() in net/core/dev.c
10 – Viz net_bh() in net/core/dev.c
11 – Viz ip_recv() in net/ipv4/-ip_input.c
12 – Viz include/-net/route.h
13 – Viz ip_build_xmit() in net/ ipv4/ip_output.c
14 – Viz ip_rcv() in net/ipv4/-ip_input.c
15 – Viz ip_build_xmit() in net/ipv4/-ip_output.c
16 – Viz rebuild_header()in net/-ethernet/eth.c
17 – Viz ip_rt_check_expire() in net/ipv4/-route.c

Kapitola

11

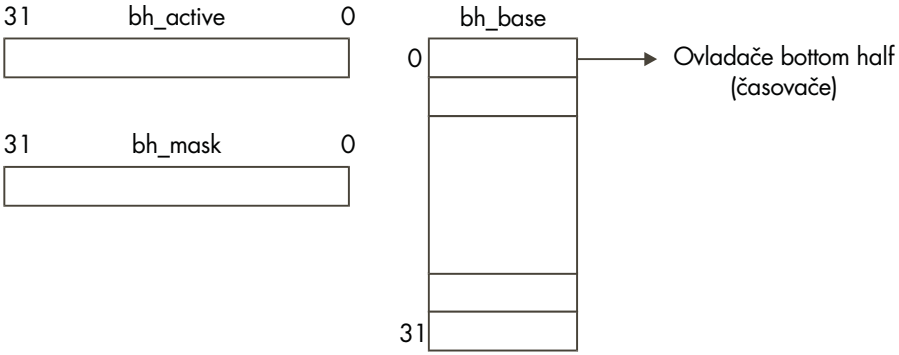
Mechanismy jádra

V této kapitole jsou popsány některé obecné činnosti a mechanismy, které musí jádro Linuxu zajišťovat, aby mohly jednotlivé jeho části efektivně spolupracovat.

11.1 Obslužný mechanismus „bottom-half“

Velmi často se v jádře stává, že nějakou činnost nechcete vykonat okamžitě. Typickým příklaem je zpracování přerušení. Když dojde k přerušení, procesor přeruší právě prováděnou činnost a operační systém doručí obsluhu přerušení příslušnému ovladači zařízení. Ovladače ale nemohou strávit obsluhou příliš mnoho času, protože v době obsluhy přerušení nemůže v systému běžet nic jiného. Často by se při obsluze prováděla činnost, kterou je možno stejně dobře vykonat až někdy později. Obslužný mechanismus bottom-half byl vyvinut právě k tomu, aby ovladače zařízení a další části jádra mohly naplánovat pozdější provedení nějaké činnosti. Na obrázku 11.1 jsou znázorněny datové struktury, které s tímto typem obsluhy souvisejí.

Může existovat až 32 bottom-half handlerů, `bh_base` je vektor ukazatelů na obslužné rutiny jednotlivých handlerů. Struktury `bh_active` a `bh_mask` mají příslušné bity nastaveny podle toho, které handlery byly instalovány a jsou aktivní. Pokud je nastaven N-tý bit struktury `bh_mask`, znamená to, že N-tý prvek pole `bh_base` obsahuje platnou adresu obslužné rutiny. Pokud je nastaven N-tý bit masky `bh_active`, znamená to, že obslužná rutina by měla být volána jakmile to plánovač uzná za vhodné. Tyto indexy jsou definovány staticky. Bottom-half handler časovače má nejvyšší prioritu (index 0), následuje bottom-half handler konzoly (s indexem 1) a tak dále.Typicky mají tyto obslužné rutiny k sobě přiřazen seznam úloh, které je zapotřebí provést. Například bottom-half handler *immediate* zpracovává úkoly z fronty akutních úloh (`tq_immediate`), která obsahuje úkoly, jež je nutno provést co nejdříve.



Obrázek 11.1
Datové struktury bottom-half obsluhy

Některé bottom-half handlery v jádře jsou specifické podle zařízení, další jsou ale obecnější:

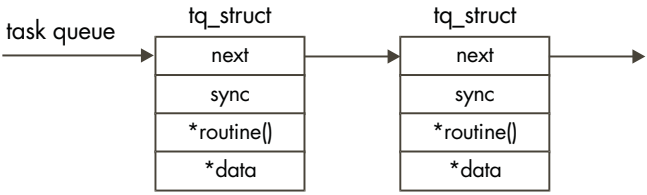
TIMER	Tento handler se aktivuje vždy, když se objeví přerušení od systémového časovače, a slouží k obsluze front časovačů v jádře.
CONSOLE	Tento handler zpracovává zprávy konzoly.
TQUEUE	Tento handler zpracovává zprávy od zařízení `tty`.
NET	Tento handler provádí obecnou obsluhu sítí.
IMMEDIATE	Obecný handler používaný různými ovladači zařízení k provádění činností, odložených na později.

Kdykoliv ovladač zařízení nebo nějaká jiná část jádra potřebuje naplánovat nějakou činnost na později, přidá úkol do vhodné systémové fronty, například do fronty časovače, a pak signalizuje jádru, že je třeba provést nějakou bottom-half obsluhu. Dosáhne se toho nastavením příslušného bitu v `bh_active`. Bit 8 se nastavuje v případě, že ovladač zařadil nějaký úkol do fronty handleru *immediate* a přeje si, aby byl handler *immediate* spuštěn a provedl úkol. Bitová mapa `bh_active` se kontroluje na konci každého systémového volání, těsně před předáním řízení zpět volajícímu procesu. Pokud má nějaký bit nastaven, volají se obslužné rutiny aktivovaných bottom-half handlerů. Nejprve se kontroluje bit 0, poté bit 1 a tak dále až k bitu 31.

2 Bit ve struktuře `bh_active` se nuluje po volání příslušného bottom-half handleru. Struktura `bh_active` je transientní, má význam pouze mezi voláními plánovače a představuje metodu, jak nevolat bottom-half handlery v případě, kdy pro ně není žádná práce.

11.2 Fronty úloh

Fronty úloh představují způsob, kterým jádro odkládá různé činnosti na pozdější dobu. Linux má obecné mechanismy pro řazení úloh do front a jejich pozdější provedení.



Obrázek 11.2
Fronta úloh

Fronty úloh se často používají společně s bottom-half obsluhou, například při volání bottom-half obsluhy časovače se zpracovává fronta úloh časovače. Fronta úloh je jednoduchá datová struktura, jak vidíme na obrázku 11.2. Skládá se z lineárního seznamu struktur `tq_struct`, z nichž každá obsahuje adresu rutiny a ukazatel na nějaká data.

Rutina se bude volat při zpracovávání daného prvku fronty a předá se jí ukazatel na data.

Jakákoliv část jádra, například ovladač zařízení, může vytvářet a používat fronty úloh, kromě toho existují tři fronty úloh vytvořené a udržované jádrem:

timer Fronta časovače slouží k řazení úloh, které se provedou s příštím tikem systémových hodin. Při každém tiku se tato fronta kontroluje, zda obsahuje nějaké položky, a pokud ano, aktivuje se bottom-half handler časovače. Bottom-half handler časovače se volá, stejně jako ostatní bottom-half handlery, při příštím běhu plánovače. Nezaměňujte tuto frontu se systémovými časovači, které představují podstatně inteligentnější mechanismus.

immediate Fronta akutních úloh se rovněž zpracovává když plánovač aktivuje bottom-half handlery. Handler obsluhy akutní fronty má nižší prioritu než obsluha časovače, takže úlohy v této frontě budou provedeny později.

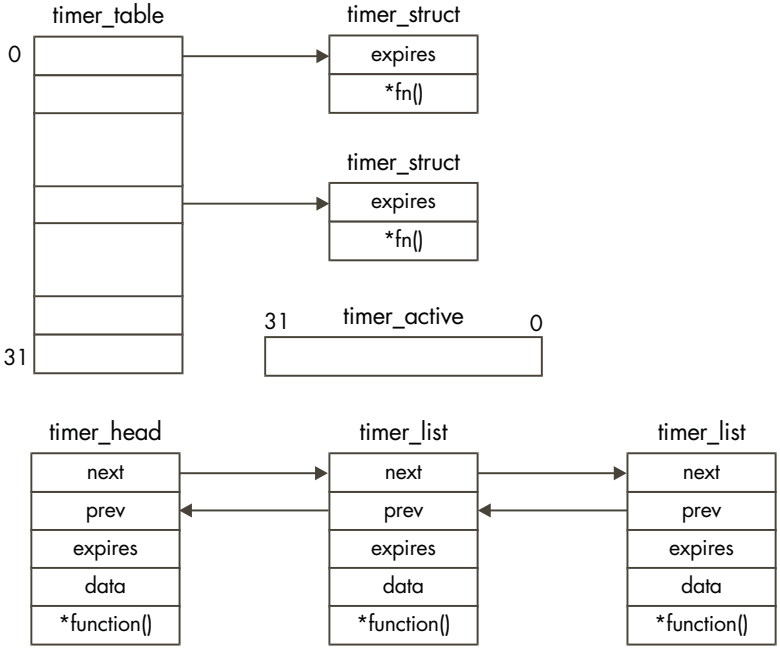
scheduler Frontu plánovače zpracovává přímo plánovač. Slouží k podpoře dalších front úloh v systému a řadí se do ní rutiny, které zpracovávají ostatní fronty úloh, například fronty úloh vytvářené ovladači zařízení.

Při zpracovávání fronty úloh se nejprve zruší ukazatel na první prvek fronty a přiřadí se mu hodnota NULL. Toto odstranění je atomická operace, tedy operace, kterou není možno přerušit. Následně se volá obslužná rutina každého prvku fronty. Prvky fronty často bývají static-

ky alokovaná data. Neexistuje však žádný obecný mechanismus pro uvolnění alokované paměti. Obsluha fronty se pouze přesouvá z jednoho prvku na druhý. Korektní uvolnění alokované paměti jádra musí zajistit přímo volaná rutina každé položky.

11.3 Časovače

Operační systém musí být schopen naplánovat provedení nějaké úlohy někdy v budoucnu. Je nutný mechanismus, který zajistí, aby se daná úloha provedla někdy v budoucnu v relativně přesně určeném okamžiku. Každý mikroprocesor musí být vybaven programovatelným intervalovým časovačem, který pravidelně přerušuje procesor. Toto pravidelné přerušení se označuje jako systémové hodiny a funguje trochu jako metronom, kdy udává rytmus systémových aktivit.



Obrázek 11.3
Systémové časovače

Linux má velmi jednoduchou představu o čase, měří čas v počtu systémových tiků, k nimž došlo od zavedení systému. Všechny časy v systému se měří v těchto jednotkách, které se označují jako `jiffies` podle globálně přístupné proměnné stejného jména.

Linux používá dva typy systémových časovačů, oba řadí do front úkoly, které se mají provést v nějakém čase, jejich implementace se však mírně liší. Na obrázku 11.3 jsou znázorněny oba tyto mechanismy.

První, starší mechanismus obsahuje statické pole 32 ukazatelů na datové struktury `timer_struct` a masku aktivních časovačů, `timer_active`. Řazení časovačů v tabulce časovačů je staticky definováno (podobně jako v tabulce `bh_base` bottom-half obsluhy). Většina položek se do této tabulky přidává v době inicializace systému. Druhý, novější mechanismus používá seznam datových struktur `timer_list`, které jsou řazeny vzestupně podle doby, kdy příslušné časovače vyprší.

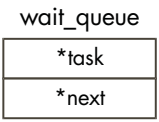
Obě metody měří čas k vypršení časovače v `jiffies`, takže časovač, který chce být aktivován za pět sekund, musí nejprve převést pět sekund na `jiffies` a přičíst získanou hodnotu k aktuálnímu systémovému času, čímž získá čas, v němž má vypršet. S každým tikem systémových hodin se aktivuje bottom-half handler časovače, takže při příštím běhu plánovače budou zpracovány položky ve frontě úloh časovače. Bottom-half handler časovače zpracovává časovače obou typů.

U starších časovačů se kontroluje bitová maska `timer_active` aby se zjistilo, zda je příslušný časovač aktivní. Pokud daný časovač vypršel (jeho čas vypršení je menší než aktuální hodnota `jiffies`), volá se jeho obslužná rutina a vynuluje se příznak jeho aktivity.

Při obsluze novějších typů časovačů se kontrolují položky v seznamu struktur `timer_list`. Každý vypršený časovač se odstraní ze seznamu a zavolá se jeho obslužná rutina. Novější mechanismus má tu výhodu, že obslužné rutině časovače je možno předávat parametry.

11.4 Čekací fronty

Často se stává, že proces musí čekat na nějaký systémový prostředek. Proces například potřebuje ke své práci VFS inode popisující nějaký adresář souborového systému, daný inode však není přítomen ve vyrovnávacích pamětech. V takovém případě musí proces počkat, dokud se inode nenačte z fyzického média obsahujícího daný souborový systém.



Obrázek 11.4
Čekací fronta

8 Linux používá jednoduchou datovou strukturu, čekací frontu (viz obrázek 11.4), která se skládá z ukazatele na strukturu `task_struct` procesu a z ukazatele na další prvek v čekací frontě.

Proces přidaný na konec čekací fronty může být buď přerušitelný, nebo nepřerušitelný. Přerušitelné procesy mohou být v čekání přerušeny událostmi jako vypršení časovače nebo doručení signálu. Tato vlastnost se odráží ve stavu čekajícího procesu, který může být buď `INTERRUPTIBLE`, nebo `UNINTERRUPTIBLE`. Jakmile je proces přidán do čekací fronty, nemůže dále pokračovat v práci a spouští se plánovač, který naplánuje běh jiného procesu.

Při zpracovávání čekací fronty se stav každého procesu mění na `RUNNING`. Pokud byl proces odstraněn z fronty běžících procesů, opět se do ní přidá. Při příštím běhu plánovače se kandidáty na spuštění stávají i procesy v čekací frontě, protože nyní už nečekají. Když se naplánuje spuštění procesu v čekací frontě, proces jako první odstraní sám sebe z čekací fronty. Čekací fronty se používají například k synchronizaci přístupu k systémovým prostředkům a používají se rovněž při implementaci semaforů (viz dále).

11.5 Zámky

Zámky (*buzz locks* nebo *spin locks*) představují jednoduchou metodu ochrany datových struktur nebo částí kódu. Umožňují, aby v daném okamžiku vykonával kritickou sekci kódu pouze jediný proces. Linux je používá při omezení přístupu k položkám svých datových struktur, jako zámek slouží jedna položka struktury. Vždy, když chce nějaký proces změnit nějakou část datové struktury, změní hodnotu zámku z 0 na 1. Pokud je hodnota rovna 1, proces ji stále testuje a běhá v krátké testovací smyčce. Přístup k paměťové oblasti, v níž je zámek uložen, musí být atomickým, operace čtení hodnoty zámku, testování, zda je nulová, a nastavení na jedničku nesmí být přerušitelná žádným jiným procesem. Většina procesorových architektur nabízí speciální instrukce pro podporu takovýchto operací, zámky je však možno implementovat i s využitím necachované hlavní paměti.

Když proces opouští kritickou sekci kódu, dekrementuje zámek a vrací tak jeho hodnotu zpět na nulu. Pokud nějaký proces právě běhá v čekací smyčce na kritickou sekci, zjistí, že hodnota je nyní nulová, inkrementuje ji na jedničku a vstoupí do kritické sekce sám.

11.6 Semafory

Semafory slouží k ochraně kritických sekcí kódu nebo datových struktur. Připomeňme si, že všechny přístupy ke kritickým částem dat, například k inodům VFS, provádí jádro jménem nějakého procesu. Bylo by velmi nebezpečné, kdyby nějaký proces mohl měnit kritická data, používaná právě jiným procesem. Jedna metoda ochrany by mohla spočívat v použití zámků

kolem kritických dat, jedná se však o velmi prostou metodu, která by nevedla k optimálnímu výkonu systému. Namísto toho používá Linux semafory, které umožňují přístup ke kritickým částem kódu a dat vždy jen jednomu procesu, ostatní procesy, které požadují přístup ke stejnému prostředku, budou čekat až do doby, než se prostředek uvolní. Čekající procesy jsou pozastaveny, takže v běhu mohou pokračovat jiné procesy.

Datová struktura `semaphore` v Linuxu obsahuje následující informace:

count Tato položka udává počet procesů, které mohou daný prostředek použít. Kladná hodnota znamená, že prostředek je možno použít. Záporná nebo nulová hodnota znamená, že nějaký proces na prostředek čeká. Počáteční hodnota 1 znamená, že daný prostředek může použít právě jeden proces. Když proces požaduje prostředek, dekrementuje tuto hodnotu, a když skončí s jeho použitím, inkrementuje ji.

waking Počet procesů čekajících na daný prostředek, tedy počet procesů, které je třeba probudit, jakmile se prostředek uvolní.

wait queue Když proces čeká na prostředek, je přesunut do čekací fronty.

lock Zámek používaný při přístupu k položce `waking`.

Předpokládejme, že počáteční hodnota semaforu je 1. První proces, který k němu dorazí, zjistí, že počitadlo je kladné a dekrementuje je o jedničku, nová hodnota tedy bude 0. Proces nyní „vlastní“ kritickou část kódu nebo prostředek, chráněný semaforem. Když proces kritickou sekci opouští, inkrementuje počitadlo semaforu. Nejoptimálnější případ nastává tehdy, pokud v dané chvíli žádný jiný proces nečeká na vlastnictví chráněné sekce. Semafory jsou v Linuxu optimalizovány tak, aby pracovaly nejefektivněji právě v tomto, nejčastějším případě.

Pokud chce vlastnictví kritické sekce, právě vlastněné nějakým procesem, získat další proces, rovněž dekrementuje počitadlo. Hodnota počitadla bude nyní záporná (-1) a proces nemůže vstoupit do kritické sekce. Namísto toho musí počkat, dokud ji vlastník neopustí. Čekající proces se přidá do čekací fronty semaforu a cyklicky testuje hodnotu `waking` s tím, že dokud tato hodnota nebude nenulová, předává vždy řízení plánovači.

Vlastník kritické sekce inkrementuje počitadlo semaforu a pokud je nová hodnota menší nebo rovna nule, pak existuje nějaký spící proces, čekající na prostředek. V nejlepším případě bude nová hodnota semaforu rovna jedné a žádná další činnost není zapotřebí. V opačném případě vlastník inkrementuje počitadlo `waking` a probudí proces čekající ve frontě semaforu. Když se proces probudí, zjistí, že hodnota počitadla `waking` je jedna a ví, že může vstoupit do kritické sekce. Dekrementuje hodnotu `waking`, vrátí ji tedy zpět na nulu, a pokračuje. Veškerý přístup k položce `waking` semaforu je chráněn zámkem v položce `lock` semaforu.

Odkazy na zdrojové texty jádra

1 – Viz include/linux/-interrupt.h
2 – Viz do_bottom_half() in kernel/-softirq.c
3 – Viz include/-linux/tqueue.h
4 – Viz include/-linux/timer.h
5 – Viz timer_bh() in kernel /sched.c
6 – Viz run_old_timers)) in kernel/sched.c
7 – Viz run_timer_list() in kernel/sched.c
8 – Viz include/-linux/wait.h
9 – Viz include/-asm/semaphore.h

Kapitola

12

Moduly

V této kapitole je popsáno, jak může jádro Linuxu dynamicky nahrávat různé funkce, například souborové systémy, pouze v případě, že je potřebuje.

Jádro Linuxu je monolitické, jedná se tedy o jediný velký program, v němž mají všechny komponenty přístup ke všem interním datovým strukturám a rutinám. Alternativním řešením by bylo použití mikrojádra, kdy by byly jednotlivé funkční celky jádra rozděleny do samostatných jednotek, které by spolu mohly komunikovat pouze přísně omezenými mechanismy. Takto by ale bylo přidávání nových komponent do jádra v době jeho konfigurace časově velmi náročné. Řekněme, že chcete použít ovladač SCSI pro řadič SCSI NCR 810 a nemáte jej vestavěn přímo v jádře. Museli byste nakonfigurovat a sestavit nové jádro s tímto ovladačem. Linux ale umožňuje dynamicky nahrávat a rušit komponenty operačního systému podle potřeby. Moduly Linuxu představují části kódu, které je možno dynamicky přilinkovat do jádra v kterémkoliv okamžiku po zavedení systému. Je možno je z jádra zrušit a odstranit ve chvíli, kdy je není déle potřeba. Většinou se jako moduly vytvářejí ovladače zařízení, ovladače pseudozařízení (například ovladače sítí) a souborové systémy.

Moduly jádra můžete nahrávat a rušit buď explicitně pomocí příkazů `insmod` a `rmmod`, nebo si jejich nahrání či odstranění může vynutit jádro prostřednictvím démona jádra (`kerneld`).*

Dynamické nahrávání kódu podle potřeby je velmi lákavé, protože umožňuje udržovat minimální velikost jádra se zachováním vysoké flexibility. V jádře mého Linuxu používám moduly poměrně často, a tak je veliké pouze 406 KB. Občas používám souborový systém `VFAT`, takže mám jádro nakonfigurováno tak, aby automaticky nahrálo modul souborového systému `VFAT` v okamžiku, kdy tento systém připojím. Když oblast `VFS` odpojím, systém

* Poznámka korektora: V verzích jádra 2.1.x je démon `kerneld` nahrazen vláknem `kmod`.

zjistí, že modul souborového systém VFAT není déle potřebný a odstraní jej. Moduly mohou být velmi užitečné také při testování nových částí kódu jádra, aniž by bylo nutné jádro znovu sestavit a znovu zavést po každé změně. Nic ale není zadarmo, takže moduly jádra s sebou přinášejí lehké snížení výkonu a zvýšení paměťové náročnosti. Nahratelný modul musí obsahovat určitý speciální kód a tento kód spolu se speciálními datovými strukturami jádra má za následek zvýšení paměťové náročnosti. Kromě toho používají moduly nepřímé přístupy, takže použití prostředků jádra z modulů je méně efektivní.

Jakmile dojde k nahrání modulu, stává se částí jádra stejně, jako veškerý ostatní kód jádra. Má stejná práva a zodpovědnost jako ostatní části jádra. Jinak řečeno, modul jádra může zhroutit celé jádro stejně, jako kterákoliv jiná část kódu jádra nebo ovladače zařízení.

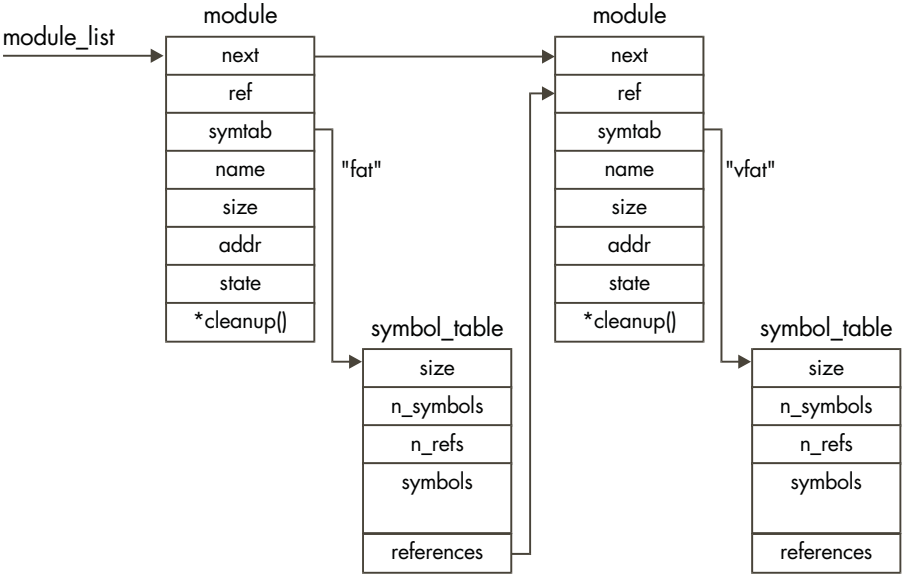
Aby mohl modul používat potřebné prostředky jádra, musí je být schopen najít. Řekněme, že modul potřebuje volat rutinu kmalloc(), alokační rutinu paměti jádra. V době svého sestavení modul neví, kde v paměti je rutina kmalloc() umístěna, takže po nahrání modulu musí jádro upravit všechny odkazy na rutinu kmalloc() v modulu, tak aby modul mohl pracovat. Jádro si udržuje seznam všech prostředků jádra v tabulce symbolů jádra, takže je schopno řešit odkazy na tyto prostředky ze všech nahraných modulů. Linux povoluje vrstvení modulů, tedy situaci, kdy jeden modul používá prostředky druhého. Například souborový systém VFAT potřebuje služby modulu souborového systému FAT, protože systém VFAT je víceméně pouze rozšířením služeb systému FAT. Když modul požaduje služby nebo prostředky jiného modulu, je to podobné situaci, kdy modul požaduje prostředky a služby samotného jádra. Rozdíl je pouze v tom, že v této situaci je požadovaná služba přítomna v jiném, dříve nahraném modulu. Vždy při nahrání modulu modifikuje jádro tabulku symbolů jádra a přidává do ní všechny prostředky nebo symboly exportované nově nahraným modulem. Znamená to tedy, že jakmile se nahraje další modul, má přístup ke službám všech dříve nahraných modulů.

Když je vznesen požadavek na odstranění modulu, musí jádro vědět, že modul je nepoužívaný a musí být schopno modulu sdělit, že bude odstraněn. Díky tomu bude modul před svým zrušením schopen uvolnit všechny jím alokované systémové prostředky, například paměť jádra nebo přerušení. Po zrušení modulu odstraní jádro z tabulky symbolů jádra všechny symboly exportované právě zrušeným modulem.

Kromě toho, že jakýkoliv špatně napsaný modul může zhroutit operační systém, hrozí zde i další nebezpečí. Co se stane, pokud nahrajete modul určený pro starší nebo novější verzi jádra, než kterou právě používáte? Může to způsobit problémy například pokud modul bude volat nějakou rutinu jádra a předá jí špatné parametry. Jádro se proti tomu dokáže bránit pomocí kontroly verze při nahrávání modulu.

12.1 Nahrání modulu

Existují dvě možnosti, jak může dojít k nahrání modulu jádra. První způsob používá příkaz `insmod`, který způsobí zavedení modulu do jádra. Druhý, chytřejší způsob, spočívá v nahrávání modulů podle potřeby, této metodě se říká vynucené nahrávání.



Obrázek 12.1
Seznam modulů jádra

Když jádro zjistí, že potřebuje nahrát nějaký modul, například pokud uživatel připojí souborový systém, který v jádře není, požádá jádro démona jádra (`kerneld`) o nahrání příslušného modulu. 1

Démon jádra je normální uživatelský proces ovšem s privilegii superuživatele. Po svém spuštění (obvykle v době zavádění jádra) otevře kanál IPC mezi sebou a jádrem. Tento kanál jádro používá k zasílání zpráv démonu `kerneld`, jimiž požaduje provedení různých operací.

Hlavním úkolem démona `kerneld` je nahrávání a rušení modulů jádra, dokáže však i jiné věci, například v případě potřeby otevřít linku PPP přes sériový kabel a posléze ji zase zavřít. Démon `kerneld` tyto úkoly neprovádí přímo, namísto toho volá příslušné příkazy, například `insmod`. Démon `kerneld` je čistě zástupce jádra, který jeho jménem provádí různé úkony. 2

Utilita `insmod` musí nejprve nalézt modul, který se má nahrát. Vynuceně nahrávané moduly jsou obvykle uloženy v adresáři `/lib/modules/verze jádra`. Moduly jádra jsou linkované objektové soubory stejně jako ostatní programy v systému s tou výjimkou, že jsou

relokovatelné. Znamená to, že obraz není slinkován tak, aby musel být nahrán na určitou adresu. Mohou být uloženy v objektovém formátu `a.out` nebo `elf`. Pomocí privilegovaných systémových volání zjišťuje utilita `insmod` jádrem exportované symboly.

3 Tyto symboly jsou organizovány v párech jména symbolu a jeho hodnoty, například adresa. Tabulka symbolů exportovaných jádrem je udržována v první datové struktuře `module` se-
4 znamu modulů, kterou udržuje jádro a ukazuje na ni ukazatel `module_list`.

Do této tabulky, která se vytváří při překladu a linkování jádra, jsou zařazovány pouze vybrané symboly jádra, jádro neexportuje modulům všechny své symboly. Příkladem může být symbol „`request_irq`“, což je rutina jádra, kterou modul musí volat, pokud chce převzít obsluhu nějakého přerušení. V mém konkrétním jádře má tento symbol hodnotu *0x0010cd30*. Jádrem exportované symboly a jejich moduly můžete zjistit výpisem souboru `/proc/ksyms` nebo příkazem `ksyms`. Utilita `ksyms` může zobrazit buď všechny jádrem exportované symboly, nebo pouze symboly exportované nahranými moduly.

Utilita `insmod` nahraje modul do své virtuální paměti a upraví všechny odkazy na rutiny a prostředky jádra pomocí tabulky symbolů, exportovaných jádrem. Tato úprava se provádí přímo přepisem obrazu modulu v paměti, utilita `insmod` fyzicky zapisuje adresy jednotlivých symbolů na příslušná místa v kódu modulu.

Když `insmod` upraví odkazy v modulu, požádá jádro o přidělení dostatečného místa pro uložení nového modulu, opět pomocí privilegovaného systémového volání. Jádro provede alokaci nové datové struktury `module` a dostatečné paměti pro uložení modulu. Strukturu `module` umís-
5 tí na konec seznamu nahraných modulů jádra. Nový modul je označen jako `UNINITIALIZED`.

Na obrázku 12.1 vidíme seznam modulů jádra poté, co byly do jádra nahrány dva moduly, `FAT` a `VFAT`. Na obrázku není znázorněn první modul v seznamu, což je pseudomodul používaný pouze k uložení tabulky symbolů exportovaných jádrem. Pomocí příkazu `lsmod` můžete vypsat seznam všech modulů nahraných v jádře a jejich vzájemných závislostí. Příkaz `lsmod` pouze přeformátuje výpis souboru `/proc/modules`, který se sestavuje ze seznamu struktur `module` v jádře. Paměť alokovaná jádrem se namapuje do adresového prostoru procesu `insmod`, takže do ní proces může přistupovat. Utilita `insmod` zkopíruje modul do alokovaného prostoru a provede jeho relokaci tak, aby běžel od té adresy, kterou mu jádro přidělilo. Tento postup je nutný, protože modul nemůže počítat s tím, že se bude dvakrát nahrávat na stejnou adresu, nebo že jej nahrají na stejnou adresu dva různé systémy. Relokace opět spočívá ve fyzické úpravě adres a odkazů v modulu.

6 Nový modul také exportuje své symboly, takže `insmod` musí sestavit tabulku těchto symbolů. Každý modul jádra musí obsahovat svůj inicializační a ukončovací kód, jejichž adresy se neexportují, `insmod` je ale musí znát a sdělit jádru. Pokud šlo všechno dobře, může nyní `insmod` provést inicializaci modulu, která se provádí privilegovaným systémovým voláním, jímž se jádru předávají adresy inicializační a ukončovací rutiny modulu.

Po přidání nového modulu do jádra je nutné aktualizovat tabulku symbolů jádra a upravit moduly, které nový modul používají. Moduly, na nichž jiné moduly závisejí, musejí na konci své tabulky symbolů udržovat seznam odkazů, na nějž se ukazuje ze struktury `module`. Na obrázku 12.1 je vidět, že modul souborového systému `vfat` závisí na modulu souborového systému `fat`. Modul `fat` tedy obsahuje odkaz na modul `vfat`, tento odkaz byl přidán, když se nahrál modul `vfat`. Jádro volá inicializační rutinu modulu a pokud rutina proběhne úspěšně, je modul považován za nahraný. Adresa ukončovací rutiny modulu se umístí do datové struktury `module` modulu a tato rutina bude volána, až jádro bude modul odstraňovat. Nakonec se stav modulu změní na `RUNNING`.

12.2 Rušení modulu

Moduly je možno odstranit pomocí příkazu *rmmod*, vynuceně nahrané moduly ale démon `kerneld` odstraňuje automaticky, jakmile nejsou zapotřebí. Vždy, když vyprší časovač démona `kerneld`, provede démon systémové volání, jímž se požaduje odstranění všech nepoužívaných vynuceně nahraných modulů ze systému. Hodnota časovače se nastavuje při spuštění démona `kerneld`, v mém systému se kontrola provádí každých 180 sekund. Pokud například připojíme souborový systém `iso9660` a tento systém máme implementován jako modul, pak po odpojení mechaniky CD-ROM dojde v krátké době k odstranění modulu `iso9600` z jádra.

Modul není možno odstranit, dokud na něm závisejí nějaké komponenty jádra. Nemůžete například odstranit modul `vfat`, pokud máte připojen jeden nebo více souborových systémů `vfat`. Když se podíváte na výpis příkazu `lsmod`, uvidíte, že každý modul má své počitadlo:

```
Module:    #pages:   Used by:
msdos            5                 1
vfat             4                 1 (autoclean)
fat              6   [vfat msdos]  2 (autoclean)
```

Počitadlo představuje počet entit jádra, které na modulu závisejí. V předchozím příkladu oba moduly `vfat` a `msdos` závisejí na modulu `fat`, který má tedy počitadlo rovno dvěma. Moduly `vfat` a `msdos` jsou každý používány jednou, a to připojeným souborovým systémem. Pokud připojím další souborový systém `vfat`, počitadlo modulu `vfat` bude 2. Počitadlo modulu je uloženo v prvním dvouslově jeho obrazu.

Toto pole je lehce přetíženo, protože kromě počitadla obsahuje také příznaky `AUTOCLEAN` a `VISITED`. Oba tyto příznaky se používají u vynuceně nahraných modulů. Vynuceně nahrané moduly jsou označeny příznakem `AUTOCLEAN`, aby systém věděl, že mají být automaticky zrušeny. Příznak `VISITED` označuje moduly, které jsou používány jednou nebo

více systémovými komponentami, nastavuje se vždy, když nějaká komponenta modul používá. Vždy když systém požaduje po démonu `kerneld` odstranění nepotřebných vynuceně nahraných modulů, prohlédnou se všechny moduly v systému a hledají se kandidáti na zrušení. Prohlížejí se pouze moduly označené jako `AUTOCLEAN` ve stavu `RUNNING`. Pokud modul nemá nastaven příznak `VISITED`, odstraní se. Je-li příznak nastaven, vynuluje se a pokračuje se dalším modulem v systému.

7 Pokud se modul bude rušit, volá se jeho ukončovací rutina, která zajistí uvolnění prostředků jádra, alokovaných modulem.

Datová struktura `module` daného modulu se označí jako `DELETED` a vyřadí se ze seznamu modulů jádra. Všem modulům, na nichž rušený modul závisel, se upraví odkazy tak, aby věděl, že modul už je nepotřebuje. Nakonec se uvolní paměť jádra, přidělená modulu.

Odkazy na zdrojové texty jádra

1 – `kerneld is in the modules package along with insmod, Ismod and rmmod.`
2 – `Viz includ/-linux/kerneld.h`
3 – `Viz sys_get_kernel_syms() in kernel/module.c`
4 – `Viz include/linux/module.h`
5 – `Viz sys_create_module() in kernel/module.c.`
6 – `Viz sys_init_module() in kernel/module.c.`
7 – `Viz sys_delete_module() in kernel/module.c`

Kapitola

13

Zdrojový kód Linuxu

V této kapitole vysvětlujeme, kde ve zdrojovém kódu Linuxu byste měli hledat různé funkce jádra.

Ke čtení této knihy nepotřebujete znát programovací jazyk C, nepotřebujete ani zdrojový kód Linuxu, abyste pochopili, jak jádro funguje. Studium kódu jádra je pouze cvičení, díky němuž získáte hlubší pochopení chování operačního systému Linux. Tato kapitola představuje přehled zdrojového kódu jádra, jak je uspořádán a kde byste měli začít hledat určitou funkci.

13.1 Jak získat zdrojový kód jádra

Všechny hlavní distribuce Linuxu (Debian, Slackware, Red Hat a další) zdrojový kód přímo obsahují. Obvykle se operační systém nainstalovaný na vašem počítači vytváří přímo z těchto zdrojových kódů. Ze své podstaty ovšem zdrojové kódy nebývají úplně aktuální, a proto možná budete chtít získat nejnovější zdrojové kódy na některé z adres, uvedených v příloze C. Zdrojové kódy jsou uloženy na adrese `ftp://ftp.cs.helsinki.fi`* a ostatní zdroje jsou pouze zrcadla tohoto serveru. Helsinky jsou tedy vždy nejaktuálnější, ovšem servery jako MIT a Sunsite nejsou nikdy příliš pozadu.

Pokud nemáte přístup k webu, existuje řada prodejců, kteří na CD-ROM nabízejí archívy hlavních světových serverů za velmi přijatelné ceny. Někteří dokonce nabízejí předplatné na čtvrtletní nebo dokonce měsíční aktualizace.

---

* Poznámka korektora: V současnosti je primárním zdrojem linuxových zdrojových textů server *ftp://ftp.kernel.org*. V České republice je zrcadlo např. na *ftp://ftp.fi.muni.cz/pub/linux/kernel.*

Zdrojový kód jádra Linuxu používá velmi jednoduchý systém číslování. Každé sudé číslo jádra (například `2.0.30`) je stabilní, oficiálně distribuované jádro, každé liché číslo (například `2.1.42`) je vývojová verze jádra. Tato kniha vychází ze stabilní verze `2.0.30`. Vývojové verze jádra obsahují ty nejnovější funkce a podporují ta nejnovější zařízení. I když mohou být nestabilní, což vám nemusí vyhovovat, je velmi důležité, aby uživatelé Linuxu zkoušeli nejnovější verze. Díky tomu dochází k jejich širokému testování. Pamatujte si ale, že je velmi rozumné vždy provést zálohu stávajícího systému předtím, než začnete zkoušet nejnovější vývojovou verzi.

Změny zdrojových kódů jádra se distrubuují jako záplaty (patche). Utilita patch provádí úpravy ve zdrojových souborech. Pokud například používáte zdrojové soubory 2.0.29 a chcete přejít na verzi 2.0.30, měli byste si pořídit patch 2.0.30 a na stávající zdrojový kód použít tuto utilitu:

```
$ cd /usr/src/linux
$ patch -p1 < patch-2.0.30
```

Tím se ušetří kopírování celých zdrojových souborů, například přes pomalá modemová spojení. Dobrým zdrojem patchů jádra (oficiálních i neoficiálních) je webová adresa `http://www.linuxhq.com`.

13.2 Členění zdrojového kódu

Na nejvyšší úrovni zdrojového stromu `/usr/src/linux` uvidíte řadu adresářů:

arch Podadresář `arch` obsahuje veškerý na architektuře závislý kód jádra. Obsahuje další podadresáře, jeden pro každou podporovanou architekturu, například `i386` a `alpha`.

include Podadresář `include` obsahuje většinu hlavičkových souborů potřebných pro sestavení jádra. I on obsahuje další podadresáře členěné podle podporovaných architektur. Podadresář `include/asm` je odkaz na skutečný adresář potřebný pro danou architekturu, například `include/asm/i386`. Architekturu změníte tak, že upravíte soubor Makefile jádra a znovu spustíte konfigurační program jádra.

init Tento adresář obsahuje inicializační kód jádra a je to dobré místo, odkud začít studovat, jak jádro funguje.

mm Tento adresář obsahuje veškerý kód správy paměti. Kód správy paměti závislý na architektuře je uložen v adresáři `arch/*/mm`, např. `arch/i386/mm/fault.c`.

drivers Adresář obsahuje všechny ovladače zařízení. Je dále členěn podle tříd ovladačů zařízení, například `block`.

ipc Adresář obsahuje kód pro meziprocesovou komunikaci.

fs Kód souborového systému. Je dále členěn na podadresáře pro jednotlivé podporované souborové systémy, například `vfat` a `ext2`.

kernel Hlavní kód jádra. Na architektuře závislé části jsou opět uloženy v `arch/*/kernel`.

net Síťový kód.

lib Tento adresář obsahuje kód knihoven jádra. Na architektuře závislé části kódu jsou uloženy v `arch/*/lib`.

scripts Tento adresář obsahuje skripty (například skripty `awk` a `tk`), používané při konfiguraci jádra.

13.3 Kde začít hledat

Představa hledání něčeho v tak velkém a složitém programu jako je jádro Linuxu může zastrašit. Působí jako obrovské klubko provázku bez konce a začátku. Prohlížení jedné části jádra často vede k hledání v dalších souvisejících částech a zanedlouho zapomenete, co jste vlastně hledali. V následujících částech jsou uvedena doporučení, kde ve struktuře zdrojových souborů co hledat.

13.3.1 Spouštění a inicializace systému

Na systémech s procesorem Intel se jádro spouští, když program `loadlin.exe` nebo *LILO* nahraje jádro do paměti a předá mu řízení. Tuto část najdete v `arch/i386/kernel/head.S`. Soubor `head.S` provede nějaké systémově specifické nastavení a poté skáče do rutiny `main()` v `init/main.c`.

13.3.2 Správa paměti

Kód je převážně soustředěn v `mm`, na architektuře závislé části najdete v `arch/*/mm`. Kód obsluhy výpadku stránky je v `mm/memory.c`, paměťové mapování a vyrovnávací paměť stránek je v `mm/filemap.c`. Vyrovnávací paměť bufferů je implementována v `mm/buffer.c` a odkládací vyrovnávací paměť v `mm/swap_state.c` a `mm/swapfile.c`.

13.3.3 Jádro

Většina důležitého obecného kódu jádra je v `kernel` se systémově závislými částmi v `arch/*/kernel`. Plánovač najdete v `kernel/sched.c`, kód rutiny fork v `kernel/fork.c`. Bottom half obsluha je implementována v `include/linux/interrupt.h`. Datová struktura `task_struct` se nachází v `include/linux/sched.h`.

13.3.4 PCI

Pseudoovladač zařízení PCI je v `drivers/pci/pci.c`, celosystémově platné definice v `include/linux/pci.h`. Každá architektura má nějaký specifický kód PCI BIOS, pro Alphu je v `arch/alpha/kernel/bios32.c`.

13.3.5 Meziprocesová komunikace

Vše je v `ipc`. Všechny IPC objekty Systemu V používají datovou strukturu `ipc_perm`, která je v `include/linux/ipc.h`. Zprávy Systemu V jsou implementovány v `ipc/msg.c`, semafory v `ipc/sem.c`. Roury jsou implementovány v `ipc/pipe.c`.

13.3.6 Obsluha přerušení

Obsluha přerušení je z největší části závislá na procesoru a architektuře systému. Obslužný kód přerušení pro Intel je v `arch/i386/kernel/irq.c`, definice v `include/asm/i386/irq.h`.

13.3.7 Ovladače zařízení

Největší část zdrojového kódu Linuxu tvoří jeho ovladače zařízení. Všechny ovladače zařízení jsou v adresáři `drivers`, který se ale dále dělí podle typu zařízení:

/block Ovladače blokových zařízení jako je třeba IDE (v `ide.c`). Pokud vás zajímá, jak se provádí obecná inicializace všech zařízení, která mohou obsahovat souborový systém, pak se podívejte na rutinu `device_setup()` v `drivers/block/genhd.c`. Provádí inicializaci nejen pevných disků, ale i například sítě, kterou potřebujete při připojení souborových systémů `nfs`. Bloková zařízení zahrnují jak zařízení IDE, tak zařízení SCSI.

/char Znaková zařízení jako `tty`, sériové porty a myš.

/cdrom Veškerý kód pro podporu CD-ROM. Zde najdete ovladače pro jednotlivá zařízení CD-ROM (například Soundblaster CDROM). Ovladač IDE CD je v `ide-cd.c` v adresáři `drivers/block`, ovladač SCSI CD je v `scsi.c` v adresáři `drivers/scsi`.

/pci Zdrojové kódy ovladače pseudozařízení PCI. Zde se dá zjistit, jak se subsystém PCI mapuje a inicializuje. PCI fixup kód pro architekturu Alpha AXP najdete v `/arch/alpha/kernel/bios32.c`.

/scsi Zde naleznete kód SCSI a ovladače všech zařízení SCSI, podporovaných Linuxem.

/net Zde najdete ovladače síťových zařízení, například ovladač ethernetové karty DECChip 21040 PCI je v `tulip.c`.

/sound Zde jsou ovladače zvukových karet.

13.3.8 Souborové systémy

Zdrojové kódy souborového systému `ext2` jsou v `fs/ext2` s definicemi datových struktur v `include/linux/ext2_fs.h`, `ext2_fs_i.h` a `ext2_fs_sb.h`. Datové struktury virtuálního souborového systému jsou popsány v `include/linux/fs.h`, kód je v `fs/*`. Vyrovnávací paměť bufferů je implementována ve `fs/buffer.c` včetně démona `update`.

13.3.9 Sítě

Kód podpory sítí je uložen v adresáři `net`, většina hlavičkových souborů v `include/net`. Kód soketů BSD je v `net/socket.c`, soketový kód pro sokety IPV4 je v `net/ipv4/af_inet.c`. Obecný kód pro podporu protokolů (včetně rutin pro obsluhu bufferů `sk_buff`) je v `net/core`, kód TCP/IP v `net/ipv4`. Ovladače síťových zařízení jsou v `drivers/net`.

13.3.10 Moduly

Kód pro podporu modulů je zčásti v jádře a zčásti v adresáři `modules`. Modulová část kódu jádra je v `kernel/modules.c`, datové struktury a démon `kerneld` v `include/linux/module.h` a `include/linux/kerneld.h`. Struktura objektového souboru `ELF` je popsána v `include/linux/elf.h`.

A

Datové struktury Linuxu

V této příloze je uveden seznam hlavních datových struktur používaných Linuxem, které jsme v této knize popisovali. Pro potřeby tisku byly jejich definice lehce upraveny.

block_dev_struct

Datová struktura `block_dev_struct` slouží k registraci blokových zařízení pro potřeby vyrovnávací paměti bufferů. Tyto struktury jsou uloženy ve vektoru `blk_dev`. 1

```
struct blk_dev_struct {
  void (*request_fn)(void);
  struct request * current_request;
  struct request   plug;
  struct tq_struct plug_tq;
};
```

buffer_head

Datová struktura `buffer_head` obsahuje informace o blokových bufferech ve vyrovnávací paměti bufferů. 2

```
/* stavové bity */
#define BH_Uptodate   0  /* 1 pokud buffer obsahuje platná data     */
#define BH_Dirty      1  /* 1 pokud je buffer modifikován           */
#define BH_Lock       2  /* 1 pokud je buffer zamčen                */
#define BH_Req        3  /* 0 je-li buffer zneplatněn               */
#define BH_Touched    4  /* 1 pracovalo-li se s bufferem (stárnutí) */
```

```
#define BH_Has_aged   5  /* 1 pokud buffer zestárnul                */
#define BH_Protected  6  /* 1 je-li buffer chráněn                  */
#define BH_FreeOnIO   7  /* 1 je-li nutné po IO zrušit buffer_head  */

struct buffer_head {
  /* První řádek cache: */
  unsigned long     b_blocknr;    /* číslo bloku                    */
  kdev_t            b_dev;        /* zařízení (B_FREE = volný)      */
  kdev_t            b_rdev;       /* skutečné zařízení              */
  unsigned long     b_rsector;    /* skutečné umístění na disku     */
  struct buffer_head *b_next;     /* seznam hashovací fronty        */
  struct buffer_head *b_this_page;      /* cyklický seznam bufferů
                                           v jedné stránce         */

  /* Druhý řádek cache: */
  unsigned long     b_state;      /* stav bufferu (viz výše)        */
  struct buffer_head  *b_next_free;
  unsigned int      b_count;      /* uživatelé tohoto bloku         */
  unsigned long     b_size;       /* velikost bloku                 */

  /* následují pro výkon nekritická data */
  char              *b_data;      /* ukazatel na datový blok        */
  unsigned int      b_list;       /* seznam, v němž je tento buffer*/
  unsigned long     b_flushtime;  /* čas, kdy by se měl buffer zapsat */
  unsigned long     b_lru_time;   /* čas posledního použití bufferu */
  struct wait_queue  *b_wait;
  struct buffer_head *b_prev;     /* obousměrný hashovací seznam    */
  struct buffer_head *b_prev_free; /* obousměrný seznam bufferů     */
  struct buffer_head *b_reqnext; /* fronta požadavků                */
};
```

device

Každé síťové zařízení v systému je reprezentováno datovou strukturou `device`.

```
struct device
{

 /*
 * Toto je první "viditelná" položka této struktury (tedy část, kterou
 * uživatel vidí v souboru "Space.c"). .c" file). Je to jméno rozhraní.
 */
 char                    *name;

  /* V/V specifické údaje */
  unsigned long          rmem_end;     /* "recv" konec sd. pam.   */
  unsigned long          rmem_start;   /* "recv" zač. sd. pam.    */
  unsigned long          mem_end;      /* konec sdíl. paměti      */
  unsigned long          mem_start;    /* zač. sdílené paměti     */
  unsigned long          base_addr;    /* V/V adresa zařízení     */
  unsigned char          irq;          /* číslo přerušení         */

  /* Nízkoúrovňové stavové příznaky */
  volatile unsigned char  start,       /* začátek operace         */
                         interrupt;    /* došlo přerušení         */
  unsigned long          tbusy;        /* vysílač obsazen         */
  struct device          *next;

  /* Inicializační funkce zařízení. Volá se jen jednou.           */
  int                      (*init)(struct device *dev);

  /* Následující položky potřebují některá zařízení, nejsou ale
     obvykle uváděny ve Space.c. */
  unsigned char          if_port;      /* volitelné AUI,TP        */
  unsigned char          dma;          /* DMA kanál               */

  struct enet_statistics* (*get_stats)(struct device *dev);
```

```
/*
 * Konec "viditelné části struktury. Všechny následující položky
 * jsou interní systémové položky a mohou se libovolně měnit
 */

/* Pro potřeby budoucího síťového kódu                          */
unsigned long           trans_start; /* čas (jiffies) posledního
                                        vysílání                */
unsigned long           last_rx;     /* čas posl. příjmu        */
unsigned short          flags;       /* příznaky rozhr. (BSD)   */
unsigned short          family;      /* ID adresové rodiny      */
unsigned short          metric;      /* směrovací metrika       */
unsigned short          mtu;         /* MTU                     */
unsigned short          type;        /* hardwarový typ          */
unsigned short          hard_header_len; /* délka HW hlavičky   */
void                    *priv;       /* privátní data           */

/* Adresové informace */
unsigned char           broadcast[MAX_ADDR_LEN];
unsigned char           pad;
unsigned char           dev_addr[MAX_ADDR_LEN];
unsigned char           addr_len;    /* délka hw adresy         */
unsigned long           pa_addr;     /* protokolová adresa      */
unsigned long           pa_brdaddr;  /* protokolová vysílací a.*/
unsigned long           pa_dstaddr   /* další protokolová adr. */
unsigned long           pa_mask;     /* síťová maska            */
unsigned short          pa_alen;     /* délka protok. adresy    */

struct dev_mc_list      *mc_list;    /* hromadné adresy         */
int                     mc_count;    /* kolik hrom. adres       */

struct ip_mc_list       *ip_mc_list; /* hrom. filtrační řetězec*/
__u32                   tx_queue_len;/* max rámců ve frontě     */

/* Kvůli vyvažování zátěže mezi páry zařízení */
unsigned long           pkt_queue;   /* paketů ve frontě        */
struct device           *slave;      /* druhé zařízení          */
struct net_alias_info   *alias_info; /* hlavní alias zařízení   */
struct net_alias        *my_alias;   /* další aliasy            */
```

```
  /* Ukazatel na buffery rozhraní */
  struct sk_buff_head     buffs[DEV_NUMBUFFS];

  /* Ukazatele na služební rutiny rozhraní */
  int                     (*open)(struct device *dev);
  int                     (*stop)(struct device *dev);
  int                     (*hard_start_xmit) (struct sk_buff *skb,
                            struct device *dev);
  int                     (*hard_header)      (struct sk_buff *skb,
                                               struct device *dev,
                                               unsigned short type,
                                               void *daddr,
                                               void *saddr,
                                               unsigned len);
  int                     (*rebuild_header)(  void *eth,
                                               struct device *dev,
                                               unsigned long raddr,
                                               struct sk_buff *skb);
  void                    (*set_multicast_list)(struct device *dev);
  int                     (*set_mac_address)  (struct device *dev,
                                               void *addr);
  int                     (*do_ioctl)(struct device *dev,
                                               struct ifreq *ifr,
                                               int cmd);
  int                     (*set_config)(struct device *dev,
                                               struct ifmap *map);
  void                    (*header_cache_bind)(struct hh_cache **hhp,
                                               struct device *dev,
                                               unsigned short htype,
                                               __u32 daddr);
  void                    (*header_cache_update)(struct hh_cache *hh,
                                               struct device *dev,
                                              unsigned char *  haddr);
  int                     (*change_mtu)(struct device *dev,
                                               int new_mtu);
  struct iw_statistics*   (*get_wireless_stats)(struct device *dev);
};
```

device_struct

Datová struktura `device_struct` slouží k registraci znakových a blokových zařízení (obsahuje jejich jméno a množinu souborových operací, které zařízení podporuje). Každý platný prvek vektorů `chrdevs` a `blkdevs` reprezentuje jedno znakové, respektive blokové zařízení.

4

```
struct device_struct {
     const char * name;
     struct file_operations * fops;
};
```

file

5 Datovou strukturou `file` je reprezentován každý otevřený soubor, soket a podobně.

```
struct file {
  mode_t f_mode;
  loff_t f_pos;
  unsigned short f_flags;
  unsigned short f_count;
  unsigned long f_reada, f_ramax, f_raend, f_ralen, f_rawin;
  struct file *f_next, *f_prev;
  int f_owner;          /* pid nebo -pgrp, kam se má poslat SIGIO
*/
  struct inode * f_inode;
  struct file_operations * f_op;
  unsigned long f_version;
  void *private_data;  /* nutné pro ovladače tty a některé další */
};
```

6 files_struct

Datová struktura `files_struct` popisuje soubory, které má proces otevřen.

```
struct files_struct {
  int count;
  fd_set close_on_exec;
  fd_set open_fds;
  struct file * fd[NR_OPEN];
};
```

fs_struct

```
struct fs_struct {
  int count;
  unsigned short umask;
  struct inode * root, * pwd;
};
```

gendisk

Datová struktra `gendisk` obsahuje informace o pevném disku. Používá se v době inicializace, kdy se naleznou pevné disky a pak se na nich hledají oblasti.

```
struct hd_struct {
    long start_sect;
    long nr_sects;
};

struct gendisk {
    int major;                  /* hlavní číslo ovladače */
    const char *major_name;     /* jméno hlavního ovladače */
    int minor_shift;            /* kolikrát se posouvá vedlejší
                                   číslo než vznikne skutečné
                                   vedlejší číslo */
    int max_p;                  /* maximum oblastí na zařízení */
    int max_nr;                 /* maximální počet zařízení */

    void (*init)(struct gendisk *);
                                /* inicializace volaná než začneme
                                  dělat naše věci */
    struct hd_struct *part;     /* tabulka oblastí */
    int *sizes;                 /* velikost zařízení v blocích,
                                   kopíruje se do blk_size[] */
    int nr_real;                /* počet reálných zařízení */

    void *real_devices;         /* pro interní použití */
    struct gendisk *next;
};
```

inode

9 Datová struktura VFS `inode` obsahuje informace o souboru nebo adresáři na disku.

```
struct inode {
    kdev_t                      i_dev;
    unsigned long               i_ino;
    umode_t                     i_mode;
    nlink_t                     i_nlink;
    uid_t                       i_uid;
    gid_t                       i_gid;
    kdev_t                      i_rdev;
    off_t                       i_size;
    time_t                      i_atime;
    time_t                      i_mtime;
    time_t                      i_ctime;
    unsigned long               i_blksize;
    unsigned long               i_blocks;
    unsigned long               i_version;
    unsigned long               i_nrpages;
    struct semaphore            i_sem;
    struct inode_operations     *i_op;
    struct super_block          *i_sb;
    struct wait_queue           *i_wait;
    struct file_lock            *i_flock;
    struct vm_area_struct       *i_mmap;
    struct page                 *i_pages;
    struct dquot                *i_dquot[MAXQUOTAS];
    struct inode                *i_next, *i_prev;
    struct inode                *i_hash_next, *i_hash_prev;
    struct inode                *i_bound_to, *i_bound_by;
    struct inode                *i_mount;
    unsigned short              i_count;
    unsigned short              i_flags;
    unsigned char               i_lock;
    unsigned char               i_dirt;
    unsigned char               i_pipe;
    unsigned char               i_sock;
    unsigned char               i_seek;
```

```
    unsigned char                       i_update;
    unsigned short                      i_writecount;
    union {
        struct pipe_inode_info          pipe_i;
        struct minix_inode_info         minix_i;
        struct ext_inode_info           ext_i;
        struct ext2_inode_info          ext2_i;
        struct hpfs_inode_info          hpfs_i;
        struct msdos_inode_info         msdos_i;
        struct umsdos_inode_info        umsdos_i;
        struct iso_inode_info           isofs_i;
        struct nfs_inode_info           nfs_i;
        struct xiafs_inode_info         xiafs_i;
        struct sysv_inode_info          sysv_i;
        struct affs_inode_info          affs_i;
        struct ufs_inode_info           ufs_i;
        struct socket                   socket_i;
        void                            *generic_ip;
    } u;
};
```

ipc_perm

10 Datová struktura `ipc_perm` popisuje přístupová práva k meziprocesovým komunikačním objektům Systemu V.

```
struct ipc_perm
{
  key_t  key;
  ushort uid;       /* euid a egid vlastníka */
  ushort gid;
  ushort cuid;      /* euid a egid autora */
  ushort cgid;
  ushort mode;      /* přístupové režimy viz příznaky režimů níže */
  ushort seq;       /* sekvenční číslo */
};
```

irqaction

11 Datová struktura `irqaction` slouží k popisu handlerů přerušení v systému.

```
struct irqaction {
  void (*handler)(int, void *, struct pt_regs *);
  unsigned long flags;
  unsigned long mask;
  const char *name;
  void *dev_id;
  struct irqaction *next;
};
```

linux_binfmt

12 Každý binární souborový formát, jemuž Linux rozumí, je reprezentován datovou strukturou `linux_binfmt`.

```
struct linux_binfmt {
  struct linux_binfmt * next;
  long *use_count;
  int (*load_binary)(struct linux_binprm *, struct  pt_regs *
regs);
  int (*load_shlib)(int fd);
  int (*core_dump)(long signr, struct pt_regs * regs);
};
```

mem_map_t

13 Datová struktura `mem_map_t` (označovaná také jako stránka) obsahuje informace o každé stránce fyzické paměti.

```
typedef struct page {
  /* tyto údaje musejí být první (kvůli manipulaci s volnou pamětí) */
  struct page             *next;
  struct page             *prev;
  struct inode            *inode;
  unsigned long           offset;
  struct page             *next_hash;
  atomic_t                count;
```

```
  unsigned                flags;       /* atomické příznaky, mohou se
                                          měnit asynchronně */
  unsigned                dirty:16,
                          age:8;
  struct wait_queue       *wait;
  struct page             *prev_hash;
  struct buffer_head      *buffers;
  unsigned long           swap_unlock_entry;
  unsigned long           map_nr; /* page->map_nr == page - mem_map */
} mem_map_t;
```

mm_struct

14 Datová struktura `mm_struct` slouží k popisu virtuální paměti procesů.

```
struct mm_struct {
  int count;
  pgd_t * pgd;
  unsigned long context;
  unsigned long start_code, end_code, start_data, end_data;
  unsigned long start_brk, brk, start_stack, start_mmap;
  unsigned long arg_start, arg_end, env_start, env_end;
  unsigned long rss, total_vm, locked_vm;
  unsigned long def_flags;
  struct vm_area_struct * mmap;
  struct vm_area_struct * mmap_avl;
  struct semaphore mmap_sem;
};
```

pci_bus

15 Každá sběrnice PCI v systému je reprezentována strukturou `pci_bus`.

```
struct pci_bus {
  struct pci_bus  *parent;       /* rodičovská sběrnice můstku */
  struct pci_bus  *children;     /* seznam můstků na této sběrnici */
  struct pci_bus  *next;         /* seznam všech sběrnic */

  struct pci_dev  *self;         /* můstek z pohledu rodiče */
```

```
  struct pci_dev  *devices;      /* zařízení pod můstkem */

  void    *sysdata;              /* odkaz na systémově závislá data */

  unsigned char  number;         /* číslo sběrnice */
  unsigned char  primary;        /* číslo primárního můstku */
  unsigned char  secondary;      /* číslo sekundárního můstku */
  unsigned char  subordinate;    /* číslo podřízené sběrnice */
};
```

pci_dev

16 Každé zařízení PCI v systému včetně můstků PCI-PCI a PCI-ISA je reprezentováno datovou strukturou `pci_dev`.

```
/*
 * Existuje jedna struktura pci_dev pro každou dvojici číslo slotu/
 * číslo funkce:
 */
struct pci_dev {
  struct pci_bus  *bus;        /* číslo sběrnice, na níž je toto zař.
*/
  struct pci_dev  *sibling;    /* další zařízení na této sběrnici */
  struct pci_dev  *next;       /* seznam všech zařízení */

  void    *sysdata;            /* odkaz na systémově specifická data
*/

  unsigned int  devfn;         /* kódovaný index zařízení a funkce */
  unsigned short  vendor;
  unsigned short  device;
  unsigned int  class;         /* 3 bajty: (base, sub, prog-if) */
  unsigned int  master : 1;    /* nastaveno, jde-li o zař. master */
  /*
   * Teoreticky je možno úroveň přerušení přečíst v konfiguraci
   * a všechno by fungovalo. Bohužel starší čipy PCI nepodporují
   * tyto registry a namísto toho vracejí nulu. Například řadič
   * Vision864-P rev 0 umí používat INTA, vrací ale 0 v registrech
   * přerušovací linky a pinu. Funkce pci_init() inicializuje
```

```
    * tuto položku hodnotou v PCI_INTERRUPT_LINE a pokud to bude
    * nutné, mění ji funkce pcibios_fixup(). Údaj nesmí být nulový,
    * ledaže zařízení vůbec negeneruje přerušení.
*/
  unsigned char  irq;      /* přerušení generované zařízením */
};
```

request

17 Datová struktura request se používá k vytváření požadavků na bloková zařízení v systému. Požadavky vždy znamenají zápis nebo čtení bloku dat do nebo z vyrovnávací paměti bufferů.

```
struct request {
    volatile int rq_status;
#define RQ_INACTIVE              (-1)
#define RQ_ACTIVE                1
#define RQ_SCSI_BUSY             0xffff
#define RQ_SCSI_DONE             0xfffe
#define RQ_SCSI_DISCONNECTING    0xffe0

    kdev_t rq_dev;
    int cmd;         /* READ nebo WRITE */
    int errors;
    unsigned long sector;
    unsigned long nr_sectors;
    unsigned long current_nr_sectors;
    char * buffer;
    struct semaphore * sem;
    struct buffer_head * bh;
    struct buffer_head * bhtail;
    struct request * next;
};
```

rtable

18 Každá datová struktura `rtable` obsahuje informace o trase používané k odesílání IP paketů. Tato struktura se používá ve vyrovnávací paměti tras.

```
struct rtable
{
    struct rtable      *rt_next;
    __u32               rt_dst;
    __u32               rt_src;
    __u32               rt_gateway;
    atomic_t            rt_refcnt;
    atomic_t            rt_use;
    unsigned long       rt_window;
    atomic_t            rt_lastuse;
    struct hh_cache    *rt_hh;
    struct device      *rt_dev;
    unsigned short      rt_flags;
    unsigned short      rt_mtu;
    unsigned short      rt_irtt;
    unsigned char       rt_tos;
};
```

semaphore

19 Semafory slouží k ochraně kritických datových struktur a oblastí kódu.

```
struct semaphore {
    int count;
    int waking;
    int lock ;                  /* aby bylo testování atomické */
    struct wait_queue *wait;
};
```

sk_buff

20 Datová struktura `sk_buff` slouží k popisu síťových dat při jejich předávání mezi protokolovými vrstvami.

```
struct sk_buff
{
  struct sk_buff    *next;        /* další buffer v seznamu        */
  struct sk_buff    *prev;        /* předchozí buffer v seznamu    */
  struct sk_buff_head *list;      /* seznam, v němž jsme           */
  int               magic_debug_cookie;
  struct sk_buff    *link3;       /* odkaz na buffery IP protokolu */
  struct sock       *sk;          /* soket, který nás vlastní      */
  unsigned long     when;         /* používáno k výpočtu rtt       */
  struct timeval    stamp         /* čas, kdy jsme dorazili        */
  struct device     *dev;         /* zařízení, na němž jsme        */
                                  dorazili, z nějž budeme odesláni */
  union
  {
      struct tcphdr       *th;
      struct ethhdr       *eth;
      struct iphdr        *iph;
      struct udphdr       *uh;
      unsigned char       *raw;
      /* pro předávání handlů souborů v soketech domény Unix */
      void                *filp;
  } h;

  union
  {
      /* Část informací fyzické vrstvy */
      unsigned char       *raw;
      struct ethhdr       *ethernet;
  } mac;

  struct iphdr            *ip_hdr;      /* pro IPPROTO_RAW         */
  unsigned long           len;          /* skutečná délka dat      */
  unsigned long        csum;            /* kontrolní součet        */
  __u32                saddr;           /* zdrojová IP adresa      */
```

```
  __u32             daddr;       /* cílová IP adresa              */
  __u32             raddr;       /* IP adresa dalšího skoku       */
  __u32             seq;         /* sekvenční číslo (TCP)         */
  __u32             end_seq;     /* seq [+ fin] [+ syn] + datalen */
  __u32             ack_seq;     /* potvrzovací sekvenční číslo TCP)
*/
  unsigned char     proto_priv[16];
  volatile char      acked,      /* jsme potvrzeni?               */
                    used,        /* jsme používáni?               */
                    free,        /* jak tento buffer uvolnit      */
                    arp;         /* dokončen překlad IP/ARP       */
  unsigned char     tries,       /* kolikrát zkoušeno             */
                    lock,        /* jsme zamčení?                 */
                    localroute,  /* lokální směrování tohoto rámce */
                    pkt_type,    /* třída paketu                  */
                    pkt_bridged, /* trasa pro můstky              */
                    ip_summed;   /* kontrolní součet IP           */
#define PACKET_HOST        0     /* nám                           */
#define PACKET_BROADCAST   1     /* všem                          */
#define PACKET_MULTICAST   2     /* skupině                       */
#define PACKET_OTHERHOST   3     /* někomu jinému                 */
  unsigned short    users;       /* počitadlo uživatelů
                                    viz datagram.c,tcp.c          */
  unsigned short    protocol;    /* protokol                      */
  unsigned int      truesize;    /* velikost bufferu              */
  atomic_t          count;       /* referenční počitadlo          */
  struct sk_buff    *data_skb;   /* odkaz na datový buffer        */
  unsigned char     *head;       /* začátek bufferu               */
  unsigned char     *data;       /* začátek dat                   */
  unsigned char     *tail;       /* ukazatel na patičku           */
  unsigned char     *end;        /* ukazatel na konec             */
  void              (*destructor)(struct sk_buff *);
                                 /* funkce rušení                 */
  __u16             redirport;   /* port pro přesměrování         */
};
```

sock

Každá datová struktura `sock` obsahuje protokolově specifické informace o BSD soketu. Například pro soket domény INET bude tato struktura obsahovat všechny informace specifické pro protokoly TCP/IP a UDP/IP.

```
struct sock
{
    /* Toto musí být první */
    struct sock           *sklist_next;
    struct sock           *sklist_prev;

    struct options        *opt;
    atomic_t              wmem_alloc;
    atomic_t              rmem_alloc;
    unsigned long         allocation;   /* režim alokace */
    __u32                 write_seq;
    __u32                 sent_seq;
    __u32                 acked_seq;
    __u32                 copied_seq;
    __u32                 rcv_ack_seq;
    unsigned short        rcv_ack_cnt;  /* počitadlo potvrzení */
    __u32                 window_seq;
    __u32                 fin_seq;
    __u32                 urg_seq;
    __u32                 urg_data;
    __u32                 syn_seq;
    int                   users;        /* počitadlo uživatelů */
  /*
   *    Ne všechny údaje jsou sice typu "volatile",
   *    některé ale jsou, takže stejně dobře mohou být všechny.
   */
    volatile char         dead,
                          urginline,
                          intr,
                          blog,
                          done,
                          reuse,
                          keepopen,
```

```
                        linger,
                        delay_acks,
                        destroy,
                        ack_timed,
                        no_check,
                        zapped,
                        broadcast,
                        nonagle,
                        bsdism;
    unsigned long       lingertime;
    int                 proc;

    struct sock         *next;
    struct sock         **pprev;
    struct sock         *bind_next;
    struct sock         **bind_pprev;
    struct sock         *pair;
    int                 hashent;
    struct sock         *prev;
    struct sk_buff      *volatile send_head;
    struct sk_buff      *volatile send_next;
    struct sk_buff      *volatile send_tail;
    struct sk_buff_head back_log;
    struct sk_buff      *partial;
    struct timer_list   partial_timer;
    long                retransmits;
    struct sk_buff_head write_queue,
                        receive_queue;
    struct proto        *prot;
    struct wait_queue   **sleep;
    __u32               daddr;
    __u32               saddr;       /* Odesílatel */
    __u32               rcv_saddr;
    unsigned short      max_unacked;
    unsigned short      window;
    __u32               lastwin_seq; /* sekv. číslo, kdy jsme
                                        naposledy aktualizovali
                                        nabízené okno */
```

```
    __u32                high_seq;     /* sekv. číslo, kdy jsme
                                          naposledy opakovali */
    volatile unsigned long  ato;       /* timeout potvrzení */
    volatile unsigned long lrcvtime;   /* jiffies při posledním
                                          příjmu dat */
    volatile unsigned long  idletime;  /* jiffies při posl.příjmu */
    unsigned int          bytes_rcv;
/*
 *    mss je min(mtu, max_window)
 */
    unsigned short        mtu;         /* mss dohodnutné při sync */
    volatile unsigned short mss;       /* efektivní mss, mění se  */
    volatile unsigned short user_mss;  /* mss požad. uživ. v ioctl*/
    volatile unsigned short max_window;
    unsigned long         window_clamp;
    unsigned int          ssthresh;
    unsigned short        num;
    volatile unsigned short cong_window;
    volatile unsigned short cong_count;
    volatile unsigned short packets_out;
    volatile unsigned short shutdown;
    volatile unsigned long  rtt;
    volatile unsigned long  mdev;
    volatile unsigned long  rto;

    volatile unsigned short backoff;
    int             err, err_soft;     /* pravidelně se objevující
                                          chyby, ne jen timeout */
    unsigned char           protocol;
    volatile unsigned char  state;
    unsigned char           ack_backlog;
    unsigned char           max_ack_backlog;
    unsigned char           priority;
    unsigned char           debug;
    int                     rcvbuf;
    int                     sndbuf;
    unsigned short          type;
    unsigned char   localroute;        /* směrovat jen lokálně */
```

```
/*
 *     Tady budou uloženy všechny specifické volitelné údaje.
 */
    union
    {
          struct unix_opt       af_unix;
#if defined(CONFIG_ATALK) || defined(CONFIG_ATALK_MODULE)
        struct atalk_sock       af_at;
#endif
#if defined(CONFIG_IPX) || defined(CONFIG_IPX_MODULE)
        struct ipx_opt          af_ipx;
#endif
#ifdef CONFIG_INET
        struct inet_packet_opt  af_packet;
#ifdef CONFIG_NUTCP
        struct tcp_opt          af_tcp;
#endif
#endif
    } protinfo;
/*
 *     'Privátní oblast' IP
 */
    int                 ip_ttl;             /* nastavení TTL */
    int                 ip_tos;             /* TOS */
    struct tcphdr       dummy_th;
    struct timer_list   keepalive_timer;    /* časovač
                                               TCP keepalive   */
    struct timer_list   retransmit_timer;   /* časovač
                                               TCP retransmit  */
    struct timer_list   delack_timer;       /* časovač potvrzovací
                                               TCP             */
    int                 ip_xmit_timeout;    /* proč běží timeout */
    struct rtable       *ip_route_cache;    /* trasa v cache */
    unsigned char       ip_hdrincl;         /* včetně hlaviček? */
#ifdef CONFIG_IP_MULTICAST
    int                 ip_mc_ttl;          /* multicasting TTL */
    int                 ip_mc_loop;         /* loopback */
```

```
    char            ip_mc_name[MAX_ADDR_LEN];  /* jméno hrom.zař. */
    struct ip_mc_socklist   *ip_mc_list;       /* pole skupiny */
#endif

/*
 *     Tato část slouží funkcím timeoutu (timer.c).
 */
    int                     timeout;      /* Na co čekáme? */
    struct timer_list       timer; /* TIME_WAIT/receive časovač */
    struct timeval          stamp;
 /*
  *     Identd
  */
    struct socket           *socket;
  /*
   *     Callbacks
   */
    void                    (*state_change)(struct sock *sk);
    void                    (*data_ready)(struct sock *sk,int bytes);
    void                    (*write_space)(struct sock *sk);
    void                    (*error_report)(struct sock *sk);

};
```

socket

Každá datová struktura `socket` obsahuje informace o BSD soketu. Neexistuje nezávisle, je vždy součástí VFS inodu.


```
struct socket {
  short             type;         /* SOCK_STREAM, ...              */
  socket_state      state;
  long              flags;
  struct proto_ops    *ops;       /* protokoly dělají téměř vše    */
  void                *data;      /* protokolová data              */
  struct socket       *conn;      /* server, k němuž jsme připojeni */
  struct socket       *iconn;     /* nedokončené klientské spojení */
  struct socket       *next;
  struct wait_queue   **wait;     /* ukazatel na místo, kde čekáme */
```

```
  struct inode        *inode;
  struct fasync_struct *fasync_list;    /* asynchronní seznam
                                           čekatelů  */
  struct file         *file;            /* souborový ukazatel      */
};
```

task_struct

23 Každá datová struktura `task_struct` popisuje jeden proces v systému.

```
struct task_struct {
/* následující údaje neměnit */
  volatile long     state;       /* -1 unrunnable, 0 runnable,
                                    >0 pozastavený */
  long              counter;
  long              priority;
  unsigned          long signal;
  unsigned          long blocked;/* bitmapa maskovaných signálů */
  unsigned          long flags;  /* procesové příznaky, viz dále */
  int errno;
  long              debugreg[8]; /* hardwarové ladicí registry */
  struct exec_domain      *exec_domain;
/* různé údaje */
  struct linux_binfmt     *binfmt;
  struct task_struct      *next_task, *prev_task;
  struct task_struct      *next_run,  *prev_run;
  unsigned long           saved_kernel_stack;
  unsigned long           kernel_stack_page;
  int                     exit_code, exit_signal;
  /* ??? */
  unsigned long           personality;
  int                     dumpable:1;
  int                     did_exec:1;
  int                     pid;
  int                     pgrp;
  int                     tty_old_pgrp;
  int                     session;
  int                     leader;
```

```
  int                     groups[NGROUPS];
  /*
   * Ukazatele na (původní) rodičovský proces, nejmladšího syna,
   * nejmladšího a nejstaršího sourozence. (p->father je možno
   * nahradit p->p_pptr->pid)
   */
  struct task_struct      *p_opptr, *p_pptr, *p_cptr,
                            *p_ysptr, *p_osptr;
  struct wait_queue       *wait_chldexit;
  unsigned short          uid,euid,suid,fsuid;
  unsigned short          gid,egid,sgid,fsgid;
  unsigned long           timeout, policy, rt_priority;
  unsigned long           it_real_value, it_prof_value,
                             it_virt_value;
  unsigned long           it_real_incr, it_prof_incr, it_virt_incr;
  struct timer_list       real_timer;
  long                    utime, stime, cutime, cstime, start_time;
/* údaje pro obsluhu výpadku a odkládání */
  unsigned long           min_flt, maj_flt, nswap, cmin_flt,
                             cmaj_flt, cnswap;
  int swappable:1;
  unsigned long      swap_address;
  unsigned long      old_maj_flt;/* stará hodnota maj_flt */
  unsigned long      dec_flt;    /* počitadlo výpadků     */
  unsigned long      swap_cnt;   /* počet příště odkládaných str */
/* limity */
  struct rlimit      rlim[RLIM_NLIMITS];
  unsigned short     used_math;
  char               comm[16];
/* informace souborového systému */
  int                link_count;
  struct tty_struct  *tty;            /* NULL, není-li tty */
/* údaje meziprocesové komunikace */
  struct sem_undo    *semundo;
  struct sem_queue   *semsleeping;
/* ldt této úlohy, používá Wine. Pokud je NULL, použije se de-
fault_ldt. */
  struct desc_struct *ldt;
```

```
/* tss této úlohy */
  struct thread_struct tss;
/* informace souborového systému */
  struct fs_struct     *fs;
/* informace o otevřených souborech */
  struct files_struct  *files;
/* informace správy paměti */
  struct mm_struct     *mm;
/* obsluha signálů */
  struct signal_struct *sig;
#ifdef __SMP__
  int                   processor;
  int                   last_processor;
  int                   lock_depth;      /* hloubka zamčení */
#endif
};
```

timer_list

24 Datová struktura `timer_list` slouží k implementaci reálných časovačů procesů.

```
struct timer_list {
  struct timer_list *next;
  struct timer_list *prev;
  unsigned long expires;
  unsigned long data;
  void (*function)(unsigned long);
};
```

tq_struct

25 Každá fronta úloh `tq_struct` obsahuje informace o úkolech, které byly zařazeny do fronty. Jedná se obvykle o činnosti požadované ovladačem zařízení, které ale není nutné provést okamžitě.

```
struct tq_struct {
    struct tq_struct *next;    /* seznam prvků */
    int sync;                  /* inicializuje se na nulu */
```

```
    void (*routine)(void *);  /* funkce, která se má volat */
    void *data;               /* parametr funkce */
};
```

vm_area_struct

Každá datová struktura `vm_area_struct` popisuje oblast virtuální paměti procesu.

```
struct vm_area_struct {
  struct mm_struct * vm_mm;  /* parametry oblasti */
  unsigned long vm_start;
  unsigned long vm_end;
  pgprot_t vm_page_prot;
  unsigned short vm_flags;
/* AVL strom oblastí řazený podle adres */
  short vm_avl_height;
  struct vm_area_struct * vm_avl_left;
  struct vm_area_struct * vm_avl_right;
/* Seznam oblastí řazený podle adres */
  struct vm_area_struct * vm_next;
/* Pro oblasti s inody kruhový seznam inode->i_mmap */
/* Pro sdílené oblasti kruhový seznam uživatelů */
/* jinak nepoužito */
  struct vm_area_struct * vm_next_share;
  struct vm_area_struct * vm_prev_share;
/* další */
  struct vm_operations_struct * vm_ops;
  unsigned long vm_offset;
  struct inode * vm_inode;
  unsigned long vm_pte;       /* sdílená paměť */
};
```

Odkazy na zdrojové texty jádra

1 – Viz include/linux/
2 – Viz include/linux/fs.h
3 – Viz include/linux/netdevice.h

```
 4 – Viz fs/devices.c
 5 – Viz include/linux/fs.h
 6 – Viz include/linux/sched.h
 7 – Viz include/linux/sched.h
 8 – Viz include/linux/genhd.h
 9 – Viz include/linux/fs.h
10 – Viz include/linux/ipc.h
11 – Viz include/linux/interrupt.h
12 – Viz include/linux/binfmts.h
13 – Viz include/linux/mm.h
14 – Viz include/linux/sched.h
15 – Viz include/linux/pci.h
16 – Viz include/linux/pci.h
17 – Viz include/linux/blkdev.h
18 – Viz include/linux/route.h
19 – Viz include/linux/semaphore.h
20 – Viz include/linux/skbuff.h
21 – Viz include/linux/net.h
22 – Viz include/linux/net.h
23 – Viz include/linux/sched.h
24 – Viz include/linux/timer.h
25 – Viz include/linux/tqueue.h
26 – Viz include/linux/mm.h
```

B

Procesor Alpha AXP

Procesory Alpha AXP mají 64bitovou architekturu RISC load/store, navrženou s ohledem na rychlost. Všechny registry jsou dlouhé 64 bitů, procesor obsahuje 32 celočíselných registrů a 32 registrů v plovoucí řádové čárce. Celočíselný registr 31 a reálný registr 31 slouží pro nulové operace. Čtením z nich se získá nulová hodnota, zápis do nich nemá žádný efekt. Všechny operace jsou dlouhé 32 bitů a v paměti je možno provádět čtení nebo zápisy. Architektura umožňuje různé implementace do té doby, dokud implementace dodržují návrh architektury.

Neexistují instrukce pro přímou manipulaci s hodnotami v paměti, všechny datové manipulace se provádějí mezi registry. Pokud tedy budete chtít inkrementovat počitadlo v paměti, musíte je nejprve načíst do registru, modifikovat je a zapsat je zpět do paměti. Pro vzájemnou interakci slouží pouze jedna instrukce pro zápis do registru nebo paměti a jiná pro čtení registru nebo paměti. Zajímavou funkcí procesoru Alpha AXP je to, že některé instrukce mohou generovat příznaky, například při testování dvou registrů na shodu, ovšem výsledek se neuloží do stavového registru procesoru, ale do třetího registru. Na první pohled to může vypadat podivně, odstranění závislosti na stavových registrech ale znamená, že se usnadňuje konstrukce procesoru, který v jednom cyklu provádí více instrukcí. Instrukce s nezávislými registry nemusejí vzájemně čekat na své dokončení jak by tomu bylo, kdyby používaly společný stavový registr. Malý počet operací s pamětí a velký počet registrů rovněž usnadňuje paralelní zpracovávání instrukcí.

Architektura Alpha AXP používá skupinu subrutin, zvanou *privileged architecture library code* (PALcode). PALcode je specifický pro operační systém, implementaci architektury Alpha AXP v procesoru a hardware systému. Tyto subrutiny představují elementární bloky operačního systému pro přepínání kontextu, přerušení, výjimky a správu paměti. Jednotlivé subrutiny je možno volat hardwarově nebo instrukcemi CALL_PAL. PALcode je napsán ve standardním assembleru Alpha AXP s některými implementačně závislými rozšířeními tak, aby

umožňoval přímý přístup k nízkoúrovňovým hardwarovým funkcím, například interním registrům procesoru. PALcode se provádí v režimu PALmode, privilegovaném režimu, který zabrání vzniku některých systémových událostí a umožňuje PALcodu převzít řízení nad hardwarem fyzického systému.

C

Užitečné adresy WWW a FTP

Dále uvádíme seznam užitečných adres WWW a FTP:

http://www.azstarnet.com/~axplinux

Jedná se o stránky Davida Mosbergera-Tanga věnované Alpha AXP Linuxu, na tomto místě naleznete odkazy na všechny dokumenty HOWTO týkající se procesoru Alpha AXP. Obsahuje také řadu odkazů na další informace o Linuxu a procesorech Alpha AXP, například datové tabulky procesoru.

http://www.redhat.com/

Webové stránky Red Hat. Obsahují řadu užitečných odkazů.

ftp://sunsite.unc.edu

Významná síť se spoustou zdarma dostupných programů. Programy pro Linux naleznete v adresáři *pub/Linux*.

http://www.intel.com

Domovská stránka firmy Intel, dobré místo k nalezení informací o čipech Intel.

http://www.ssc.com/lj/index.html

Linux Journal je velmi dobrý časopis věnovaný Linuxu a díky řadě skvělých článků se předplatné rozhodně vyplatí.

http://www.blackdown.org/java-linux.html

Primární síť informací o Javě na Linuxu.

ftp://tsx-11.mit.edu/~ftp/pub/linux

FTP server Linuxu na MITu.

ftp://ftp.cs.helsinki.fi/pub/Software/Linux/Kernel

Zdrojové kódy jádra Linuxu.

http://www.linux.org.uk

Linux User Group ve Velké Británii.

http://sunsite.unc.edu/mdw/linux.html

Domovská stránka Linux Documentation Projectu.

http://www.digital.com

Domovská stránka firmy Digital Equipment Corporation.

http://altavista.digital.com

Vyhledávací stroj Altavista firmy Digital. Velmi dobré místo pro hledání informací na webu.

http://www.linuxhq.com

Stránky Linux HQ obsahují aktuální oficiální i neoficiální patche a dále rady a odkazy, které vám mohou pomoci získat ty nejvhodnější zdrojové kódy jádra pro váš systém.

http://www.amd.com

Domovské stránky firmy AMD.

http://www.cyrix.com

Domovské stránky firmy Cyrix.

D

Slovníček

ARP Address Resolution Protocol. Slouží k překladu IP adres na fyzické hardwarové adresy.

ASCII American Standard Code for Information Interchange. Každé písmeno abecedy je reprezentováno 8bitovou hodnotou. ASCII kód se často používá k ukládání psaných textů.

Bajt 8 bitů dat.

Bit Elementární jednotka dat, která je reprezentována jako 0 nebo jako 1 (zapnuto nebo vypnuto).

Bottom-half handler Obslužný systém pro práci s úlohami řazenými ve frontách.

C Programovací jazyk vysoké úrovně. Většina jádra Linuxu je napsána v jazyce C.

CPU Central Processing Unit. Hlavní zařízení počítače, viz též *procesor* a *mikroprocesor.*

Datová struktura Skupina dat v paměti uspořádaná do položek.

DMA Direct Memory Access. Přímý přístup do paměti.

EIDE Extended IDE.

ELF Executable and Linkable Format. Objektový formát souborů navržený v Unix System Laboratories je dnes uznáván jako nejčastěji používaný formát v Linuxu.

Fronta úloh Mechanismus pro odkládání úloh v Linuxu.

Funkce Kus programu zajišťující nějakou činnost, například vrácení většího ze dvou čísel.

IDE Integrated Disk Electronics.

IP Internet Protocol.

IPC Interprocess Communication, meziprocesová komunikace.

IRQ Interrupt Request Queue, fronta požadavků na přerušení.

ISA Industry Standard Architecture. Dnes poněkud zastaralý standard rozhraní datové sběrnice pro komponenty jako řadič disketových mechanik.

Kilobajt Tisíc bajtů dat (přesně 1 024), často se zapisuje jako KB.

Megabajt Milión bajtů dat (přesně 1 048 576 B nebo 1 024 KB), zapisuje se jako MB.

Mikroprocesor Integrovaný CPU. Většina moderních CPU jsou *mikroprocesory*.

Modul jádra Dynamicky nahrávaná funkce jádra jako například souborový systém nebo ovladač zařízení.

Modul Soubor obsahující instrukce CPU buď v podobě instrukcí assembleru, nebo instrukcí vysokoúrovňového jazyka jako je například C.

Objektový soubor Soubor obsahující strojový kód a data, který ještě nebyl slinkován s dalšími objektovými soubory a knihovnami do *spustitelného obrazu*.

Obraz Viz *Spustitelný obraz.*

Ovladač zařízení Program řídící určité zařízení, například ovladač zařízení NCR 810 slouží k řízení řadiče SCSI NCR 810.

Parametr Funkce a rutiny přebírají parametry, které zpracovávají.

PCI Peripheral Component Interconnect. Standard popisující jak se vzájemně propojují komponenty počítačového systému.

Periferie Inteligentní procesor pracující místo procesoru systému. Například čip řadiče IDE.

Proces Entita, která může provádět *program*. Proces si je možno představit jako *program* v akci.

Procesor Zkrácený název pro mikroprocesor, ekvivalentní termín pro *CPU*.

Program Koherentní množina instrukcí procesoru, která provádí nějakou úlohu, například vytiskne text „hello world“. Viz též *spustitelný obraz*.

Protokol Síťový *jazyk* používaný k přenosu aplikačních dat mezi dvěma spolupracujícími procesy nebo síťovými vrstvami.

Registr Místo na čipu, používané k uložení informací nebo instrukcí.

Rozhraní Standardní metoda volání rutin a předávání datových struktur. Například rozhraní mezi dvěma vrstvami kódu může být popsáno rutinami, které přebírají a vracejí určité datové struktury. Dobrým příkladem rozhraní je virtuální souborový systém Linuxu.

Rutina Podobá se funkci s tou výjimkou, že, přísně vzato, nevrací výslednou hodnotu.

SCSI Small Computer Systems Interface.

Shell Program působící jako rozhraní mezi operačním systémem a živým uživatelem. Označuje se také jako *příkazový interpret*, v Linuxu se nejčastěji používá shell *bash*.

SMP Symetrický multiprocessing. Systém, v němž je více než jeden procesor, a ty si mezi sebou spravedlivě rozdělují práci.

Software CPU instrukce (ať už v assembleru, nebo ve vyšším jazyce) a data. Prakticky ekvivalent pojmu *program*.

Soket Reprezentuje jeden konec síťového spojení. Linux podporuje soketové rozhraní BSD.

Spustitelný obraz Strukturovaný soubor obsahující strojové instrukce a data. Soubor je možno nahrát do virtuální paměti procesu a spustit. Viz též *program*.

Stránka Fyzická paměť se dělí na stejně velké stránky.

System V Klon Unixu uvolněný v roce 1983, který, mimo jiné, obsahoval mechanismy meziprocesové komunikace.

TCP Transmission Control Protocol.

UDP	User Datagram Protocol.
Ukazatel	Místo v paměti, které obsahuje adresu jiného místa v paměti.
Virtuální paměť	Hardwarové a softwarové mechanismy zajišťující, že fyzická paměť v systému se jeví větší, než jaká doopravdy je.

E

Seznam literatury

Richard L. Sites: *Alpha Architecture Reference Manual*, Digital Press

Matt Welsh, Lar Kaufman: *Running Linux*, O'Reilly&Associates

PCI Special Interest Group: *PCI Local Bus Specification*

PCI Special Interest Group: *PCI BIOS ROM Specification*

PCI Special Interest Group: *PCI to PCI Bridge Architecture Specification*

Intel: *Peripheral Components*, Intel 296467

Brian W. Kernihan, Dennis M. Ritchie: *The C Programming Language*, Prentice Hall

Steven Levy: *Hackers*, Penguin Books

Intel: *Intel486 Processor Family: Programmer's Reference Manual*, Intel

Comer D. E.: *Internetworking with TCP/IP, Volume 1 - Principles, Protocols and Architecture*, Prentice Hall

David Jagger: *ARM Architectural Reference Manual*, Prentice Hall

LINUX

Praktické návody

Kapitola

1

Linux NET-3, použití Linuxu na sítích

Terry Dawson, VK2KTJ, **terry@perf.no.itg.telstra.com.au** v1.2, 20. srpna 1997

Operační systém Linux má podporu jádra pro sítě napsánu téměř od základu nově. Výkon implementace TCP/IP dělá z posledních verzí jader plnohodnotnou konkurenci i pro nejlepší srovnatelné systémy. Tento dokument si klade za cíl popsat instalaci a konfiguraci síťového softwaru a příslušných nástrojů v Linuxu.

1 Změny oproti předchozí verzi

Navíc:

- Odkaz na dokument PLIP – díky Claes IP NAT
- Úpravy/aktualizace
- Mnoho oprav od Alessandra Rubiniho – díky!
- Aktualizovaná e-mailová adresa Larryho Stefaniho – díky Larry
- Opravené umístění síťových nástrojů *ftp.linux.uk.org* – díky Rone
- Opravený nesprávný příkaz pro směrování – díky Johne
- Více nefunkčních směrovacích příkazů! – díky Jean-Pierre
- Adresy IPv6 jsou 16bitové, ne 32bitové, aha – díky Erezi

Ještě schází:

- Přidat omezovač provozu (shaper)
- Popsat nový směrovací algoritmus
- Přidat volby IPv6 pro kompilaci jádra

- Popsat údaje /proc/sys/net/*
- Zařízení WanRouter

2 Úvod

Původní NET-FAQ napsal Matt Welsh a já, abychom ještě před spuštěním projektu Linux Documentation Project odpověděli na často pokládané dotazy o sítích pod Linuxem. Pokrývali jsme zde rané vývojové verze síťového jádra Linuxu. Po NET-FAQ následovalo NET-2-HOWTO a jednalo se o jeden z prvních dokumentů celého projektu LDP. Nyní byla používána verze 2 a později i 3 síťového softwaru jádra Linuxu. Tento dokument bezprostředně navazuje a vztahuje se již jen k verzi 3.

Předchozí verze dokumentu byly poměrně velké, protože takový byl i rozsah tématu. Proto byly vyčleněny některé další dokumenty, které se zabývají specifickými síťovými otázkami. Tento dokument se na ně na příslušných místech odkazuje a pokrývá oblasti, které ostatní dokumenty ještě nepopsaly.

2.1 Ohlasy

Vždy vítám ohlasy a obzvláště cenné příspěvky. Vše mi prosím odesílejte na můj *e-mail* **<mailto:terry@perf.no.itg.telstra.com.au>**[*].

3 Jak používat tento dokument

Formát tohoto dokumentu se od starších verzí liší. Přeskupil jsem části tak, aby bylo možné přeskočit informativní části na začátku, pak následují obecné předpoklady, které je nutné zvládnout, a potom teprve přichází specializované části.

Čtěte obecné části

Tyto části se vztahují ke každému nebo téměř každému postupu popisovanému později, takže je pro vás velmi důležité, abyste je pochopili.

Berte v úvahu vaši síť

Měli byste vědět, jak je (nebo má být) navržena vaše síť a jaký typ hardwaru a technologií využívá.

[*] Poznámka korektora: Současným správcem NET-3 HOWTO je Alessandro Rubini.

Přečtěte si části, zaměřené na vámi požadované technologie

Jakmile víte, co chcete, můžete nalézt jakoukoliv komponentu. Tyto části obsahují pouze podrobnosti k určitým technologiím.

Proveďte konfigurační práci

Měli byste se pokusit nakonfigurovat vaši síť a pečlivě se zaměřit na jakékoliv problémy, které vyvstanou.

Podle potřeby vyhledejte další pomoc

Jestliže se objeví problémy, se kterými vám tento dokument nepomůže, přečtěte si v příslušné části, kde je možné nalézt pomoc nebo nahlásit chyby.

Bavte se!

Používání sítí je zábava, tak si to užijte.

4 Obecné informace o používání Linuxu na sítích

4.1 Stručná historie vývoje podpory sítí v jádře Linuxu

Vývoj jádra se zcela novou implementací protokolů TCP/IP, která měla být alespoň tak dobrá, jako již existující implementace, nebyl vůbec snadný. Rozhodnutí o nevyužití existující implementace bylo provedeno v čase, kdy vznikla určitá obava z přetížení existující implementace díky určitým omezením práv ze strany U.S.L. Tehdy také bylo dost elánu k takové práci, mělo se to provést jinak a snad i lépe než kdy jindy.

Prvním dobrovolníkem, který vedl vývoj síťového kódu jádra, byl Ross Biro **<biro@yggdrasil.com>**. Ross vytvořil jednoduchou a nekompletní, ale většinou využitelnou implementační sadu rutin, které byly doplněny ethernetovým ovladačem pro síťovou kartu WD-8003. To stačilo, aby se softwarem experimentovalo dostatečně mnoho lidí, a aby se s tím někteří lidé dokonce připojili k živému Internetu. Takže na vývoj podpory sítě vznikl v linuxové komunitě určitý tlak, ale tomuto tlaku (někdy nevybíravému) podlehl Ross a vzdal se pozice vedoucího vývoje. Jeho snaha však odstartovala práci dobrým směrem a naznačila, že vývoj je možný.

Orest Zborowski **<obz@Kodak.COM>** vytvořil pro jádro Linuxu původní BSD socketové programové rozhraní. To byl velký skok, umožňující mnoha existujícím síťovým aplikacím úpravu pro Linux bez vážnějších potíží.

Někdy v této době vyvinul Laurence Culhane **<loz@holmes.demon.co.uk>** první ovladače pro Linux, podporující protokol SLIP. Takto mohli s novým síťovým softwarem experimentovat i lidé, kteří neměli přístup k ethernetovým sítím. Opět to někteří zkoušeli i na Internetu. Tak mnozí lidé nabyli dojmu, že je možné Linux na sítích využívat a počet experimentátorů se síťovým softwarem výrazně stoupl.

Jedním z lidí, kteří na problému vytvoření síťové podpory pracovali, byl Fred van Kempen **<waltje@uwalt.nl.mugnet.org>**. V období určité nejistoty po Rossově odstoupení z pozice vedoucího vývoje Fred obětoval svůj čas a námahu a tuto roli přijal. Fred měl v určitých směrech s Linuxem jisté ambice a také zde podpořil další pokrok. Vytvořil série síťového kódu, nazývané kód „NET-2“ jádra (kód „NET“ byl Rossův), které byli mnozí lidé schopni velmi dobře využívat. Do vývojové agendy Fred formálně vložil mnoho inovací, jako je dynamické rozhraní zařízení, podpora protokolu Amateur Radio AX.25 a více modulárně navržená implementace síťové podpory. Fredův kód NET-2 využívalo velké množství nadšenců, jejichž počet se zvyšoval s tím, jak se šířilo poznání o dobré funkčnosti. Síťový software byl v této době tvořen mnoha záplatami (patch) standardního kódu jádra a nebyl začleněn v jeho normální šířené verzi. Aby to všechno dobře fungovalo, byly zde dokumenty NET-FAQ a následný NET-2-HOWTO. Fred se soustředil na vývoj vylepšení standardních síťových implementací a to bralo hodně času. Komunita uživatelů začala být netrpělivá, i když zde byla spolehlivost, která uspokojila 80 % uživatelů. Stejně jako u Rosse začal stoupat tlak na Freda jako hlavního vývojáře.

Alan Cox **<iialan@www.uk.linux.org>** navrhl řešení situace. Předpokládal, že by si vzal Fredův kód NET-2 a odladil by jej, aby byl spolehlivý a stabilní a uspokojil nedočkavou uživatelskou základnu. Sejmul by tak z Freda tlak a Fred by mohl nadále pokračovat ve své práci. Alan nezůstal jen u návrhů a jeho první verze linuxového síťového kódu se nazývala „Net-2D“ (debugged - odladěný). Kód fungoval v mnoha běžných konfiguracích spolehlivě a uživatelská základna byla spokojená. Alan měl vhodné nápady a um, takže nepopiratelně přispěl do projektu i do řešení problému, kterým směrem se má ubírat vývoj kódu NET-2. Byly zde dva proudy, jeden zastával názor, že „nejdříve to musí fungovat a pak to musí fungovat lépe“, zatímco druhý chtěl, „aby to hned fungovalo lépe“. Linus nakonec nabídl podporu Alanovým vývojovým snahám a začlenil jeho kód do standardní distribuce kódu jádra. Tak se dostal do nevýhodné pozice Fred. Jakékoliv pokračování vývoje by postrádalo velkou uživatelskou základnu, která by kód aktivně využívala a testovala. Fred sice ještě v práci chvíli pokračoval, ale pak odpadl a na jeho pozici vedoucího vývoje linuxového síťového jádra se dostal Alan.

Donald Becker **<becker@cesdis.gsfc.nasa.gov>** brzy rozvinul svůj talent pro aspekty nižší úrovně použití sítí a vytvořil velký balík ethernetových ovladačů. Téměř všechny, které se nyní v jádrech používají, pochází od Donalda. Významně zde přispěli i mnozí jiní lidé, ale Donaldova práce je plodná, a proto si zaslouží speciální zmínku.

Alan určitou dobu vybrušoval kód NET-2-Debugged, přičemž pracoval na některých záležitostech, které zůstaly nedořešeny v seznamu „TODO“ (zbývá dokončit). V době, kdy byl zdrojový text jádra Linuxu 1.3.* teprve v plenkách, se síťový kód jádra přesunul do verze NET-3, na které jsou založeny i současné verze. Alan pracoval na mnoha různých aspektech síťového kódu a přitom mu pomáhali mnozí další talentovaní lidé ze síťové části komunity uživatelů Linuxu, takže kód rostl ve všech směrech. Alan vytvořil dynamická síťová zařízení a první standardní implementace AX.25 a IPX. Dále pokračoval ve vypiplávání kódu, pomalu jej přeorganizoval a vylepšil do dnešní podoby.

Podpora PPP byla přidána Michaelem Callahanem **<callahan@maths.ox.ac.uk>** a Alem Longyearem **<longyear@netcom.com>**. Tohle byl také jeden z faktorů, který ovlivnil zvýšení počtu uživatelů, kteří Linux aktivně využívali pro sítě.

Jonathon Naylor **<jsn@cs.nott.ac.uk>** přispěl značným vylepšením Alanova kódu AX.25, kde přidal podporu protokolů NetRom a Rose. Podpora AX.25/NetRom/Rose je sama o sobě dost významná, protože kromě Linuxu nemá standardně tuto podporu žádný jiný operační systém.

K vývoji linuxového síťového softwaru značnou měrou samozřejmě přispěly i stovky dalších lidí. S některými se setkáte v částech, vztahujících se k technologiím, další lidé přispěli moduly, ovladači, opravami chyb, návrhy, hlášeními o testech a morální podporou. Všichni si mohou říci, že se zúčastnili a přispěli tím, čím mohli. Síťový kód linuxového jádra je skvělým příkladem výsledku linuxového stylu anarchistického vývoje, který přitom stále není ukončen.

4.2 Kde získat další informace o sítích pod Linuxem

Míst, ze kterých můžete získat dobré informace o použití sítí pod Linuxem, je spousta.

Alan Cox, současný správce síťového kódu linuxového jádra, má stránku WWW, která obsahuje výtah ze současného a nového vývoje síťové podpory Linuxu: `http://www.uk.linux.org/NetNews.html`.

Dalším místem je kniha od Olafa Kircha, nazvaná **Network Administrators Guide**. Jedná se o práci pod *Linux Documentation Project* `http://sunsite.unc.edu/LDP/` a můžete si ji interaktivně přečíst na adrese `http://sunsite.unc.edu/LDP/LDP/nag/nag.html`

nebo ji můžete přes ftp získat v různých formátech z archívu ftp LDP `ftp://sunsite.unc.edu/pub/Linux/docs/LDP/network-guide/`. Olafova kniha je srozumitelná a nabízí dobrý vyšší standard síťové konfigurace pod Linuxem.

V linuxové hierarchii news je skupina, věnovaná sítím a příbuzným záležitostem. Jedná se o *comp.os.linux.networking* **<news:comp.os.linux.networking>**.

Existuje poštovní konference, ve které se můžete ptát na záležitosti kolem sítí pod Linuxem. Přihlásíte se odesláním zprávy:

```
To: majordomo@vger.rutgers.edu
Subject: anything at all
Zpráva: subscribe linux-net
```

Na různých sítích IRC jsou často kanály **#linux**, na kterých jsou lidé schopni odpovídat na dotazy kolem sítí pod Linuxem.

Při popisu jakéhokoliv problému prosím nezapomeňte na veškeré nutné podrobnosti. Zejména se jedná o verzi vámi používaného softwar – hlavně jádra, nástrojů, jako jsou *pppd* nebo *dip* - a o přesnou povahu problému. To znamená přesné zaznamenání chybových hlášení a předtím použitých příkazů.

4.3 Kde získat některé informace o sítích, které nejsou specifické pro Linux

Jestliže vám jde o některé základní obecné informace k použití sítí TCP/IP, doporučuji vám tyto dokumenty:

TCP/IP introduction (úvod)

Tento dokument existuje v *textové verzi <ftp://athos.rutgers.edu/runet/tcp-ip-intro.doc>* a v *postscriptové verzi <ftp://athos.rutgers.edu/runet/tcp-ip-intro.ps>*.

Jestliže na dané téma potřebujete ještě podrobnější informace, potom vřele doporučuji:

```
"Internetworking with TCP/IP"
od Douglase E. Comera
ISBN 0-13-474321-0
Prentice Hall publications.
```

Jestliže se chcete naučit psát síťové aplikace pro prostředí, kompatibilní s Unixem, pak vám doporučuji:

```
"Unix Network Programming"
od W. Richarda Stevense
ISBN 0-13-949876-1
Prentice Hall publications.
```

Nahlédněte také do newsové skupiny *comp.protocols.tcp-ip* **<news:comp.protocols.tcp-ip>**.

Důležitým zdrojem konkrétních technických informací k Internetu a protokolům TCP/IP jsou RFC. RFC je zkratka pro „Request For Comment“ a jedná se o standardní typ dokumentace k protokolům Internetu. RFC je možné nalézt na mnoha místech. Buď se jedná o ftp-servery nebo o WWW, kdy je možné prohledávat databázi RFC na základě klíčových slov.

Jedním ze zdrojů RFC je: *Nexorská RFC-databáze*

<http://pubweb.nexor.co.uk/public/rfc/index/rfc.html>.

5 Informace k běžné konfiguraci sítě

Následující části budete potřebovat znát a pochopit ještě předtím, než se skutečně dostanete ke konfiguraci vaší sítě. Jedná se o základní principy, platné nezávisle na povaze použité sítě.

5.1 Co potřebuji na začátku?

Před vytvořením nebo konfigurací sítě budete potřebovat pár věcí. Nejdůležitější jsou:

5.1.1 Aktuální zdrojový text jádra

Protože jádro, které nyní používáte, nemusí mít podporu pro zamýšlené síťové typy nebo karty, budete pravděpodobně potřebovat zdrojový text jádra, aby pak bylo možné jádro překompilovat s patřičnými volbami.

Nejnovější zdrojový text jádra je vždy možné získat na: *ftp.kernel.org* nebo

<ftp://ftp.funet.fi/pub/Linux/PEOPLE/Linus/v2.0>.

Normálně bývá zdrojový text jádra rozbalen do adresáře `/usr/src/linux`. Informace k aplikaci úprav (patch) a sestavení jádra naleznete v dokumentu k jádru **<Kernel-HOWTO.html>**. Informace ke konfiguraci modulů jádra naleznete v dokumentu o modulech **<Module-HOWTO.html>**.

Pokud není řečeno něco jiného, doporučuji zůstat u standardních verzí jádra (mají druhou číslici ve verzi sudou). Vývojové verze jader (ty s druhou číslicí ve verzi lichou) mohou mít strukturální nebo jiné změny, které mohou při práci s jiným softwarem ve vašem systému způsobovat problémy. Jestliže si nejste jisti, že byste takovýto druh problémů zvládli (kromě dalších potenciálních softwarových chyb), tyto verze nepoužívejte.

5.1.2 Aktuální síťové nástroje

Síťové nástroje jsou programy, které používáte ke konfiguraci linuxových síťových zařízení. Tyto nástroje umožňují například přiřazení adres zařízením a konfigurování směrovacích cest.

Většina současných distribucí Linuxu je dodávána se síťovými nástroji, takže pokud instalujete z distribuce a ještě jste nenainstalovali síťové nástroje, učiňte tak.

Jestliže neinstalujete z distribuce, musíte nástroje získat a zkompilovat sami. To ale není obtížné.

Síťové nástroje jsou nyní spravovány Berndem Eckenfelsem a jsou k dispozici na adrese <*ftp://ftp.inka.de/pub/comp/Linux/networking/NetTools/*> a jejich mirror najdete na adrese <*ftp://ftp.uk.linux.org/pub/linux/Networking/base/*>.

Vyberte verzi, která se nejlépe hodí k vámi používanému jádru. Při instalaci se řiďte přiloženými instrukcemi.

Při instalaci a konfiguraci verze, aktuální v době psaní, postupujte následovně:

```
#
# cd /usr/src
# tar xvfz net-tools-1.33.tar.gz
# cd net-tools-1.33
# make config
# make
# make install
#
```

Navíc, pokud předpokládáte konfiguraci firewallu nebo využití funkce IP-masquerade, budete potřebovat příkaz `ipfwadm`. Jeho poslední verzi je možné získat na <*ftp://ftp.xos.nl/pub/linux/ipfwadm*>. Opět jsou zde k dispozici různé verze. Vyberte vždy tu verzi, která nejvíce vyhovuje vašemu jádru.

Při instalaci a konfiguraci verze, aktuální v době psaní, postupujte následovně:

```
#
# cd /usr/src
```

```
# tar xvfz ipfwadm-2.3.0.tar.gz
# cd ipfwadm-2.3.0
# make
# make install
#
```

5.1.3 Programy síťových aplikací

Programy síťových aplikací jsou programy, jako je `telnet` a `ftp` a jejich příslušné programy pro server. Distribuci těch nejběžnějších nyní spravuje David Holland **<dholland@hcs.harvard.edu>**. Získat je můžete na adrese *<ftp://ftp.uk.linux.org/pub/linux/Networking/base/>*.

Při instalaci a konfiguraci verze, aktuální v době psaní, postupujte následovně:

```
#
# cd /usr/src
# tar xvfz /pub/net/NetKit-B-0.08.tar.gz
# cd NetKit-B-0.08
# more README
# vi MCONFIG
# make
# make install
#
```

5.1.4 Adresy

IP-adresy se skládají ze čtyř bajtů. Je zvykem je zapisovat v desítkové podobě, oddělené tečkou. Každý bajt je převeden na desítkové číslo (0-255), nuly na začátku se nepíší a čísla se oddělují tečkou „.“. IP-adresu má každé rozhraní hostitele nebo routeru. Za určitých okolností je sice povoleno na jednom stroji u všech rozhraní použít stejnou IP-adresu, ale obvykle má každé rozhraní svoji vlastní.

IP-sítě jsou souvislé řady IP-adres. Všechny adresy v rámci sítě mají určitou část čísel adresy společnou. Tato společná část se nazývá „síťová část“ adresy. Zbývající čísla jsou nazývána „hostitelská část“ adresy. Počet bitů, sdílených všemi adresami sítě, se nazývá síťová maska a tato maska také určuje, které adresy do sítě patří a které ne. Vezměme si například:

Adresa hostitele	192.168.110.23
Síťová maska	255.255.255.0
Síťová část	192.168.110.
Hostitelská část	.23
Síťová adresa	192.168.110.0
Vysílací adresa	192.168.110.255

Každá adresa, která odpovídá síťové masce, ukazuje na síť, do které náleží. Síťová adresa má proto mezi adresami na síti nejnižší číslo a svoji hostitelskou část má nulovou.

Vysílací adresa je speciální adresa, na kterou slyší všichni hostitelé sítě (kromě své vlastní). Na tuto adresu se odesílají datagramy, které má obdržet každý hostitel sítě. Odesílají se sem určité typy dat, jako směrovací informace a různá upozornění. Existují dvě běžné vysílací adresy. Nejčastěji se používá nejvyšší možná adresa sítě. V předchozím příkladu by to byla 192.168.110.255. Někdy se jako vysílací používá síťová adresa. V praxi se nic nemění, ale všichni hostitelé sítě musí mít hlavně nakonfigurovánu stejnou vysílací adresu.

Z administrativních důvodů byly již v dobách vývoje protokolu IP vyčleněny určité skupiny adres do sítí, které byly seskupeny do tříd. Tyto třídy nabízí různý rozsah sítí, které je možné alokovat. Rozsahy jsou následující:

Třída sítě	Síťová maska	Síťové adresy
A	255.0.0.0	0.0.0.0 - 127.255.255.255
B	255.255.0.0	128.0.0.0 - 191.255.255.255
C	255.255.255.0	192.0.0.0 - 223.255.255.255
Multicast	240.0.0.0	224.0.0.0 - 239.255.255.255

Využívání adres závisí na tom, co děláte. Můžete využít i kombinace těchto tříd.

Instalace linuxového stroje na již existující IP-síti

Jestliže se chystáte linuxový stroj instalovat na již existující IP-síť, musíte kontaktovat osobu, která ji spravuje, a zeptat se na následující informace:

- IP-adresa hostitele
- Adresa IP-sítě
- Vysílací IP-adresa
- Síťová IP-maska

- Adresa routeru
- Adresa DNS

S takto získanými informacemi pak nakonfigurujete vaše linuxové síťové zařízení. Nemůžete si je vymyslet a pak očekávat, že by fungovaly.

Vytvoření zcela nové sítě, která se nebude připojovat k Internetu

Jestliže vytváříte soukromou síť a neočekáváte, že se kdy budete připojovat k Internetu, můžete si adresy vybrat jaké chcete. Z důvodů jednotnosti (a také bezpečnostních) jsou zde však pro tyto účely rezervovány určité adresy IP-sítě. Specifikace podle RFC1597 je následující:

REZERVOVANÉ ALOKACE SOUKROMÝCH SÍTÍ

Třída sítě	Síťová maska	Síťové adresy
A	255.0.0.0	10.0.0.0 – 10.255.255.255
B	255.255.0.0	172.16.0.0 – 172.31.255.255
C	255.255.255.0	192.168.0.0 – 192.168.255.255

Nejprve se rozhodnete, jak má být vaše síť velká. Potom vyberete tolik adres, kolik bude třeba.

5.2 Kam mám vložit konfigurační příkazy?

K procedurám zavádění systému Linuxu existují různé přístupy. Po zavedení jádra se vždy spouští program „`init`“. Tento program pak čte svůj konfigurační soubor `/etc/inittab` a začne proces zavádění. `Init` má pár odrůd a to jsou také největší rozdíly mezi jednotlivými distribucemi nebo stroji.

`/etc/inittab` obvykle obsahuje přibližně takovýto údaj:

```
si::sysinit:/etc/init.d/boot
```

Tento řádek určuje název skritpu příkazového interpretu, který spravuje zaváděcí sekvenci. Tento soubor je obdobou souboru `AUTOEXEC.BAT` v operačním systému MS-DOS.

Jsou zde obvykle i další skripty, volané zaváděcím skriptem, přičemž síť se často konfiguruje právě jedním z nich.

Jako průvodce vaším systémem může sloužit následující tabulka:

Distribuce	Konfigurace rozhraní a směrovacích cest	Inicializace
Debian	/etc/init.d/network	/etc/init.d/netbase
		/etc/init.d/netstd_init
		/etc/init.d/netstd_nfs
		/etc/init.d/netstd_misc
Slackware	/etc/rc.d/rc.inet1	/etc/rc.d/rc.inet2
RedHat	/etc/sysconfig/network-scripts/ifup-<ifname>	
		/etc/rc.d/init.d/network

Většina současných distribucí zahrnuje program, který umožňuje konfiguraci mnoha běžných síťových rozhraní. Jestliže jednu z nich máte, před pokusem o ruční konfiguraci se podívejte, jestli bude dělat to, co požadujete.

Distribuce	Konfigurační program sítě
RedHat	/sbin/netcfg
Slackware	/sbin/netconfig

5.3 Vytvoření vašich síťových rozhraní

V mnoha unixových operačních systémech se síťová zařízení objevují v adresáři /dev. V Linuxu tomu tak není. V Linuxu jsou síťová zařízení vytvářena dynamicky softwarem, takže nevyžadují přítomnost souborů zařízení.

Ve většině případů je síťové zařízení automaticky vytvořeno ovladačem zařízení při inicializaci a nalezení hardwaru. Například ovladač ethernetového zařízení vytvoří rozhraní eth[0..n] postupně po nalezení ethernetového hardwaru. První nalezená ethernetová karta se stane eth0, druhá eth1 atd.

V některých případech, například *slip* a *ppp*, jsou síťová zařízení vytvořena zásahem některého uživatelského programu. Použije se stejné postupné číslování zařízení, ale zařízení nejsou automaticky vytvářena v době zavádění systému. Důvod je v tom, že na rozdíl od ethernetových zařízení se počet aktivních zařízení *slip* a *ppp* za běhu počítače mění. Tyto případy si později ještě podrobněji popíšeme.

5.4 Konfigurace síťového rozhraní

Jakmile máte všechny programy, které potřebujete, a informace o adrese a síti, můžete přistoupit ke konfiguraci vašich síťových rozhraní. Když mluvíme o konfiguraci síťového rozhraní,

mluvíme o procesu přiřazení příslušných adres síťovému zařízení a o nastavení příslušných hodnot pro další nastavitelné parametry síťového zařízení. Nejčastějším programem, který se k tomuto účelu používá, je příkaz `ifconfig`.

Většinou příkaz použijete v následující podobě:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

V tomto případě konfiguruji ethernetové rozhraní „eth0“ s IP-adresou „**192.168.0.1**“ a síťovou maskou „**255.255.255.0**“. Dodatek „*up*“ za příkazem sděluje rozhraní, že se má zaktivovat.

Jádro předpokládá při konfiguraci rozhraní určité implicitní hodnoty. Například pro rozhraní můžete určit síťové adresy a vysílací adresy, ale když je neurčíte (viz můj příklad), jádro je odvodí z dodané síťové masky. Když nedodáte masku, jádro ještě může využít třídu sítě nakonfigurované IP-adresy. V mém případě by jádro předpokládalo na rozhraní konfiguraci sítě třídy C a u rozhraní by nastavilo síťovou adresu „**192.168.0.0**“ a vysílací adresu „**192.168.0.255**“.

Příkaz `ifconfig` má mnoho dalších voleb. Nejdůležitější jsou:

up Tato volba aktivuje rozhraní.

down Tato volba deaktivuje rozhraní.

[- arp] Tato volba u rozhraní nastavuje nebo ruší využití protokolu ARP.

[- allmulti] Tato volba nastavuje nebo ruší příjem všech hardwarových multicastových paketů. Hardwarový multicast umožňuje skupinám hostitelů obdržet pakety, adresované na speciální místa určení. Tohle může být důležité v případě využití aplikací, jako jsou videokonference, které normálně nepoužíváte.

mtu N Tento parametr umožňuje nastavit MTU tohoto zařízení.

netmask addr Tento parametr umožňuje nastavit síťovou masku sítě, do které toto zařízení patří.

irq addr Tento parametr je funkční pouze na určitých typech hardwaru a umožňuje nastavení IRQ tohoto zařízení.

[- broadcast [addr]] Tento parametr umožňuje nastavit přijetí datagramů, určených na vysílací adresu, nebo jejich příjem zrušit.

[- pointopoint [addr]] Tento parametr umožňuje nastavení adresy stroje na vzdáleném konci vazby typu point to point, jako je *slip* nebo *ppp*.

hw <typ> <adresa> Tento parametr umožňuje nastavení hardwarových adres určitých typů síťových zařízení. Není vždy nutný pro Ethernet, ale hodí se k jiným typům sítě, jako je AX.25.

Příkaz `ifconfig` můžete použít na jakémkoliv síťovém rozhraní. Některé uživatelské programy, jako je `pppd` a `dip`, automaticky konfigurují síťová zařízení ve chvíli, kdy je vytváří, takže ruční využití `ifconfig` není nutné.

5.5 Konfigurace resolveru

Resolver je částí standardní linuxové knihovny. Jeho základní funkcí je poskytnutí služeb převodu „lidských“ názvů, jako je „*ftp.funet.fi*“, na strojové IP-adresy, jako je **128.214.248.6**.

5.5.1 Co je v názvu?

Se vzhledem jmen hostitelů Internetu jste se již asi seznámili, ale jak se odvozují, to už asi chápete méně. Názvy lokací tvoří hierarchii, to znamená, že jejich struktura se podobá stromu. „*Doména*“ je rodina nebo skupina názvů. Tu můžeme rozdělit na „*subdomény*“. „*Doménou nejvyšší úrovně*“ rozumíme doménu, která není subdoménou. Domény nejvyšší úrovně jsou specifikovány v RFC-920. Příklady nejběžnějších domén nejvyšší úrovně:

COM komerční organizace

EDU vzdělávací organizace

GOV vládní organizace

MIL vojenské organizace

ORG jiné organizace

Domény nejvyšší úrovně jednotlivých zemí

dvou písmenný kód, reprezentující určitou zemi

Každá z těchto domén nejvyšší úrovně má subdomény. Domény nejvyšší úrovně, rozdělené podle zemí, jsou dále rozděleny na subdomény podle domén **com**, **edu**, **gov**, **mil** a **org**. Například komerční a vládní organizace v Austrálii by měly **com.au** a **gov.au**. Z historických důvodů je většina domén nejvyšší úrovně, které nejsou podle zemí, určena organizacím v USA, i když v Spojené státy mají také vlastní kód země - „**.us**“.

Další úroveň rozdělení většinou reprezentuje název organizace. Další subdomény se liší podle povahy. Často je vyjádřením oddělení organizace, ale může to být i jakékoliv jiné kritérium, které dává síťovým administrátorům organizace smysl.

Část názvu, která je nejvíce vlevo, je vždy jedinečný název hostitelského stroje - „*název hostitele*“. Část napravo je pak „*název domény*“ a celý název je *plně kvalifikované doménové jméno.*

Když jako příklad využiji svého hostitele pro e-mail, plně kvalifikované doménové jméno je „**perf.no.itg.telstra.com.au**“. To znamená, že název hostitele je „**perf**“ a název domény je „**no.itg.telstra.com.au**“. Doménové jméno je odvozeno z domény nejvyšší úrovně, což je v tomto případě moje země - Austrálie. Moje adresa pak náleží komerční organizaci, takže jako doménu následující úrovně máme „**.com**“. Název společnosti je (byl) „**telstra**“ a naše vnitřní pojmenovávací struktura je založena na organizační struktuře. V mém případě můj stroj náleží do Information Technology Group, sekce Network Operations.

5.5.2 Jaké informace budete potřebovat

Budete potřebovat vědět, do které domény náleží název vašeho hostitele. Tuto překladovou službu poskytuje nameresolver provedením žádosti na DNS-server, takže budete muset znát IP-adresu lokálního nameserveru, který můžete využívat.

Editovat musíte tři soubory a nyní je postupně všechny probereme.

5.5.3 /etc/resolv.conf

Jedná se o hlavní konfigurační soubor resolveru. Jeho formát je velmi jednoduchý. Jedná se o textový soubor s jedním klíčovým slovem na každém řádku. Většinou se zde používají tři klíčová slova:

domain Toto klíčové slovo určuje název lokální domény.

search Toto klíčové slovo určuje seznam dalších domén, kde je možné hledat název hostitele.

nameserver Toto klíčové slovo může být použito vícekrát a určuje IP-adresu DNS-serveru, na který se můžete obrátit při zjišťování názvů.

Ukázkový `/etc/resolv.conf`:

```
domain maths.wu.edu.au
search maths.wu.edu.au wu.edu.au
nameserver 192.168.10.1
nameserver 192.168.12.1
```

Tento příklad určuje, že implicitní název domény, připojované za nepřesné názvy (názvy hostitelů bez domény), je **maths.wu.edu.au** nebo (v případě neúspěchu) přímo **wu.edu.au**. Jsou zde dva nameservery, přičemž při zjišťování názvu je možné využít oba.

5.5.4 /etc/host.conf

Zde se konfigurují některé položky, ovlivňující chování kódu resolveru. Formát tohoto souboru je podrobně popsán na manuálové stránce „**resolv+**“. Téměř za všech okolností by vám měl fungovat následující příklad:

```
order hosts,bind
multi on
```

Tato konfigurace sdělí resolveru, aby před pokusem o žádost na nameserver kontroloval soubor `/etc/hosts` a aby podle tohoto souboru vracel všechny platné adresy.

5.5.5 /etc/hosts

Sem se vkládají názvy a IP-adresy lokálních hostitelů. Jestliže sem vložíte hostitele, nemusíte při zjišťování jeho IP-adresy obtěžovat DNS-server. Nevýhodou je to, že tento soubor musíte sami ručně aktualizovat (dochází-li ke změnám). V dobře spravovaném systému se zde objevují pouze údaje pro zpětnovazebné rozhraní loopback a názvy lokálních hostitelů.

```
# /etc/hosts
127.0.0.1 localhost loopback
192.168.0.1 this.host.name
```

Můžete použít i více než jeden název hostitele na řádek, viz první údaj v příkladu.

5.6 Konfigurace zpětnovazebného rozhraní

Rozhraní „**loopback**“ je speciálním typem rozhraní, které umožňuje provést připojení na sebe samého. K tomu existují různé důvody, například testování síťového softwaru bez zasahování do sítě. Většinou se pro loopback přidává IP-adresa „**127.0.0.1**“. Takže pokud zkusíte telnet na **127.0.0.1**, vždy se dostanete na lokálního hostitele.

Konfigurace rozhraní loopback je jednoduchá a určitě ji zvládnete.

```
# ifconfig lo 127.0.0.1
# route add -host 127.0.0.1 lo
```

O příkazu `route` si více povíme v následující části.

5.7 Směrování

Směrování je velké téma. Dalo by se o něm napsat opravdu hodně. Většina z vás má na směrování velmi jednoduché požadavky, někteří ale ne. Já se budu zabývat pouze základy. Jestliže se zajímáte o podrobnější informace, doporučuji vám odkazy ze začátku tohoto dokumentu.

Začněme definicí. Co je IP-směrování? Takto to chápu já: IP-směrování je proces, kterým hostitel s více síťovými připojeními rozhodne, kam odeslat obdržené IP-datagramy.

K tomu by se hodil příklad. Představte si typický router, mohl by mít linku PPP, mimo Internet několik ethernetových segmentů pro pracovní stanice a další linku PPP do jiného úřadu. Když router na kterémkoliv ze svých síťových připojení obdrží datagram, bude směrování mechanizmem, rozhodujícím, na které rozhraní se datagram dále odešle. Směrování potřebují i jednodušší hostitelé. Všechny internetové hostitele mají alespoň dvě síťová zařízení - jedním je výše popsané rozhraní loopback a druhým je rozhraní použité pro komunikaci se zbytkem sítě - snad Ethernet, PPP nebo sériové rozhraní SLIP.

Dobrá, tak jak tedy směrování funguje? Každý hostitel má speciální seznam směrovacích pravidel, nazývaný směrovací tabulka. Tabulka obsahuje řádky většinou s alespoň třemi políčky - adresa určení, název rozhraní, kam se má datagram směrovat a třetí je volitelná IP-adresa dalšího stroje, který má datagram provést přes další krok na jeho cestě sítí. V Linuxu je možné tabulku zobrazit následujícím příkazem:

```
# cat /proc/net/route
```

nebo použitím jednoho z následujících příkazů:

```
# /sbin/route -n
# /bin/netstat -r
```

Směrovací proces je velice jednoduchý: příchozí datagram je obdržen, adresa určení (pro koho to je) je porovnána s údaji v tabulce. Vybere se údaj tabulky, který adrese nejlépe odpovídá, a na takto získané rozhraní je datagram propuštěn. Jestliže je zaplněno pole brány, datagram je předán přes specifické rozhraní, jinak se předpokládá, že adresa určení je dosažitelná uvedeným rozhraním.

Pro manipulaci s touto tabulkou se používá speciální příkaz. Tento příkaz bere argumenty příkazového řádku a převádí je na volání systému jádra, kde se žádá o přidání, smazání nebo úpravu údajů směrovací tabulky. Příkaz se nazývá „`route`“.

Jednoduchý příklad. Představte si ethernetovou síť. Bylo vám řečeno, že se jedná o síť třídy C s adresou **198.168.1.0**. Pro vaše využití vám byla přidělena IP-adresa **192.168.1.10** a řekli vám, že **192.168.1.1** je router, připojený na Internet.

Prvním krokem je konfigurace rozhraní (viz výše) - k tomu využijete příkaz, podobný tomuto:

```
# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
```

Nyní potřebujete údaj přidat do směrovací tabulky, aby jádro vědělo, že datagramy pro všechny hostitele s adresou, odpovídající **192.168.1.*,** by měly být odeslány na ethernetové zařízení. Použili byste přibližně takovýto příkaz:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
```

Povšimněte si využití argumentu „**-net**“, který sděluje směrovacímu programu, že tento údaj je síťový směr. Vaše další volba je zde směr „**-host**“, což je směr, specifický pro jednu IP-adresu.

Tento směr vám umožní vytvořit IP-spojení se všemi hostiteli na vašem ethernetovém segmentu. Ale co pak se všemi IP-hostiteli, kteří na vašem ethernetovém segmentu nejsou?

Nastavení směrů pro všechna možná místa určení na síti by bylo zhola nemožné, takže se zde využívá určitý trik. Jedná se o směr „**default**“, který odpovídá všem možným místům určení, ale pokud se nalezne i jiný směr, je použit místo **default**. Filozofií je zde umožnit prohlásit „a všechno ostatní pošlete sem“. V tomto případě by tedy údaj vypadal takto:

```
# route add default gw 192.168.1.1 eth0
```

Argument „**gw**“ sděluje směrovacímu příkazu, že následujícím argumentem je IP-adresa (nebo název) brány nebo routeru, kterému budou všechny odpovídající datagramy předány pro další směrování.

Takže vaše kompletní konfigurace by vypadala takto:

```
# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
# route add default gw 192.168.1.1 eth0
```

Jestliže se blíže podíváte na vaše síťové soubory „**rc**“, zjistíte, že alespoň jeden z nich vypadá dost podobně. Tato konfigurace je velmi běžná.

Nyní se zaměřme na trochu komplikovanější směrovací konfiguraci. Představme si konfiguraci dříve zmiňovaného routeru, podporujícího linku PPP na Internet a segmenty LAN pro pracovní stanice na úřadě. Představme si, že router má tři ethernetové segmenty a jednu linku PPP. Naše směrovací konfigurace by vypadala takto:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
# route add -net 192.168.2.0 netmask 255.255.255.0 eth1
# route add -net 192.168.3.0 netmask 255.255.255.0 eth2
# route add default ppp0
```

Každá pracovní stanice by využívala jednodušší formu (prezentovanou výše), pouze router potřebuje určit všechny síťové směry odděleně, protože směrovací mechanismus **default** u pracovních stanic by všechny odchytil a nechal na routeru, aby je rozdělil. Možná se divíte, zde implicitní směr neurčuje „**gw**“. Důvod je jednoduchý - protokoly sériových linek (PPP a SLIP) mají na svých sítích pouze dva hostitele - jeden na každém konci. Určit hostitele z druhého konce linky jako bránu je zbytečné, protože nic jiného stejně nezbývá. U těchto typů síťových připojení tedy není nutné určit bránu. Jiné typy sítí, jako je Ethernet, arcnet nebo token ring, určení brány vyžadují, protože tyto sítě v sobě podporují velké množství hostitelů.

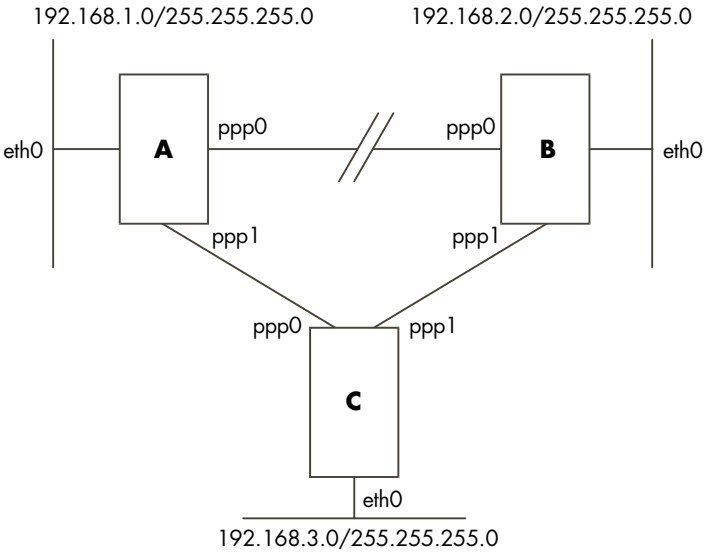
5.7.1 Takže co dělá program routed?

Výše popsaná směrovací konfigurace se nejlépe hodí pro jednodušší sítě, kde je většinou jen jedna možná cesta k cíli. Když je síť složitější, věci se hodně zkomplikují - naštěstí se to většiny z vás netýká a nemusíte se tím zabývat.

Velkým problémem „ručního směrování“ nebo „statického směrování“ (jak bylo popsáno) je situace, kdy selže stroj nebo linka a jediným způsobem směrování jinou cestou (existuje-li vůbec taková) je osobní zásah a provedení příslušných příkazů. Tohle je samozřejmě hloupé, pomalé, nepraktické a riskantní. Tohle je možné provádět i automaticky - existuje mnoho postupů, souhrnně nazývaných „dynamické směrovací protokoly“.

Možná jste o těch běžnějších slyšeli - nejznámější jsou RIP a OSPF. RIP je velmi běžný na malých sítích, jako jsou sítě v malých až středních společnostech nebo v budovách. OSPF je modernější a schopnější zvládnout konfigurace velkých sítí. Je také vhodnější v případě, že existuje velké množství možností cesty sítí. Známé implementace těchto protokolů jsou: „`routed`“ - RIP a „`gated`“ - RIP, OSPF a další. Program „`routed`“ se normálně dodává ve vaší distribuci Linuxu nebo je obsažen v balíku „NetKit“, viz výše.

Příklad kdy a jak můžete využít dynamický směrovací protokol vypadá následovně:



Máme tři routery A, B a C. Každý podporuje jeden ethernetový segment s IP-sítí třídy C (síťová maska 255.255.255.0). Každý router má také linku PPP ke zbylým routerům. Síť je ve tvaru trojúhelníku.

Mělo by být jasné, že směrovací tabulka na routeru A by měla vypadat takto:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
# route add -net 192.168.2.0 netmask 255.255.255.0 ppp0
# route add -net 192.168.3.0 netmask 255.255.255.0 ppp1
```

Tohle by mělo spolehlivě fungovat až do chvíle, kdy by selhala linka mezi routerem A a B. V takovém případě by hostitelé na ethernetovém segmentu A nemohli dosáhnout hostitele na ethernetovém segmentu B, protože jejich datagramy by byly směrovány na linku PPP routeru A, která je nefunkční. Mohou ale stále komunikovat s hostiteli na segmentu C a ti zase s hostiteli na segmentu B, protože linka mezi B a C je funkční.

Když ale A komunikuje s C a C s B, proč by A nemohlo poslat svoje datagramy do B přes C? To je zhruba druh problémů, které řeší dynamické směrovací protokoly jako RIP. Jestliže všechny routery A, B a C měly puštěný směrovací démon, pak jsou jejich směrovací tabulky při poruše některé linky automaticky upraveny, aby odrážely nový stav sítě. Konfigurace takové sítě je snadná - na každém routeru potřebujete provést pouze dvě věci. V našem případě pro router A:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
# /usr/sbin/routed
```

Směrovací démon „*routed*“ při spuštění automaticky nalezne všechny aktivní síťové porty a odesílá a přijímá zprávy na každé síťové zařízení, aby bylo možné určit a aktualizovat směrovací tabulku na hostiteli.

Tohle bylo velice stručné vysvětlení dynamického směrování a jeho použití. Jestliže potřebujete více informací, měli byste se vrátit na začátek dokumentu a pročíst si odkazy.

Důležité postřehy, vztahující se k dynamickému směrování, jsou:

1. Démona pro dynamické směrování potřebujete na stroji s Linuxem spouštět pouze v případě, že máte na výběr více možných směrů na místo určení.
2. Tento démon automaticky upravuje vaši směrovací tabulku, čímž odráží změny v síti.
3. RIP je vhodný pro malé až středně velké sítě.

5.8 Konfigurace vašich síťových serverů a služeb

Síťové servery a služby jsou ty programy, které umožňují vzdálenému uživateli, aby se stal uživatelem vašeho linuxového stroje. Serverové programy naslouchají na síťových portech. Síťové porty jsou významem adresace určité služby na určitém hostiteli. Podle nich server pozná rozdíl mezi přicházejícím telnetovým a ftp-připojením. Vzdálený uživatel vytvoří síťové připojení k vašemu stroji a serverovému programu. Program síťového démona, který na tomto portu naslouchá, připojení přijme a provede. Existují dva způsoby, jakými mohou síťoví démoni fungovat. Oba jsou běžně využívány v praxi:

standalone (samostatný)
: Síťový démon naslouchá na vyznačeném síťovém portu a jakmile je provedeno přicházející připojení, spravuje samotné síťové spojení, aby poskytovalo svoje služby.

slave (podřízený) **pod** `inetd` **serverem**
: Server `inetd` je síťový démon, který se specializuje na příchozí síťová spojení. Má konfigurační soubor, kde se říká, který program má být spuštěn po navázání spojení. Na protokoly TCP nebo UDP je možné nakonfigurovat jakýkoliv servisní port. Porty jsou popsány v dalším souboru, ke kterému se brzy dostaneme.

Musíme nakonfigurovat dva důležité soubory. Jedná se o `/etc/services` (přiřazuje názvy číslům portů) a o `/etc/inetd.conf`, který je konfiguračním souborem síťového démona `inetd`.

5.8.1 /etc/services

Tento soubor je jednoduchou databází, která přiřazuje „lidské“ názvy strojovým portům služeb. Jeho formát je jednoduchý. Soubor je textový, přičemž každý řádek reprezentuje údaj databáze. Každý údaj se skládá ze tří polí, oddělených libovolným množstvím prázdného místa (tabulátory nebo mezery). Pole jsou:

```
jméno           port/protokol     přezdívky     # komentář
```

jméno Jednoslovný název, který reprezentuje popisovanou službu.

port/protokol Toto pole je rozděleno na dvě:

port číslo, které určuje číslo portu, na kterém bude pojmenovaná služba k dispozici. Většina běžných služeb má přiřazena standardní čísla. Jsou popsána v **RFC-1340**.

protokol Toto pole je možné nastavit na **tcp** nebo **udp**.

Důležité je, že údaj **18/tcp** se značně liší od **18/udp** a není zde žádný technický důvod, proč by stejná služba měla existovat na obou. Většinou převládá obecný smysl a údaj pro oba se objeví pouze v případě, že je určitá služba také pro oba k dispozici.

přezdívky Další názvy, které je možné použít při odkazu na údaje této služby.

Jakýkoliv text, který se na řádku objeví za znakem „**#**“, je ignorován a posuzován jako komentář.

Ukázkový soubor /etc/services

Všechny současné distribuce Linuxu nabízí důležitý soubor `/etc/services`. Jen pro případ, že byste vytvářeli stroj úplně od základů, sem umisťuji kopii souboru `/etc/services`, který je v distribuci *Debian <http://www.debian.org/>*.

```
# /etc/services:
# $Id: services,v 1.3 1996/05/06 21:42:37 tobias Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single
# well-known
# port number for both TCP and UDP; hence, most entries here have
# two entries
```

```
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1340, ''Assigned Numbers'' (July 1992). Not all
# ports are included, only the more common ones.

tcpmux          1/tcp # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp  sink null
discard         9/udp  sink null
systat          11/tcp users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp quote
msp             18/tcp # message send protocol
msp             18/udp # message send protocol
chargen         19/tcp ttytst source
chargen         19/udp ttytst source
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp # SSH Remote Login Protocol
ssh             22/udp # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp mail
# 26 - unassigned
time            37/tcp timserver
time            37/udp timserver
rlp             39/udp resource # resource location
nameserver      42/tcp name # IEN 116
whois           43/tcp nicname
re-mail-ck      50/tcp # Remote Mail Checking Protocol
re-mail-ck      50/udp # Remote Mail Checking Protocol
domain          53/tcp nameserver # name-domain server
domain          3/udp nameserver
mtp             57/tcp # deprecated
bootps          67/tcp # BOOTP server
```

```
bootps          67/udp
bootpc          68/tcp # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp # Internet Gopher
gopher          70/udp
rje             77/tcp netrjs
finger          79/tcp
www             80/tcp http # WorldWideWeb HTTP
www             80/udp # HyperText Transfer Protocol
link            87/tcp ttylink
kerberos        88/tcp kerberos5 krb5 # Kerberos v5
kerberos        88/udp kerberos5 krb5 # Kerberos v5
supdup          95/tcp
# 100 - reserved
hostnames       101/tcp hostname # usually from sri-nic
iso-tsap        102/tcp tsap # part of ISODE.
csnet-ns        105/tcp cso-ns # also used by CSO name server
csnet-ns        105/udp cso-ns
rtelnet         107/tcp # Remote Telnet
rtelnet         107/udp
pop-2           109/tcp postoffice # POP version 2
pop-2           109/udp
pop-3           110/tcp # POP version 3
pop-3           110/udp
sunrpc          111/tcp portmapper # RPC 4.0 portmapper TCP
sunrpc          111/udp portmapper # RPC 4.0 portmapper UDP
auth            113/tcp authentication tap ident
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp readnews untp # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp # Network Time Protocol
netbios-ns      137/tcp # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/tcp # NETBIOS Datagram Service
netbios-dgm     138/udp
netbios-ssn     139/tcp # NETBIOS session service
```

```
netbios-ssn   139/udp
imap2         143/tcp # Interim Mail Access Proto v2
imap2         143/udp
snmp          161/udp # Simple Net Mgmt Proto
snmp-trap     162/udp snmptrap # Traps for SNMP
cmip-man      163/tcp # ISO mgmt over IP (CMOT)
cmip-man      163/udp
cmip-agent    164/tcp
cmip-agent    164/udp
xdmcp         177/tcp # X Display Mgr. Control Proto
xdmcp         177/udp
nextstep      178/tcp NeXTStep NextStep # NeXTStep window
nextstep      178/udp NeXTStep NextStep # server
bgp           179/tcp # Border Gateway Proto.
bgp           179/udp
prospero      191/tcp # Cliff Neuman's Prospero
prospero      191/udp
irc           194/tcp # Internet Relay Chat
irc           194/udp
smux          199/tcp # SNMP Unix Multiplexer
smux          199/udp
at-rtmp       201/tcp # AppleTalk routing
at-rtmp       201/udp
at-nbp        202/tcp # AppleTalk name binding
at-nbp        202/udp
at-echo       204/tcp # AppleTalk echo
at-echo       204/udp
at-zis        206/tcp # AppleTalk zone information
at-zis        206/udp
z3950         210/tcp wais # NISO Z39.50 database
z3950         210/udp wais
ipx           213/tcp # IPX
ipx           213/udp
imap3         220/tcp # Interactive Mail Access
imap3         220/udp # Protocol v3
ulistserv     372/tcp # UNIX Listserv
ulistserv     372/udp
#
```

```
# UNIX specific services
#
exec          512/tcp
biff          512/udp comsat
login         513/tcp
who           513/udp whod
shell         514/tcp cmd   # no passwords used
syslog        514/udp
printer       515/tcp spooler # line printer spooler
talk          517/udp
ntalk         518/udp
route         520/udp router routed # RIP
timed         525/udp timeserver
tempo         526/tcp newdate
courier       530/tcp rpc
conference    531/tcp chat
netnews       532/tcp readnews
netwall       533/udp # -for emergency broadcasts
uucp          540/tcp uucpd  # uucp démon
remotefs      556/tcp rfs_server rfs # Brunhoff remote filesystem
klogin        543/tcp # Kerberized 'rlogin' (v5)
kshell        544/tcp krcmd  # Kerberized 'rsh' (v5)
kerberos-adm  749/tcp # Kerberos 'kadmin' (v5)
#
webster       765/tcp # Network dictionary
webster       765/udp
#
# From ''Assigned Numbers'':
#
#> The Registered Ports are not controlled by the IANA and on most
#> systems can be used by ordinary user processes or programs
   executed by
#> ordinary users.
#
#> Ports are used in the TCP [45,106] to name the ends of logical
#> connections which carry long term conversations. For the
```

```
#> purpose of
#> providing services to unknown callers, a service contact port is
#> defined. This list specifies the port used by the server process
#> as its contact port. While the IANA can not control uses of these
   ports
#> it does register or list uses of these ports as a convienence
#> to the community.
#
ingreslock    1524/tcp
ingreslock    1524/udp
prospero-np   1525/tcp  # Prospero non-privileged
prospero-np   1525/udp
rfe           5002/tcp  # Radio Free Ethernet
rfe           5002/udp  # Actually uses UDP only
bbs           7000/tcp  # BBS service
#
#
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4 and are unofficial. Sites
# running v4 should uncomment these and comment out the v5 entries
  above.
#
kerberos4            750/udp kdc # Kerberos (server) udp
kerberos4            750/tcp kdc # Kerberos (server) tcp
kerberos_master      751/udp # Kerberos authentication
kerberos_master      751/tcp # Kerberos authentication
passwd_server        52/udp # Kerberos passwd server
krb_prop             754/tcp # Kerberos slave propagation
krbupdate 7          60/tcp kreg # Kerberos registration
kpasswd              761/tcp kpwd # Kerberos "passwd"
kpop                 1109/tcp  # Pop with Kerberos
knetd                2053/tcp  # Kerberos de-multiplexor
zephyr-srv           2102/udp  # Zephyr server
zephyr-clt           2103/udp  # Zephyr serv-hm connection
zephyr-hm            2104/udp  # Zephyr hostmanager
eklogin              2105/tcp  # Kerberos encrypted rlogin
#
# Unofficial but necessary (for NetBSD) services
```

```
#
supfilesrv    871/tcp # SUP server
supfiledbg    1127/tcp  # SUP debugging
#
# Datagram Delivery Protocol services
#
rtmp          1/ddp # Routing Table Maintenance Protocol
nbp           2/ddp # Name Binding Protocol
echo          4/ddp # AppleTalk Echo Protocol
zip           6/ddp # Zone Information Protocol
#
# Debian GNU/Linux services
rmtcfg        1236/tcp  # Gracilis Packeten remote config server
xtel          1313/tcp  # french minitel
cfinger       2003/tcp  # GNU Finger
postgres      4321/tcp  # POSTGRES
mandelspawn   9359/udp mandelbrot # network mandelbrot

# Local services
```

5.8.2 /etc/inetd.conf

Jedná se o konfigurační soubor pro serverového démona `inetd`. Jeho funkcí je sdělit `inetd` co dělat, když obdrží žádost o spojení na určitou službu. U každé služby, pro kterou chcete přijmout spojení, musíte `inetd` sdělit, jak a který démon síťového serveru spustit.

Jeho formát je také velice jednoduchý. Jedná se o textový soubor s každým řádkem popisujícím službu, kterou chcete poskytnout. Jakýkoliv text, který se na řádku objeví za znakem „**#**“, je ignorován a posuzován jako komentář. Každý řádek obsahuje sedm polí, oddělených libovolným množstvím prázdného místa (tabulátory nebo mezery). Obecný formát vypadá takto:

```
služba  typ soketu  protokol  volby  uživatel  cesta_k_serveru\
  argumenty_serveru
```

služba — Je služba, příslušná této konfiguraci, převzatá ze souboru `/etc/services`.

typ_soketu — Toto pole popisuje typ socketu. Možné hodnoty jsou: **stream**, **dgram**, **raw**, **rdm** nebo **seqpacket**, což je sice trochu více technické, ale platí zde pravidlo, že skoro všechny služby **tcp** používají **stream** a skoro všechny služby **udp** používají **dgram**. Pouze velmi speciální typy serverových démonů využívají některé ze zbývajících hodnot.

protokol Protokol, platný pro tento údaj. Měl by odpovídat příslušnému údaji v souboru `/etc/services` a bývá **tcp** nebo **udp**. Servery Sun RPC budou využívat **rpc/tcp** nebo **rpc/udp**.

volby Toto pole má jen dvě možná nastavení. Zde si `inetd` zjistí, jestli program síťového serveru po spuštění uvolní socket a jestli má tedy `inetd` při následující žádosti o spojení spustit další, nebo jestli má `inetd` čekat a předpokládat, že nějaký již běžící serverový démon se o žádost o nové připojení postará. Opět je zde chápání trochu zjednodušené, ale pravidlem je, že všechny servery **tcp** by měly mít nastaveno **nowait** a většina serverů **udp** by měla mít **wait**. Jsou zde však časté výjimky, takže pokud si nejste jisti, raději se řiďte podle příkladu.

uživatel Toto pole popisuje, který uživatelský účet z `/etc/passwd` bude pro spuštěného síťového démona nastaven jako vlastník. Užitečné zejména z hlediska bezpečnosti. Uživatele můžete nastavit jako **nobody**, takže pokud se naruší bezpečnost serveru, možné ztráty budou minimální. Většinou je zde ale nastaven **root**, protože mnohé servery ke své správné funkci vyžadují práva superuživatele.

cesta_k_serveru Cesta k aktuálnímu serverovému programu, spouštěnému pro tento údaj.

argumenty_serveru Tohle pole tvoří zbytek řádku a je volitelné. Zde se umisťují všechny parametry příkazového řádku, které chcete předat serverovému démonu při jeho spuštění.

Ukázkový soubor /etc/inetd.conf

Stejně jako u `/etc/services` všechny současné distribuce Linuxu nabízí správný soubor `/etc/inetd.conf`. Pro doplnění sem umisťuji kopii souboru `/etc/inetd.conf`, který je v distribuci *Debian <http://www.debian.org/>*.

```
# /etc/inetd.conf: see inetd(8) for further informations.
#
# Internet server configuration database
#
#
# Modified for Debian by Peter Tobias <tobias@et-inf.fho-emden.de>
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path>
#   <args>
```

```
#
# Internal services
#
#echo                stream      tcp    nowait   root    internal
#echo                dgram       udp    wait     root    internal
discard              stream      tcp    nowait   root    internal
discard              dgram       udp    wait     root    internal
daytime              stream      tcp    nowait   root    internal
daytime              dgram       udp    wait     root    internal
#chargen             stream      tcp    nowait   root    internal
#chargen             dgram       udp    wait     root    internal
time                 stream      tcp    nowait   root    internal
time                 dgram       udp    wait     root    internal
#
# These are standard services.
#
telnet  stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.telnetd
ftp     stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.ftpd
#fsp    dgram   udp  wait   root  /usr/sbin/tcpd /usr/sbin/in.fspd
#
# Shell, login, exec and talk are BSD protocols.
#
shell   stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.rshd
login   stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.rlogind
#exec   stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.rexecd
talk    dgram   udp  wait   root  /usr/sbin/tcpd /usr/sbin/in.talkd
ntalk   dgram   udp  wait   root  /usr/sbin/tcpd /usr/sbin/in.ntalkd
#
# Mail, news and uucp services.
#
smtp    stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.smtpd
#nntp   stream  tcp  nowait news  /usr/sbin/tcpd /usr/sbin/in.nntpd
#uucp   stream  tcp  nowait uucp  /usr/sbin/tcpd /usr/lib/uucp/uucico
#comsat dgram   udp  wait   root  /usr/sbin/tcpd /usr/sbin/in.comsat
#
#       Pop     et   al
#
#pop-2  stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.pop2d
```

```
#pop-3 stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.pop3d
#
# 'cfinger' is for the GNU finger server available for Debian.
# (NOTE: The current implementation of the 'finger' daemon allows it
# to berun as 'root'.)
#
#cfinger stream tcp nowait  root   /usr/sbin/tcpd /usr/sbin/in.cfingerd
#finger  stream tcp nowait  root   /usr/sbin/tcpd /usr/sbin/in.fingerd
#netstat stream tcp nowait  nobody /usr/sbin/tcpd /bin/netstat
#systat  stream tcp nowait  nobody /usr/sbin/tcpd /bin/ps -auwwx
#
# Tftp service is provided primarily for booting. Most sites
# run this only on machines acting as "boot servers."
#
#tftp    dgram  udp wait    nobody /usr/sbin/tcpd /usr/sbin/in.tftpd
#tftp    dgram  udp wait    nobody /usr/sbin/tcpd /usr/sbin/in.tftpd/boot
#bootps  dgram  udp wait    root   /usr/sbin/bootpd bootpd -i -t 120
#
# Kerberos authenticated services (these probably need to be
# corrected)
#
#klogin stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.rlogind -k
#eklogin stream tcp nowait root  /usr/sbin/tcpd /usr/sbin/in.rlogind
#-k -x
#kshell stream  tcp  nowait root  /usr/sbin/tcpd /usr/sbin/in.rshd -k
#
# Services run ONLY on the Kerberos server (these probably need to be
# corrected)
#
#krbupdate stream tcp nowait root /usr/sbin/tcpd /usr/sbin/registerd
#kpasswd   stream tcp nowait root /usr/sbin/tcpd /usr/sbin/kpasswdd
#
# RPC based services
#
#mountd/1    dgram rpc/udp wait root /usr/sbin/tcpd /usr/sbin/rpc.mountd
#rstatd/1-3  dgram rpc/udp wait root /usr/sbin/tcpd /usr/sbin/rpc.rstatd
#rusersd/2-3 dgram rpc/udp wait root /usr/sbin/tcpd /usr/sbin/rpc.rusersd
#walld/1     dgram rpc/udp wait root /usr/sbin/tcpd /usr/sbin/rpc.rwalld
```

```
#
# End of inetd.conf.
ident  stream tcp nowait nobody /usr/sbin/identd identd -i
```

5.9 Další síťové konfigurační soubory

V Linuxu existuje množství různých síťových konfiguračních souborů, které by vás mohly zajímat. Nebudete je muset nikdy upravovat, ale stojí zato si je popsat, abyste věděli, co obsahují a k čemu slouží.

5.9.1 /etc/protocols

Tento soubor je databází, která mapuje identifikační čísla protokolů na názvy protokolů. Využívají jej programátoři, aby mohli ve svých programech specifikovat protokoly prostřednictvím názvů, a také některé programy (například `tcpdump`), aby na výstupu zobrazily názvy místo čísel. Obecná syntaxe je:

```
jméno  číslo  přezdívky
```

Soubor `/etc/protocols` je v distribuci *Debian <http://www.debian.org/>* v následující podobě:

```
# /etc/protocols:
# $Id: protocols,v 1.1 1995/02/24 01:09:41 imurdock Exp $
#
# Internet (IP) protocols
#
# from: @(#)protocols 5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July
1992).

ip         0   IP         # internet protocol, pseudo protocol number
icmp       1   ICMP       # internet control message protocol
igmp       2   IGMP       # Internet Group Management
ggp        3   GGP        # gateway-gateway protocol
ipencap    4   IP-ENCAP   # IP encapsulated in IP (officially ''IP'')
st         5   ST         # ST datagram mode
tcp        6   TCP        # transmission control protocol
egp        8   EGP        # exterior gateway protocol
```

```
pup         12 PUP        # PARC universal packet protocol
udp         17 UDP        # user datagram protocol
hmp         20 HMP        # host monitoring protocol
xns-idp     22 XNS-IDP    # Xerox NS IDP
rdp         27 RDP        # "reliable datagram" protocol
iso-tp4     29 ISO-TP4    # ISO Transport Protocol class 4
xtp         36 XTP        # Xpress Tranfer Protocol
ddp         37 DDP        # Datagram Delivery Protocol
idpr-cmtp   39 IDPR-CMTP  # IDPR Control Message Transport
rspf        73 RSPF       # Radio Shortest Path First.
vmtp        81 VMTP       # Versatile Message Transport
ospf        89 OSPFIGP    # Open Shortest Path First IGP
ipip        94 IPIP       # Yet Another IP encapsulation
encap       98 ENCAP      # Yet Another IP encapsulation
```

5.9.2 /etc/networks

Tento soubor má podobnou funkci, jakou má `/etc/hosts`. Poskytuje jednoduchou databázi síťových názvů a síťových adres. Jeho formát se liší v tom, že na řádku jsou jen dvě pole a ty vypadají následovně:

```
jméno_sítě    adresa_sítě
```

Příklad může vypadat takto:

```
loopnet 127.0.0.0
localnet 192.168.0.0
amprnet 44.0.0.0
```

Když používáte příkazy, jako je `route`, s místem určení nějaké sítě, o které existuje v `/etc/networks` záznam, směrovací příkaz zobrazí u sítě místo adresy název.

5.10 Zabezpečení sítě a kontrola přístupu

Dovolte mi začít tuto část upozorněním, že zabezpečení sítě proti nežádoucím útokům je složité umění. Na tohle téma se necítím příliš povolaným odborníkem. I když je mechanismus, který popíši, jistě přínosem, jestli chcete zabezpečení brát vážně, doporučuji vám používat vlastní postupy, které vám nejlépe vyhovují. Na dané téma je možné z Internetu získat mnoho dobrých odkazů.

Důležité pravidlo zní: „**Nespouštějte servery, které nehodláte používat.**“ Mnoho distribucí má nakonfigurovány různé služby, které se automaticky spustí. Alespoň minimální úroveň zabezpečení zajistíte, když si projdete soubor `/etc/inetd.conf` a vložíte do komentáře (*umístěním znaku „#“ na začátek řádku*) údaje ke službám, které nehodláte používat. Dobrými kandidáty jsou služby, jako **shell**, **login**, **exec**, **uucp**, **ftp** a informační služby, jako **finger**, **netstat** a **systat**.

Existují různé druhy zabezpečovacích a přístupových mechanismů, já zde ale popíši jenom ty nejzákladnější.

5.10.1 /etc/ftpusers

Tento soubor je jednoduchým mechanismem, který vám umožňuje pro některé uživatele zakázat přístup na váš stroj pomocí FTP. Soubor je čten ftp-démonem (`ftpd`) při obdržení přicházejícího ftp-spojení. Seznam je jednoduchým seznamem těch uživatelů, kterým není povolen přístup. Může vypadat asi takto:

```
# /etc/ftpusers - uživatelé, kterým není povoleno ftp-spojení
root
uucp
bin
mail
```

5.10.2 /etc/securetty

Soubor, umožňující nastavení zařízení **tty**, na která se může přihlásit **root**. Tento soubor je čten programem `login` (obvykle `/bin/login`). Jeho formátem je seznam názvů zařízení **tty**, na která se **root** může přihlásit (na zbývající nemůže):

```
# /etc/securetty - zařízení tty, na která se může přihlásit root
tty1
tty2
tty3
tty4
```

5.10.3 Mechanismus kontroly přístupu tcpd

Program *tcpd*, který jste viděli v **/etc/inetd.conf**, poskytuje přihlašovací a přístupové kontrolní mechanismy službám, ke kterým je nakonfigurován, aby je chránil.

Po vyvolání programem `inetd` čte dva soubory, obsahující přístupová pravidla, a umožňuje nebo zakazuje přístup na server, který chrání.

Prohledává soubory s pravidly, dokud nenalezne první shodu. Jestliže není nalezena žádná shoda, usoudí, že je možné povolit přístup každému. Soubory, které postupně prochází, jsou `/etc/hosts.allow`, `/etc/hosts.deny`. Oba dva ještě popíši. Kompletní popis naleznete na příslušných manuálových stránkách (dobrým začátkem je **host_access(5)**).

/etc/hosts.allow

Konfigurační soubor programu `/usr/sbin/tcpd`. Soubor `hosts.allow` obsahuje pravidla, popisující, kteří hostitelé *mohou* přistoupit ke službám na vašem stroji.

Formát souboru je jednoduchý:

```
# /etc/hosts.allow
#
# <seznam služeb>: <seznam hostitelů> [: příkaz]
```

seznam služeb Čárkami oddělený seznam názvů serverů, na které se tohle pravidlo vztahuje. Příklady názvů serverů jsou: `ftpd`, `telnetd` a `fingerd`.

seznam hostitelů Čárkami oddělený seznam názvů hostitelů. Můžete zde také použít IP-adresy. Názvy hostitelů nebo adresy můžete dodatečně určit pomocí zástupných znaků, aby odpovídaly celým skupinám. Příklady jsou: **gw.vk2ktj.ampr.org**, odpovídající jednomu hostiteli; **uts.edu.au**, odpovídající jakémukoliv názvu, končícímu tímto řetězcem; **44.**, odpovídající jakékoliv IP-adrese, začínající na toto číslo. Existuje pár speciálních slov, která zjednodušují konfiguraci - některé z nich jsou: **ALL** (odpovídá všemu), **LOCAL** (odpovídá všemu, co neobsahuje tečku, což znamená, že je ve stejné doméně jako váš stroj) a **PARANOID** (odpovídá všem hostitelům, jejichž název neodpovídá jejich adrese). Ještě je užitečné **EXCEPT**, umožňující nastavení seznamu s výjimkami. Později uvedeme příklad.

příkaz Operační parametr. Tento parametr určuje plnou cestu k příkazu, spouštěnému vždy, když je pravidlo splněno. Může například spouštět příkaz, který se pokusí identifikovat, kdo je přihlášen na připojujícím se hostiteli, nebo generovat zprávu nebo jiné upozornění systémovému správci, že se někdo snaží připojit. Je možné využít množství rozšíření, nejběžnějšími příklady jsou: **%h** se expanduje na název připojovaného hostitele nebo adresu, jestliže nemá název, **%d** na název démona, který je volán.

Příklad:

```
# /etc/hosts.allow
#
# Poštu povolíme všem.
in.smtpd: ALL
# Telnet a ftp povolíme pouze počítačům z naší domény nebo počítači
doma.
telnetd, ftpd: LOCAL, myhost.athome.org.au
# Službu finger povolíme všem, ale poznamenáme si je.
fingerd: ALL: (finger @%h | mail -s "finger from %h" root)
```

/etc/hosts.deny

Konfigurační soubor programu `/usr/sbin/tcpd`. Soubor `hosts.deny` obsahuje pravidla, popisující, kteří hostitelé *nesmí* přistoupit ke službám na vašem stroji.

Jednoduchá ukázka vypadá následovně:

```
# /etc/hosts.deny
#
# Zakážeme všechny služby těm, kteří nemají v pořádku jméno hostitele
ALL: PARANOID
#
# Zakážeme vše
ALL: ALL
```

Údaj **PARANOID** je zde nadbytečný, protože ostatní údaje stejně všechno odchytí. V závislosti na svých požadavcích si některé z těchto údajů můžete nastavit jako implicitní.

Nejbezpečnější konfigurací je v `/etc/hosts.deny` implicitní hodnota **ALL: ALL** a následné umožnění služeb některým hostitelům v `/etc/hosts.allow`.

5.10.4 /etc/hosts.equiv

Soubor se využívá k zajištění přístupu na váš stroj bez hesla pro některé hostitele nebo uživatele. To je užitečné v bezpečném prostředí, kde ovládáte všechny stroje, jinak je to velmi riskantní. Váš stroj je zabezpečen tak, jako nejméně zabezpečený hostitel, kterému důvěřujete. Aby se zabezpečení zvýšilo, nepoužívejte tento mechanismus a zařiďte, aby vaši uživatelé nevyužívali soubor `.rhosts`.

5.10.5 Řádně konfigurujte vašeho ftp-démona

Množství hostitelů využívá anonymní server, umožňující vkládání a stahování souborů bez specifické uživatelské identifikace. Jestliže se rozhodnete tuto možnost nastavit, musíte pro anonymní přístup řádně nakonfigurovat ftp-démona. Většina manuálových stránek pro `ftpd(8)` v určité délce popisuje, jak se s tím vypořádat. Vždy se řiďte doporučenými instrukcemi. Důležitým tipem je nepoužívat kopii vašeho souboru `/etc/passwd` v adresáři `/etc` anonymního účtu. Odstiňte všechny podrobnosti o účtech, kromě těch nejnutnějších, jinak budete zranitelní metodami rozkódování hesla hrubou silou.

5.10.6 Firewally

Neumožnit datagramům dosáhnout na váš stroj nebo server je skvělý příklad zabezpečení. Viz dokument **<Firewall-HOWTO.html>**.

5.10.7 Další návrhy

Zde je pár dalších, potenciálně zbožných návrhů, které můžete vzít v úvahu.

sendmail

Navzdory popularitě se démon `sendmail` objevuje se železnou pravidelností v hlášeních o nedostatcích v zabezpečení. Je to na vás, ale já bych jej nepoužíval.

NFS a další služby Sun RPC

Pozor na ně. Tyto služby umožňují všechny druhy možných průniků. Ke službám, jako je NFS, se obtížně hledá náhrada, ale pokud je použijete, ujistěte se, komu umožňujete právo připojení.

6 Informace, specifické k síťovým technologiím

Následující části jsou specifické pro určité síťové technologie. Informace zde obsažené se nemusí nutně vztahovat na jiné síťové technologie.

6.1 ARCNet

Názvy zařízení ARCNet jsou „arc0e“, „arc1e“, „arc2e“ atd. nebo „arc0s“, „arc1s“, „arc2s“ atd. První detekované kartě je přiřazen název „arc0e“ nebo „arc0s“ a zbylé názvy jsou přiřazovány postupně v pořadí, v jakém jsou detekovány. Písmeno na konci názvu označuje, zda jste vybrali paket formátu ethernetové zapouzdření nebo paket formátu RFC1051.

Volby překladu jádra:

```
Network device support  --->
      [*] Network device support
      <*> ARCnet support
      [ ]   Enable arc0e (ARCnet "Ether-Encap" packet format)
      [ ]   Enable arc0s (ARCnet RFC1051 packet format)
```

Jakmile máte sestaveno jádro s podporou ethernetové karty, je již vlastní konfigurace karty jednoduchá:

Zpravidla byste měli použít zápis podobný tomu následujícímu:

```
# ifconfig arc0e 192.168.0.1 netmask 255.255.255.0 up
# route add -net 192.168.0.0 netmask 255.255.255.0 arc0e
```

Podrobnější informace najdete v souborech `/usr/src/linux/Documentation/networking/arcnet.txt` a `/usr/src/linux/Documentation/networking/arcnet-hardware.txt`.

Autorem podpory ARCNet je Avery Pennarun, `apenwarr@foxnet.net`.

6.2 Appletalk (AF_APPLETALK)

Podpora protokolu Appletalk nepoužívá žádné speciální názvy zařízení, protože využívá síťová zařízení.

Volby překladu jádra :

```
Networking options  --->
<*> Appletalk DDP
```

Podpora Appletalk umožňuje vašemu počítači spolupracovat se sítěmi Appletalk. Má důležité využití při sdílení zdrojů jako jsou tiskárny a disky mezi linuxovými počítači a počítači Apple. Potřebujete k domu dodatečný software, který se nazývá `netatalk`. Wesley Craig **<netatalk@umich.edu>** reprezentuje tým nazvaný „Research Systems Unix Group“ na University of Michigan, který tento balík vytvořil. Poskytuje software, který implementuje protokol Appletalk a některé užitečné utility. Balík `netatalk` bude buď součástí vaší distribuce Linuxu nebo si ho musíte stáhnout pomocí služby z University of Michigan *<ftp://terminator.rs.itd.umich.edu/unix/netatalk/>*.

Balík sestavíte a nainstalujete pomocí následujících příkazů:

```
# cd /usr/src
# tar xvfz .../netatalk-1.4b2.tar.Z
- V tuto chvíli možná bude nutné upravit soubor ‚Makefile‘, přesněji
  změnit proměnnou DESTDIR, která definuje místo pozdější instalaci
  souborů. Implicitní adresář je /usr/local/atalk je ovšem poměrně
  bezpečný.
# make
- jako root:
# make install
```

6.2.1 Konfigurace softwaru Appletalk

Nejprve je třeba zkontrolovat, zda jsou v souboru `/etc/services` příslušné záznamy. Jsou to tyto záznamy:

```
rtmp     1/ddp    # Routing Table Maintenance Protocol
nbp      2/ddp    # Name Binding Protocol
echo     4/ddp    # AppleTalk Echo Protocol
zip      6/ddp    # Zone Information Protocol
```

Dalším krokem je vytvoření konfiguračních souborů Appletalk v adresáři `/usr/local/atalk/etc` (nebo v adresáři, do kterého jste nainstalovali balík Appletalk).

První soubor má název `/usr/local/atalk/etc/atalkd.conf`. Zpočátku stačí, když bude tento soubor obsahovat pouze jediný řádek, na kterém bude uveden název síťového zařízení, které podporuje síť, k niž jsou připojeny vaše počítače Apple:

```
eth0
```

Démon Appletalk do něj po svém spuštění přidá další podrobnosti.

6.2.2 Export souborových systémů prostřednictvím protokolu Appletalk

Souborové systémy z vašeho linuxového počítače lze exportovat do sítě, aby je mohl sdílet počítač Apple připojený k téže síti.

Aby to bylo možné, je třeba nejdříve upravit soubor `/usr/local/atalk/etc/AppleVolumes.system`. Existuje ještě jeden konfigurační soubor jménem `/usr/local/atalk/etc/AppleVolumes`, který má úplně stejný formát a popisuje souborové systémy, které budou mít k dispozici uživatelé přihlašující se na účet *guest*.

Podrobnosti týkající se způsobu konfigurace těchto souborů a popis různých voleb najdete v manuálových stránkách programu `afpd`.

Toto je jednoduchý příklad:

```
/tmp Scratch
/home/ftp/pub "Public Area"
```

Tyto příkazy způsobí export vašeho souborového systému `/tmp` jako svazek AppleShare „Scratch“ a váš veřejný ftp-adresář jako svazek AppleShare „Public Area“. Názvy svazků nejsou povinné, démon si nějaké zvolí.

6.2.3 Sdílení linuxové tiskárny prostřednictvím protokolu Appletalk

Sdílení linuxové tiskárny na počítači Apple je poměrně jednoduché. Potřebujete spustit program `papd`, což je Appletalk Printer Access Protocol Daemon. Tento program bude po spuštění přijímat požadavky počítačů Apple a předávat tiskové úlohy vašemu démonu tiskárny k vytištění.

Konfigurace démona se provádí v souboru `/usr/local/atalk/etc/papd.conf`. Syntaxe tohoto souboru je stejná jako souboru `/etc/printcap`. Název, který přidělíte definici, je registrován pomocí názvového protokolu Appletalk, NBP.

Vzorový konfigurační soubor by mohl například obsahovat následující záznam:

```
TricWriter:\
:pr=lp:op=cg:
```

Ten by zpřístupnil tiskárnu jménem „TricWriter“ vaší síti Appletalk a všechny přijaté úlohy by byly vytištěny na linuxové tiskárně „`lp`“ (dle definice v souboru `/etc/printcap`) pomocí `lpd`. Zápis „`op=cg`“ říká, že operátorem tiskárny je uživatel „`cg`“.

6.2.4 Spuštění softwaru appletalk

Nyní bychom mohli otestovat základní konfiguraci. S balíkem `netatalk` je dodáván soubor `rc.atalk`, který by nám měl vyhovovat, takže stačí přidat jeho cestu do některého startovacího souboru

```
# /usr/local/atalk/etc/rc.atalk
```

a vše by mělo fungovat tak, jak má. Neměly by se objevit žádné chybové zprávy a software by měl posílat zprávy o svém stavu na konzolu.

6.2.5 Otestování softwaru appletalk

Chcete-li otestovat správnou funkci softwaru, vyvolejte na některém z počítačů menu Apple, zvolte položku Chooser, klepněte na AppleShare a mělo by se objevit okno Linux.

6.2.6 Výhrady proti softwaru appletalk

- Možná bude nutné spustit podporu Appletalk před konfigurací sítě IP. Máte-li problémy se spouštěním programů Appletalk nebo po jejich spuštění nefunguje IP-síť, zkuste spustit software Appletalk ještě před spuštěním souboru `/etc/rc.d/rc.inet1`.
- Démon `afpd` (Apple Filing Protocol Daemon) vážně zaneřádí váš pevný disk. Pod přípojnými body vytvoří dvojici adresářů nazvaných `.AppleDesktop` a `Network Trash Folder`. Potom pro každý adresář, do kterého vstoupíte vytvoří adresář .AppleDouble, aby do něj mohl uložit resource forks atd. Takže dříve než exportujete adresář `/`, si to pořádně rozmyslete, jinak budete mít spoustu práce s čištěním disku.
- Program `afpd` očekává od počítačů Apple hesla v čistě textové podobě. To může být vážný bezpečnostní problém, takže pokud budete spouštět tohoto démona na počítači připojenému k Internetu, můžete za případný průnik do systému vinit jen sami sebe.
- Existující diagnostické nástroje, jako je `netstat` a `ifconfig`, nepodporující Appletalk. Hrubé informace najdete v adresáři `/proc/net/`.

6.2.7 Více informací

Podrobnější popis způsobu konfigurace softwaru Appletalk pro Linux najdete na stránce *Linux Netatalk-HOWTO* Andrease Brownwothe na adrese *http://thehamptons.com/anders/netatalk/.*

6.3 ATM

Werner Almesberger **<werner.almesberger@lrc.di.epfl.ch>** vede projekt, který by umožnil podporu režimu ATM (Asynchronous Transfer Mode) v Linuxu. Aktuální informace o stavu tohoto projektu získáte na adrese *http://lrcwww.epfl.ch/linux-atm/.*

6.4 AX25 (AF_AX25)

Názvy zařízení protokolu AX.25 jsou ve verzi 2.0 „sl0“, „sl1“ atd. nebo „ax0“, „ax1“ atd. ve verzi jádra 2.1.*.

Volby kompilace jádra:

```
Networking options  --->
[*] Amateur Radio AX.25 Level 2
```

Protokol AX25, Netrom a Rose jsou popisovány v dokumentu *AX25-HOWTO* <AX25-HOWTO.html>. Tyto protokoly používají radioamatéři po celém světě k experimentům s paketovým rádiem.

Velký díl práce při implementaci těchto protokolů má na svědomí Jonathon Naylor, **jsn@cs.nott.ac.uk.**

6.5 DECNet

Na podpoře protokolu DECNet se v současné době pracuje. Předpokládá se, že se objeví ve verzi jádra č. 2.1.*.

6.6 EQL – multiple line traffic equaliser

Název zařízení protokolu EQL je „eq1“. Ve standardním zdrojovém kódu jádra můžete mít pouze jediné zařízení EQL na počítač. EQL poskytuje způsob využití více linek point to point, např. PPP, SLIP nebo PLIP, jako jediného logického spojení pro přenos protokolu TCP/IP. Vždy je levnější použít větší množství pomalejších linek, než mít nainstalovanou jedinou vysokorychlostní linku.

Volby kompilace jádra

```
Network device support  --->
[*] Network device support
<*> EQL (serial line load balancing) support
```

Aby mohl být tento mechanismus podporován, musí počítač na druhém konci spojení také podporovat EQL. Linux, Livingstone Portmasters a novější přístupové servery podporují kompatibilní prostředky.

Ke konfiguraci EQL budeme potřebovat nástroje `eql`, které lze získat na adrese *ftp://sunsite.unc.edu/pub/linux/system/Serial/eql-1.2.tar.gz.*

Konfigurace je poměrně jednoduchá. Začnete konfigurací rozhraní eql. Je to rozhraní jako každé jiné síťové rozhraní. Pomocí utility `ifconfig` nastavíte IP-adresu a mtu, například takto:

```
ifconfig eql 192.168.10.1 mtu 1006
```

Potom musíte ručně navázat všechna spojení, která budete používat. Může se jednat o kombinaci síťových služeb point to point. Způsob navázání spojení bude záviset na jejich typu. Podrobnosti najdete v příslušných statích tohoto dokumentu.

nakonec je potřeba sdružit sériové spojení se zařízením EQL, kterému se říká „enslaving" a provádí se příkazem `eql_enslave` níže uvedeným způsobem:

```
eql_enslave eql sl0 28800
eql_enslave eql ppp0 14400
```

Parametr „*estimated speed*", který předáte programu `eql_enslave`, sám o sobě nic neprovádí. Ovladač EQL podle něj určí, jaký typ sdílení datagramů má zařízení obdržet, takže pomocí této hodnoty můžete doladit zatížení linek.

Ke zrušení přiřazení mezi linkou a zařízením EQL slouží příkaz `eql_emancipate`, který se použije následujícím způsobem:

```
eql_emancipate eql sl0
```

Směrování přidáte stejně jako byste to provedli pro kterékoliv jiné spojení point to point, jen vaše trasy by spíše než na skutečná sériová zařízení měly odkazovat na zařízení eql. Typický zápis bude vypadat takto:

```
route add default eql
```

Ovladač EQL napsal Simon Janes, **simon@ncm.com**.

6.7 Ethernet

Názvy ethernetových zařízení jsou „eth0", „eth1", „eth2" atd. První detekované kartě je přiřazen název „eth0" a delším pak názvy v tom pořadí, v jakém jsou detekovány.

Jak rozchodit vaši ethernetovou kartu pod Linuxem se dozvíte v dokumentu *Ethernet-HOWTO*.

Jakmile máte sestavené jádro s podporou vaší ethernetové karty, je již vlastní konfigurace karty jednoduchá.

Typicky použijete následující zápis:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
# route add -net 192.168.0.0 netmask 255.255.255.0 eth0
```

Většinu ethernetových ovladačů vytvořil Donald Becker, **becker@CESDIS.gsfc.nasa.gov**.

6.8 FDDI

Názvy zařízení FFDI jsou „fddi0“, „fddi1“, „fddi2“ atd. První detekované kartě je přiřazen název „fddi0“ a delším pak postupně názvy v tom pořadí, v jakém jsou detekovány.

Larry Stefani, `lstefani@ultranet.com`, napsal ovladač pro karty Digital Equipment Corporation FDDI EISA a PCI.

Volby kompilace jádra

```
Network device support  --->
[*] FDDI driver support
[*] Digital DEFEA and DEFPA adapter support
```

Jakmile máte sestavené jádro s podporou ovladače FDDI, je vlastní konfigurace rozhraní FDDI takřka identická s konfigurací ethernetového rozhraní. Stačí zadat příslušný název rozhraní FDDI příkazům `ifconfig` a `route`.

6.9 Frame Relay

Názvy zařízení Frame Relay jsou „dlci00“, „dlci01“ atd pro DLCI-zařízení zapouzdření a „sdla0“, „sdla1“ atd. pro FRAD.

Frame Relay je nová síťová technologie, která je navržena pro datové komunikace, které mají impulzivní nebo střídavou povahu. K síti Frame Relay se připojíte pomocí zařízení zvaného Frame Relay Access Device (FRAD). Frame Relay pro Linux podporuje IP přes Frame Relay, dle popisu v dokumentu RFC-1490.

Volby kompilace jádra

```
Network device support  --->
<*> Frame relay DLCI support (EXPERIMENTAL)
(24)    Max open DLCI
(8)    Max DLCI per device
<*>    SDLA (Sangoma S502/S508) support
```

Autorem podpory Frame Relay a konfiguračních nástrojů je Mike McLagan, **mike.mclagan@linux.org.**

V současné době je FRAD podporován pouze firmou Sangoma Technologies <*http://www.sangoma.com/*> `S502A, S502E` a `S508`.

Ke konfiguraci zařízení FRAD a DLCI po sestavení jádra budete potřebovat konfigurační nástroje Frame Relay. Ty lze získat na adrese *ftp://ftp.invlogic.com/pub/linux/fr/frad-0.15.tgz*. Kompilace a instalace těchto nástrojů je poměrně jednoduchá, ale díky absenci souboru `Makefile` je to potřeba provést manuálně:

```
# cd /usr/src
# tar xvfz .../frad-0.15.tgz
# cd frad-0.15
# for i in common dlci frad; do make -C $i clean; make -C $i; done
# mkdir /etc/frad
# install -m 644 -o root -g root bin/*.sfm /etc/frad
# install -m 700 -o root -g root frad/fradcfg /sbin
# install -m 700 -o root -g root dlci/dlcicfg /sbin
```

Po nainstalování těchto nástrojů je třeba vytvořit soubor `/etc/frad/router.conf`. Můžete k tomu využít tuto šablonu, která je upravenou verzí jednoho z příkladů:

```
# /etc/frad/router.conf
# This is a template configuration for frame relay.
# All tags are included. The default values are based on the code
# supplied with the DOS drivers for the Sangoma S502A card.
#
# A '#' anywhere in a line constitutes a comment
# Blanks are ignored (you can indent with tabs too)
# Unknown [] entries and unknown keys are ignored
#

[Devices]
Count=1                    # number of devices to configure
Dev_1=sdla0                # the name of a device
#Dev_2=sdla1               # the name of a device

# Specified here, these are applied to all devices and can be
# overriden for
# each individual board.
#
```

```
Access=CPE
Clock=Internal
KBaud=64
Flags=TX
#
# MTU=1500          # Maximum transmit IFrame length, default is # #
                    # 4096
# T391=10           # T391 value     5 - 30, default is 10
# T392=15           # T392 value     5 - 30, default is 15
# N391=6            # N391 value     1 - 255, default is 6
# N392=3            # N392 value     1 - 10, default is 3
# N393=4            # N393 value     1 - 10, default is 4

# Specified here, these set the defaults for all boards
# CIRfwd=16         # CIR forward    1 - 64
# Bc_fwd=16         # Bc forward     1 - 512
# Be_fwd=0          # Be forward     0 - 511
# CIRbak=16         # CIR backward   1 - 64
# Bc_bak=16         # Bc backward    1 - 512
# Be_bak=0          # Be backward    0 - 511

# Device specific configuration
#

# The first device is a Sangoma S502E
#
[sdla0]
Type=Sangoma        # Type of the device to configure, currently
                    # only SANGOMA is recognised
#
# These keys are specific to the 'Sangoma' type
#
# The type of Sangoma board - S502A, S502E, S508
Board=S502E
#
# The name of the test firmware for the Sangoma board
# Testware=/usr/src/frad-0.10/bin/sdla_tst.502
#
```

```
# The name of the FR firmware
# Firmware=/usr/src/frad-0.10/bin/frm_rel.502
#
Port=360            # Port for this particular card
Mem=C8              # Address of memory window, A0-EE, depending
                    # on card
IRQ=5               # IRQ number, do not supply for S502A
DLCIs=1             # Number of DLCI's attached to this device
DLCI_1=16           # DLCI #1's number, 16 - 991
# DLCI_2=17
# DLCI_3=18
# DLCI_4=19
# DLCI_5=20
#
# Specified here, these apply to this device only,
# and override defaults from above
#
# Access=CPE        # CPE or NODE, default is CPE
# Flags=TXIgnore,RXIgnore,BufferFrames,DropAborted,Stats,MCI,AutoDLCI
# Clock=Internal    # External or Internal, default is Internal
# Baud=128          # Specified baud rate of attached CSU/DSU
# MTU=2048          # Maximum transmit IFrame length, default is # #
                    # 4096
# T391=10           # T391 value    5 - 30, default is 10
# T392=15           # T392 value    5 - 30, default is 15
# N391=6            # N391 value    1 - 255, default is 6
# N392=3            # N392 value    1 - 10, default is 3
# N393=4            # N393 value    1 - 10, default is 4

#
# The second device is some other card
#
# [sdla1]
# Type=FancyCard    # Type of the device to configure.
# Board=            # Type of Sangoma board
# Key=Value         # values specific to this type of device

#
```

```
# DLCI Default configuration parameters
# These may be overridden in the DLCI specific configurations
#
CIRfwd=64           # CIR forward    1 - 64
# Bc_fwd=16         # Bc forward     1 - 512
# Be_fwd=0          # Be forward     0 - 511
# CIRbak=16         # CIR backward   1 - 64
# Bc_bak=16         # Bc backward    1 - 512
# Be_bak=0          # Be backward    0 - 511

#
# DLCI Configuration
# These are all optional. The naming convention is
# [DLCI_D<devicenum>_<DLCI_Num>]
#

[DLCI_D1_16]
# IP=
# Net=
# Mask=
# Flags defined by Sangoma: TXIgnore,RXIgnore,BufferFrames
# DLCIFlags=TXIgnore,RXIgnore,BufferFrames
# CIRfwd=64
# Bc_fwd=512
# Be_fwd=0
# CIRbak=64
# Bc_bak=512
# Be_bak=0

[DLCI_D2_16]
# IP=
# Net=
# Mask=
# Flags defined by Sangoma: TXIgnore,RXIgnore,BufferFrames
# DLCIFlags=TXIgnore,RXIgnore,BufferFrames
# CIRfwd=16
# Bc_fwd=16
# Be_fwd=0
```

```
# CIRbak=16
# Bc_bak=16
# Be_bak=0
```

Po sestavení souboru `/etc/frad/router.conf` zbývá nakonfigurovat vlastní zařízení. Pouze tato část je o něco obtížnější, než konfigurace normálních síťových zařízení. Nesmíte zapomenout předložit zařízení FRAD ještě před zapouzdřenými zařízeními DLCI.

```
# Configure the frad hardware and the DLCI parameters
/sbin/fradcfg /etc/frad/router.conf || exit 1
/sbin/dlcicfg file /etc/frad/router.conf
#
# Bring up the FRAD device
ifconfig sdla0 up
#
# Configure the DLCI encapsulation interfaces and routing
ifconfig dlci00 192.168.10.1 pointopoint 192.168.10.2 up
route add -net 192.168.10.0 netmask 255.255.255.0 dlci00
#
ifconfig dlci01 192.168.11.1 pointopoint 192.168.11.2 up
route add -net 192.168.11.0 netmask 255.255.255.0 dlci00
#
route add default dev dlci00
#
```

6.10 IP-účetnictví (accounting)

IP-účetnictví jádra Linuxu vám umožňuje shromažďovat a analyzovat některá data týkající se využití sítě. Mezi sesbíraná data patří počet paketů a počet bajtů nahromaděných od posledního vynulování součtů. Ke kategorizaci součtů lze zadat množství nejrůznějších pravidel tak, aby výsledek vyhovoval vašim potřebám.

Volby kompilace jádra

```
Networking options  --->
[*] IP: accounting
```

Po kompilaci a instalaci jádra je třeba pomocí příkazu `ipfwadm` nastavit IP-účetnictví. Existuje mnoho různých způsobů specifikací účetnictví, z nichž můžete vybírat. Vybral jsem jednoduchý příklad zahrnující to podstatné. Další informace najdete v manuálových stránkách příkazu `ipfwadm`.

Scénář: Máte ethernetovou síť, která je připojena k Internetu prostřednictvím spojení PPP. Na Ethernetu máte počítač, který nabízí několik služeb, a zajímá vás, jaký provoz způsobuje telnet, rlogin, FTP a WWW.

Můžete použít následující sadu příkazů:

```
#
# Vyprázdnit účetní pravidla
ipfwadm -A -f
#
# Pravidla pro lokální ethernetový segment
ipfwadm -A in -a -P tcp -D 44.136.8.96/29 20
ipfwadm -A out -a -P tcp -S 44.136.8.96/29 20
ipfwadm -A in -a -P tcp -D 44.136.8.96/29 23
ipfwadm -A out -a -P tcp -S 44.136.8.96/29 23
ipfwadm -A in -a -P tcp -D 44.136.8.96/29 80
ipfwadm -A out -a -P tcp -S 44.136.8.96/29 80
ipfwadm -A in -a -P tcp -D 44.136.8.96/29 513
ipfwadm -A out -a -P tcp -S 44.136.8.96/29 513
ipfwadm -A in -a -P tcp -D 44.136.8.96/29
ipfwadm -A out -a -P tcp -D 44.136.8.96/29
ipfwadm -A in -a -P udp -D 44.136.8.96/29
ipfwadm -A out -a -P udp  -D 44.136.8.96/29
ipfwadm -A in -a -P icmp -D 44.136.8.96/29
ipfwadm -A out -a -P icmp -D 44.136.8.96/29
#
# Pravidla pro ostatní
ipfwadm -A in -a -P tcp -D 0/0 20
ipfwadm -A out -a -P tcp -S 0/0 20
ipfwadm -A in -a -P tcp -D 0/0 23
ipfwadm -A out -a -P tcp -S 0/0 23
ipfwadm -A in -a -P tcp -D 0/0 80
ipfwadm -A out -a -P tcp -S 0/0 80
ipfwadm -A in -a -P tcp -D 0/0 513
```

```
ipfwadm -A out -a -P tcp -S 0/0 513
ipfwadm -A in -a -P tcp -D 0/0
ipfwadm -A out -a -P tcp -D 0/0
ipfwadm -A in -a -P udp -D 0/0
ipfwadm -A out -a -P udp  -D 0/0
ipfwadm -A in -a -P icmp -D 0/0
ipfwadm -A out -a -P icmp -D 0/0
#
# Výpis pravidel
ipfwadm -A -l -n
#
```

Poslední příkaz vypíše všechna pravidla účetnictví a zobrazí shromážděné souhrny.

Je důležité vědět, že při analýze IP-účetnictví se budou **celkové výsledky všech vyhovujících pravidel zvyšovat**, takže k získání různých součtů je třeba příslušná matematika. Pokud bych chtěl například vědět, jakou část tvořila data, která nepatří ke službě FTP, telnet, rlogin nebo WWW, musel bych odečíst jednotlivé součty od pravidla, které vyhovuje všem portům.

```
# ipfwadm -A -l -n
  IP accounting rules
pkts   bytes  dir   prot   source          destination       ports
   0       0  in    tcp    0.0.0.0/0       44.136.8.96/29    * -> 20
   0       0  out   tcp    44.136.8.96/29  0.0.0.0/0         20 -> *
   0       0  in    tcp    0.0.0.0/0       44.136.8.96/29    * -> 23
   0       0  out   tcp    44.136.8.96/29  0.0.0.0/0         23 -> *
  10    1166  in    tcp    0.0.0.0/0       44.136.8.96/29    * -> 80
  10     572  out   tcp    44.136.8.96/29  0.0.0.0/0         80 -> *
 242    9777  in    tcp    0.0.0.0/0       44.136.8.96/29    * -> 513
 220   18198  out   tcp    44.136.8.96/29  0.0.0.0/0         513 -> *
 252   10943  in    tcp    0.0.0.0/0       44.136.8.96/29    * -> *
 231   18831  out   tcp    0.0.0.0/0       44.136.8.96/29    * -> *
   0       0  in    udp    0.0.0.0/0       44.136.8.96/29    * -> *
   0       0  out   udp    0.0.0.0/0       44.136.8.96/29    * -> *
   0       0  in    icmp   0.0.0.0/0       44.136.8.96/29    *
   0       0  out   icmp   0.0.0.0/0       44.136.8.96/29    *
   0       0  in    tcp    0.0.0.0/0       0.0.0.0/0         * -> 20
   0       0  ou    tcp    0.0.0.0/0       0.0.0.0/0         20 -> *
   0       0  in    tcp    0.0.0.0/0       0.0.0.0/0         * -> 23
```

```
   0       0  out   tcp    0.0.0.0/0       0.0.0.0/0          23 -> *
  10    1166  in    tcp    0.0.0.0/0       0.0.0.0/0          * -> 80
  10     572  out   tcp    0.0.0.0/0       0.0.0.0/0          80 -> *
 243    9817  in    tcp    0.0.0.0/0       0.0.0.0/0          * -> 513
 221   18259  out   tcp    0.0.0.0/0       0.0.0.0/0          513 -> *
 253   10983  in    tcp    0.0.0.0/0       0.0.0.0/0          * -> *
 231   18831  out   tcp    0.0.0.0/0       0.0.0.0/0          * -> *
   0       0  in    udp    0.0.0.0/0       0.0.0.0/0          * -> *
   0       0  out   udp    0.0.0.0/0       0.0.0.0/0          * -> *
   0       0  in    icmp   0.0.0.0/0       0.0.0.0/0          *
   0       0  out   icmp   0.0.0.0/0       0.0.0.0/0          *
#
```

6.11 Přidělování IP-přezdívek

U některých aplikací je výhodné přidělit jednomu síťovému zařízení více IP-adres. Poskytovatelé internetových služeb využívají tuto vlastnost k poskytování zákaznických nabídek služeb World Wide Web a FTP.

Volby kompilace jádra

```
Networking options  --->
      ....
      [*] Network aliasing
      ....
      <*> IP: aliasing support
```

Po kompilaci a instalaci jádra s podporou IP_Alias je vlastní konfigurace velice jednoduchá. Přezdívky (aliasy) jsou přidány k virtuálním síťovým zařízením sdruženým se skutečným síťovým zařízením. Jednoduchá názvová konvence pak vypadá třeba takto: <název zařízení>:<číslo virtuálního zařízení>, tj. eth0:0, ppp0:10 atd.

Předpokládejme například, že máte ethernetovou síť, která současně podporuje dvě různé podsítě IP, a vy chcete, aby měl váš počítač přímý přístup k oběma. V takovém případě můžete použít tento zápis:

```
#
# ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
```

```
# ifconfig eth0:0 192.168.10.1 netmask 255.255.255.0 up
# route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0
#
```

Budete-li chtít odstranit nějakou přezdívku, stačí přidat na konec jejího názvu pomlčku, jako v následujícím výpisu:

```
# ifconfig eth0:0- 0
```

Všechny trasy sdružené s tímto aliasem budou také automaticky odstraněny.

6.12 IP-firewall

IP-firewall a problémy kolem firewallů jsou podrobněji probírány v dokumentu *Firewall-HOWTO*. IP-firewally chrání váš počítač před neautorizovaným síťovým přístupem, což provádějí filtrováním nebo propouštěním datagramů z a na vámi nominované IP-adresy. Existují tři různé třídy pravidel: příchozí filtrování, odchozí filtrování a postupující filtrování. Příchozí pravidla jsou aplikována na datagramy, které obdrží síťové zařízení. Odchozí pravidla jsou aplikována na datagramy, které mají být síťovým zařízením přenesena. Postupující pravidla jsou aplikována na datagramy, které zařízení obdrží, ale nejsou určeny pro daný počítač, tj. datagramy, které je třeba nasměrovat.

Volby kompilace jádra

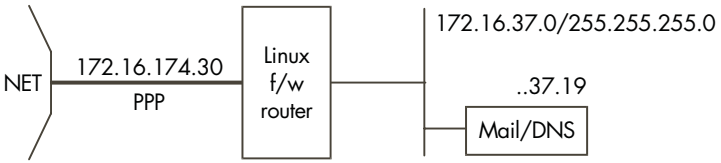
```
Networking options  --->
       [*] Network firewalls
       ....
       [*] IP: forwarding/gatewaying
       ....
       [*] IP: firewalling
       [ ] IP: firewall packet logging
```

Konfigurace pravidel IP-firewallu se provádí pomocí příkazu `ipfwadm`. Jak jsem již uvedl, nepatřím v otázkách bezpečnosti mezi experty, takže ačkoli vám zde představím jednoduchý příklad, který můžete použít, je-li pro vás bezpečnost prvořadá, měli byste zkoumat sami na vlastní pěst a vytvořit si svá vlastní pravidla.

Pravděpodobně nejběžnějším využitím IP-firewallu je případ, kdy používáte váš linuxový počítač jako router a firewallovou bránu k ochraně vaší lokální počítačové sítě před neautorizovaným přístupem z vnějšku.

Následující konfigurace vychází z příspěvku, jehož autorem je Arnt Gulbrandsen, **<agulbra@troll.no>**.

Příklad popisuje konfiguraci pravidel firwallu na počítači-firewallu/bráně ilustrovaném na tomto diagramu:



Následující příkazy by byly normálně umístěny v souboru `rc`, aby došlo k jejich automatickému spuštění při startu systému. Pro zajištění maximální bezpečnosti by měly být tyto příkazy provedeny až po konfiguraci síťových rozhraní, ale před vlastním spuštěním těchto zařízení, aby nemohl nikdo získat přístup v době, kdy počítač startuje.

```
#!/bin/sh

# Flush the 'Forwarding' rules table
# Change the default policy to 'accept'
#
/sbin/ipfwadm -F -f
/sbin/ipfwadm -F -p accept
#
# .. and for 'Incoming'
#
/sbin/ipfwadm -I -f
/sbin/ipfwadm -I -p accept

# First off, seal off the PPP interface
# I'd love to use '-a deny' instead of '-a reject -y' but then it
# would be impossible to originate connections on that interface
# too.
# The -o causes all rejected datagrams to be logged. This trades
# disk space against knowledge of an attack of configuration error.
#
/sbin/ipfwadm -I -a reject -y -o -P tcp -S 0/0 -D 172.16.174.30

# Throw away certain kinds of obviously forged packets right away:
```

```
# Nothing should come from multicast/anycast/broadcast addresses
#
/sbin/ipfwadm -F -a deny -o -S 224.0/3 -D 172.16.37.0/24
#
# and nothing coming from the loopback network should ever be
# seen on a wire
#
/sbin/ipfwadm -F -a deny -o -S 127.0/8 -D 172.16.37.0/24

# accept incoming SMTP and DNS connections, but only
# to the Mail/Name Server
#
/sbin/ipfwadm -F -a accept -P tcp -S 0/0 -D 172.16.37.19 25 53
#
# DNS uses UDP as well as TCP, so allow that too
# for questions to our name server
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 -D 172.16.37.19 53
#
# but not "answers" coming to dangerous ports like NFS and
# Larry McVoy's NFS extension.  If you run squid, add its port here.
#
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 53 \
              -D 172.16.37.0/24 2049 2050

# answers to other user ports are okay
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 53 \
              -D 172.16.37.0/24 53 1024:65535

# Reject incoming connections to identd
# We use 'reject' here so that the connecting host is told
# straight away not to bother continuing, otherwise we'd experience
# delays while ident timed out.
#
/sbin/ipfwadm -F -a reject -o -P tcp -S 0/0 -D 172.16.37.0/24 113

# Accept some common service connections from the 192.168.64 and
```

```
# 192.168.65 networks, they are friends that we trust.
#
/sbin/ipfwadm -F -a accept -P tcp -S 192.168.64.0/23 \
              -D 172.16.37.0/24 20:23

# accept and pass through anything originating inside
#
/sbin/ipfwadm -F -a accept -P tcp -S 172.16.37.0/24 -D 0/0

# deny most other incoming TCP connections and log them
# (append 1:1023 if you have problems with ftp not working)
#
/sbin/ipfwadm -F -a deny -o -y -P tcp -S 0/0 -D 172.16.37.0/24

# ... for UDP too
#
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 -D 172.16.37.0/24
```

Správná konfigurace firewallu je poměrně obtížná. Tento příklad by vám měl posloužit jako odrazový můstek. Manuálové stránky příkazu `ipfwadm` nabízí jistou pomoc při používání tohoto nástroje. Budete-li konfigurovat firewall, poptejte se kolem a snažte se získat co nejvíce rad ze zdrojů, které považujete za spolehlivé, a požádejte někoho, aby rozumně otestoval vaši konfiguraci z vnějšku.

6.13 IPIP-zapouzdření

Proč byste chtěli zapouzdřovat IP-datagramy do IP-datagramů? Pokud jste o tom zatím neslyšeli, bude vám to asi připadat zvláštní. Zde je tedy dvojice využití: Mobile-IP a IP-Multicast. Pravděpodobně nejrozšířenější využití má tato technika v nejméně známé oblasti, kterou je amatérské rádio.

Volby kompilace jádra

```
Networking options  --->
       [*] TCP/IP networking
       [*] IP: forwarding/gatewaying
       ....
       <*> IP: tunneling
```

Zařízení provádějící IP-tunelování se nazývají „tunl0“, „tunl1“ atd.

„Ale proč?” No protože konvenční IP-směrování nařizuje, že síťová IP-adresa se skládá ze síťové adresy a síťové masky. Takto vzniká množství po sobě jdoucích adres, které mohou být všechny směrovány přes jeden směrovací záznam. To je poměrně výhodné, ale znamená to, že všechny konkrétní IP-adresy můžete používat jen v případě, kdy jste připojeni ke konkrétní části sítě, které tyto adresy náleží. Ve většině případů je to v pořádku, ale patříte-li k mobilním uživatelům sítě, pak nemůžete zůstat po celou dobu připojen k jednomu místu. IPIP zapouzdření (IP-tunelování) dovoluje toto omezení překonat tím, že dovolí IP-datagramy určené pro vaši IP-adresu zabalit a přesměrovat na jinou IP-adresu. Víte-li dopředu, že budete nějakou dobu pracovat v jiné IP-síti, můžete nastavit počítač ve vaší domovské síti tak, aby přijímal datagramy určené pro vaši IP-adresu a přesměroval je na adresu, kterou budete ve skutečnosti dočasně používat.

6.13.1 Konfigurace tunelované sítě

Věřím, že diagram mi jako vždy ušetří spoustu matoucího textu, takže tady je:

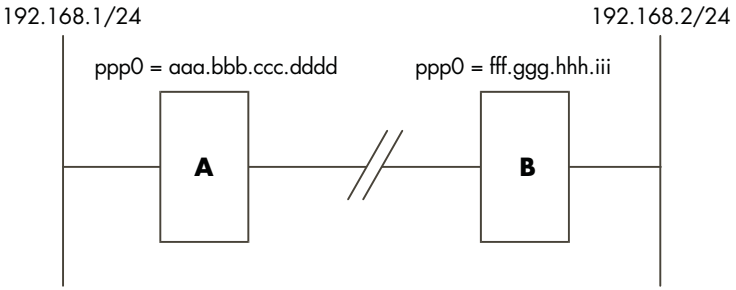


Diagram ilustruje další možný důvod použití IPIP zapouzdření, kterým je virtuální soukromá síť (VPN). Tento příklad předpokládá, že máte dva počítače, každý s jedním vytáčeným internetovým připojením. Každému hostiteli je přidělena právě jedna IP-adresa. Za těmito počítači je určitá soukromá lokální počítačová síť s rezervovanými síťovými IP-adresami. Dejme tomu, že chcete libovolnému uživateli v síti A povolit připojení k libovolnému hostiteli v síti B, stejně jako by byli řádně připojeni k Internetu prostřednictvím síťového routeru. Právě tento stav nám pomůže dosáhnou IPIP zapouzdření. Všimněte si, že zapouzdření neřeší problém ohledně způsobu, jakým hostitelé v sítích A a B komunikují s kterýmkoliv jiným hostitelem v Internetu. Stále k tomu budete potřebovat triky typu IP-maškaráda (Masquerade). Zapouzdřování normálně provádí počítač fungující jako router.

Linuxový router „A“ by byl nakonfigurován takto:

```
#
PATH=/sbin:/usr/sbin
```

```
#
# Ethernet configuration
ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.1 up
route add -net 192.168.2.0 netmask 255.255.255.0 gw fff.ggg.hhh.iii tunl0
```

Linuxový router „B“ by byl nakonfigurován takto:

```
#
PATH=/sbin:/usr/sbin
#
# Ethernet configuration
ifconfig eth0 192.168.2.1 netmask 255.255.255.0 up
route add -net 192.168.2.0 netmask 255.255.255.0 eth0
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.2.1 up
route add -net 192.168.1.0 netmask 255.255.255.0 gw aaa.bbb.ccc.ddd tunl0
```

Příkaz

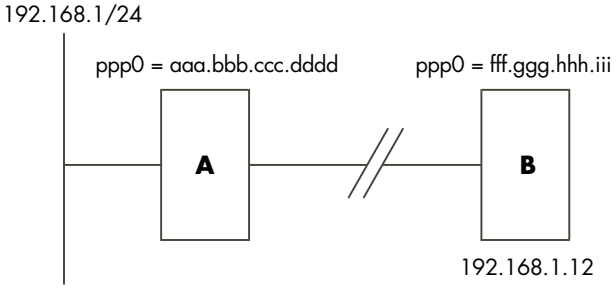
```
route add -net 192.168.1.0 netmask 255.255.255.0 gw aaa.bbb.ccc.ddd tunl0
```

říká: „Všechny datagramy určené pro adresu `192.168.1.0/24` pošli do IPIP zapouzdřeného datagramu s cílovou adresou `aaa.bbb.ccc.ddd`.“

Všimněte si, že na sebe obě konfigurace působí. Tunelovací zařízení používá ‚gw“ v trase jako cíl IP-datagramu, kam pošle datagram, který obdrželo pro směrování. Tento počítač musí umět zapouzdřovat IPIP datagramy, tj. musí být také nakonfigurován s tunelovacím zařízením.

6.13.2 Konfigurace tunelovacího hostitele

Směrování nemusí pokrývat celou síť. Můžete třeba směrovat pouze jedinou IP-adresu. V takovém případě stačí nastavit zařízení tun1 na „vzdáleném“ počítači s jeho IP-adresou a na konci A použít hostitelský router (a Proxy Arp) a nikoli síťový router přes tunelovací zařízení. Pojďme tedy naši konfiguraci příslušným způsobem upravit. Nyní máme pouze hostitele „B“, u kterého chceme, aby byl připojen k Internetu a zároveň byl součástí vzdálené sítě podporované hostitelem „A“:



Linuxový router „A“ by byl nakonfigurován takto:

```
#
PATH=/sbin:/usr/sbin
#
# Ethernet configuration
ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.1 up
route add -host 192.168.1.12 gw fff.ggg.hhh.iii tunl0
#
# Proxy ARP for the remote host
arp -s 192.168.1.12 xx:xx:xx:xx:xx:xx pub
```

Linuxový router „B“ by byl nakonfigurován takto:

```
#
PATH=/sbin:/usr/sbin
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.12 up
route add -net 192.168.1.0 netmask 255.255.255.0 \
  gw aaa.bbb.ccc.ddd tunl0
```

Tento druh konfigurace je typičtější pro aplikace Mobile-IP, kdy se chce jeden hostitel potulovat po Internetu a přitom si po celou dobu zachovat jedinou použitelnou IP-adresu. Zajímá-li vás praktická stránka tohoto tématu, pak si přečtěte stať Mobile-IP, kde najdete více informací.

6.14 IPX (AF_IPX)

Protokol IPX je častěji využíván v lokálních počítačových sítích Novell NetWare(tm). Linux obsahuje podporu tohoto protokolu a lze ho nakonfigurovat tak, aby se choval jako síťový koncový bod nebo router IPX.

Volby kompilace jádra

```
Networking options  --->
       [*] The IPX protocol
       [ ] Full internal IPX network
```

Protokol IPX a NCPFS jsou podrobněji rozebírány v dokumentu *IPX-HOWTO*.

6.15 IPv6

Když už máte pocit, že začínáte rozumět protokolu IP, změní se síťová pravidla. IPv6 je zkratka 6. verze internetového protokolu (IP). Tento protokol byl vyvinut zejména k vyřešení nedostatku IP-adres v Internetu. Adresy protokolu IPv6 jsou dlouhé 16 bajtů (128 bitů). Protokol IPv6 obsahuje také několik dalších změn, většinou se jedná o zjednodušení, což přispěje k lepší ovladatelnosti sítí IPv6 ve srovnání se sítěmi IPv4.

V jádru Linuxu `2.1.*` již je zabudována fungující, nikoli však úplná, implementace protokolu IPv6.

Pokud si chcete zaexperimentovat s internetovou technologií příští generace nebo ji už potřebujete, měli byste si přečíst dokument IPv6-FAQ, který najdete na adrese *http://www.terra.net/ipv6/*.

6.16 ISDN

ISDN (Intergrated Services Digital Network) je sada standardů, které specifikují přepínané digitální datové sítě pro obecné použití. „Volání" ISDN vytvoří synchronní datovou službu point to point směrem k cíli. ISDN je obecně posílán po vysokorychlostních linkách, které jsou rozděleny na několik oddělených kanálů. Rozeznáváme dva různé typy kanálů, tzv. „B kanály" slouží k vlastnímu přenosu dat a jediný „D kanál", který slouží k posílání řídicích informací službě ISDN o navázání spojení a dalších funkcí. V Austrálii například může být ISDN doručován po lince o rychlosti 2 Mbps, která je rozdělena na 30 samostatných B kanálů o rychlosti 64 kbps a jeden D kanál o rychlosti 64 kbps. Současně může být využíván libovolný počet a kombinace kanálů. Je možné například navázat 30 samostatných hovorů o rychlosti 64 kbps s 30 různými místy nebo 15 hovorů s 15 různými místy o rychlosti 128 kbps (každý hovor bude využívat dva kanály) případně jen malý počet hovorů a zbytek kanálů nechat zahálet. Kanál může být využíván buď pro příchozí, nebo odchozí hovor. Původním záměrem ISDN bylo umožnit telekomunikačním společnostem poskytovat jedinou datovou službu, která by umožnila doručování buď telefonních (prostřednictvím digitalizace hlasu), nebo datových služeb do domácností nebo firem bez nutnosti provádění zvláštních změn v konfiguraci.

Existují dva různé způsoby, jakými se může váš počítač připojit ke službě ISDN. První způsob spočívá ve využití zařízení zvaného „Terminal Adaptor" (terminálový adaptér), který se zapojí do jednoho ze sériových rozhraní Network Terminating Unit, kterou vám nainstalovali pracovníci telekomunikací při zavádění služby ISDN. Jedno z těchto rozhraní slouží k zadávání příkazů pro navázání hovorů a konfiguraci a ostatní jsou skutečně připojeny k síťovým zařízením, které budou využívat datové obvody po jejich vytvoření. Linux bude v této konfiguraci fungovat bez jakýchkoliv úprav. S portem na terminálovém adaptéru se zachází stejným způsobem, jako s kterýmkoliv jiným sériovým zařízením. Jiný způsob, pro který je navržena podpora jádra ISDN, spočívá v nainstalování karty ISDN do linuxového počítače a v nechání softwaru obsluhovat protokoly a provádět vlastní hovory.

Volby kompilace jádra

```
ISDN subsystem  --->
<      *> ISDN support
       [ ] Support synchronous PPP
```

```
[ ] Support audio via ISDN
< > ICN 2B and 4B support
< > PCBIT-D support
< > Teles/NICCY1016PC/Creatix support
```

Linuxová implementace ISDN podporuje několik různých typů interních karet ISDN.

- ICN 2B a 4B
- Octal PCBIT-D
- Karty Teles ISDN a kompatibilní

Některé z těchto karet vyžadují pro správnou funkci natažení softwaru. K tomu slouží samostatná utilita.

Všechny podrobnosti o konfiguraci linuxové podpory ISDN najdete v adresáři /usr/src/linux/Documentation/isdn/ a v dokumentu FAQ věnovanému programu *isdn4linux*, který je dostupný na adrese *http://www.lrz-muenchen.de/~ui161ab/www/isdn/*. (Anglickou verzi tohoto dokumentu získáte klepnutím na anglickou vlajku.)

Poznámka o PPP: Sada protokolů PPP bude fungovat po asynchronních i synchronních sériových linkách. Běžně dodávaný démon protokolu PPP pro Linux s názvem „pppd“ podporuje pouze asynchronní režim. Pokud hodláte provozovat protokoly PPP přes službu ISDN, potřebujete speciálně upravenou verzi. Podrobnosti ohledně jejího získání najdete ve výše uvedené dokumentaci.

6.17 IP-Masquerade (Maškaráda)

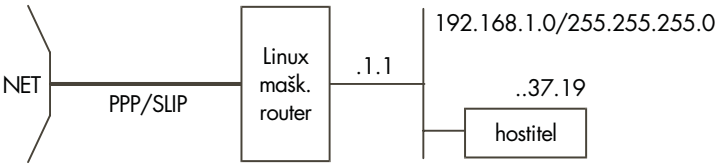
Spousta lidí vlastní jednoduchý vytáčený účet pro připojení k Internetu. Téměř každému, kdo používá tuto konfiguraci, přidělil poskytovatel internetových služeb (Internet Service Provider) jednu IP-adresu. Za normálních okolností to stačí ke zprostředkování plného přístupu k síti pouze jednomu hostiteli. IP-Masquerade je trik, díky němuž může jedinou IP-adresu využívat více počítačů, protože vypadají jako jiní hostitelé, proto se používá termín maškaráda. Nevýhodou je, že tato maškaráda funguje skoro vždy jen jedním směrem, to znamená, že maškarádovaní hostitelé mohou volat ven, ale nemohou přijímat síťová spojení od vzdálených hostitelů. To znamená, že zde nefungují některé síťové služby, například *talk* a další, třeba *ftp*, je třeba nastavit na pasivní režim (PASV). Naštěstí nejpoužívanější síťové služby, jako je *telnet*, World Wide Web a *irc* fungují dobře.

Volby kompilace jádra

```
Code maturity level options  --->
       [*] Prompt for development and/or incomplete code/drivers
Networking options  --->
       [*] Network firewalls
       ....
       [*] TCP/IP networking
       [*] IP: forwarding/gatewaying
       ....
       [*] IP: masquerading (EXPERIMENTAL)
```

Normálně byste svůj linuxový počítač podporující protokol SLIP nebo PPP nechali vytáčet stejně, jako by se jednalo o samostatný počítač Kromě toho by měl nastaveno další síťové zařízení, možná Ethernet, s jednou z rezervovaných síťových adres. Hostitelé, kteří mají být maskováni, by byli na této druhé síti. Každý z těchto hostitelů by měl nastavenu IP-adresu ethernetového portu linuxového počítače jako implicitní bránu nebo router.

Typická konfigurace by mohla vypadat asi takto:



Nejdůležitější příkazy této konfigurace jsou:

```
# Network route for ethernet
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
# Default route to the rest of the internet.
route add default ppp0
#
# Cause all hosts on the 192.168.1/24 network to be masqueraded.
ipfwadm -F -a m -S 192.168.1.0/24 -D 0.0.0.0/0
```

Více informací o IP-maškarádě pod Linuxem získáte na stránce *IP-Masquerade Resource Page http://www.hwy401.com/achau/ipmasq/.*

6.18 IP-transparentní proxy

IP-transparentní proxy je rys, který umožňuje přesměrovat servery nebo služby určené pro jiný počítač na tyto služby na daném počítači. Typicky se používá v případě linuxového počítače pracujícího jako router a zároveň i proxy server. Všechna spojení určená pro tuto službu byste měli vzdáleně přesměrovat na lokální proxy server.

Volby kompilace jádra

```
Code maturity level options  --->
      [*] Prompt for development and/or incomplete code/drivers
Networking options  --->
      [*] Network firewalls
      ....
      [*] TCP/IP networking
      ....
      [*] IP: firewalling
      ....
      [*] IP: transparent proxy support (EXPERIMENTAL)
```

Konfigurace transparetního proxy se provádí pomocí příkazy `ipfwadm`.

Toto je jeden z užitečných příkladů:

```
ipfwadm -I -a accept -D 0/0 telnet -r 2323
```

Tento příklad způsobí přesměrování všech pokusů o spojení libovolného hostitele s portem 23 (telnet) na port 2323 na tomto hostiteli. Takto je možné zajistit automatické přesměrování veškerého telnetového provozu z vaší sítě na lokální port.

6.19 Mobilní IP

Termín „IP-mobilita“ popisuje schopnost hostitele, který umí přesunout své síťové spojení z jednoho místa na Internetu do jiného bez změny své IP-adresy nebo ztráty konektivity. Když IP-hostitel změní místo svého připojení, musí zpravidla změnit také svoji IP-adresu. Mobilita tento problém překonává alokováním pevné IP-adresy mobilnímu hostiteli a používáním IP-zapouzdření (tunelování) s automatickým směrováním, čímž se zajistí, že datagramy určené pro tohoto hostitele jsou směrovány na skutečnou IP-adresu, kterou právě používá.

V současné době se pracuje na projektu, jehož cílem je kompletní sada IP-mobilních nástrojů pro Linux. Stav tohoto projektu a nástrojů zjistíte na stránce *Linux Mobile IP Home Page http://anchor.cs.binghamton.edu/~mobileip/.*

6.20 Multicast

IP-Multicast umožňuje simultánní směrování IP-datagramů na libovolný počet IP-hostitelů ve vzájemně neslučitelných IP-sítích. Tento mechanismus je využíván k šíření materiálů „broadcast" po Internetu, například audio a video přenosů a dalších nových aplikací.

Volby kompilace jádra

```
Networking options  --->
       [*] TCP/IP networking
       ....
       [*] IP: multicasting
```

Je nutná sadů nástrojů a některé malé úpravy sítě. Zdroj informací týkajících se způsobu instalace a konfigurace pro Linux najdete na adrese *www.teksouth.com <http://www.teksouth.com/linux/multicast/>*.

6.21 NAT – Network Address Translation

IP Network Address Translation (převod síťových adres) je více než standardizovaným bratrem linuxové IP-maškarády. Podrobně je tato vlastnost popisována v dokumentu RFC-1631. NAT disponuje vlastnostmi, které IP-maškarádě chybí, takže je neobyčejně vhodná pro použití ve firewallových routerech a velkých instalacích.

Jádro alfa-verze NAT pro Linux 2.0.29 vytvořil Michael Hasenstein, **Michael.Hasenstein@informatik.tu-chemnitz.de**. Jeho dokumentaci a implementaci lze získat na stránce:

Linux IP Network Address Web Page http://www.csn.tu-chemnitz.de/HyperNews/get/linux-ip-nat.html.

Novější jádra Linuxu 2.1.* obsahují také některé funkce NAT ve směrovacím algoritmu.

6.22 NetRom (AF_NETROM)

Názvy zařízení NetRom jsou „nr0", „nr1" atd.

Volby kompilace jádra

```
Networking options  --->
       [*] Amateur Radio AX.25 Level 2
       [*] Amateur Radio NET/ROM
```

Protokoly AX25, NetRom a Rose jsou popsány v dokumentu *AX25-HOWTO*. Tyto protokoly používají amatérští rádiooperátoři při experimentech s paketovým rádiem.

Většinu práce na implementaci těchto protokolů má na svědomí Jonathon Naylor, **jsn@cs.nott.ac.uk.**

6.23 PLIP

Názvy zařízení PLIP jsou „plip0“, „plip1“ atd.

Volby kompilace jádra

```
Networking options  --->
       <*> PLIP (parallel port) support
```

Protokol *PLIP* (Parralel Line IP) je podobný protokolu SLIP v tom, že slouží k poskytování síťových spojení typu *point to point* mezi dvěma počítači. ale je navržen pro paralelní tiskový port místo portu sériového (schéma zapojení najdete v příslušné stati na konci dokumentu). Protože pomocí paralelního portu je možné přenášet více než jeden bit současně, dosahuje se u rozhraní *PLIP* vyšších rychlostí než u standardního sériového zařízení. Kromě toho i ten nejjednodušší paralelní port lze použít místo poměrně drahého rozhraní 16550AFN UART pro sériové porty. Protokol PLIP využívá v porovnání se spojením přes sériový port velkou část CPU a pokud můžete získat nějaké levné ethernetové karty, není dobrou volbou. Pokud však nemáte k dispozici nic jiného, můžete se na něj spolehnout. Při dobrém spojení můžete očekávat přenosové rychlosti kolem 20 kilobajtů za sekundu.

Ovladače zařízení protokolu PLIP soutěží s ovladačem paralelních zařízení o paralelní port. Pokud chcete používat oba tyto ovladače, pak byste je měli přeložit jako moduly, abyste mohli volit, který port chcete pro PLIP a které porty pro ovladač tiskárny. Podrobnější informace najdete v dokumentu *Modules-HOWTO* `<Modules-HOWTO.html>`.

Některé přenosné počítače používají chipsety, které nebudou s protokolem PLIP fungovat, protože neumožňují některé kombinace signálů, na které tento protokol spoléhá a tiskárny je nepoužívají.

Linuxové rozhraní *PLIP* je kompatibilní s *Crynwyr Packet Driver PLIP*, což znamená, že prostřednictvím *PLIP* můžete spojit váš linuxový počítač s dosovým počítačem, na kterém běží kterýkoliv jiný druh softwaru TCP/IP.

V jádru verzí 2.0.* jsou zařízením PLIP přiděleny následující V/V porty a IRQ:

Zařízení	V/V	IRQ
plip0	0x3bc	5
plip1	0x378	7
plip2	0x278	2

Pokud vašemu paralelnímu portu žádná z těchto kombinací nevyhovuje, můžete změnit IRQ portu pomocí příkazu `ifconfig`, jemuž předáte parametr „irq“. Pokud to váš BIOS umožňuje, nezapomeňte na tiskových portech povolit příslušná IRQ.

V pozdějších verzích jádra 2.1.* s podporou Plug‘n‘Play jsou zařízení PLIP alokována postupně v tom pořadí, v jakém jsou detekována, podobně jako ethernetová zařízení. První alokované zařízení je označeno plip0.

Při kompilaci jádra je třeba upravit pouze jediný soubor. Jmenuje se `/usr/src/linux/driver/net/CONFIG` a obsahuje časovače *PLIP* v milisekundách. Implicitní hodnoty jsou ve většině případů vyhovující. Zvýšit je bude nutné jen pokud máte zvláště pomalý počítač, a zvyšovat budete časovače na **druhém** počítači. Program nazvaný `plipconfig` umožňuje změnit tato nastavení bez nutnosti rekompilace jádra. Je dodáván jako součást většiny distribucí Linuxu.

Konfigurace rozhraní *PLIP* znamená **doplnění** následujících řádků do síťového souboru `rc`:

```
#
# Attach a PLIP interface
#
#  configure first parallel port as a plip device
/sbin/ifconfig plip0 IPA.IPA.IPA.IPA pointopoint IPR.IPR.IPR.IPR up
#
# End plip
```

Kde:

IPA.IPA.IPA.IPA zastupuje vaši IP-adresu

IPR.IPR.IPR.IPR zastupuje IP-adresu vzdáleného počítače

Parametr *pointopoint* má stejný význam jako u protokolu SLIP v tom, že specifikuje adresu počítače na druhém konci spojení.

Téměř ve všech ohledech můžete zacházet s rozhraním *PLIP* jako by se jednalo o rozhraní *SLIP*, jen s tou výjimkou, že není zapotřebí ani možné používat příkazy `dip` a `slattach`.

Další informace najdete v dokumentu *PLIP-mini-HOWTO*.

6.24 PPP

Názvy zařízení PPP jsou „ppp0“,‘ppp1“ atd. Zařízení jsou číslována sekvenčně, přičemž první nalezené zařízení bude mít označení „`0`“.

Volby kompilace jádra

```
Networking options  --->
       <*> PPP (point-to-point) support
```

Konfigurace protokolu PPP je podrobněji probírána v dokumentu *PPP-HOWTO*.

6.24.1 Zajištění permanentního připojení k síti pomocí pppd

Máte-li to štěstí, že vlastníte „polopermanentní“ připojení k síti a byli byste rádi, kdyby váš počítač v případě ztráty spojení PPP toto automaticky znovu navázal, mám pro vás jednoduché řešení.

Pomocí následujícího příkazu nastavte PPP tak, jako by ho spustil uživatel *root*.

```
# pppd
```

Nezapomeňte v souboru `/etc/ppp/options` uvést volbu „`-detach`“. Potom vložte do sekce *getty* souboru `/etc/inittab` následující řádek:

```
pd:23:respawn:/usr/sbin/pppd
```

Takto bude program `init` monitorovat běh programu `pppd` a pokud skončí, automaticky ho znovu spustí.

6.25 Protokol Rose (AF_ROSE)

Názvy zařízení Rose jsou ve verzi jádra 2.1.* „rs0“, „rs1“ atd. Protokol Rose je dostupný od verzí 2.1.*.

Volby kompilace jádra

```
Networking options  --->
       [*] Amateur Radio AX.25 Level 2
       <*> Amateur Radio X.25 PLP (Rose)
```

Protokoly AX25, NetRom a Rose jsou popsány v dokumentu *AX25-HOWTO*. Tyto protokoly používají amatérští radiooperátoři při experimentech s paketovým rádiem.

Většinu práce na implementaci těchto protokolů má na svědomí Jonathon Naylor, **jsn@cs.nott.ac.uk**.

6.26 SAMBA – podpora 'NetBEUI', 'NetBios'

SAMBA je implementací protokolu Session Management Block. Umožňuje systémům firmy Microsoft a jiným připojovat a používat disky a tiskárny.

Protokol SAMBA a jeho konfigurace jsou rozebírány v dokumentu *SMB-HOWTO*.

6.27 SLIP-klient

Názvy zařízení SLIP jsou „sl0“, „sl“ atd. První nalezené zařízení bude mít označení „0“ a zbytek pak rostoucí čísla v pořadí, v jakém byly konfigurovány.

Volby kompilace jádra

```
Network device support  --->
       [*] Network device support
       <*> SLIP (serial line) support
       [ ]  CSLIP compressed headers
       [ ]  Keepalive and linefill
       [ ]  Six bit SLIP encapsulation
```

Protokol SLIP (Serial Line Internet Protocol) dovoluje provozovat protokol TCP/IP po sériové lince, ať už se jedná o telefonní linku s vytáčeným modemem nebo nějaký druh pronajaté linky. Samozřejmě, že abyste mohli používat protokol SLIP, potřebujete mít ve vaší oblasti přístup k *SLIP-serveru*. Spousta univerzit a společností po celém světě takovýto přístup poskytuje.

Zařízení SLIP používá sériové porty vašeho počítače k přenosu IP-datagramů. K tomu potřebuje převzít kontrolu nad sériovým zařízením. Zařízení SLIP jsou pojmenována „sl0“,

„s/l“ atd. Jak to koresponduje s vašimi sériovými zařízeními? Síťový kód pomocí tzv. volání `ioctl` (řízení V/V operací) změní sériová zařízení na SLIP zařízení. Tuto proceduru zvládají dva programy, nazývají se `dip` a `slattach`.

6.27.1 dip

`dip` (Dialup IP) je chytrý program, který umí nastavit rychlost sériového zařízení, přikázat vašemu modemu, aby vytočil vzdálený konec linky, automaticky vás přihlásí ke vzdálenému serveru, hledá zprávy, které vám server poslal, a vytáhne z nich informace, jako je IP-adresa, a spustí volání *ioctl*, které je nutné pro přepnutí sériového portu do režimu SLIP. Program `dip` má mocnou podporu skriptů, což můžete využít k automatizaci přihlašovací procedury.

Tento program najdete na adrese *ftp://sunsite.unc.edu/pub/Linux/system/Network/serial/dip/dip337o-uri.tgz*.

Při instalaci proveďte následující posloupnost příkazů:

```
#
# cd /usr/src
# gzip -dc dip337o-uri.tgz | tar xvf -
# cd dip-3.3.7o

<editace souboru Makefile>

# make install
#
```

Soubor `makefile` předpokládá existenci skupiny jménem *uucp*, ale tento název můžete v závislosti na vaší konfiguraci změnit buď na `dip` nebo *SLIP*.

6.27.2 slattach

Program `slattach` je ve srovnání s programem `dip` velice jednoduchý, snadno se používá, ale není tak důmyslný, jako `dip`. Nedisponuje podporou skriptů a veškerá jeho činnost spočívá v konfiguraci vašeho sériového zařízení jako zařízení SLIP. Předpokládá, že máte všechny potřebné informace a před jeho spuštěním je navázáno spojení po sériové lince. Program `slattach` je ideální tam, kde máte trvalé spojení se serverem, například fyzickým kabelem nebo vyhrazenou linkou.

6.27.3 Kdy který program použít?

Program `dip` byste měli použít, pokud se ke SLIP-serveru připojujete prostřednictvím modemu nebo nějakého jiného dočasného spojení. Program `slattach` byste měli použít, je-li váš počítač spojen se serverem vyhrazenou linkou, třeba kabelem, a pro správnou funkci tohoto spojení není třeba provádět nějaké speciální činnosti. Více informací získáte ve stati „Trvalé spojení SLIP“.

Konfigurace protokolu SLIP je podobná konfiguraci ethernetového rozhraní (nahlédněte do stati „Konfigurace ethernetového rozhraní výše). Jsou zde však některé klíčové rozdíly.

Předně spojení SLIP se liší od ethernetových sítí tím, že zde jsou v síti vždy jen dva hostitelé, na každém konci jeden. Na rozdíl od Ethernetu, který lze používat ihned po připojení kabelů, u protokolu SLIP je třeba, v závislosti na typu spojení, určitým speciálním způsobem síťové spojení inicializovat.

Pokud používáte program `dip`, nedojde k tomu normálně při startu systému, ale o něco později, až budete připraveni navázat spojení. Tuto proceduru je možné automatizovat. Používáte-li program `slattach`, bude nutné přidat do souboru `rc.inet1` novou sekci, viz dále.

Existují dva základní typy SLIP-serverů: Servery s dynamickou IP-adresou a servery se statickou IP-adresou. Skoro každý SLIP-server vás po vytočení vyzve k přihlášení za pomoci vašeho uživatelského jména a hesla. Program `dip` vás může přihlásit automaticky.

6.27.4 Statický SLIP-server s vytáčenou linkou a programem DIP

Statickému SLIP-serveru poskytujete IP-adresu, která je výlučně vaše. Pokaždé, když se připojíte k tomuto serveru, nastavíte SLIP-port na tuto adresu. Statický SLIP-server odpoví vašemu modemu, pravděpodobně vás vyzve k zadání vašeho uživatelského jména a hesla a potom vám přes toto spojení pošle veškeré datagramy určené pro vaši adresu. Máte-li server se statickou IP-adresou, můžete název vašeho hostitele a IP-adresu (pokud je dopředu znáte) umístit do souboru `/etc/hosts`. Také bude potřeba překonfigurovat některé další soubory: `rc.inet2`, `host.conf`, `resolv.conf`, `/etc/HOSTNAME` a `rc.local`. Nezapomeňte, že při úpravě souboru `rc.inet1` do něj nemusíte přidávat žádné speciální příkazy týkající se SLIP-spojení, protože program `dip` provede veškerou práci související s konfigurací rozhraní za vás. Stačí, když programu `dip` poskytnete příslušné informace, a on nejprve přikáže modemu, aby navázal spojení a přihlásil se k vašemu SLIP-serveru a potom sám rozhraní nakonfiguruje.

Pokud váš SLIP-server funguje tímto způsobem, pak přeskočte na další stať – „Používání programu DIP“, kde se dozvíte informace o jeho konfiguraci.

6.27.5 Dynamický SLIP-server s vytáčenou linkou a programem DIP

Dynamický SLIP-server vám při každém přihlášení přidělí IP-adresu náhodně ze skupiny adres. To znamená, že nelze zaručit, že budete mít pokaždé jednu konkrétní adresu a že tuto adresu může po vašem odhlášení využívat někdo jiný. Správce sítě, který prováděl konfiguraci SLIP-serveru, mu přidělil skupinu adres, které může používat. Když server obdrží nový hovor, vyhledá první nepoužívanou adresu, provede volajícího přihlašovacím procesem a potom zobrazí uvítací zprávu, která obsahuje přidělenou IP-adresu a tuto adresu pak bude používat po celou dobu trvání hovoru.

Konfigurace tohoto typu serveru je podobná konfiguraci statického serveru. Pouze obsahuje jeden krok navíc, ve kterém je třeba získat IP-adresu, kterou vám server vyhradil, a předat ji zařízení SLIP.

I v tomto případě za vás provede veškerou práci program `dip`, jehož novější verze vás nejenom přihlásí, ale umí také automaticky vytáhnout a uložit z uvítací zprávy IP-adresu, kterou pak můžete použít při konfiguraci zařízení SLIP.

6.27.6 Používání programu DIP

Již jsme si řekli, že program `dip` je mocný nástroj, který vám může za pomocí příkazů `ifconfig` a `route` ulehčit a zcela automatizovat proces vytáčení SLIP-serveru, přihlášení, navázání spojení a konfigurace SLIP-zařízení.

V podstatě napíšete skript tvořený příkazy, kterým program `dip` rozumí a na jejichž základě provede příslušné akce. Představu o fungování programu `dip` si můžete utvořit podle následujícího vzorového skriptu `sample.dip`, který je dodáván společně s tímto programem. Program `dip` je poměrně výkonný a obsahuje množství voleb.

Nebudeme se jimi zde zabývat, ale v případě zájmu vás odkazuji na manuálové stránky, soubor `README` a vzorové soubory, které získáte současně s vaší verzí programu `dip`.

Možná jste si všimli, že skript `sample.dip` předpokládá, že používáte statický SLIP-server, takže budete dopředu vědět svou IP-adresu. Pro dynamické SLIP-servery obsahují novější verze programu `dip` příkaz, který slouží k automatickému přečtení IP-adresy, kterou vám dynamický server přidělil, a následnému nakonfigurování zařízení SLIP. Následující ukázka je upravenou verzí souboru `sample.dip`, který je součástí balíku `dip337j-uri.tgz` a má vám posloužit jako jakýsi odrazový můstek. Soubor uložte pod názvem `/etc/dipscript` a upravte podle vaší konfigurace.

```
#
# sample.dip    Dialup IP connection support program.
#
#                This file (should show) shows how to use the DIP
#       This file should work for Annex type dynamic servers, if you
#       use a static address server then use the sample.dip file that
#       comes as part of the dip337-uri.tgz package.
#
#
# Version:      @(#)sample.dip  1.40    07/20/93
#
# Author:       Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
#

main:
# Next, set up the other side's name and address.
# My dialin machine is called 'xs4all.hacktic.nl' (== 193.78.33.42)
get $remote xs4all.hacktic.nl
# Set netmask on sl0 to 255.255.255.0
netmask 255.255.255.0
# Set the desired serial port and speed.
port cua02
speed 38400

# Reset the modem and terminal line.
# This seems to cause trouble for some people!
reset

# Note! "Standard" pre-defined "errlevel" values:
#  0 - OK
#  1 - CONNECT
#  2 - ERROR
#
# You can change those grep'ping for "addchat()" in *.c...

# Prepare for dialing.
```

```
send ATQ0V1E1X4\r
wait OK 2
if $errlvl != 0 goto modem_trouble
dial 555-1234567
if $errlvl != 1 goto modem_trouble

# We are connected.  Login to the system.
login:
sleep 2
wait ogin: 20
if $errlvl != 0 goto login_trouble
send MYLOGIN\n
wait ord: 20
if $errlvl != 0 goto password_error
send MYPASSWD\n
loggedin:
# We are now logged in.
wait SOMEPROMPT 30
if $errlvl != 0 goto prompt_error

# Command the server into SLIP mode
send SLIP\n
wait SLIP 30
if $errlvl != 0 goto prompt_error

# Get and Set your IP address from the server.
#   Here we assume that after commanding the SLIP server into SLIP
#   mode that it prints your IP address
get $locip remote 30
if $errlvl != 0 goto prompt_error

# Set up the SLIP operating parameters.
get $mtu 296
# Ensure "route add -net default xs4all.hacktic.nl" will be done
default

# Say hello and fire up!
done:
```

```
print CONNECTED $locip ---> $rmtip
mode CSLIP
goto exit

prompt_error:
print TIME-OUT waiting for sliplogin to fire up...
goto error

login_trouble:
print Trouble waiting for the Login: prompt...
goto error

password:error:
print Trouble waiting for the Password: prompt...
goto error

modem_trouble:
print Trouble occurred with the modem...
error:
print CONNECT FAILED to $remote
quit

exit:
exit
```

Výše uvedený příklad předpokládá volání *dynamického* SLIP-serveru. Voláte-li *statický* SLIP-server, použijte soubor `sample.dip`, který je součástí balíku `dip337j-uri.tgz`.

Když předáte programu `dip` příkaz `get $local`, prohledá příchozí text, zda neobsahuje řetězec, který vypadá jako IP-adresa, to znamená řetězec čísel oddělených tečkami. Tato úprava byla do programu přidána kvůli *dynamickým* SLIP-serverům, aby mohl být proces čtení IP-adresy poskytnuté serverem automatizován.

Výše uvedený příklad automaticky vytvoří implicitní trasu přes vaše SLIP-spojení. Pokud si to nepřejete, může ponechat ethernetové spojení jako implicitní trasu, potom odstranit příkaz `default` ze skriptu. Po skončení tohoto skriptu provedete příkaz `ifconfig`, zjistíte, že máte nové zařízení jménem sl0. To je vaše zařízení SLIP. Dle potřeby můžete manuálně upravit jeho konfiguraci po skončení příkazu `dip` pomocí příkazů `ifconfig` a `route`.

Všimněte si, že příkaz `mode` programu `dip` umožňuje výběr několika různých protokolů, nejpoužívanějším příkladem je CSLIP, což znamená SLIP s kompresí. Všimněte si také, že musí souhlasit oba konce spojení, takže cokoliv zvolíte by mělo souhlasit s nastavením vašeho serveru.

Výše uvedený příklad je poměrně robustní a měl by odolat většině chyb. Podrobnější informace najdete v manuálových stránkách programu `dip`. Přirozeně byste mohli třeba vytvořit skript, který by prováděl opakované vytáčení serveru, pokud by nebylo navázáno spojení po uplynutí stanovené doby nebo vyzkoušet skupinu serverů, pokud přistupujete k více než jednomu serveru.

6.27.7 Trvalé spojení SLIP s vyhrazenou linkou a programem slattach

Máte-li dva počítače propojeny kabelem, případně vyhrazenou linkou nebo nějakým jiným typem trvalého sériového spojení, nepotřebujete se zatěžovat všemi problémy kolem používání programu `dip` na sériové linky. Při konfiguraci spojení úplně vystačíte s jednoduchou utilitou jménem `slattach`.

Protože bude vaše spojení trvalé, bude nutné přidat určité příkazy do souboru *rc.inet1*. V podstatě je potřeba nastavit správnou rychlost sériového zařízení a přepnou ho do režimu SLIP. Díky programu `slattach` vám k tomu stačí jeden příkaz. Následující řádky **přidejte** do souboru `rc.inet1`.

```
#
# Attach a leased line static SLIP connection
#
#  configure /dev/cua0 for 19.2kbps and cslip
/sbin/slattach -p cslip -s 19200 /dev/cua0 &
/sbin/ifconfig sl0 IPA.IPA.IPA.IPA pointopoint IPR.IPR.IPR.IPR up
#
# End static SLIP.
```

Kde:

IPA.IPA.IPA.IPA zastupuje vaši IP-adresu

IPR.IPR.IPR.IPR zastupuje IP-adresu vzdáleného počítače

Program `slattach` přidělí první volné zařízení SLIP specifikovanému sériovému zařízení. Začne označením sl0. Takto první příkaz `slattach` přiřadí specifikovanému zařízení SLIP zařízení sl0, další pak sl1 atd.

Program `slattach` umožňuje pomocí argumentu `-p` konfigurovat různé protokoly. Ve vašem případě použijete buď *SLIP*, nebo *CSLIP*, podle toho, zda chcete nebo nechcete používat kompresi.

6.28 SLIP-server

Pokud máte počítač, který je připojen k síti, a chcete, aby se k němu mohli připojovat i jiní uživatelé, budete muset tento počítač nakonfigurovat jako server. Chcete-li jako protokol sériové linky používat SLIP, máte, co se týče způsobu konfigurace vašeho linuxového počítače jako SLIP-server, tři možnosti. Já osobně bych dal přednost první z nich, která využívá program `sliplogin` a připadá mi nejjednodušší co se konfigurace a pochopení týče. Nicméně představím vám všechny tři, abyste si mohli udělat vlastní úsudek.

6.28.1 SLIP-server používající program sliplogin

Program `sliplogin` lze použít místo klasického přihlašovacího programu pro uživatele, kteří převedou terminálovou linku na SLIP-linku. Umožňuje nastavit váš linuxový počítač jako *statický server* (uživatelé získají při každém zavolání vždy stejnou adresu) nebo jako *dynamický server* (uživatelům je přidělena adresa, která nutně nemusí být stejná, jako při předchozím spojení).

Volající se přihlásí jako při normálním přihlašování, zadá své uživatelské jméno a heslo. Ovšem místo příkazového interpretu se po přihlášení spustí program *sliplogin*, který vyhledá v konfiguračním souboru (`/etc/slip.hosts`) záznam s uživatelským jménem, které odpovídá jménu volajícího. Pokud takový záznam najde, nastaví linku jako 8bitovou a pomocí udání *ioctl* ji převede na SLIP-linku. Po skončení tohoto procesu začne poslední fáze konfigurace, kdy program `sliplogin` spustí skript příkazového interpretu, který přidělí rozhraní SLIP IP-adresu, síťovou masku a nastaví příslušné směrování. Tento skript se většinou nazývá `/etc/slip.login`, ale vyžadují-li někteří volající speciální inicializaci, můžete vytvořit konfigurační skripty nazvané `/etc/slip.login.přihlašovacíjméno`, které budou spouštěny místo tohoto implicitního.

Pro správnou funkci programu `sliplogin` potřebujete tři nebo čtyři soubory. Podrobně vám popíši, jak a kde získáte příslušný software a jak ho nakonfigurujete. Jedná se o tyto soubory:

- `/etc/passwd` pro vytáčené uživatelské účty
- `/etc/slip.hosts` pro ukládání informací, které jsou pro každého uživatele jedinečné
- `/etc/slip.login` řídí konfiguraci směrování, kterou je třeba provádět pro každého uživatele
- `/etc/slip.tty` je nutný pouze v případě, že má server nastavenu *dynamickou alokaci adres* a obsahuje tabulku přidělovaných adres

- `/etc/slip.logout` obsahuje příkazy, které se provedou po zavěšení nebo odhlášení uživatele

Kde získáte program `sliplogin`**?** Je možné, že jste balík `sliplogin` získali jako součást vaší distribuce Linuxu. Pokud nikoliv, získáte program `sliplogin` na adrese *ftp://sunsite.unc.edu/pub/linux/system/Network/serial/sliplogin-2.1.1.tar.gz*. Archiv ve formátu tar obsahuje zdrojový kód, předkompilované binární soubory a manuálovou stránku.

Aby mohli program `sliplogin` používat pouze autorizovaní uživatelé, měli byste do souboru `/etc/group` přidat záznam podobný tomu následujícímu:

```
..
slip::13:radio,fred
..
```

Po nainstalování balíku `sliplogin` změní soubor `Makefile` vlastnictví skupiny programu `sliplogin` na *slip*, což znamená, že ho mohou spouštět pouze uživatelé patřící do této skupiny. Výše uvedený zápis povolí spouštění tohoto programu pouze uživatelům *radio* a *fred*.

Následující posloupnost příkazů nainstaluje binární soubory do adresáře `/sbin` a manuálové stránky do sekce 8.

```
# cd /usr/src
# gzip -dc .../sliplogin-2.1.1.tar.gz | tar xvf -
# cd sliplogin-2.1.1
# <..editujte Makefile pokud nepoužíváte stínová hesla..>
# make install
```

Pokud budete chtít binární soubory ještě před instalací překompilovat, spusťte před příkaze `make install` ještě příkaz `make clean`. Budete-li chtít nainstalovat binární soubory někam jinam, musíte upravit pravidlo *install* v souboru `Makefile`.

Další informace získáte v souboru `README`, který je součástí balíku.

Konfigurace souboru `/etc/passwd` **pro SLIP hostitele**. Normálně byste pro tyto uživatele vytvořili v souboru `/etc/passwd` speciální účty. Běžně se ovšem před jméno volajícího hostitele přidává písmeno „S“. Takže když se například volající hostitel jmenuje radio, vytvoříte v souboru `/etc/passwd` následující záznam:

```
Sradio:FvKurok73:1427:1:radio SLIP login:/tmp:/sbin/sliplogin
```

Pokud je pro vás účet smysluplný, nezáleží na tom, jak se nazývá.

Poznámka: Volající nepotřebuje mít žádný speciální adresář, protože nebude mít přístup k příkazovému interpretu, takže postačí adresář `/tmp`. Nezapomeňte také, že místo normálního přihlašovacího příkazového interpretu se používá program `sliplogin`.

Konfigurace souboru `/etc/slip.hosts`. V souboru `/etc/slip.hosts` hledá program `sliplogin` záznamy vyhovující uživatelskému jménu, aby získal konfigurační informace pro příslušného volajícího. V tomto souboru specifikujete IP-adresu a síťovou masku, která bude přidělena volajícímu. Následují vzorové záznamy pro dva hostitele, první je statická konfigurace pro hostitele radio a druhá dynamická pro uživatele *albert*:

```
#
Sradio    44.136.8.99    44.136.8.100   255.255.255.0   normal       -1
Salbert   44.136.8.99    DYNAMIC        255.255.255.0   compressed   60
#
```

Záznamy v souboru `/etc/slip.hosts` jsou složeny z následujících položek:

1. Přihlašovací jméno volajícího uživatele.
2. IP-adresa serveru, to znamená tohoto počítače.
3. IP-adresa, která bude přidělena volajícímu. Je-li v tomto poli uveden řetězec „`DYNAMIC`“, bude IP-adresa přidělena v závislosti na informaci obsažené v souboru `/etc/slip.tty`, který jsme probírali před chvílí.
   Poznámka: Aby vše fungovalo tak, jak má, musíte používat program `sliplogin` alespoň verze 1.3.
4. Síťová maska přidělená volajícímu počítači v tečkové notaci, tj. 255.255.255.0 pro síťovou masku třídy C.
5. Nastavení režimu SLIP, kterým můžete povolit nebo zakázat kompresi. Povolené jsou hodnoty „`normal`“ nebo „`compressed`“.
6. Časový údaj, který specifikuje dobu, po kterou může linka zůstat v nečinnosti (nepřijímá žádné datagramy), než dojde k automatickému zrušení spojení. Záporná hodnota tuto vlastnost ruší.
7. Volitelné argumenty.

Poznámka: Pole 2 a 3 mohou obsahovat buď názvy hostitelů, nebo IP-adresy. Použijete-li názvy hostitelů, musí být váš počítač schopen zjistit příslušnou IP-adresu, jinak se skript po spuštění zhroutí. Můžete si to ověřit příkazem `telnet`, kterému předáte jako parametr název hostitele. Pokud se objeví zpráva „*Trying nnn.nnn.nnn…*“, podařilo se vašemu počítači nalézt IP-

adresu odpovídající tomuto názvu hostitele. Objeví-li se zpráva „*Unknown host*“, potom se to nepodařilo. V takovém případě buď použijte IP-adresu převedenou na tečkovou desítkovou notaci, nebo upravte konfiguraci resolveru (viz. stať Konfigurace resolveru).

Nejpoužívanější režimy SLIP jsou:

normal - zapíná normální nekomprimovaný SLIP

compressed – zapíná van Jacobsonovu kompresi hlaviček (CSLIP)

Samozřejmě, že se tyto režimy navzájem vylučují, takže musíte použít buď jeden, nebo druhý. Další informace o ostatních volbách najdete v manuálových stránkách.

Konfigurace souboru `/etc/slip.login`. Jakmile program `sliplogin` nalezl vyhovující záznam v souboru `/etc/slip.hosts`, pokusí se zpracovat soubor `/etc/slip.login` a dojde na vlastní konfiguraci rozhraní SLIP, kterému je předána IP-adresa a síťová maska.

Vzorový soubor `/etc/slip.login`, který je dodáván s balíkem `sliplogin`, vypadá takto:

```
#!/bin/sh -
#
#       @(#)slip.login  5.1 (Berkeley) 7/1/90
#
# generic login file for a SLIP line.  sliplogin invokes this with
# the parameters:
#     $1        $2       $3     $4, $5, $6 ...
#   SLIPunit ttyspeed   pid   the arguments from the slip.host entry
#
/sbin/ifconfig $1 $5 pointopoint $6 mtu 1500 -trailers up
/sbin/route add $6
arp -s $6 <hw_addr> pub
exit 0
#
```

Jistě si všimnete, že tento skript používá ke konfiguraci SLIP-zařízení příkazy `ifconfig` a `route`, kterým předá IP-adresu, vzdálenou IP-adresu a síťovou masku, a vytvoří směrování pro vzdálenou adresu prostřednictvím rozhraní SLIP. Stejného výsledku bychom dosáhli i pomocí programu `slattach`.

Nezapomeňte také použít *Proxy ARP*, aby ostatní hostitelé, kteří jsou připojeni ke stejnému Ethernetu jako server, věděli, jak se dostanou k volajícímu hostiteli. Pole <hw_addr> by mělo obsahovat hardwarovou adresu ethernetové karty v tomto počítači. Pokud váš server není součástí ethernetové sítě, můžete tento řádek vynechat.

Konfigurace souboru /etc/slip.logout. Jakmile volající zavěsí, budete chtít obnovit normální stav sériového zařízení, aby se mohli dovolat další volající. Tuto funkci plní soubor /etc/slip.logout. Jeho formát je poměrně jednoduchý a je spouštěn se stejným argumentem, jako soubor /etc/slip.login.

```
#!/bin/sh -
#
#                 slip.logout
#
/sbin/ifconfig $1 down
arp -d $6
exit 0
#
```

Jeho cílem je „shodit“ rozhraní, které odstraní dříve vytvořené manuální směrování. Pomocí příkazu arp také smaže všechny záznamy v tabulce ARP. I zde platí, že pokud server neobsahuje ethernetový port, nemusí být ve skriptu příkaz arp.

Konfigurace souboru /etc/slip.tty. Používáte-li dynamické přidělování adres (máte u některých hostitelů nastaveno v souboru /etc/slip.hosts klíčové slovo DYNAMIC) musíte v souboru /etc/slip.tty uvést seznam adres přidělených jednotlivým portům. K dynamickému přidělování adres potřebujete pouze tento soubor.

Soubor /etc/slip.tty je vlastně tabulka obsahující zařízení *tty*, která budou podporovat vytáčené SLIP-spojení a IP-adresy, které budou přiděleny uživatelům, kteří na příslušný port zavolají.

Formát souboru vypadá takto:

```
# slip.tty     tty -> IP address mappings for dynamic SLIP
# format: /dev/tty?? xxx.xxx.xxx.xxx
#
/dev/ttyS0       192.168.0.100
/dev/ttyS1       192.168.0.101
#
```

Tato tabulka říká, že volajícímu, který vytočí port `/dev/ttyS0` a má v souboru `/etc/slip.hosts` nastavené pole pro vzdálenou adresu na `DYNAMIC`, přiřadí adresu `192.168.0.100`.

Takto vám pro všechny uživatele, kteří nevyžadují vyhrazené adresy, stačí alokovat jednu adresu pro každý port. Vyhnete se tak plýtvání adresovým prostorem.

6.28.2 SLIP-server používající program dip8

Dovolte mi začít konstatováním, že některé níže uvedené informace pocházejí z manuálových stránek programu `dip`, kde je stručně popisována konfigurace SLIP-serveru pod Linuxem. Předesílám též, že následující fakta vycházejí z balíku `dip337o-uri.tgz` a pravděpodobně se nebudou úplně shodovat s jinými verzemi tohoto programu.

Program `dip` může pracovat v tzv. vstupním režimu, kdy automaticky vyhledá záznam uživatele, který ho spustil, a nakonfiguruje sériovou linku jako SLIP linku podle informací uložených v souboru `/etc/diphosts`. Tento vstupní režim aktivujete, když program `dip` spustíte jako `diplogin`. Tímto způsobem používáte program `dip` jako SLIP-server, vytvoříte speciální účty, kde program `diplogin` funguje jako přihlašovací příkazový interpret.

Nejprve je třeba vytvořit následující symbolický odkaz:

```
# ln -sf /usr/sbin/dip /usr/sbin/diplogin
```

Potom doplníte určité záznamy do souborů `/etc/passwd` a `etc/diphosts`. Tyto záznamy mají níže popsaný formát:

Při konfiguraci Linuxu coby SLIP-serveru potřebujete vytvořit pro uživatele určité speciální SLIP-účty, přičemž program `dip` (ve vstupním režimu) slouží jako přihlašovací příkazový interpret. Doporučuje se označovat všechny účty SLIP s počátečním písmenem „S“, např. „Sfredm“.

Takto vypadá vzorový záznam SLIP-uživatele v souboru `/etc/passwd`:

```
Sfredm:ij/SMxiTlGVCo:1004:10:Fred:/tmp:/usr/sbin/diplogin
```

Jakmile se uživatel přihlásí, program `login`, pokud nalezne a ověří uživatele, spustí příkaz `diplogin`. Program `dip`, je-li spuštěn jako `diplogin`, ví, že má automaticky předpokládat, že je používán jako přihlašovací příkazový interpret. Pokud je spuštěn jako `diplogin`, zjistí nejprve pomocí funkce *getuid()* id uživatele, který ho spustil. Potom vyhledá v souboru `/etc/diphosts` první záznam, který odpovídá buď id uživatele, nebo názvu zařízení *tty*, ze kterého hovor pochází a příslušným způsobem se nakonfiguruje. Moudrým rozhodnutím, zda vyhradit uživateli záznam v souboru `diphosts` nebo mu přidělit implicitní konfiguraci, získáte směs uživatelů s dynamicky a staticky přidělovanými adresami.

Program `dip`, je-li spuštěn ve vstupním režimu, automaticky přidá záznam „Proxy-ARP“, takže se nemusíte zabývat manuálním přidáváním těchto záznamů.

Konfigurace souboru /etc/diphosts.

Soubor `/etc/diphosts` slouží programu `dip` k vyhledávání přednastavených konfigurací vzdálených hostitelů. Tito vzdálení hostitelé mohou být uživatelé, kteří se přihlašují k vašemu linuxovému počítači, nebo to mohou být stroje, ke kterým se přihlašujete z vašeho počítače.

Následuje obecný formát souboru `/etc/diphosts`:

```
..
Suwalt::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
ttyS1::145.71.34.3:145.71.34.2:255.255.255.0:Dynamic ttyS1:CSLIP,296
..
```

Toto jsou jednotlivá pole každého záznamu:

1. přihlašovací jméno: vrácené funkcí getpwuid(getuid()) nebo název tty
2. nevyužito: slučitelné s heslem
3. Vzdálená adresa: IP-adresa volajícího hostitele, buď číselná, nebo názvová
4. Lokální adresa: IP-adresa tohoto počítače, opět buď číselná, nebo názvová
5. Síťová maska: v tečkové notaci
6. Pole komentáře: sem můžete napsat cokoliv
7. Protokol: SLIP, CSLPI atd.
8. MTU: desítkové číslo

Příklad záznamu v souboru `/etc/diphosts` pro vzdáleného SLIP-uživatel může vypadat třeba takto:

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:SLIP,296
```

což udává SLIP-spojení se vzdálenou adresou 145.71.34.1 a MTU 296 nebo jiný příklad:

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
```

který specifikuje spojení CSLIP se vzdálenou adresou 145.71.34.1a MTU 1006.

Proto všichni uživatelé, kterým chcete povolit staticky přidělovaný vytáčený IP-přístup, musí mít záznam v souboru `/etc/diphosts`. Pro uživatele, kteří vytáčejí konkrétní port, s nímž si dynamicky domlouvají detaily spojení, potřebujete mít záznam pro zařízení *tty*. Nezapomeňte nakonfigurovat alespoň jeden záznam pro každé zařízení *tty*, které používají tito uživatelé, aby pro ně byla k dispozici vhodná konfigurace bez ohledu na modem, se kterým se spojí.

Když se uživatel přihlásí, odpoví na normální přihlašovací výzvu svým přihlašovacím SLIP id a heslem. Po ověření neuvidí uživatel žádné zprávy a zařízení se přepne do režimu SLIP. Uživatel by se potom měl bez potíží připojit a příslušnému zařízení budou nastaveny parametry ze souboru `diphosts`.

6.28.3 SLIP-server používající balík dSLIP

Matt Dillon **<dillon@apollo.west.oic.com>** napsal balík, který se umí nejenom přihlásit, ale také odhlásit. Jde o kombinaci malých programů a skriptů, které se starají o vaše spojení. Potřebujete mít nainstalovaný program `tcsh`, protože ho ke své činnosti vyžaduje minimálně jeden skript. Matt Dillon dodává také utilitu `expect` (v binárním tvaru), kterou také vyžaduje jeden ze skriptů. Ke zprovoznění tohoto balíku budete potřebovat určitou zkušenost s programem `expect`, ale tím se nenechte odradit.

Matt Dillon popsal instalaci v souboru `README`, takže se jí zde nebudu zabývat.

Balík `dSLIP` získáte na jeho domovské adrese:

apollo.west.oic.com `/pub/linux/dillon_src/dSLIP203.tgz`

nebo na adrese:

sunsite.unc.edu `/pub/Linux/system/Network/serial/dSLIP203.tgz`

Před spuštěním příkazu `make install` si přečtěte soubor `README` a vytvořte příslušné záznamy v souborech `/etc/passwd` a `/etc/group`.

6.29 Podpora protokolu STRIP (Standard Radio IP)

Názvy zařízení STRIP jsou „st0“, „st1“ atd.

Volby kompilace jádra

```
Network device support  --->
       [*] Network device support
       ....
       [*] Radio network interfaces
       < > STRIP (Metricom starmode radio IP)
```

Protokol STRIP byl navržen speciálně pro rádiové modemy Metricom v rámci výzkumného projektu vedeného na Stanford University a pojmenovaného MosquitoNet Project *http://mosquitonet.Stan-ford.EDU/mosquitonet.html.* Je to velice zajímavé čtení, a to i pro ty, kdo se o tento projekt přímo nezajímají.

Rádiové modemy Metricom se připojují k sériovému portu, využívají široké spektrum technologií a zvládají přenosové rychlosti 100 kbps. Informace o rádiových modemech Metricom získáte na adrese *http://www.metricom.com/.*

V současné době nepodporují standardní síťové nástroje ovladač protokolu STRIP, takže si budete muset ze serveru MosquitoNet stáhnout nějaké upravené nástroje. Podrobnosti ohledně softwaru, který potřebujete, najdete na stránce *http://mosquitonet.Stanford.EDU/strip.html.*

Co se konfigurace týče, pomocí upraveného programu `slattach` nastavíte chování linky sériového zařízení *tty* na STRIP a potom nakonfigurujete výsledné zařízení „st[0-9]“ jako pro Ethernet, jen s jednou výjimkou. Z technických důvodů nepodporuje protokol STRIP protokol ARP, takže je třeba ručně upravit záznamy ARP pro každého hostitele ve vaší podsíti. To by nemělo být obtížné.

6.30 Token Ring

Názvy zařízení token ring jsou „tr0“, „tr1“ atd. Token Ring je standardní LAN protokol vytvořený firmou IBM, který zabraňuje kolizím pomocí mechanismu, dávajícího v určitou dobu pouze jedné stanici v síti LAN právo vysílat. „Token“ vlastní vždy pouze jedna stanice a pouze ona může přenášet data. Po skončení přenosu předá token další stanici. Token putuje dokola po všech aktivních stanicích, odtud tedy název „Token Ring“.

Volby kompilace jádra

```
Network device support  --->
       [*] Network device support
       ....
       [*] Token Ring driver support
       < > IBM Tropic chipset based adaptor support
```

Konfigurace protokolu Token Ring je kromě názvu síťového zařízení identická s konfigurací Ethernetu.

6.31 X.25

X.25 je kruhový protokol založený na přepínání paketů definovaný organizací C.C.I.T.T. (Sdružení, které navrhuje standardy, jež se používají telekomunikačními společnostmi téměř na celém světě). Na implementaci protokolů AX.25 a LAPB se pracuje a poslední jádra verzí 2.1.* je již obsahují.

Vývoj vede Jonathon Naylor **jsn@cs.nott.ac.uk** a za účelem řešení problémů kolem protokolu AX.25 byla zřízena diskusní skupina. Chcete-li se do ní přihlásit, pošlete zprávu na adresu **majordomo@vger.rutgers.edu** a do jejího těla napište text „`subscribe linux-x25`“.

Dřívější verze konfiguračních nástrojů získáte na Jonathonově anonymním FTP na adrese *ftp://ftp.cs.nott.ac.uk/jsn/*.

6.32 Karta WaveLan

Názvy zařízení WaveLan jsou „eth0“, „eth1“ atd.

Volby kompilace jádra

```
Network device support  --->
       [*] Network device support
       ....
       [*] Radio network interfaces
       ....
       <*> WaveLAN support
```

Karta WaveLan je bezdrátová síťová karta. Při používání je velmi podobná ethernetové kartě a stejným způsobem se i konfiguruje.

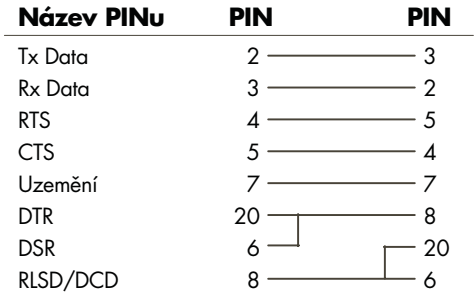
Informace o kartě WaveLan získáte na adrese *http://www.wavelan.com/*.

7 Kabely a kabeláž

Ti z vás, kteří mají zkušenosti s pájením, si možná budou chtít vyrobit vlastní kabely, kterými by propojili své linuxové počítače. Následující diagramy kabelů by vám měly pomoci.

7.1 Sériový kabel null modem

Ne všechny kabely null modem jsou stejné. Většina těchto kabelů umí jen simulovat přítomnost všech příslušných signálů a přehodit vstupní a výstupní data. To je sice pěkné, ale znamená to, že musíte používat softwarovou kontrolu toku dat (XON/XOFF), která je méně efektivní než hardwarová. Následující kabel poskytuje nejlepší možnou signalizaci mezi počítači a umožňuje používat hardwarovou (RTS/CTS) kontrolu toku dat.



7.2 Kabel pro paralelní port (kabel PLIP)

Pokud hodláte používat protokol PLIP ke komunikaci dvou počítačů, můžete použít tento kabel bez ohledu na typ instalovaného paralelního portu.

Poznámky:

- Nespojujte piny označené hvězdičkou („*“).
- Zvlášť uzemněny jsou piny 18,19,20,21,22,23 a 24.
- Má-li vámi použitý kabel kovové stínění, měl by být **na jednom konci** opatřen kovovou zástrčkou DB-25.

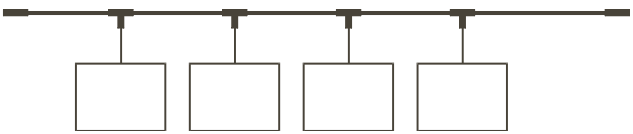
Upozornění: Špatně zapojený kabel PLIP může zničit kartu řadiče. Při zapojování buďte proto velice opatrní a dvakrát zkontrolujte každé spojení, abyste se vyhnuli problémům.

Název PINu	PIN		PIN
STROBE	1*		
D0 - ERROR	2	———	15
D1 - BUSY	3	———	13
D2 - PAPOUT	4	———	12
D3 - ACK	5	———	10
D4 - BUSY	6	———	11
D5	7*		
D6	8*		
D7	9*		
ACK - D3	10	———	5
BUSY - D4	11	———	6
PAPOUT - D2	12	———	4
SLCT - D1	13	———	3
FEED	14*		
ERROR - D0	15	———	2
INIT	16*		
SLCTIN	17*		
Uzemnění	25	———	25

I když můžete kabel PLIP používat na velké vzdálenosti, měli byste se tomu, je-li to možné, vyhnout. Specifikace tohoto kabelu umožňuje délku kolem 1 metru. Při používání dlouhých kabelů tohoto typu dávejte proto pozor na zdroje elektromagnetických polí, jako je blesk, vodiče elektrického proudu a rádiové vysílače, které je mohou rušit a někdy též zničit váš řadič. Pokud opravdu potřebujete spojit dva počítače na velkou vzdálenost, měli byste si opatřit dvojici ethernetových karet a použít raději koaxiální kabel.

7.3 Ethernetový kabel 10base2 (tenký koaxiál)

10base2 je standard ethernetového kabelu, který specifikuje použití 52ohmového koaxiálního kabelu o průměru kolem 5 milimetrů. Při spojování počítačů kabelem 10base2 je třeba dodržet dvě důležitá pravidla. Zaprvé musíte použít terminátory **na obou koncích** kabelu. Terminátor je rezistor 52 ohmů, který zajišťuje, aby byl signál po dosažení konce absorbován a nikoliv odražen. Bez terminátoru na každém konci kabelu může být Ethernet nespolehlivý nebo nemusí vůbec fungovat. Normálně se používají k propojení počítačů konektory ve tvaru T, takže získáte následující zapojení:




na každém konci kabelu reprezentuje terminátor

reprezentuje délku koaxiálního kabelu s BNC-konektory na obou koncích


reprezentuje konektor ve tvaru T. Délka kabelu mezi T-konektorem a vlastní ethernetovou kartou by měl být co nejmenší, v ideálním případě je T-konektor zapojen přímo do ethernetové karty.

7.4 Ethernetový kabel Twisted pair

Pokud máte pouze dvě ethernetové karty twisted pair a chcete je propojit, nepotřebujete rozbočovač. Tyto karty můžete propojit přímo. Schéma zapojení najdete v dokumentu *Ethernet-HOWTO*.

8 Některé termíny používané v tomto dokumentu

Následuje seznam některých důležitých termínů používaných v tomto dokumentu.

ARP Zkratka názvu protokolu Address Resolution Protocol a jedná se o způsob, jakým síťový počítač sdružuje IP-adresy se síťovými adresami.

ATM Zkratka režimu Asynchronous Transfer Mode. Síť ATM balí data do bloků standardních velikostí, které může efektivně dopravovat z jednoho místa na druhé. ATM je síťová technologie založená na kruhovém přepínání paketů.

klient Většinou se jedná o software na straně uživatele. Existují výjimky z tohoto tvrzení, například v okenním systému X11 běží na straně uživatele server a na vzdáleném počítači klient. Klient je program, který přijímá nějakou službu poskytovanou serverem. V případě systému *peer to peer*, jako je *SLIP* a *PPP*, navazuje klient spojení a vzdálený konec, ten, který je volán, se nazývá server.

datagram Datagram je vyhrazený balík dat a hlaviček, který obsahuje adresy. Jedná se o základní přenosovou jednotku v síti IP. Někdy se také datagramům říká pakety

DLCI DLCI znamená Data Link Connection Identifier a slouží k identifikaci jedinečného virtuálního spojení typu point to point v síti Frame Relay. Identifikátor DLCI normálně přiděluje poskytovatel sítě Frame Relay.

Frame Relay Frame Relay je síťová technologie, která je ideální pro impulsní nebo ojedinělý přenos. Sdílí-li stejnou kapacitu sítě mnoho zákazníků, kteří používají síť v jinou dobu, mohou se značně zredukovat síťové náklady.

Hardwarová adresa Jedná se o číslo, které slouží k jedinečné identifikaci hostitele v rámci fyzické sítě na úrovni přístupu k médiu. Příkladem jsou *ethernetové adresy* a *adresy AX.25.*

ISDN Zkratka Integrated Services Digital Network. ISDN je standard, s jehož pomocí mohou telekomunikační společnosti doručovat hlasové nebo datové informace do domovů svých zákazníků. Z technického hlediska je ISDN kruhově přepínaná datová síť.

ISP Zkratka Internet Service Provider (Poskytovatelé internetových služeb). Jsou to organizace nebo společnosti, které lidem poskytují připojení k Internetu.

IP-adresa Jedná se o číslo, které slouží k jedinečné identifikaci TCP/IP-hostitele v síti. Adresa je 4 bajty dlouhá a zpravidla se zapisuje v tzv. tečkové notaci, kde je každý bajt uveden jako desítkové číslo a ta jsou vzájemně oddělená tečkami.

MSS Maximum Segment Size (maximální velikost segmentu) je největší objem dat, který lze najednou přenést. Chcete-li zabránit lokální fragmentaci, měla by být MSS rovna velikosti MTU IP-hlavičky.

MTU Maximum Transmission Unit (maximální přenosová jednotka) je parametr, který určuje největší datagram, který lze přenést přes IP-rozhraní, aniž by ho bylo nutné rozdělit na menší jednotky. Hodnota MTU by měla být větší než největší datagram, který chcete přenášet nefragmentován. Uvědomte si, že tak zabráníte pouze místní fragmentaci. Některá jiná linka totiž může mít nastavenu nižší hodnotu MTU a k fragmentaci datagramu pak dojde zde. Typické hodnoty jsou pro Ethernet 1500 bajtů a 576 bajtů pro rozhraní SLIP.

trasa (route) Trasa je cesta, po které putují datagramy síti ke svému cíli.

server Zpravidla se jedná o software na vzdáleném konci spojení. Server poskytuje určité služby jednomu nebo více klientům. Mezi servery patří *FTP*, *NFS* nebo *DNS*. V případě systémů *peer to peer*, jakým je například *SLIP* nebo *PPP*, je server na straně, která přijímá spojení.

okno Okno je největší množství dat, které je přijímací strana schopna v určitém okamžiku přijmout.

9 Linux pro ISP?

Pokud vás zajímá využití Linuxu pro účely ISP, doporučuji vám navštívit *Linux ISP homepage http://www.anime.net/linuxisp/,* kde najdete velký seznam odkazů na informace, které se vám mohou hodit.

10 Poděkování

Rád bych poděkoval následujícím lidem za jejich příspěvky do tohoto dokumentu (bez ohledu na pořadí): Terry Dawson, Axel Boldt, Arnt Gulbrandsen, Gary Allpike, Cees de Groot, Alan Cox, Jonathon Naylor, Claes Ensson, Ron Nessim, John Minack, Jean-Pierre Cocatrix, Erez Strauss.

Zvláštní díky patří Alessandru Rubinimu za jeho připomínky a korektury.

11 Autorská práva

Kapitola

2

Linux Intranet Server

Pramod Karnad, **karnad@indiamail.com** v2.11, 7. srpna 1997

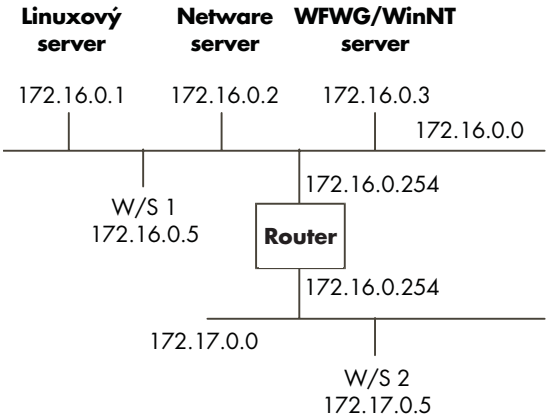
Tento dokument popisuje, jak nastavit Intranet s linuxovým serverem, který propojuje Unix, NetWare, NT a Windows. Tak po připojení na Linux získáte přístup ke všem různým platformám. Je zde podrobný popis k nastavení HTTP pomocí NCSA-serveru a k připojení k němu pomocí TCP/IP-klientů na Novellu; Microsoft pod Windows3.1, WFWG, Win95 a WinNT; a MacTCP na Apple PowerMac.

1 Úvod

Intranet je zjednodušeně řečeno implementací technologií Internetu v rámci nějaké uzavřené organizace bez nutnosti připojení ke globálnímu Internetu. Tato implementace se provádí takovým způsobem, aby bylo možné při minimálních nákladech, času a námaze poskytnout veškeré informační zdroje každému počítači. Tento dokument se pokouší osvětlit, jak vytvořit Intranet pomocí nástrojů, které jsou k dispozici, ale které jsou přesto velmi levné nebo jsou zcela zdarma.

Dokument předpokládá, že již víte, jak na vašem linuxovém serveru instalovat TCP/IP a jak jej fyzicky připojit pomocí ethernetové karty k vaší LAN. Předpokládá se zde také základní znalost systémů NetWare, WinNT a Mac. Konfigurace NetWare-serveru je zde ukázána na bázi verze 3.1x. Stejného výsledku dosáhnete za pomoci INETCFG. Na straně klienta se dokument zabývá systémy Windows 3.1x, Windows for Workgroups a Win95, WinNT a Apple PowerMac.

Soukromé síťové adresy (RFC-1918) 172.16.0.0 a 172.17.0.0 používám pouze jako příklady. V závislosti na vaší konfiguraci si můžete vybrat odpovídající adresy.



1.1 Co je nutné

Před instalací budete potřebovat následující software:

- Software HTTP-serveru, který je možné získat z One Step NCSA HTTPd Downloader na stránce *http://hoohoo.ncsa.uiuc.edu/docs/setup/OneStep.html.*[*]
- Novell NetWare Client, který je k dispozici na *http://support.novell.com/* (s klientem se dodávají i TCP/IP-soubory).
- Microsoft TCP/IP client, který je k dispozici na *http://www.microsoft.com/*.
- Apple MacTCP client, který je k dispozici na *http://www.apple.com/*.
- Prohlížeče WWW, jako je Netscape na *http://home.netscape.com/* nebo MS Internet Explorer na *http://www.microsoft.com/* nebo NCSA Mosaic na adrese *http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html.*

1.2 Nové verze tohoto dokumentu

Nové verze tohoto dokumentu budou pravidelně posílány do comp.os.linux.announce a comp.os.linux.help. Budou také nahrávány na různé ftp-servery s Linuxem, včetně sunsite.unc.edu. Poslední verze tohoto dokumentu je k dispozici ve formátu HTML na adrese *http://www.inet.co.th/cyberclub/karnadp/http.html*.

[*] Poznámka korektora: Vhodnějším HTTP-serverem je server Apache, který můžete získat na adrese `http://www.apache.org`.

1.3 Ohlasy

Jestliže máte k dokumentu dotazy nebo komentáře, nebojte se je odeslat Pramodu Karnadovi - **karnad@indiamail.com**. Návrhy, kritika a pošta jsou vždy vítány. Jestliže v dokumentu naleznete chyby, dejte mi prosím vědět, abych je mohl v příští verzi opravit. Díky.

2 Instalace HTTP-serveru

Když chcete získat server, máte dvě možnosti: získat zdroj a sami si jej zkompilovat nebo získat již zkompilované binární soubory. Binární soubory pro verzi Linux (ELF) jsou k dispozici v NCSA, ale ne starší verze.

2.1 Přípravy před stažením

Server NCSA vás provede kroky s konfiguračními volbami a připraví pro vás různé soubory. Před stažením HTTPd si ale připravte odpovědi na následující dotazy.

2.1.1 Operační systém

Nejprve musíte vybrat, jestli se má stahovat zdroj nebo předkompilovaná verze softwaru. Jestliže v menu není váš systém, musíte vzít zavděk implicitním zdrojem a sami jej zkompilovat.

Zjištění verze vašeho Linuxu provedete v jeho příkazovém řádku pomocí

```
linux:~$ uname -a
```

a získáte tak zhruba takovouto odpověď

```
linux:~$ uname -a
Linux 2.0.29 #4 Tue Sep 13 04:05:51 CDT 1994 i586
linux:~$
```

Zbývající parametry mohou být určeny před stažením nebo nakonfigurovány později úpravou souboru `srm.conf` v adresáři `/usr/local/etc/httpd/conf`. Názvy direktiv, které se právě ukazují v souboru `httpd.conf`, jsou v závorkách. Jedinou výjimkou je zde DocumentRoot, který se objevuje v souboru `srm.conf`.

2.1.2 Typ procesu (ServerType)

Zde se určí, jak na vašem stroji poběží HTTPd-server. Základní metodou je „standalone“ (samostatně). Takto běží HTTP-démon nepřetržitě. Jestliže nahrajete HTTPd pod „inetd“, bude binární soubor serveru nahráván do paměti při každé žádosti, což by mohlo server zpomalit.

2.1.3 Navázání portu (Port)

Určuje, na který port vašeho stroje se HTTPd-démon naváže a bude naslouchat HTTP-žádostem. jestliže se můžete přihlásit jako „root“, použijte implicitní nastavení 80. Jinak použijte hodnotu v rozsahu 1 025 až 65 536.

2.1.4 Identifikace uživatele na serveru (User)

Jedná se o id uživatele, na které se server změní při reakcích na žádosti a při práci se soubory. Tento dotaz zodpovídáte pouze pokud server používáte jako „standalone“. Jestliže nemáte rootovská práva, použijte vlastní uživatelské jméno. Jestliže jste systémovým správcem, můžete vytvořit speciálního uživatele, aby bylo možné kontrolovat přístupová práva k souborům.

2.1.5 Identifikace skupiny na serveru (Group)

Jedná se o id skupiny, na které se server změní při reakcích na žádosti a při práci se soubory. Je to podobné jako u Server User a používá se pouze pokud server používáte jako „standalone“.

Jestliže nemáte rootovská práva, použijte název vaší primární skupiny. Vaše skupiny zjistíte v Linuxu v příkazovém řádku pomocí příkazu **groups**.

2.1.6 E-mailová adresa administrátora serveru (ServerAdmin)

Toto je e-mailová adresa, na kterou by měl uživatel odesílat poštu s hlášením problémů na serveru. Sem můžete nastavit svoji osobní adresu.

2.1.7 Umístění adresáře serveru (ServerRoot)

Místo na vašem systému, na kterém sídlí server. Jestliže máte rootovská práva, ponechejte jej v doporučené podobě v adresáři `/usr/local/etc/httpd`. Jestliže se jako root přihlásit nemůžete, zvolte podadresář ve vašem domovském adresáři. Cestu k vašemu domovskému adresáři zjistíte příkazem `pwd`.

2.1.8 Umístění HTML-souborů (DocumentRoot)

Zde jsou uloženy HTML-soubory. Implicitní umístění je **/usr/local/etc/httpd/htdocs.** Můžete je nastavit do domovského adresáře speciálního uživatele, kterého vyberete ze Server User identify, nebo do podadresáře svého domovského adresáře, pokud se nemůžete přihlásit jako root.

Když nevíte, použijte implicitní nastavení. Nyní, když máte odpovědi na předchozí otázky, můžete si stáhnout NCSA HTTPd z *http://hoohoo.ncsa.uiuc.edu/docs/setup/OneStep.html.* Před instalací si na *http://hoohoo.ncsa.uiuc.edu/docs/* pročtěte dokumentaci k HTTPd. Jestliže se chystáte kompilovat kód, musíte upravit `Makefile` v každém z adresářů `support`, `src` a `cgi-src`. Jestliže je vaše verze Linuxu podporována, stačí vám v hlavním adresáři (např. `/usr/local/etc/httpd`) zadat `make linux`.

2.2 Kompilace HTTPd

Kompilace je snadná, zadejte v kořenovém adresáři serveru do příkazového řádku `make linux`.

Poznámka: Uživatelé předELFového Linuxu musí před kompilací HTTPd v souboru `portability.h` zrušit komentář u řádku `#define NO_PASS` a v souboru `Makefile` nastavit `DBM_LIBS= -ldbm`.

3 Testování HTTPd

Po instalaci HTTPd se přihlaště jako root a spustěte jej zadáním `httpd &` (předpokládáme instalaci jako standalone). Nyní by měl být vidět v seznamu příkazu `ps`. Nejjednodušším způsobem testování HTTPd je `telnet`. V příkazovém řádku Linuxu zadejte:

```
linux:~$ telnet 172.16.0.1 80
```

kde 80 je implicitní port pro HTTP. Jestliže jste „Port“ nastavili na jinou hodnotu, napište ji sem místo 80. Odpověď bude vypadat asi takto:

```
Trying 172.16.0.1...
Connected to linux.mydomain
Escape character is '^]'.
```

Nyní, když zadáte jakýkoliv znak a stisknete Enter, objeví se odpověď, podobná té následující.

```
HTTP/1.0 400 Bad Request
```

```
Date: Wed, 10 Jan 1996 10:24:37 GMT
Server: NCSA/1.5
Content-type: text/html

<HEAD><TITLE>400 Bad Request < /TITLE> < /HEAD>
<BODY><H1>400 Bad Request < /H1>
Your client sent a query that this server could
not understand.<P>
Reason: Invalid or unsupported method.<P>
</BODY>
```

Nyní jsme připraveni se na tento server připojit pomocí jiného PC a prohlížeče WWW.

4 Připojování na linuxový server

Použité adresovací schéma je vidět z diagramu v první části. Pracovní stanice 1 (W/S1) je na síti 172.16.0.0 a může se připojit na linuxový server přímo, zatímco pracovní stanice 2 (W/S2) je na síti 172.17.0.0 a musí pro připojení na Linux využít bránu (router) 172.17.0.254. Tato informace o bráně musí být použita pouze při konfiguraci klientů na W/S2. NetWare se na bránu odkazuje jako na „ip_router“ (ip-router).

Já pro ilustraci nastavení klienta použiji W/S2. Při nastavení W/S1 stačí zaměnit adresy 172.17.0.5 na 172.16.0.5 a ignorovat všechny odkazy na bránu/router.

Jestliže nemáte router, můžete tuto část přeskočit a postoupit až k

- 4.2, jestliže používáte NetWare server
- 4.4, jestliže používáte Microsoft Client

4.1 Nastavení linuxového serveru

Jestliže nemáte router, můžete tuto část vynechat.

Musíte nakonfigurovat linuxový server, aby rozpoznal router a umožnil W/S2-připojení na webový server. Aby bylo možné nastavit linuxový server, musíte se přihlásit jako root. Na příkazovém řádku serveru zadejte

```
route add gw default 172.16.0.254
```

Aby bylo možné tuto bránu využívat při každém zavedení systému linuxového serveru, můžete editovat soubor `/etc/rc.d/rc.inet1`* a změnit řádek, obsahující definici brány na `GATEWAY = „172.16.0.254“`. Tento řádek nesmí být v komentáři.

Nebo můžete směrování přidat do sítí na druhé straně routeru. To se provádí pomocí

```
route add -net 172.17.0.0 gw 172.16.0.254
```

Směrování pak přidáním příkazu do souboru `/etc/rc.d/rc.local` nastavíte pro každé zavedení systému.

4.2 Nastavení NetWare-serveru

Aby bylo možné nastavit NetWare-server, musíte mít přístupová práva Supervisor nebo alespoň Console operator. Jestliže je nemáte, musíte požádat vašeho síťového správce (Network Administrator), aby vám s nastavením pomohl. Serveru nastavte typ rámce LAN na Ethernet_II. K tomu použijte tyto příkazy (můžete je také vložit do souboru `AUTOEXEC.ncf`.

```
load NE2000 frame=Ethernet_II name=IPNET
load TCPIP
bind IP to IPNET addr=172.16.0.2 mask=FF.FF.FF.0
```

Při nahrávání ovladače NE2000 můžete v závislosti na konfiguraci vašeho stroje určit slot nebo číslo desky (například: `load NE2000 slot=3 frame=.....`).

4.3 Nastavení NetWare-klienta

Na PC máte možnosti Win3.1, WFWG nebo Win95. Instalační procedura se mezi Win95 a staršími verzemi liší podle toho, jestli používáte 32bitového klienta od Microsoftu nebo Novellu. Jestliže použijete 16bitového klienta, procedura bude stejná a vy se můžete řídit instalačními instrukcemi pro Windows 3.x. Při instalaci 32bitového klienta pro Win95 přejděte k části 4.3.2.

4.3.1 Windows 3.x

Jestliže používáte Win3.1 nebo WFWG, můžete nainstalovat NetWare Client (VLMs) a některé další soubory, které jsou na TCP/IP-disketě, jmenovitě

`TCPIP.exe`, `VTCPIP.386`, `WINSOCK.dll` a `WLIBSOCK.dll`

* Poznámka překladatele: V konkrétní distribuci může být umístění tohoto souboru jiné, případně mohou být soubory zcela jinak uspořádány.

Povšimněte si, že soubor se liší od stejného souboru z Win95 a Trumpet. Nainstalujte NetWare Client s podporou pro Windows. Okopírujte `VTCPIP.386`, `WINSOCK.dll` a `WLIBSOCK.dll` do adresáře `SYSTEM` a `TCPIP.exe` do adresáře `NWCLIENT`. Nyní upravte `STARTNET.bat` v adresáři `NWCLIENT` na

```
lsl
ne2000                      ---> ovladač vaší síťové karty
c:\windows\odihlp.exe       ----> pokud používáte WFWG
ipxodi
tcpip                       ---> doplňte tento řádek
nwip                        ---> pokud používáte NetWare/IP
vlm
```

Vytvořte podadresář (například) `\NET\TCP` a okopírujte soubory `HOSTS`, `NETWORKS`, `PROTOCOLS` a `SERVICES` z `/etc` vašeho linuxového serveru nebo adresáře `SYS:ETC` vašeho NetWare-serveru. Editujte okopírovaný soubor `HOSTS`, kam přidáte řádek pro váš nový linuxový server. Díky tomu se budete moci ve vašem prohlížeči WWW na linuxový server odkazovat jako na *http://linux.mydomain/* místo *http://172.16.0.1/*

```
127.0.0.1     localhost
172.16.0.1 linux.mydomain
```

Editujte v adresáři `NWCLIENT` soubor `NET.cfg`

```
Link Driver NE2000
port 300
int 3
MEM D0000
FRAME Ethernet_802.2

; ---- přidejte tyto řádky ----

FRAME Ethernet_II

Protocol TCPIP
PATH TCP_CFG C:\NET\TCP
ip_address 172.17.0.5
ip_netmask 255.255.255.0
ip_router 172.17.0.254      ---> přidejte adresu vaší brány pouze
```

```
                              ---> pokud musíte tuto bránu použít
                              ---> k dosažení vašeho HTTP-serveru

Link Support
MemPool 6192 ---> minimum je 1024. Zkuste jiné hodnoty.
Buffers 10 1580 ---> Tato hodnota může být opět vyladěna.

;---------------------------------
; Pokud používáte NetWare/IP, budete muset přidat následující řádky
;
NWIP
       NWIP_DOMAIN_NAME mydomain
       NSQ_BROADCAST ON
       NWIP1_1 COMPATIBILITY OFF
       AUTORETRIES 1
       AUTORETRY SECS 10
```

Editujte v adresáři `WINDOWS` soubor `SYSTEM.ini` a pro `VTCPIP.386` přidejte tento údaj

```
[386Enh]
.....
network=*vnetbios, vipx.386, vnetware.386, VTCPIP.386
.....
```

Proveďte restart vašeho PC, spusťte `STARTNET.bat` a nyní můžete k přístupu na vaše webové stránky používat svůj oblíbený prohlížeč WWW. Nepotřebujete se přihlašovat na NetWare a nemusíte spouštět `TCPMAN` (pokud používáte Trumpet Winsock).

4.3.2 Windows 95

Tato část vysvětluje, jak ve Win95 instalovat 32bitového klienta. Nejprve musíte nainstalovat následující:

```
Client for NetWare Networks (od Microsoftu nebo Novellu)
Microsoft TCP/IP Protocol
Network Adapter
```

Tyto položky instalujete po klepnutí na My Computer, Control Panel, Networks. Klepněte na Add. Nyní se ocitnete v okně, které zobrazuje Client, Adapter, Protocol a Service. Při instalaci Client for NetWare Networks:

```
1. Klepněte dvakrát na Client
2. Klepněte na Microsoft nebo Novell
3. Klepněte dvakrát na Client for NetWare Networks
```

Při instalaci TCP/IP Protocol:

```
1. Klepněte dvakrát na Protocol
2. Klepněte na Microsoft
3. Klepněte dvakrát na TCP/IP
```

Windows 95 implicitně automaticky instaluje několik dalších protokolů. Odstraníte je tak, že na ně klepnete a pak klepnete na tlačítko Remove. Win95 většinou instaluje protokol Microsoft NetBeui a kompatibilní protokol IPX/SPX. Protokol NetBEUI můžete smazat, ale protokol IPX/SPX budete potřebovat, pokud se budete chtít přihlásit na NetWare Server.

Nastavení TCP/IP provedete klepnutím na TCP/IP, klepnutím na Properties, klepnutím na tabulku IP-adress

```
V boxu Specify an IP address zadejte vaši IP-adresu 172.17.0.5
V boxu Subnet Mask zadejte 255.255.255.0
```

vyberte tabulku Gateway

```
V boxu New gateway zadejte adresu vaší brány 172.17.0.254
Klepněte na tlačítko Add
```

Adresa brány by se nyní měla objevit v boxu s instalovanými branami. Nyní klepněte na OK.

Obdržíte zprávu o restartu systému. Proveďte restart. Nyní budete moci k připojení na váš HTTP-server využívat prohlížeč.

4.4 Nastavení Microsoft Client

Jestliže pro přístup na síť používáte Microsoft Client, tato část popisuje, jak instalovat TCP/IP pro:

- 4.4.1 Windows for Workgroups
- 4.4.2 Windows 95
- 4.4.3 Windows NT

Poznámka: Abyste se ve svém prohlížeči WWW a všech internetových příkazech mohli na linuxový server odkazovat jako na *http://linux.mydomain/* místo *http://172.16.0.1/,* musíte editovat soubor `hosts`. Pro každý server (NetWare, Unix, WinNT) můžete přidat více údajů. Windows obsahují soubor `HOSTS` v adresáři `\WINDOWS` nebo `\WINDOWS\SYSTEM`, v závislosti na verzi. Editujte tento soubor a pro váš linuxový server přidejte řádek:

```
127.0.0.1 localhost
       172.16.0.1    linux.mydomain
       172.16.0.2    netware.mydomain
       172.16.0.3    winNT.mydomain
       172.16.0.5    ws_1
```

4.4.1 Windows for Workgroups

Tato část popisuje, jak instalovat 32bitového klienta na WFWG. Nejprve si musíte od Microsoftu obstarat ovladače TCP/IP pro Windows. Aktuální verzí je 3.11b a k dispozici je na adrese `ftp://ftp.microsoft.com` i na jiných místech, jako `tcp32b.exe`. Před nahráním 32bitového ovladače TCP/IP musíte mít ale nahrán Win32s.

Po vložení souborů TCP/IP do dočasného adresáře (například `C:\TEMP`) zkontrolujte adresář `\WINDOWS\SYSTEM`, zda neobsahuje kopie `OEMSETUP.INF`. Jestliže zde nějaké jsou, přejmenujte je. Nyní okopírujte soubor `OEMSETUP.INF` z adresáře TEMP do adresáře `\WINDOWS\SYSTEM`. Jestliže máte v systému jiné TCP/IP-ovladače, nejprve je odstraňte.

Spusťte nastavení Network Setup nebo Windows Setup/Change Network

```
Klepněte na tlačítko Networks
Klepněte na Install Microsoft Windows Network
Zvolte podporu pro další sítě (je-li to nutné)
Klepněte na OK
```

Měli byste být vyzváni k výběru vašeho síťového adaptéru - vyberte jej. Jestliže k výzvě nedojde, potom

```
Klepněte na tlačítko Adapter
       vyberte adaptér (řekněme NE2000)
       Klepněte na OK
Klepněte na tlačítko Protocol
       vyberte protokol MS TCP/IP-32
       klepněte na OK
```

Nyní budete vyzváni ke konfiguraci sady TCP/IP-protokolů. Vždy je můžete překonfigurovat pomocí nastavení TCP/IP-protokolu v boxu Adapters a klepnutí na tlačítko Setup.

```
V boxu IP address zadejte 172.17.0.5
V boxu Subnet Mask zadejte 255.255.255.0
V boxu Default gateway zadejte adresu vaší brány 172.17.0.254
```

Klepněte na OK. Obdržíte zprávu o restartu systému. Proveďte restart. Nyní budete moci k připojení na váš HTTP server využívat prohlížeč.

4.4.2 Windows 95

Tato část popisuje, jak instalovat 32bitového klienta pro Microsoft na Win95. Nejprve musíte nainstalovat následující

```
Client for Microsoft Networks
Microsoft TCP/IP Protocol
Network Adapter
```

Tyto položky instalujete po klepnutí na My Computer, Control Panel, Networks. Klepněte na Add. Nyní se ocitnete v okně, které zobrazuje Client, Adapter, Protocol a Service. Při instalaci Client for Microsoft Networks:

```
1. Klepněte dvakrát na Client
2. Klepněte na Microsoft
3. Klepněte dvakrát na Client for Microsoft Networks
```

Při instalaci TCP/IP Protocol:

```
1. Klepněte dvakrát na Protocol
2. Klepněte na Microsoft
3. Klepněte dvakrát na TCP/IP
```

Windows 95 implicitně automaticky instaluje několik dalších protokolů. Odstraníte je tak, že na ně klepnete a pak klepnete na tlačítko Remove. Win95 většinou instaluje protokol Microsoft NetBeui.

Nastavení TCP/IP provedete klepnutím na TCP/IP, klepnutím na Properties, klepnutím na tabulku IP-adress

```
V boxu Specify an IP address zadejte vaši IP-adresu 172.17.0.5
V boxu Subnet Mask zadejte 255.255.255.0
```

vyberte tabulku Gateway

```
V boxu New gateway zadejte adresu vaší brány 172.17.0.254
Klepněte na tlačítko Add
```

Adresa brány by se nyní měla objevit v boxu s instalovanými branami. Nyní klepněte na OK.

Obdržíte zprávu o restartu systému. Proveďte restart. Nyní budete moci k připojení na váš HTTP-server využívat prohlížeč.

4.4.3 Windows NT

Tato část popisuje instalaci TCP/IP-klienta pro WinNT 4.0. Spusťte Control Panel/Network

```
Vyberte tabulku Adapter.
Klepněte na Add (jestliže ještě nemáte, tak přidáte nový adaptér)
```

Měli byste být vyzváni k výběru vašeho síťového adaptéru - vyberte jej. Pro přidání protokolů:

```
Vyberte tabulku protocols
       Klepněte na Add
       Vyberte protokol TCP/IP
       Klepněte na OK
```

Nyní budete vyzváni ke konfiguraci sady TCP/IP-protokolů. Vždy je můžete překonfigurovat pomocí nastavení TCP/IP-protokolu a klepnutí na tlačítko Properties.

```
Vyberte tabulku IP Address
       Označte box 'Specify an IP address'
       V boxu IP address zadejte 172.17.0.5
       V boxu Subnet Mask zadejte 255.255.255.0
       V boxu Default gateway zadejte adresu vaší brány 172.17.0.254
```

Klepněte na OK. Obdržíte zprávu o restartu systému. Proveďte restart. Nyní budete moci k připojení na váš HTTP-server využívat prohlížeč.

4.5 Nastavení TCP/IP na počítači Macintosh

Jestliže pro přístup na síť používáte Macintosh, tato část popisuje, jak instalovat MacTCP na PowerMac.

Poznámka: Abyste se ve svém prohlížeči WWW a všech internetových příkazech mohli na linuxový server odkazovat jako na *http://linux.mydomain/* místo *http://172.16.0.1/,* musíte editovat soubor `hosts`. Formát souboru `hosts` se liší od formátu, který používá Unix. Soubor `hosts` je zde vytvořen na základě RFC-1035. Pro každý server (NetWare, Unix, WinNT) můžete přidat více údajů. MacOS ukládá soubor `HOSTS` do **Preferences folder** pod **System folder**. Editujte tento soubor a pro váš linuxový server přidejte řádek:

```
linux.mydomain      A  172.16.0.1
netware.mydomain    A  172.16.0.2
winNT.mydomain      A  172.16.0.3
ws_1                A  172.16.0.5
```

4.5.1 MacTCP

Tato část popisuje instalaci MacTCP. Nejprve musíte od Applu získat soubory MacTCP nebo je nainstalovat z Internet Connection CD. Při konfiguraci MacTCP klepněte na Apple Menu/ Control Panels/ TCP/IP. Na obrazovce změňte nastavení pro „Connect via:“ na „Ethernet“.

Změňte nastavení „Configure“ na „Manually“.

```
V boxu IP address zadejte 172.17.0.5
V boxu Subnet Mask zadejte 255.255.255.0
V boxu Router address zadejte adresu vaší brány 172.17.0.254
```

Klepněte na OK. Nyní budete moci k připojení na váš HTTP-sserver využívat prohlížeč.

5 Nastavení Intranetu

Intranet by nebyl Intranetem bez sdílení zdrojů na různých platformách. Budete potřebovat podporu pro další systémy souborů, aby bylo možné využívat data, která jsou na nich k dispozici. Tento dokument nabízí instrukce pro připojení k těmto populárním systémům souborů.

- 5.1 NCPFS pro NetWare
- 5.2 SMBFS pro Windows
- 5.3 NFS pro Unix

Tyto systémy souborů mohou být kompilovány do jádra Linuxu nebo přidány jako moduly, v závislosti na verzi Linuxu. Jestliže nejste v kompilaci jádra příliš sběhlí, můžete se více do-

zvědět z dokumentu o jádru a o modulech. Tyto moduly jsou k dispozici na adrese *http://sunsite.unc.edu/mdw/HOWTO/Kernel-HOWTO.html* a *http://sunsite.unc.edu/mdw/HOWTO/Module-HOWTO.html.*

5.1 Ncpfs

Při sdílení souborů na NetWare-serveru budete potřebovat podporu pro NCP (ncpfs). Ncpfs funguje s jádrem verze 1.2.x a od 1.3.71 výše. Nefunguje se starými jádry 1.3.x. Nevyužívá NDS-databázi NetWare 4.0, ale dokáže využít bindery. Jestliže používáte NetWare 4.x, můžete na konzole na určitých místech pomocí příkazu `Set Bindery Context` umožnit podporu bindery:

```
set Bindery Context = CORP.MYDOM;WEBUSER.MYDOM
```

V předchozím případě byla podpora bindery umožněna na dvou místech.

Z URL *ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/ncpfs.tgz* (aktuálně `ncpfs-2.0.10`) ze Sunsite musíte získat utility pro ncpfs.

5.1.1 Instalace

Utility `ncpfs` nainstalujete zadáním

```
zcat ncpfs.tgz | tar xvf -
```

soubory se tak vloží do vlastního adresáře. V tomto případě budete mít adresář `ncpfs-2.0.10`. Před započetím instalace sem změňte aktuální adresář. Pročtěte si `README` a je-li to nutné, editujte `Makefile`.

Instalace ncpfs závisí na verzi jádra, kterou používáte. Pro jádro 1.2 stačí zadat „`make`“. Následné „`make install`“ nainstaluje spustitelné soubory a manuálové stránky.

Jestliže používáte Kernel 1.3.71 nebo pozdější, možná budete muset svoje jádro překompilovat. U těchto jader je část jádra pro `ncpfs` již začleněna v hlavním zdrojovém stromu. Jestli jádro musíte překompilovat, zjistíte zadáním

```
cat /proc/filesystems
```

Měl by se objevit řádek, sdělující, že jádro zná `ncpfs`.

Jestliže zde `ncpfs` není, můžete buď překompilovat jádro, nebo přidat `ncpfs` jako modul. Při kompilaci jádra byste měli zadat „`make config`“ a jakmile se vás dotáže na

```
The IPX protocol (CONFIG_IPX) [N/y/?]
```

odpovězte jednoduše „y“. Plnou vnitřní síť, která bude v dotazech následovat, pravděpodobně potřebovat nebudete. Jakmile je jádro úspěšně nainstalováno, proveďte restart systému, zkontrolujte `/proc/filesystems`, a jestliže je vše v pořádku, pokračujte v instalaci utilit ncpfs. Změňte aktuální adresář na adresář s přetaženými soubory ncpfs a zadejte „`make`“. Po dokončení kompilace zadejte „`make install`“, čímž se nainstalují různé utility a manuálové stránky.

5.1.2 Připojení Ncpfs

Kontrolu instalace proveďte pomocí

```
ipx_configure --auto_interface=on --auto_primary=on
```

```
....počkejte 10 sekund a napište
```

```
slist
```

Měli byste být schopni vidět seznam vašich NetWare-serverů. Nyní jsme připraveni ke sdílení souborů z NetWare-serveru.

Předpokládejme, že potřebujeme využít HTML-soubory z adresáře `\home\htmldocs` z `VOL1`: na serveru MYDOM_NW. Doporučuji na tomto serveru vytvořit nového uživatele (například) „EXPORT“ s heslem „EXP123“, kterému pomocí `SYSCON` nebo `NWADMIN` dáte příslušná přístupová práva do tohoto adresáře.

Na linuxovém stroji vytvořte nový adresář `/mnt/MYDOM_NW`. Nyní zadejte příkaz

```
ncpmount -S MYDOM_NW -U EXPORT -P EXP123 /mnt/MYDOM_NW
```

kterým se připojí systém souborů Netware. Zadáním příkazu

```
ls /mnt/MYDOM_NW/vol1/home/htmldocs
```

zobrazíte seznam všech souborů v `MYDOM_NW/VOL1:\HOME\HTMLDOCS` (v systému pojmenování souborů podle NetWare). Jestliže máte nějaké problémy, pročtěte si dokument o IPX, který naleznete na *http://sunsite.unc.edu/mdw/HOWTO/IPX-HOWTO.html.*

5.2 Smbfs

Aby bylo možné sdílet soubory na serveru Windows, budete potřebovat podporu pro SMB (smbfs).

Z *ftp://sunsite.unc.edu/pub/Linux/system/filesystems/smbfs/smbfs.tgz* (aktuálně smbfs-2.0.1) ze Sunsite musíte získat utility pro smbfs.

5.2.1 Instalace

Utility nainstalujete zadáním

```
zcat smbfs.tgz | tar xvf -
```

soubory se tak vloží do vlastního adresáře. V tomto případě budete mít adresář `smbfs-2.0.10`. Před započetím instalace sem změňte aktuální adresář. Pročtěte si `README` a je-li to nutné, editujte `Makefile`.

Instalace ¨smbfs závisí na verzi jádra, kterou používáte. Pro jádro 1.2 stačí zadat „`make`“. Následné „`make install`“ nainstaluje spustitelné soubory a manuálové stránky.

Jestliže používáte Kernel 2.0 nebo pozdější, možná budete muset svoje jádro překompilovat. U těchto jader je jejich část pro smbfs již začleněna v hlavním zdrojovém stromu. Zda jádro musíte překompilovat, zjistíte zadáním

```
cat /proc/filesystems
```

Měl by se objevit řádek, sdělující, že jádro zná smbfs.

Jestliže zde smbfs není, můžete buď překompilovat jádro, nebo přidat smbfs jako modul. Při kompilaci jádra byste měli zadat „`make config`“ a jakmile se vás dotáže na podporu SMBFS, odpovězte jednoduše „`y`“. Jakmile je jádro úspěšně nainstalováno, proveďte restart systému, zkontrolujte `/proc/filesystems`, a jestliže je vše v pořádku, pokračujte v instalaci utilit smbfs. Změňte aktuální adresář na adresář s přetaženými soubory smbfs a zadejte „`make`“. Po dokončení kompilace zadejte „`make install`“, čímž se nainstalují různé utility a manuálové stránky.

5.2.2 Připojení smbfs

V našem příkladu předpokládejme, že WinNT server se nazývá „MYDOM_NT“ a sdílí svůj adresář `C:\PUB\HTMLDOCS` se sdíleným názvem „`HTMLDOCS`“ a bez hesla. Na linuxovém stroji vytvořte nový adresář `/mnt/MYDOM_NT`. Nyní zadejte příkaz

```
smbmount //MYDOM_NT/HTMLDOCS /mnt/MYDOM_NT -n
```

kterým se připojí smbfs (sdílení ve Windows). Jestliže příkaz nefunguje, vyzkoušejte

```
smbmount //MYDOM_NT/COMMON /mnt/MYDOM_NT -n -I 172.16.0.3
```

Zadáním příkazu

```
ls /mnt/MYDOM_NT
```

zobrazíte seznam všech souborů v `\\MYDOM_NT\PUB\HTMLDOCS` (v systému pojmenování souborů podle Windows).

5.3 NFS

Nejprve budete potřebovat jádro s NFSFS buď zakompilovaným, nebo k dispozici jako modul.

Předpokládejme, že máte server s Unixem, na kterém běží NFS s názvem MYDOM_UNIX a IP-adresou 172.16.0.4. Kontrolu zde exportovaných (sdílených) adresářů provedete zadáním příkazu

```
showmount -e 172.16.0.4
```

Jakmile známe exportované adresáře, můžeme je připojit zadáním příslušného příkazu `mount`. Doporučuji pod „`/mnt`“ vytvořit podadresář (řekněme) „`MYDOM_UNIX`“ a využít jej jako vaše připojovací centrum.

```
mount -o rsize=1024,wsize=1024 172.16.0.4:/pub/htmldocs\
  /mnt/MYDOM_UNIX
```

`Rsize` a `wsize` mohou být v závislosti na vašem prostředí změněny.

Jestliže máte nějaké problémy, přečtěte si dokument k NFS, který je k dispozici na *http://sunsite.unc.edu/mdw/HOWTO/NFS-HOWTO.html.*

6 Přístup na Web

Nyní, když máme nastavený HTTP-server, klienty a propojili jsme linuxový server s ostatními servery, musíme na linuxovém serveru provést některé menší úpravy, aby bylo možné přistupovat k těmto systémům souborů i z WWW-prohlížeče.

6.1 Přístup k připojeným systémům souborů

Při přístupu k připojeným adresářům na vaší stránce HTML máte dvě možnosti:

Vytvořit odkaz (link) na DocumentRoot (`/usr/local/etc/httpd/htdocs`), aby se odkazoval na připojený adresář jako

```
ln -s /mnt/MYDOM_NW/vol1/home/htmldocs netware
```

nebo

```
ln -s /mnt/MYDOM_NT winNT
```

nebo

```
ln -s /mnt/MYDOM_UNIX unix
```

Editovat ve vašem adresáři `/usr/local/etc/httpd/conf` soubor `srm.conf` a přidat nový alias.

```
# Přezdívka pro skutečné jméno
Alias /icons/ /usr/local/etc/httpd/icons/

# alias pro Netware server
Alias /netware/ /mnt/MYDOM_NW/vol1/home/htmldocs/
Alias /winNT/ /mnt/MYDOM_NT/
Alias /unix/ /mnt/MYDOM_UNIX
```

A znovu spustit váš server HTTPd. K dokumentům na serveru NetWare přistoupíte pomocí *http://linux.mydomain/netware/index.htm* podobně u ostatních.

6.2 Připojení k Internetu

Svůj Intranet můžete nakonec připojit k Internetu, čímž využijete e-mail a všechny ty báječné informace, co tam jsou. Stručnou poznámku k této problematice pravděpodobně připíši v některých dalších verzích dokumentu. Podrobný popis naleznete v dokumentech ISP Hookup na adrese *http://sunsite.unc.edu/mdw/HOWTO/ISP-Hookup-HOWTO.html* a Diald na adrese *http://sunsite.unc.edu/mdw/HOWTO/mini/Diald.*

6.3 Další využití

Server HTTP je možné využít na úřadu, aby poskytl přímý přístup k informacím na různých serverech, místech a adresářích. Data mohou být jednoduché dokumenty Wordu, tabulky Lotusu nebo složité databáze.

Aplikace této technologie jsou většinou následující:

- Vydávání dokumentů v organizaci. Tyto dokumenty mohou obsahovat vývěsky, roční zprávy, mapy, budovy v organizaci, ceníky, literaturu s informacemi o výrobcích a jakékoliv dokumenty, které mohou mít v rámci organizace nějakou hodnotu.
- Přístup k adresářům, ve kterých je možné vyhledávat. Rychlý přístup k telefonním seznamům v organizaci nebo podobným věcem. Tato data je možné zobrazit na serveru WWW nebo může server WWW (pomocí skriptu CGI) sloužit jako brána k připravovaným nebo novým aplikacím. To znamená, že využitím stejného standardního přístupového mechanismu může být informace lépea snadněji k dispozici. To znamená, že je možné pro generování informací v reálném čase vytvořit rozhraní s RDBMS, jako ORACLE, a SYBASE. Zde je seznam odkazů na takové servery WWW.

  - Web Access - *http://cscsun1.larc.nasa.gov/~beowulf/db/web_access.html/* - CGI gateways - *http://www.w3.org/hypertext/WWW/RDBGate/Overview.html/*
- Stránka organizace/oddělení/osobní. Jak se mění v organizaci přístup směrem k větší samostatnosti oddělení, technologie Intranetu nabízí ideální médium pro rozšíření aktuálních informací oddělením i jednotlivcům. Mocné vyhledávací procedury nabízí služby lidem, kteří hledají skupinu nebo jednotlivce s odpovědí na běžné každodenní otázky, vznikající při chodu organizace.
- Jednoduché aplikace Groupware. S podporou HTML mohou servery poskytovat podpisové archy, průzkumy a jednoduché plánování.
- Distribuce softwaru. Administrátoři mohou Intranet využít k předávání software a aktualizací uživatelům v rámci organizace. K tomu slouží „Java“, umožňující vytvoření a přímé dodání požadovaných objektů, ne tedy jen dat a aplikací. To je více podporováno v nových verzích Linuxu se zabudovanou podporou Javy.
- Pošta. S přesunem k využívání intranetových poštovních produktů se standardními a jednoduchými metodami přiřazení dokumentu, zvuku, videa a dalších multimédií mezi jednotlivci, se pošta stává jednoduchou komunikační metodou. Pošta je brána zejména jako komunikace mezi jednotlivci nebo jednotlivcem a malou skupinou. K nastavení systému elektronické pošty je pro Linux k dispozici několik systémů, jako jsou `sendmail`, `pop3d`, `imapd`.
- Uživatelské rozhraní. Technologie Intranetu se vyvíjí tak rychle, že použité nástroje (zde HTML) je možné využít k podstatné změně naší komunikace se systémy. Pomocí HTML můžete vytvořit rozhraní, ovlivněné pouze představivostí tvůrce. Na technologiích Intranetu je pěkné to, že se snadno používají. Klepnutím na hyperlink se můžete v HTML přesunout na další stránku, spustit poplach, spustit několikaměsíční proceduru nebo cokoliv jiného, co zvládají počítačové programy.

7 Další možnosti

Následuje seznam dalších zajímavých věcí, které je možné s vaším linuxovým intranetovým serverem provádět. Veškerý níže zmíněný software spadá do kategorie free software nebo sharewaru.

- Prohlížení linuxového serveru pomocí Network Neighbourhood ve Win95/NT; Nastavte server NBT, podobný WINS. Projděte si stránku WWW SAMBA na *http://lake.canberra.edu.au/pub/samba/samba.html.*
- Implementace vyhledávacích procedur na vašem Intranetu. Připojte se k ht://Dig na *http://htdig.sdsu.edu/.*
- Využití CUSeeMe nastavením lokálního reflektoru viz domovské stránky v Cornellu na http://cu-seeme.cornell.edu/.
- Založení WWW-konferencí. Použijte COW z *http://thecity.sfsu.edu/COW/.*
- Založení databáze SQL. Viz domovská stránka mSQL na *http://Hughes.com.au/.*
- Nastavení FTP, Gopher, Finger, Bootp-serverů na serveru NetWare. Získáte je na *http://mft.ucs.ed.ac.uk/.*
- Emulace serveru NetWare. Vyzkoušejte NCP Utilities na internetové adrese *ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/.*

Jestliže naleznete další možnosti využití pro linuxový intranetový server, nebojte se mi svoje návrhy zaslat.

8 Autorské a právní záležitosti

8.1 Poděkování

Děkuji lidem z NCSA za to, že mi poskytli perfektní dokumentaci. Davidu Andersonovi a všem ostatním za to, že dokument přečetli a poslali své komentáře. Podrobnosti o NetWare/IP pochází od Romela Florese (**rom@mnl.sequel.net**).

8.2 Informace o autorských právech

- Dokumenty projektu Linux HOWTO (jak na to) mohou být kopírovány a šířeny celé nebo i částečně, na jakémkoliv fyzickém i elektronickém médiu, ale tato poznámka o autorských právech zde musí vždy zůstat zachována. Komerční šíření je povoleno a vítáno; autor by ale na takové šíření byl rád upozorněn.
- Všechny překlady, odvozené práce nebo výtahy dokumentů Linux HOWTO musí být prováděny s dodržením této poznámky o autorských právech. To znamená, že nemůžete vytvářet od těchto dokumentů odvozenou práci a uvalit na ni dodatečná omezení šíření. Výjimky z tohoto pravidla mohou být poskytnuty za určitých podmínek; zde prosím kontaktujte koordinátora celého projektu na níže uvedené adrese.
- Jestliže máte dotazy, kontaktujte prosím na adrese **gregh@sunsite.unc.edu** Grega Hankinse, koordinátora pro Linux HOWTO. Telefonní číslo a adresy získáte pomocí `finger`.

Kapitola

3

Elektronická pošta a Linux

Guylhem Aznar **<guylhem at danmark.linux.eu.org>** verze 2.0, leden 1998

Tento dokument popisuje nastavení, údržbu a provoz elektronické pošty pod operačním systémem Linux. Tento dokument byste si měli pročíst v případě, že budete lokálně nebo přes síť posílat elektronickou poštu. Pokud elektronickou poštu nepoužíváte (ať už mezi uživateli jednoho systému nebo mezi uživateli různých systémů), dokument si číst **nemusíte**.

1 Úvod, autorská práva a zodpovědnost za správnost

1.1 E-mail

Jestliže se kdekoliv v elektronické adrese vyskytne řetězec „at“, nahraďte jej znakem „@“.

Pro lidi je to jednoduché, ale ne pro roboty prohledávající WWW; proto je to dostatečná ochrana před nevyžádanou poštou pro přispěvatele tohoto dokumentu.

1.2 Zaměření

Účelem tohoto dokumentu je nalézt odpovědi na některé často se vyskytující otázky a komentáře (FAQ) obecně k aplikacím elektronické pošty pro Linux a specificky k verzím pro distribuce Linux Debian a RedHat.

1.3 Nové verze

Nové verze tohoto dokumentu budou pravidelně posílány do diskusních skupin **comp.os.linux.announce**, **comp.answers** a **mail.answers**. Objeví se také na různých anonymních ftp-serverech, které archivují takový typ informací.

Kromě toho by mělo být vždy možné nalézt tento dokument na domovské stránce WWW pro Linux, která je na *http://sunsite.unc.edu/mdw/linux.html.*

1.4. Ohlasy

Zajímají mě jakékoliv vaše ohlasy (elektronickou poštou), týkající se tohoto dokumentu, kladné i záporné. Určitě mi dejte vědět, pokud objevíte chyby nebo zjevné překlepy.

Všechny dopisy obdržené elektronickou poštou, si přečtu, ale ne na všechny odpovídám. Žádosti budou zvažovány a vyřizovány v závislosti na mém volném čase, na stupni žádosti a na mém krevním tlaku :-).

Co se mi nebude líbit, to skončí v `/dev/null`, takže můžete být klidní.

Ohlasy na formát dokumentu by měly být odesílány koordinátorovi projektu HOWTO: Gregu Hankinsovi (**gregh at sunsite.unc.edu**).

1.5 Autorská práva

Vlastníkem autorských práv k dokumentu Mail-HOWTO je (c) 1998 Guylhem Aznar. Distribuce se provádí v licenci LDP. Všechny dotazy prosím směřujte na koordinátora Linux HOWTO - Grega Hankinse (**gregh at sunsite.unc.edu**).

1.6. Omezení záruky

Samozřejmě se distancuji od jakékoliv zodpovědnosti za obsah tohoto dokumentu. Použití návodů, příkladů a jiného obsahu dokumentu provádíte zcela na vlastní riziko.

2 Další zdroje informací

2.1 USENET

Na konfiguraci a provozu aplikací elektronické pošty pro Linux není nic „specifického“ (jednou provždy). Vzhledem k tomu *NEPOSÍLEJTE* dotazy, týkající se využití elektronické pošty, do skupin **comp.os.linux**.*

Do hierarchie **comp.os.linux** posílejte pouze dotazy, specifické pro Linux, jako jsou: „Se kterými volbami byl kompilován Debian 1.2 sendmail?“ nebo „RedHat 5.0 smail se po spuštění zhroutí.“

Dovolte mi to ještě zopakovat.

Pro posílání čehokoliv, co se týká pošty, do hierarchie **comp.os.linux** již není prakticky žádný důvod. Na *VŠECHNY* vaše otázky by měly stačit skupiny v hierarchii comp.mail.*

JESTLIŽE POSÍLÁTE NA COMP.OS.LINUX. OTÁZKY, KTERÉ SE NETÝKAJÍ KONKRÉTNĚ LINUXU, HLEDÁTE POMOC NA ŠPATNÉM MÍSTĚ. ODBORNÍCI NA ELEKTRONICKOU POŠTU VYŘIZUJÍ DOTAZY NA MÍSTECH, ZMÍNĚNÝCH VÝŠE, PŘIČEMŽ NEMUSÍ NUTNĚ POUŽÍVAT LINUX.*

POSÍLÁNÍM OTÁZEK, KTERÉ SE NETÝKAJÍ KONKRÉTNĚ LINUXU, DO LINUXOVÉ HIERARCHIE JEDINĚ MARNÍTE SVŮJ A CIZÍ ČAS, NAVÍC SE TÍM PRODLOUŽÍ VAŠE ČEKÁNÍ NA ODPOVĚĎ.

PATŘIČNÁ MÍSTA jsou:

comp.mail.elm	poštovní systém `ELM`
comp.mail.mh	systém Rand Message Handling
comp.mail.mime	Multipurpose Internet Mail Extensions
comp.mail.misc	všeobecná diskuze o počítačové poště
comp.mail.multi-	media Multimedia Mail
comp.mail.mush	Mail User's Shell (MUSH)
comp.mail.sendmail	BSD `sendmail`
comp.mail.smail	`smail`
comp.mail.uucp	pošta v prostředí uucp

2.2 Emailové konference

Pro `sendmail`, `smail` a `qmail` existuje velké množství e-mailové konference.

Adresy je možné nalézt v `/usr/doc/tenkterýjstesizvolil`.

2.3. Další dokumenty z LDP

V jiných dokumentech Linux HOWTO a v projektu Linux DOC je k dispozici množství zajímavého materiálu.

Konkrétně se můžete podívat na následující:

- na vašem vlastním počítači je adresář `/usr/doc/smail` nebo `/usr/doc/sendmail` :-)
- Příručka správce sítě
- Serial Communications HOWTO (komunikace po sériové lince)
- Ethernet HOWTO
- UUCP HOWTO, jestliže používáte UUCP

2.4 Knihy

Následuje sada knih, které by vám mohly být užitečné:

- **„Managing UUCP and USENET“** od O'Reilly and Associates je podle mého názoru nejlepší knihou o programech a protokolech, používaných při provozu serveru pro USENET.
- **„Unix Communications“** od The Waite Group obsahuje perfektní popis všech součástí a způsobu jejich spojení.
- **„Sendmail“** od O'Reilly and Associates je nejlepší příručkou pro sendmail-v8 a sendmail+IDA. Nutná koupě pro každého, kdo chce využívat sendmail a nehodlá se učit až z vlastních chyb při jeho provozu.
- **„The Internet Complete Reference“** od Osborne je dobrou příručkou, vysvětlující různé služby, dostupné na Internetu, a současně je skvělým zdrojem informací o různých zdrojích Internetu, jako jsou news, a elektronická pošta.
- **„Příručka správce sítě“** od Olafa Kircha z Linux Documentation Project je k dispozici na síti a určitě ji vydává O'Reilly a SSC.

Je dobré si zjistit vše, co vlastně o síťovém používání systému Unix potřebujete vědět.

3 Požadavky

3.1 Hardware

Pro používání pošty v Linuxu neexistují žádné specifické požadavky.

Budete potřebovat nějaký druh „transportních“ programů, používaných pro připojení ke vzdáleným systémům, což zde znamená buď TCP/IP, nebo UUCP.

To znamená, že v závislosti na nastavení budete potřebovat modem nebo ethernetovou kartu.

Ve většině případů je vhodné mít nejrychlejší dostupný modem, dnes se jedná o modem s přenosovou rychlostí 57 600 bps. Obecně je vhodné mít na sériovém portu nebo přímo v modemu 16550 UART, aby bylo možné zvládnout rychlosti nad 9 600 baudů.

Jestliže nechápete smysl poslední věty, informace si zjistěte ve skupině **comp.dcom.modems** nebo využijte USENET a jeho FAQ a periodicky zasílané informace, týkající se komunikace pomocí modemu a sériové komunikace.

3.2 Software

Problém zní: K čemu bude váš poštovní software sloužit?

1. Počet hostitelů

   Využíváte více než 100 hostitelů se složitými volbami pro názvy domén? *Zvolte* `sendmail`*!*

   Využíváte méně než 100 hostitelů, přičemž s názvy domén se moc nezatěžujete? *Zvolte* `smail`*!*

2. Zabezpečení

   Využíváte více nebo méně než 100 hostitelů, ale s vysokým zabezpečením? *Zvolte* `qmail`*!*

   Využíváte méně než 100 hostitelů, ale se standardní úrovní zabezpečení? *Zvolte* `smail`*!*

3. Různé způsoby získávání pošty

   Poštu získáváte a odesíláte přes UUCP nebo FIDO (pomocí ifmail)? Poštu získáváte a odesíláte pomocí POP a internetového SMTP? *Zvolte* `smail`*!*

Samozřejmě, že ve volbě poštovního software záleží rozhodnutí na vás. Předchozí informace vám mohou toto rozhodnutí usnadnit. `Sendmail` je vhodný pro větší množství hostitelů se složitými volbami, `qmail` zajišťuje vysoké zabezpečení; mezi `sendmailem` a `qmailem` leží kompromis, `smail`. Jestliže víte co děláte, zvolte `sendmail` (pak asi nebudete číst tento dokument); obecně však doporučuji `smail`.

4 Smail verze 3.1

`Smail 3.1` je pro hostitele UUCP a pro některé hostitele SMTP vlastně téměř standardním přenosovým agentem. Snadno se konfiguruje, kompiluje se bez připojení zdrojů a jeho zabezpečení je dostatečné.

4.1 Konfigurování smailu

Nainstalujte si binární soubor pro `smail` z vaší distribuce Linuxu (doporučuji) nebo si získejte zdroje pro `smail` a vytvořte jej. Jestliže `smail` vytváříte ze zdrojových textů, musíte mít ve vašem souboru `os/linux` následující řádek (aby „`sed`“ dával scripty příkazového interpretu, které správně fungují).

```
CASE_NO_NEWLINES=true
```

Po nainstalování se do `/etc/smail` zapíší konfigurační soubory (pokud používáte starší verze zdroje Linuxu, situace se může lišit). Můžeme začít s jejich editací!

4.2 Soubor config

```
# Odkud
smart_path=polux
smart_transport=uux
# Na
hostname=danmark
domains=linux.eu.org
visible_name=danmark.linux.eu.org
uucp_name=danmark.linux.eu.org
# max_message_size=512k
# auth_domains=foo.bar
# more_hostnames=barberouge:barberouge.polux.freenix.fr
```

Takže zaprvé, kdo vám poštu doručuje? U mě je to „polux“ přes UUCP (transport pomocí `uux`); tento soubor si přirozeně změňte podle vlastní situace. Poštu vám může doručovat například „bargw.bar.foobar.com“ přes „smtp“, přičemž zde nepotřebujete transportní soubor, což definujete pomocí „`-transport_file`“.

Můžete také použít „`postmaster_address = vaše_jméno`“, pomocí „`visible_name`“ skrýt síťovou topologii v adresách pro odesílání (jestliže jste brána) a pomocí „`more_hostnames`“ nastavit, které přezdívky mají být využity i pro doručenou poštu.

Více podrobností naleznete v dokumentaci pro `smail` nebo si prohlédněte příklady z `/usr/doc/smail/examples`, jestli vám náhodou některý nevyhovuje.

4.3 Soubor directors

```
#  aliasinclude -  expandovat ":include:filename"  adresy, vytvořené
#  aliasovými soubory. Tento a následující údaj je v podstatě
#  dost běžný.
#  K provedení větších úprav je jen málo důvodů, snad jen úprava
#  adres ve formě:
#      :include:pathname
#  které se mohou objevit v aliasových souborech nebo poštovních
#  seznamech/předávacích souborech.
aliasinclude:
  driver = aliasinclude,   # použijte tento ovladač speciálních znaků
  nobody;                  # přiřazení uživatele nobody (nikdo)
                           # v případě narušení přístupových práv
  copysecure,              # získání práv z alias directoru
  copyowners,              # získání vlastníků z alias directoru
# forwardinclude - expandovat ":include:filename" adresy, vytvořené
# předávacími soubory
forwardinclude:
  driver = forwardinclude, # použijte tento ovladač speciálních znaků
  nobody;
  copysecure,              # získání práv z předávacího directoru
  copyowners,              # získání vlastníků z předávacího
                           # directoru

# aliases - vyhledání aliasových expanzí, uložených v databázi.
# Jedná se o standardní soubor aliases. Používá se pro běžné věci,
# jako je mapování roota, postmastera, MAILER-DAEMONa a uucp
# při administraci sítě, vytváření některých aliasových expanzí
# menších systémů a podobně. V konfiguraci této lokace je soubor
# aliases využit zejména pro aliasové a předávací informace,
# specifické pro jednotlivé stroje. Celkové předávací informace
# se vkládají do databáze "forward".
```

```
aliases:
  driver=aliasfile,        # víceúčelový aliasový director
  -nobody,                 # všechny adresy jsou s nobody spojeny
                           # implicitně, takže toto nastavení není
                           # užitečné
  sender_okay,             # neodstraňujte sender (odesilatele)
                           # z výrazů
  owner=owner-$user;       # problémy se směrují na vlastníkovu adresu
  file=/etc/aliases,
  modemask=002,            # nemělo by to být globálně zapisovatelné
  optional,                # ignoruj, pokud soubor neexistuje
  proto=lsearch,           # nesetříděný ASCII-soubor

# forward - vyhledání expanzí, uložených v předávací databázi.
# Jedná se o uživatelskou subdoménovou předávací databázi. Údaje jsou
# udržovány pro aktuální nebo minulé uživatele, kterým se pošta
# předává na preferované místo určení. Předávací databáze se
# po provedení změn dodává po síti TCP/IP, aby byla síť jednotná.
# forward:
# driver = aliasfile,      # víceúčelový aliasový director
# -nobody,                 # všechny adresy jsou s nobody spojeny
                           # implicitně, takže toto nastavení není
                           # užitečné
# owner = real-$user;      # problémy se směrují na vlastníkovu adresu
# file = /etc/forward,
# modemask = 002,
# proto = dbm,             # k přístupu využijte knihovnu dbm(3X)

# dotforward - expanduje soubory .forward v domovských adresářích
# uživatelů
# Pro uživatele, kteří mají údaj v databázi "forward", se soubor
# ".forward" využije pouze tehdy, pokud je na "domovském" stroji,
# definovaném v předávací databázi. Je-li použit, bere se spíše jako
# seznam adres, na které má být pošta doručena, než jako (nebo navíc
# jako) uživatel, určený v lokální adrese.
dotforward:
  driver = forwardfile,    # víceúčelový předávací director
  owner = postmaster, nobody, sender_okay;
```

```
  file = ~/.forward,       # soubor .forward v domovských adresářích
  checkowner,              # uživatel může tento soubor vlastnit
  owners = root,           # nebo root může tento soubor vlastnit
  modemask = 002,          # nemělo by to být globálně zapisovatelné
  caution = daemon:root,   # nespouštět nic jako root nebo daemon
  # věnovat zvláštní pozornost na vzdáleně přístupné domovské
  # adresáře
  unsecure = "~uucp:/tmp:/usr/tmp:/var/tmp"

# forwardto - expanduje a "Forward to " (předat na) z uživatelských
# souborů v mailboxu
# Emuluje předávací mechanismus V6/V7/System-V, který využívá řádek
# předávacích adres, uložený na začátku uživatelských souborů
# mailboxu, s předponou "Forward to".
forwardto:
  driver = forwardfile,
  owner = postmaster, nobody, sender_okay;
  file = /var/spool/mail/${lc:user},    # ukazuje na uživatelské
                                        # soubory mailboxu
  forwardto,               # umožňuje funkci "Forward to "
  checkowner,              # uživatel může tento soubor vlastnit
  owners = root,           # nebo root může tento soubor vlastnit
  modemask = 0002,         # pod System V může zapisovat group mail
  caution = daemon:root,   # nespouštět nic jako root nebo daemon
# user - odevzdá uživatelům na jejich lokálních strojích do mailboxu
user:   driver = user;     # ovladač pro zjištění uživatelského jména
  transport = local        # lokální transport směřuje do mailboxů

# real_user - nalezne uživatelská jména s předponou "real-"
# Toto je užitečné při umožnění adres, které explicitně dodávají
# do uživatelova souboru mailboxu. Sem mohou být dodány například
# chyby při expanzi souboru .forward nebo tak mohou být rozřešeny
# předávací smyčky mezi více stroji. Uživatelé mohou poštou také
# na svůj "nedomovský" stroj předávat data, přičemž využijí adresu
# ve tvaru real-login@vzdálený.hostitel.
real_user:
  driver = user;
  transport = local,
```

```
  prefix = "real-"         # odpovídá například real-root

# lists - expanduje poštovní seznamy, uložené v adresáři list
# poštovní seznamy mohou být vytvořeny jednoduchým vytvořením
# souboru v adresáři /etc/smail/lists.
lists:  driver = forwardfile,
  caution,                 # všechny adresy označit upozorněním
  nobody,                  # a potom přiřadit uživatele nobody
  owner = owner-$user;     # lokace se system V mohou využít o-$user
                           # - owner-$user by bylo moc dlouhé
  file = lists/${lc:user}  # lists je pod $smail_lib_dir

# owners - expanduje poštovní seznamy, uložené v adresáři list owner
# seznamy vlastníků mohou být vytvořeny vytvořením souboru
# v adresáři/etc/smail/lists/owner. Vlastníkům seznamů jsou posílány
# lokálně vzniklé chyby při práci se seznamy stejného názvu. Seznam
# vlastníků pro poštovní seznam vytvoříte souborem s názvem seznamu
# v /etc/smail/lists/owner. Tak se vytvoří seznam adres vlastníků-
# názvů seznamů, jak jej výše používá director "lists".
owners: driver = forwardfile,
  caution,                 # všechny adresy označit upozorněním
  nobody,                  # a potom přiřadit uživatele nobody
  owner = postmaster;      # lokace se system V mohou využít o-$user
                           # - owner-$user by bylo moc dlouhé
  prefix = "owner-",
  file = lists/owner/${lc:user}    # lists je pod $smail_lib_dir

# request - expanduje poštovní seznamy, uložené v adresáři list
# request seznamy požadavků mohou být vytvořeny vznikem souboru
# v adresáři /etc/smail/lists/request. Zde vypsané adresy se
# využívají jako standardní adresy pro dotazy k poštovním seznamům.
# Například žádosti o přidání nebo smazání ze seznamu budou odesílány
# na "list-request", což je nutné nastavit na předání k příslušným
# osobám.
request: driver = forwardfile,
  caution,                 # všechny adresy označit upozorněním
  nobody,                  # a potom přiřadit uživatele nobody
  owner = postmaster;      # lokace se system V mohou využít o-$user
```

```
                              # - owner-$user by bylo moc dlouhé
  suffix = "-request",
  file = lists/request/${lc:user} # lists je pod $smail_lib_dir
```

Tady nemusíte nic měnit, snad jen volbu pro poštovní konference (mailing list), jestliže ji v `smailu` budete využívat, nebo volbu `forwards`, jestliže budete chtít například znemožnit přeposílání pošty.

4.4 Soubor fidopaths

```
.f105.n324.z2.fidonet.org f105.n324.z2.fidonet.org!%s
.n324.z2.fidonet.org f105.n324.z2.fidonet.org!%s
.z2.fidonet.org  f105.n324.z2.fidonet.org!%s
.fidonet.org  f105.n324.z2.fidonet.org!%s
```

Takový soubor vytvořte pouze pokud používáte `ifmail` a FIDO.

4.5 Soubor routers

```
# forces - nastavit určité cesty
# Tato databáze existuje z důvodu nastavení cest na různé stroje
# nebo domény. Využívá se při vytváření dočasných převodů k dalším
# databázím cesty. Změny zde provedete editací souboru
# maps/force.path a zadáním make v adresáři maps/.
forces:
  driver = pathalias,            # směrovač, prohledávající
                                 # soubor paths
  method = /etc/smail/maps/table;# přenosy jsou v tomto souboru
  file = forcepaths,             # soubor, obsahující force path
                                 # info
  proto = lsearch,               # použití setříděného souboru
  optional,
 reopen                          # zavřít, když se nepoužívá

uucp_neighbors:
  driver=uuname,                 # použít program, který vrací
                                 # sousedy
  transport=uux;
```

```
  cmd="/usr/bin/uuname -a",       # použít program uuname
# domain=uucp                     # oddělit koncovku .uucp

# smart_host - částečně určený director smarthost
# Jestliže je atribut smart_path v souboru config definován jako
# cesta z lokálního stroje na vzdálený, budou všechny stroje, které
# neodpovídají ničemu jinému, odeslány na vyznačený vzdálený stroj.
# Atribut smart_transport je možné použít k určení jiného
# transportu. Jestliže není určen atribut smart_path, tento router
# je ignorován.
smart_host:
  driver = smarthost,             # ovladač speciálních znaků
  transport = uux                 # implicitně dodávat přes UUCP
# path=phreak

# ifmail - posílání pošty na fidonet a obráceně
ifmail:
  driver=pathalias,
  transport=ifmail;
  file=fidopaths,
  proto=lsearch
```

Pokud pro poštu z FIDO používáte `ifmail`, měli byste sem zahrnout část `ifmail`. Povšimněte si, že je možné změnit transportní mód z „uux“ (přes UUCP) například na „SMTP“ nebo je dokonce možné vypsat cesty k různým strojům nebo doménám podle „`/etc/smail/maps/table`“.

4.6 Soubor transports

```
# local - dodání pošty lokálním uživatelům
# Smail se má připojit přímo na soubory uživatelských mailboxů
# v /var/spool/mail
#local: driver = appendfile,     # připojit zprávu do souboru
# -return_path,                  # začlenit pole Return-Path:
# local,                         # při předání využít lokální formy
# from,                          # dodat řádek From_
# unix_from_hack;                # v těle vložit > před From
#
# file = /var/spool/mail/${lc:user}, # použít tuto lokaci pro Linux
```

```
#                       # Všimněte si, že mail spool musí být 1777
# file = ~/mailfile,    # toto umístění zajišťuje lepší zabezpečení
# group = mail,         # skupina, vlastnící soubor pro System V

# mode = 0660,          # pod System V má group mail přístup
# suffix = "\n",        # připojení nového řádku navíc
# append_as_user,

# Takto může mít každý uživatel soubor ~/.procmailrc, kterým
# kontroluje filtrování pošty a její ukládání z poštovních seznamů
# v oddělených mailboxech (podle potřeby).
local:  +inet,
  -uucp,
  driver = pipe,                # připojit zprávu do souboru
  return_path,                  # začlenit pole Return-Path:
  local,                        # při předání využít lokální formy
  from,                         # dodat řádek From_
  unix_from_hack;               # v těle vložit > před From

  cmd = "/usr/bin/procmail",    # použít procmail k lokálnímu
                                # předání
  parent_env,                   # info o prostředí z nadřazeného
                                # adresáře
  pipe_as_user,                 # použít user-id, asociovaný
                                # s adresou
  umask = 0022,                 # umask pro dceřiný proces
# -ignore_status,               # status exit by měl být akceptován
# -ignore_write_errors,         # navazovat na broken pipes

# pipe - dodání pošty příkazům shellu
# To se využívá implicitně, když smail zjistí adresy, začínající
# svislou čarou, jako "|/usr/lib/news/recnews talk.bizarre".
# Svislá čára se z adresy odstraňuje před předáním k transportu.
# pipe:  driver = pipe,         # předání zprávy jinému programu
# return_path, local, from, unix_from_hack;
#
# cmd = "/bin/sh -c $user",     # odeslání adresy do Bourne Shell
```

```
# parent_env,              # info o prostředí z nadřazeného
                           # adresáře
# pipe_as_user,            # použít user-id, asociovaný s adresou
# umask = 0022,            # umask pro dceřiný proces
# -log_output,             # nelogovat stdout/stderr
# ignore_status,           # na status exit se nehledí
# ignore_write_errors,     # ignorování broken pipes

# file - dodání pošty souborům
# To se využívá implicitně, když smail zjistí adresy, začínající
# šikmou čarou nebo tildou, jako "/usr/info/list_messages" nebo
# "~/Mail/inbox".
# file:  driver = appendfile,
# return_path, local, from, unix_from_hack;
#
# file = $user,            # soubor se vezme z adresy
# append_as_user,          # použít user-id, asociovaný s adresou
# expand_user,             # expandovat ~ a $ v adrese
# check_path,
# suffix = "\n",
# mode = 0644

# uux - předání programu rmail na vzdáleném hostiteli UUCP
#
# Při jednom UUCP-převodu bude na vzdálenou lokaci předáno právě
# 5 adres.

uux:    driver = pipe,
  -uucp,
  inet,
# uucp,                    # použít adresové formuláře ve stylu UUCP
  from,                    # dodat řádek From_
  max_addrs = 5,           # maximálně 5 adres na jedno vyvolání
  max_chars = 200;         # maximálně 200 znaků adres

# přepínač -r zabraňuje okamžitému dodání, závorky kolem proměnné
# $user zabraňují speciální interpretaci v uux.
 cmd = "/usr/bin/uux - -r -g$grade $host!rmail $((${strip:user})$)",
```

```
# cmd="/usr/bin/uux - $host!rmail $(($user)$)",
  ignore_write_errors,     # ignorování broken pipes
  umask = 0022,
# pipe_as_sender,

# uux_one_addr - dodání pošty přes UUCP na vzdálenou lokaci, která
# zvládá v jednom okamžiku jednu adresu
# Toto je často nutné při dodávání na lokaci s neupravenou verzí
# 4.1BSD.

uux_one_addr:
  driver = pipe,
  uucp,                    # použít adresové formuláře ve stylu UUCP
  from;                    # dodat řádek From_
# přepínač -r zabraňuje okamžitému dodání
  cmd = "/usr/bin/uux - -r -g$grade $host!rmail (${strip:user})",
  umask = 0022,
  pipe_as_sender

queueonly:
  driver = pipe;           # odeslat zprávu na rouru
 cmd = "/usr/lib/sendmail -Q -f $sender -bm $user",
                           # použít getmail pro lokální předání
  user=root,               # spustit getmail jako "root"
 group=mail,               # spustit getmail jako "mail"
  parent_env,              # info o prostředí z nadřazeného adresáře
  -pipe_as_user,           # použít user-id, asociovaný s adresou
  umask = 0007,            # umask pro dceřiný proces

# dodání zprávy. smtp transport se začlení, pouze pokud existuje síť
# BSD. Pro přenosy v zóně UUCP je možné určit atribut uucp. Při
# přenosech na Internetu je nutné nastavit atribut inet.
# POZNÁMKA: Toto je optimální, ke zvládnutí více zpráv v jednom
# připojení by měla existovat nástavba.
# TAKÉ: Možná bude nutné omezit max_addrs na 100, protože toto je
# počet, pro který SMTP požaduje implementaci k obsluze jedné zprávy.
smtp:   driver=tcpsmtp,
  inet,                    # jestliže UUCP_ZONE není definováno
```

```
# uucp,                    # jestliže UUCP_ZONE je definováno
  -max_addrs, -max_chars;  # žádné omezení v počtu adres

  short_timeout=5m,        # timeout pro krátké operace
  long_timeout=2h,         # timeout pro delší operace smtp
  service=smtp,            # připojení na tento servisní port
# Při použití na Internetu: zrušte komentář u následujících 4 řádků
  use_bind,                # rozlišení MX a multiple A záznamů
  defnames,                # použití standardního vyhledávání domén
  defer_no_connect,        # nový pokus, je-li nameserver vypnutý
  local_mx_okay,           # selhat MX na lokálního hostitele

ifmail:
  from,received,max_addrs=5,max_chars=200,
  driver=pipe;
  pipe_as_sender,
  cmd="/usr/local/bin/ifmail -x9 -r$host $((${strip:user})$)"
```

Pokud pro poštu z FIDO používáte `ifmail`, měli byste sem zahrnout část `ifmail`. Kromě toho byste neměli v tomto souboru editovat nic, co definuje agenty pro transport (uux, smtp...). Tyto parametry určíte v jiných konfiguračních souborech.

Povšimněte si, že některé části (jako „`pipe`“ nebo „`file`“) jsem dal do komentáře, aby se rozšířilo zabezpečení.

4.7 Adresář maps/

Obsahuje soubory `map` a `table`:

Nejprve soubor `map`

```
#N       foo.bar foo2.bar2
        #S          AT 486/RedHat Linux 1.2.13
        #O          organizace
        #C          kontakt
        #E          administrace (email)
        #T          telefon
        #P          adresa
        #R
```

```
#U          hostitelé, připojení přes uucp
#W          vytvořeno/editováno kým
#
hname polux

hname linux.eu.org

hname = polux
hname = polux.linux.eu.org
```

Tento soubor si opět upravte podle vlastní situace (já poštu dostávám z polux.linux.eu.org). Nyní soubor `table`

```
* uux
```

Zde můžete určit různý způsob transportu do různých cílů. Například pro lokální síť určíte „smtp“ a pro zbytek světa „uux“ (přes UUCP) nebo obráceně (já využívám UUCP pro veškerou odesílanou poštu, proto jsem použil „`*`“).

4.8 Další dobré příklady

Předchozí soubory skutečně využívám, takže by neměly nastat problémy s jejich využitím jako základ pro vlastní soubory.

Následující soubory nabízím pouze jako příklady konfigurace `smailu` jiným způsobem.

```
#ident "@(#) transports,v 1.2 1990/10/24 05:20:46 tron Exp"

# Viz smail(5) s kompletním popisem obsahu tohoto souboru.

# local - předání pošty lokálním uživatelům
#
# Smail se má připojit přímo na soubory uživatelských mailboxů
# v /usr/mail
local: driver = appendfile,       # připojit zprávu do souboru
  return_path,                    # začlenit pole Return-Path:
  local,                          # při předání využít lokální formy
  from,                           # dodat řádek From_
  unix_from_hack;                 # v těle vložit > před From
```

```
  file = /usr/mail/${lc:user},    # použít tuto lokaci pro System V
  group = mail,             # skupina, vlastnící soubor pro System V
  mode = 0660,              # pod System V má group mail přístup
  suffix = "\n",            # připojení nového řádku navíc
  append_as_user,

# pipe - dodání pošty příkazům shellu
#
# To se využívá implicitně, když smail zjistí adresy, začínající
# svislou čarou, jako "|/usr/lib/news/recnews talk.bizarre".
# Svislá čára se z adresy odstraňuje před předáním k transportu.
pipe:  driver = pipe,             # předání zprávy jinému programu
  return_path, local, from, unix_from_hack;

  cmd = "/bin/sh -c $user",       # odeslání adresy do Bourne Shell
  parent_env,               # info o prostředí z nadřazeného adresáře
  pipe_as_user,             # použít user-id, asociovaný s adresou
  umask = 0022,             # umask pro dceřiný proces
  -log_output,              # nelogovat stdout/stderr
  ignore_status,            # na status exit se nehledí
  ignore_write_errors,      # ignorování broken pipes

# file - dodání pošty souborům
#
# To se využívá implicitně, když smail zjistí adresy, začínající
# šikmou čarou nebo tildou, jako "/usr/info/list_messages" nebo
# "~/Mail/inbox".

file:  driver = appendfile,
  return_path, local, from, unix_from_hack;

  file = $user,             # soubor se vezme z adresy
  append_as_user,           # použít user-id, asociovaný s adresou
  expand_user,              # expandovat ~ a $ v adrese
  check_path,
  suffix = "\n",
  mode = 0644
```

```
# uux - předání programu rmail na vzdálené UUCP lokaci
#
# Při jednom UUCP převodu bude na vzdálenou lokaci předáno právě
# 5 adres.
uux:    driver = pipe,
  uucp,                    # použít adresové formuláře ve stylu UUCP
  from,                    # dodat řádek From_
  max_addrs = 5,           # maximálně 5 adres na jedno vyvolání
  max_chars = 200;         # maximálně 200 znaků adres

  # přepínač -r zabraňuje okamžitému dodání, závorky kolem proměnné
  # $user zabraňují speciální interpretaci v uux.
  cmd = "/usr/bin/uux - -r -g$grade $host!rmail $((${strip:user})$)",
  umask = 0022,
  pipe_as_sender,

# uux_one_addr - dodání pošty přes UUCP na vzdálenou lokaci, která
# zvládá v jednom okamžiku jednu adresu
#
# Toto je často nutné při dodávání na lokaci s neupravenou verzí
# 4.1BSD.
uux_one_addr:
  driver = pipe,
  uucp,                    # použít adresové formuláře ve stylu UUCP
  from;                    # dodat řádek From_

  # přepínač -r zabraňuje okamžitému dodání
  cmd = "/usr/bin/uux - -r -g$grade $host!rmail (${strip:user})",
  umask = 0022, pipe_as_sender

# demand - dodat vzdálenému programu rmail, který podává žádost
demand: driver = pipe,
  uucp, from, max_addrs = 5, max_chars = 200;

  # bez přepínače -r se bude vzdálená lokace kontaktovat okamžitě
  cmd = "/usr/bin/uux - -g$grade $host!rmail $(($user)$)",
  umask = 0022, pipe_as_sender
```

```
# uusmtp - dodat programu rsmtp na vzdálené hostitele UUCP
#
# Dodat pomocí jednoduchého protokolu Batched SMTP na vzdálený stroj.
# Umožňuje využít více libovolných adres. Odstraňuje také omezení
# adresátů na jedno vyvolání uux.
uusmtp: driver = pipe,
  bsmtp,                  # odeslat příkazy batched SMTP
  -max_addrs,             # není zde žádné omezení počtu nebo
 -max_chars;              # celkové velikosti adres adresátů

  # přepínač -r zamezuje okamžitému dodání, adresáti jsou uloženi
  # v datech, odesílaných na standardní výstup rsmtp.
  cmd = "/usr/bin/uux - -r -g$grade $host!rsmtp",
  umask = 0022, pipe_as_sender

# demand_uusmtp - dodat vzdálenému programu rsmtp, který podává
# žádost
demand_uusmtp:
  driver = pipe,
  bsmtp, -max_addrs, -max_chars;

  # bez přepínače -r se bude vzdálený hostitel kontaktovat okamžitě
  cmd = "/usr/bin/uux - -g$grade $host!rsmtp",
  umask = 0022, pipe_as_sender

# smtp - dodání pomocí SMTP přes TCP/IP
#
# Připojení ke vzdálenému hostiteli pomocí TCP/IP a inicializace
# SMTP-konverzace o dodání zprávy. SMTP transport je začleněn pouze
# s existující sítí BSD.

# POZNÁMKA: Možná bude nutné omezit max_addrs na 100, protože toto
# je počet, pro který SMTP požaduje implementaci k obsluze jedné
# zprávy.
smtp:   driver = smtp,
  -max_addrs,
  -max_chars
```

```
#ident "@(#) table,v 1.2 1990/10/24 05:20:31 tron Exp"

# Tento soubor vyjmenovává transporty, které se při dodání na
# specifické lokace z bargw využijí.

#lokace       transport
#--------     ---------
curdsgw       demand_uusmtp        # pomocí batched SMTP
oldbsd        uux_one_addr         # lokace s 4.1BSD nemohou vzít
                                   # více než jednu adresu
sun           demand               # když odesíláte poštu,
                                   # volejte sun
*             uux                  # u všeho ostatního volte intervaly
```

4.9 Restart inetd

Aby se `smail` spustil jako SMTP démon, přidejte do `/etc/inetd.conf`:

```
smtp stream tcp nowait root /usr/bin/smtpd smtpd
```

nebo:

```
smtp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.smtpd
```

Odesílaná pošta se bude při použití `elmu` odesílat automaticky.

4.10 Smail s SMTP

ISP většinou používají SMTP, takže s posíláním pošty byste neměli mít žádné problémy. Jestliže je vaše připojení na Internet v době odeslání pošty nefunkční, pošta se usadí do „`/var/spool/smail/input`“. Při obnovení spojení se spouští „`runq`“, který poštu odešle. S obdržením pošty je ale v takovém případě problém, protože váš poskytovatel Internetu nemá na starosti jen vás, ale i mnoho jiných klientů!

Vaši poštu je obvykle možné zachránit pomocí protokolu POP, viz níže.

5 Sendmail+IDA

Pro velké servery stojí za zvážení výběr sendmailu, už pro snadnost jeho použití v daném případě. Ale musíte si vybrat mezi verzemi sendmailu+IDA a sendmailu 8.x:

- Jestliže používáte staré jádro (1.0): sendmailu+IDA
- Předchozí jádro (1.2): sendmailu+IDA a editace zdrojového kódu
- Současné jádro (2.0): sendmailu 8.x

Ale linuxoví začátečníci nebo lidé, zaměření na zabezpečení nebo snadnost konfigurace, by si měli raději vybrat smail nebo qmail.

5.1 Instalace zdrojového textu

- cd / ; tar -zxvf sendmail5.67b+IDA1.5.tgz
- cd /usr/local/lib/mail/CF a okopírujte sample.m4 local.m4 na „jménovašehohostitele.m4“.

Proveďte editaci dodaných jmen hostitelů, přezdívek, serverů a upravte je podle vaší situace. Implicitní soubor je pouze pro čistě UUCP-síť, kde jsou hlavičky v doménovém tvaru a používá se chytřejší hostitel. Potom make jménovašehohostitele.cf a výsledný soubor přesuňte do /etc/sendmail.cf.

- Jestliže využíváte čistě UUCP, **NEPOTŘEBUJETE** vytvářet žádné z tabulek, které jsou zmíněny v souboru README.linux

Aby fungoval Makefile, stačí na soubory použít touch. Editujte soubor .m4, vytvořte sendmail.cf a začněte jej testovat.

- Jestliže využíváte čistě UUCP a využíváte „chytřejší hostitele“, musíte pro každou lokaci přidat údaje uucpxtable (jinak by pošta pro ně také chodila přes chytřejší hostitele) a proti upraveným uucpxtable musíte spustit dbm.
- Jestliže spouštíte původní binární distribuci Riche Brauna 5.67a a změníte soubor .cf pomocí „/usr/lib/sendmail -bz“, musíte přerušit konfiguraci, aby se změny projevily.

Také byste měli přejít na verzi alespoň 5.67b, protože ve verzích 5.67a a starších je nepěkná díra v zabezpečení. Další zajímavou věcí je to, že když máte nastaven mail.debug a spustíte syslogd, vaše doručená a odeslaná pošta bude zaznamenávána. Podrobnosti viz soubor „/etc/syslog.conf“.

Zdroje pro sendmail+IDA naleznete na vixen.cso.uiuc.edu; pokud například používáte jádro 1.00, nevyžadují žádné úpravy pro běh v Linuxu.

Jestliže používáte jádro > 1.1.50, budete si muset pohrát s obrácením linuxových specifikací, které naleznete ve zdrojových textech (já jsem vám to říkal, že `sendmail` je pouze pro „stará“ jádra.

Jak se to provede je jistě jasné: stačí zadat „`make`“, počkat si, podívat se na příslušné řádky zdrojových textů a dát do komentáře kód, specifický pro Linux.

Jestliže chcete používat `sendmail+IDA`, silně doporučuji až verzi `sendmail5.67b+IDA1.5`, protože všechny nutné linuxové odlišnosti jsou nyní ve zdrojových textech a bylo odstraněno několik bezpečnostních děr, které BYLY (!) ve starších verzích, vytvořených přibližně před 1. prosincem 1993.

Nyní je poslední verzí jádra 2.0.x. Místo `sendmail+IDA` byste měli používat `sendmail 8.x`.

5.2 Soubor sendmail.m4

`Sendmail+IDA` požaduje spíše vytvoření souboru `sendmail.m4` místo přímé editace souboru `sendmail.cf`. Na tom je pěkné, že lze jednoduše nastavit konfigurace v `smailu` nebo normálním `sendmailu` prakticky nenastavitelné (pro některé lidi).

Soubor `sendmail.m4`, který odpovídá předchozímu příkladu pro `smail`, vypadá následovně:

```
dnl #------------------ UKÁZKOVÝ SOUBOR SENDMAIL.M4 ---------------
dnl #
dnl # (řetězec 'dnl' je m4 ekvivalentem řádku v komentáři)
dnl #
dnl # asi nebudete chtít přepsat LIBDIR z kompilovaných cest
dnl #define(LIBDIR,/usr/local/lib/mail)dnl     # kam jdou všechny
dnl                                            # podpůrné soubory
define(LOCAL_MAILER_DEF, mailers.linux)dnl     # poštovní program pro
dnl                                            # lokální dodávku
define(POSTMASTERBOUNCE)dnl                    # chyby na adresu
                                               # postmaster
define(PSEUDODOMAINS, BITNET UUCP)dnl          # zde nezkoušejte DNS
dnl #
dnl #-------------------------------------------------------------
dnl #
dnl # jména, pod kterými nás znají
define(PSEUDONYMS, myhostname.subdomain.domain myhostname.UUCP)
dnl #
```

```
dnl # naše primární jméno
define(HOSTNAME, myhostname.subdomain.domain)
dnl #
dnl # naše UUCP-jméno
define(UUCPNAME, myhostname)dnl
dnl #
dnl #---------------------------------------------------------------
dnl #
define(UUCPNODES, |uuname|sort|uniq)dnl  # naši UUCP-sousedé
define(BANGIMPLIESUUCP)dnl               # zajistěte tohle UUCP
define(BANGONLYUUCP)dnl                  # pošta je brána korektně
define(RELAY_HOST, my_uucp_neighbor)dnl  # náš chytrý relay host
define(RELAY_MAILER, UUCP-A)dnl          # do moria přes UUCP
dnl #
dnl #---------------------------------------------------------------
dnl #
dnl # různé vyhledávací tabulky dbm
dnl #
define(ALIASES, LIBDIR/aliases)dnl      # systémové aliasy
define(DOMAINTABLE, LIBDIR/domaintable)dnl  # doménové lokace
define(PATHTABLE, LIBDIR/pathtable)dnl  # databáze cest
define(GENERICFROM, LIBDIR/generics)dnl # obecné adresy "z"
define(MAILERTABLE, LIBDIR/mailertable)dnl  # poštovní programy na
dnl                                     # hostitele nebo doménu
define(UUCPXTABLE, LIBDIR/uucpxtable)dnl       # cesty na hostitele,
dnl                                             # které zásobujeme
define(UUCPRELAYS, LIBDIR/uucprelays)dnl # cesty krátkých okruhů
dnl #
dnl #---------------------------------------------------------------
dnl #
dnl # začlenění 'skutečného' kódu, který to vše zprovozní
dnl # (se zdrojovým kódem)
dnl #
include(Sendmail.mc)dnl                  # POŽADOVANÝ ÚDAJ !!!
dnl #
dnl #------------ KONEC UKÁZKOVÉHO SOUBORU SENDMAIL.M4 -------
```

5.3 Určení lokálního doručovatele

Na rozdíl od většiny verzí Unixu se Linux implicitně nedodává s lokálním doručovatelem. Nyní se většinou instalují `deliver` nebo `procmail`, takže tohle velice složité nastavení se již dále nekomplikuje. Doporučuji tedy využít běžně dostupné programy `deliver` a `procmail`, které se v některých distribucích Linuxu mohou vyskytovat jako volitelné komponenty.

Musíte pak v souboru `sendmail.m4` definovat `LOCAL_MAILER_DEF`, který bude ukazovat na takovýto soubor:

```
# -- /usr/local/lib/mail/mailers.linux --
#      (lokální doručovatelé pro použití v Linuxu )
Mlocal, P=/usr/bin/deliver, F=SlsmFDMP, S=10, R=25/10, A=deliver $u
Mprog,  P=/bin/sh,          F=lsDFMeuP,   S=10, R=10, A=sh -c $u
```

V souboru `Sendmail.mc` je také zabudovaná implicitní hodnota pro `deliver`, která se dostane i do souboru `sendmail.cf`. K jejímu určení nepoužijete soubor `mailers.linux`, ale v souboru `sendmail.m4` místo toho nadefinujete:

```
dnl --- (v souboru sendmail.m4) ---
define(LOCAL_MAILER_DEF, DELIVER)dnl     # doručovatel pro lokální
dnl                                      # dodávku
```

Naneštěstí `Sendmail.mc` předpokládá instalaci `deliver` v `/bin`, což není správné v případě Slackware1.1.1 (instaluje jej do `/usr/bin`). V takovém případě si musíte buď vypomoci ošálením pomocí odkazu, nebo podle zdrojových textů `deliver` přeinstalovat do `/bin`. Povšimněte si, že `procmail` je ve většině případů lepší než `deliver`, například při filtrování pošty.

5.4 Tabulky sendmail+IDA dbm

Nastavení speciálního chování hostitelů nebo domén se provádí přes množství volitelných tabulek **dbm** místo přímé editace souboru `sendmail.cf`.

Podrobnější informace naleznete v červencovém čísle časopisu **Linux Journal** z roku 1994 (jestli ale naleznete ten časopis, v dokumentacích ve zdrojích nebo v kapitole o `sendmailu` v nejnovější verzi **Příručky správce sítě** v Linux DOC Project, která je součástí této knihy.

- `mailertable` - určuje zvláštní chování vzdálených hostitelů nebo domén
- `uucpxtable` - nastavuje doručování pošty pomocí UUCP pro adresy ve formátu DNS

- `pathtable` - definuje UUCP bang-path na vzdálené hostitele nebo domény
- `uucprelays` - zkracuje cestu `pathalias` na známé vzdálené hostitele
- `genericfrom` - převádí vnitřní adresy na obecné, viditelné zvenku
- `xaliases` - převádí na/z platných vnitřních adres
- `decnetxtable` - převádí RFC-822 adresy na adresy ve stylu DECnet

5.5 Takže které údaje jsou opravdu nutné?

Jestliže nejsou použity žádné volitelné tabulky dbm, `sendmail` poštu doručuje pomocí `RELAY_HOST` a `RELAY_MAILER`, v závislosti na souboru `sendmail.m4`, použitého pro vytvoření `sendmail.cf`. Takové chování je možné snadno upravit pomocí údajů v `domaintable` nebo `uucpxtable`.

Normální hostitel, který je na Internetu a slyší na systém DNS, nebo je čistě UUCP a předává veškerou poštu pomocí UUCP přes `RELAY_HOST`, nepotřebuje žádné zvláštní tabulkové údaje.

Prakticky všechny systémy by měly nastavovat makra `DEFAULT_HOST` a `PSEUDONYMS`, která určují kanonické jméno hostitele a přezdívky, pod kterými je známa.

Pravděpodobně také nastavení `RELAY_MAILER` a `RELAY_HOST`, umožňující automatické směrování pomocí předávání pošty na chytřejšího hostitele.

Použitý přenos pošty je definován v `RELAY_MAILER` a pro UUCP hostitele by měl být obvykle nastaven na UUCP-A. Jestliže je váš hostitel čistě SMTP a rozumí `DNS`, měli byste změnit `RELAY_MAILER`.

Jestliže je váš hostitel SLIP, můžete využít jednoduchý způsob směrování veškeré odesílané pošty k vašemu poskytovateli služeb, který si s ní už poradí. K tomu je nutné definovat `ISOLATED_DOMAINS` a `VALIDATION_DOMAINS` na vaši doménu, `RELAY_HOST` musí být váš poskytovatel služeb a `RELAY_MAILER` bude TCP. K takovému nastavení systému pro převádění byste také samozřejmě měli vyžádat povolení.

5.6 Sendmail 8.x

`Sendmail 8.7.x` byl poslední větší inovací od dob `sendmail5`. Měl nádhernou zabudovanou podporu pro kompilaci v Linuxu: „`make linux`“ a všechno je nastavené.

Nejlépe vám pravděpodobně poslouží nějaká binární podoba programu, kterou si přetáhnete z některých linuxových archivních serverů. Je to lepší než se potýkat například s Berkeley dbm.

Na sunsite.unc.edu/pub/Linux/system/Mail/delivery/sendmail-8.6.12-bin.tgz naleznete skvělou distribuci pro sendmail 8.6.12 od Jasona Haara - **j.haar at lazerjem.demon.co.uk**, která má dokumentaci ke zdrojovým textům a pěkný rychlý popis pro běh sendmailu v8 při běžných konfiguracích.

5.7 Ukázkový soubor mc pro 8.7.x

Podobně jako sendmail+IDA využívá sendmail v8 m4 pro převod konfiguračního souboru config na plný sendmail.cf, využívaný sendmailem. Následuje můj aktuální mc soubor, využívaný na mém hostiteli (PPP na Internet pro odesílanou poštu, UUCP pro doručenou poštu).

```
dnl divert(-1)
#-----------------------------------------------------------------
#
# tohle je soubor .mc pro linuxového hostitele, nastavenou následovně:
#
#      - připojená na Internet pro výchozí poštu (je zde ppp)
#      - připojená přes UUCP pro příchozí poštu
#      - doménové hlavičky
#      - žádný lokální doručovatel (místo toho 'deliver')
#      - žádný běžící DNS, takže nic výchozího tudy neprojde
#      - veškerá nelokální výchozí pošta jde na RELAY_HOST přes SMTP
# (používáme ppp a necháváme našeho poskytovatele, aby se staral)
#
#                                              vds 3/31/95
#
#-----------------------------------------------------------------


include('../m4/cf.m4')
VERSIONID('linux nodns relays to slip service provider
smarthost')dnl
Cwmyhostname.myprimary.domain myhostname.UUCP localhost
OSTYPE(linux)
FEATURE(nodns)dnl
FEATURE(always_add_domain)dnl
FEATURE(redirect)
FEATURE(nocanonify)
dnl MAILER(local)dnl
```

```
MAILER(smtp)dnl
MAILER(uucp)dnl
define('RELAY_HOST', smtp:my.relay.host.domain)
define('SMART_HOST', smtp:my.relay.host.domain)
define('UUCP_RELAY', smtp:my.relay.host.domain)
define('LOCAL_MAILER_PATH', '/bin/deliver')
define('LOCAL_MAILER_ARGS', 'deliver $u')
```

5.8 Lahůdky v sendmailu v8

Existuje několik rozdílů, které předpokládám u uživatelů IDA. Zatím jsem se setkal s tímto: místo „`runq`“ pro spuštění fronty zadáváte „`sendmail -q`“.

5.9 Agenti pro lokální doručování

Na rozdíl od většiny operačních systémů neměl Linux poštu „přímo v sobě“ (built-in): potřebovali jste program, který by ji lokálně doručoval (například „`lmail`“, „`procmail`“ a „`deliver`“).

Nyní již každá nová distribuce obsahuje lokálního doručovatele!

Dokumentaci k jejich využití ve vaší distribuci naleznete v binární podobě `sendmail5.67b+IDA1.5` (na sunsite).

6 POP mail

Na síti pracovních stanic byla pošta vždy problémová:

- Buď využijete „**uživatel@počítač.foo.com**“ s problémy, když je „počítač“ vypnutý, s identifikací vaší sítě pro zbytek světa, s různými adresami pro jednu osobu u různých počítačů,
- nebo využijete poštovní přípojku na „poštovníserver.foo.com“ s přepisovacími právy, takže každý uživatel má poštu z jedné adresy, i když sedí u různých počítačů.

V takovém případě ale nastává problém, jak bude uživatel poštu číst? Pomocí `rsh` a `elmu`? :-) Tím by se přetížila poštovní přípojka! Řešením by mohlo být přeposílání, UUCP, SMTP..., ale to by bylo příliš složité.

Pak přišli POP/IMAP (oba zpočátku s problémy v zabezpečení v nových verzích jsou problémy překonány použitím `ssh`): poštovní program někdy musí být nastaven lokálně (jako `sendmail`, `smail`, `qmail` při použití například `elmu`, ale u mozilly se tomu vyhnete). Odesílání a příjem pošty je pak ale jednodušší.

6.2 Příjem pošty

Zde se dostáváme k hlavnímu neduhu protokolu POP: heslo se přes síť posílá jako čistý text a někteří příjemci POP neznají: musíte zvolit takový program, který POP umí (například `pine`, `emacs`, `netscape`, `mutt`...).

Problém s heslem se odstraní vytvořením šifrovaného „kanálu“ pro POP nebo použitím rozšíření APOP nebo RPOP. Problém s příjmem je možné vyřešit buď změnou programu pro čtení pošty (mozilla, `emacs`, `pine`,) nebo použitím lokálního poštovního programu s podporou protokolu POP.

`Gwpop` se hodí k vytvoření šifrovaného „kanálu“ a vložení pošty přímo do poštovní přihrádky. Ale závisí na `perlu`...

Mohu také doporučit `fetchmail`, který je aktivně podporován.

Jinak je možné použít jeden z mnoha POP klientů, dostupných pro Linux. Pokud je vaše uživatelské jméno `john` a heslo `PrisneTajne`, spustíte:

```
$ popclient -3 -v mail.acme.net -u john -p "PrisneTajne"
  -k -o JOHN-INET-MAIL
```

(Přesný význam předchozích voleb naleznete na manuálových stránkách programu `popclient`.)

6.3 Odeslání pošty

Zde musíte použít software, který umí SMTP, jako je `qmail`, `sendmail` nebo mozilla (tenhle umí všechno: čte poštu, obdrží POP a odešle přes SMTP!).

Vraťte se k jedné z předchozích částí, podle které nainstalujete a nakonfigurujete ten program, který vám vyhovuje. Jakmile začnete testovat, zkuste odeslat nějakou poštu na lokální účet na poštovní přípojce.

6.4 Čtení pošty

Jestliže váš program nezvládá všechno sám, můžete si nainstalovat `elm`, `pgp`, `mush`, `pine`... k dispozici je mnoho dobrých programů pro Linux!

6.5 Testování

To, že váš poštovní server zvládá POP, vyzkoušíte takto:

```
$ telnet mailhost 110
```

Jestliže to funguje, objeví se něco jako: „`OK Pop server (...) starting`“. Zadejte „quit“! Instalaci šifrovaného „kanálu“ `ssh` proveďte po testování vašeho poštovního serveru:

```
$ ssh mailhost date
```

Jestliže obdržíte datum, je vše v pořádku. Povšimněte si, že `ssh` se neptá na heslo, proto musíte na poštovním serveru vytvořit soubor „.shost“, obsahující jméno klienta. Testování `ssh` přesměrování portů (které využívá `gwpop`) provedete takto:

```
$ ssh -n -f -L 12314:localhost:110 mailhost sleep 30
```

potom

```
$ telnet localhost 12314
```

Pak můžete očekávat objevení uvítací zprávy vzdáleného POP-serveru. Jestliže nepoužíváte `ssh`, nezapomeňte ve skriptu pro `gwpop` dát do komentáře `$ssh`. Zkoušku `procmailu` provedete zadáním „`procmail -v`“.

6.6 Používání

Nyní můžete editovat skript perlu pro `gwpop`, zkontrolovat, jestli je vše v pořádku, a spustit `gwpop`:

```
$ gwpop -v vašeuživatelskéjméno
POP password on mailhost: vašeheslo
```

Jestliže jsou „chybová hlášení“ `gwpopu` normální, pošta bude z poštovní přípojky přehrána na váš lokální systém, kamkoliv určíte (tohle si prosím vyzkoušejte).

Gwpop je možné použít také jako démon:

```
$ gwpop -d $HOME/tmp vašeuživatelskéjméno
```

Zprávy gwpopu jsou potom odeslány do souboru syslog a gwpop bude stále běžet; signál „HUP“ přinutí gwpop vyzvednout vaši poštu.

POP-software je možné získat například zde:

```
ftp://ftp.pasteur.fr/pub/Network/gwpop
ftp://ftp.informatik.rwth-aachen.de/pub/packages/procmail
http://www.cs.hut.fi/ssh/
```

7 Poštovní „uživatelské agenty“

Tato část obsahuje informace, vztahující se k „uživatelským agentům“, což znamená software, který uživatel vidí a používá. Tento software spoléhá na transportní software, zmiňovaný výše. Nyní je k dispozici množství dalších „uživatelských agentů“ (pine, mush...), ale k těm neexistují žádné informace, specifické pro Linux. Dejte mi prosím vědět, jestli jsem na něco nezapomněl!

7.1 Elm

Elm se pod Linuxem kompiluje, instaluje a spouští bez problémů. Více informací naleznete ve zdrojových textech k elmu a v instalačních instrukcích. Elm a filter musí mít práva 2755 (skupin mail), /var/spool/mail 775 a skupin mail.

Jestliže používáte binární distribuci, budete muset vytvořit soubor „/usr/local/lib/elm/elm.rc“, čímž přepíšete zakompilovaný název serveru a informace o doméně:

- nahraďte „subdomain.domain“ názvem vaší domény
- nahraďte „myhostname“ vaším názvem serveru (bez názvu domény)

```
#---------- /usr/local/lib/elm/elm.rc ------------------
#
# toto je nekvalifikovaný název hostitele
hostname = myhostname
#
# toto je lokální doména
hostdomain = subdomain.domain
```

```
#
# toto je plně kvalifikovaný název hostitele
hostfullname = myhostname.subdomain.domain
#
#-------------------------------------------------------
```

Jednu věc si ale uvědomte, pokud máte totiž `Elm` kompilován s nastaveným MIME, musíte mít nainstalován a v cestě `metamail`, jinak by `Elm` nedokázal číst obdrženou MIME-poštu. `Metamail` je k dispozici na **thumper.bellcore.com** a samozřejmě také přes „`archie`“.

Do kategorie „příliš dobré, než aby to byla pravda“ spadá distribuce `Elm-2.4.24`, která umí PGP. Naleznete ji na adrese *ftp://ftp.viewlogic.com/pub/elm-2.4pl24pgp3.tar.gz,* což je `elm2.4.24` s přidanou podporou PGP. Konfiguruje se a instaluje jako normální `Elm`, což pravděpodobně znamená přidání výše zmíněných dodatků. Za zmínku stojí, že já tuto verzi používám a jsem s ní spokojen. Samozřejmě, že k dispozici je jistě i mnoho novějších verzí, včetně `elm-ME+`.

Následující věc sice není specifická pro Linux, ale často je považována (neprávem) za chybu v `Elmu`. Slyšeli jsme, že Elm někdy padá a hlásí, že nemůže alokovat v paměti nějaký vysoký počet bajtů. Náprava je v odstranění zpracovaných globálních poštovních přezdívek (`aliases.dir` a `aliases.pag`).

TOHLE ALE NENÍ CHYBA ELMU, je to chyba v konfiguraci `Elmu`, kterou prováděl ten, od koho máte binární distribuci.

`Elm` má rozšířený a nekompatibilní formát přezdívek; musíte zajistit, aby cesta, kterou `Elm` pro přezdívky využívá, byla odlišná od cesty, kterou využívá `sendmail/smail`. Vzhledem k množství zpráv o tomhle problému je zřejmé, že alespoň jedna z větších distribucí „z ulice“ byla špatně nakonfigurována (**scot at catzen.gun.de (Scot W. Stevenson)**).

Aktuální balík `metamailu` vyžaduje pro některé skripty `csh`. Nemáte-li `csh` (nebo `tcsh`), objeví se zajímavé chyby...

7.2 Mailx

Ušetřete si námahu: sežeňte si ze Slackware verze 2.1.0 nebo pozdější `mailx` kit, který obsahuje pěknou implementaci `mailx5.5`. Jestliže chcete kompilovat ze zdrojů, `mailx` v5.5 se v Linuxu kompiluje bez dodatků, pokud máte nainstalován „**pmake“.** Jestli je to ještě aktuální, doporučuji náhradu starého „`edmailu`“ ze SLS1.00 za `mailx`.

7.3 Další uživatelské agenty

Tyto by pod Linuxem také měly běžet. Podrobnosti k jejich nalezení viz `archie`...

- `Pine` - z Univ. of Washington
- `Metamail` - umožňuje podporu MIME
- `mh` - další způsob zpracování pošty
- `deliver` - shromažďuje a zpracovává poštu podle zadaných pravidel
- `procmail` - shromažďuje a zpracovává poštu podle zadaných pravidel
- `Majordomo` - spravuje poštovní konference
- `Mserv` - poskytuje soubory poštou

8 Poděkování

Následující lidé přispěli svými informacemi a zkušenostmi, čímž napomohli dokončení tohoto dokumentu:

Steve Robbins, Ian Kluft, Rich Braun, Ian Jackson, Syd Weinstein, Ralf Sauther, Martin White, Matt Welsh, Ralph Sims, Phil Hughes, Scot Stevenson, Neil Parker, Stephane Bortzmayer a zvláštní díky patří Vince Skahanovi za jeho jedinečnou spolupráci.

Jestliže jsem na někoho zapomněl, omlouvám se: stačí mi poslat email!

Kapitola

4

DNS

Nicolai Langfeldt **janl@math.uio.no** verze 2.0.3, 13. března 1998

Jak se stát úplně malým administrátorem systému DNS.

1 Předmluva

Klíčová slova: DNS, bind, bind-4, bind-8, named, dialup, ppp, slip, isdn, Internet, domain, name, hosts, resolving.

1.1 Autorská práva

1.2 Podíly a žádosti o pomoc

Chtěl bych poděkovat Arntu Gulbrandsenovi, který mnohokrát četl návrhy této práce a navrhl mnohá užitečná vylepšení. Chtěl bych také poděkovat lidem, kteří mi poslali návrhy a poznámky.

Tento dokument nebude nikdy dokončen. Pošlete mi prosím zprávy o vašich problémech a úspěších, dokument tak mohu vylepšovat. Peníze, komentáře nebo otázky posílejte na **janl@math.uio.no**. Jestliže budete odesílat elektronickou zprávu, *zajistěte prosím*, aby byla návratová adresa funkční. Předtím, než mi napíšete, si také **prosím** přečtěte část 8.

Jestliže chcete tento dokument překládat, sdělte mi to, abych věděl, ve kterých jazycích jsem byl publikován, současně vás mohu upozornit na změny v dokumentu.

1.3 Věnování

Dokument bych rád věnoval Anne Line Norheim Langfeldtové. I když jej pravděpodobně nikdy nebude číst, protože není takovým typem dívky, která by jej četla.

2 Úvod

Co to je a co není

Pro začátečníky: DNS je Domain Name System (systém pojmenování domén). DNS převádí názvy strojů na IP-adresy, dále mapuje názvy na adresy a adresy na názvy. Tento dokument ukazuje, jak takové mapování provádět v Linuxu. Mapování je vlastně vztah mezi dvěma věcmi, v našem případě mezi názvem stroje (např. *ftp.linux.org*) a IP-adresou stroje (zde 199.249.150.4).

DNS je pro nezasvěcené jednou z méně průhledných oblastí síťové správy. Tento dokument by měl alespoň něco osvětlit. Popisuje, jak nastavit *jednoduchý* DNS-server. Začneme přitom s DNS-serverem pouze s vyrovnávací pamětí a přejdeme až k nastavení primárního DNS-serveru pro doménu. Složitější nastavení naleznete v 8. části. Jestliže zde nenaleznete to, co potřebujete, musíte si pročíst opravdovou dokumentaci. K té se dostanu v 9. části.

Před dalším postupem je vhodné nakonfigurovat váš stroj, aby bylo možné se z něj a na něj přihlásit pomocí telnetu a provést všechny druhy připojení na síť. Zejména je nutné vyzkoušet `telnet 127.0.0.1` a mít vlastní stroj (zkuste to ihned!). Jako počáteční bod potřebujete také prospěšné soubory `/etc/nsswitch.conf` (nebo `/etc/host.conf`), `/etc/resolv.conf` a `/etc/hosts`. Nebudu zde totiž vysvětlovat jejich funkci. Jestliže ještě nemáte všechno nastavené a funkční, pomohou vám dokumenty NET-3 HOWTO a PPP HOWTO. Pročtěte si je.

Když říkám „vlastní stroj“, myslím stroj, na kterém má DNS být. Ne tedy jakýkoliv další váš stroj, který používáte ve vašem síťovém snažení.

Předpokládám, že se nenacházíte za žádným druhem firewallu, který by blokoval DNS-dotazy. Jestliže se ale za ním nacházíte, budete potřebovat speciální konfiguraci, viz část 8.

Převody názvů jsou v Unixu prováděny programem `named`. Ten je součástí balíku, který je koordinován Paulem Vixiem. `Named` je obsažen ve většině distribucí Linuxu a instaluje se jako `/usr/sbin/named`. Jestliže už named máte, pravděpodobně jej také můžete používat; jest-

liže jej nemáte, můžete si z linuxového FTP-serveru přetáhnout binární podobu, nebo si můžete z `ftp.isc.org:/isc/bind/src/cur/bind-8/` stáhnout poslední zdrojovou podobu. Tento dokument se zabývá bind verzí 8. Používáte-li `bind` verzi 4, stará verze tohoto dokumentu (vztahující se k `bind` verzi 4) je pro vás stále k dispozici na `http://www.math.uio.no/janl/DNS/`. Jestliže manuálová stránka k `named` hovoří o `named.conf`, máte `bind` 8, jestliže hovoří o `named.boot`, máte `bind` 4. Jestliže máte 4 a staráte se o zabezpečení, měli byste přejít na 8.

DNS je databází rozloženou na síti. Starejte se o to, co do ní vkládáte. Jestliže sem vložíte nesmysly, budou je vidět všichni. Svůj systém DNS udržujte konzistentní a jasný - získáte tak kvalitnější služby. Naučte se jej používat, spravovat, odlaďovat a bude z vás další „hodný" síťový správce, který udržuje síť před přetížením ze špatné správy.

V tomto dokumentu nastíním několik věcí, které nejsou zcela pravdivé (jsou to spíše polopravdy). Vše v zájmu zjednodušení. Když mi budete věřit, všechno bude (pravděpodobně) fungovat.

Tip: U všech souborů, které vám navrhuji změnit, si vytvořte záložní kopie. Kdyby se stalo, že nebude nic fungovat, můžete se ještě vrátit ke staré konfiguraci.

3 DNS-server pouze s vyrovnávací pamětí

První zásah do konfigurace DNS, velmi užitečný pro uživatele modemů

DNS-server pouze s vyrovnávací pamětí nalezne odpověď na dotazy na názvy a bude si pro příště pamatovat odpověď. Tím se příště zkrátí doba čekání, zejména pokud se nacházíte na pomalých připojeních.

Nejprve potřebujete soubor, nazvaný `/etc/named.conf`. Ten je čten při spuštění `named`. Pro tentokrát by měl jednoduše obsahovat:

```
// Config file for caching only DNS server

options {
  directory "/var/named";
  // Zrušení následujícího komentáře vám může pomoci
  // pokud při průchodu firwallem narazíte na problémy
  // query-source address * port 53;
};
```

```
zone "." {
  type hint;
  file "root.hints";
};

zone "0.0.127.in-addr.arpa" {
  type master;
  file "pz/127.0.0";
};
```

Řádek „`directory`“ sděluje `named`, kde se mají soubory hledat. Zde budou všechny soubory, které bude `named` používat. Proto je `pz` adresářem pod `/var/named`, tedy `/var/named/pz`. `/var/named` je adresář, který odpovídá *Linux File System Standard*.

Soubor `/var/named/root.hints` by měl obsahovat následující:

```
.       6D IN NS     G.ROOT-SERVERS.NET.
.       6D IN NS     J.ROOT-SERVERS.NET.
.       6D IN NS     K.ROOT-SERVERS.NET.
.       6D IN NS     L.ROOT-SERVERS.NET.
.       6D IN NS     M.ROOT-SERVERS.NET.
.       6D IN NS     A.ROOT-SERVERS.NET.
.       6D IN NS     H.ROOT-SERVERS.NET.
.       6D IN NS     B.ROOT-SERVERS.NET.
.       6D IN NS     C.ROOT-SERVERS.NET.
.       6D IN NS     D.ROOT-SERVERS.NET.
.       6D IN NS     E.ROOT-SERVERS.NET.
.       6D IN NS     I.ROOT-SERVERS.NET.
.       6D IN NS     F.ROOT-SERVERS.NET.

G.ROOT-SERVERS.NET. 5w6d16h IN A        192.112.36.4
J.ROOT-SERVERS.NET. 5w6d16h IN A        198.41.0.10
K.ROOT-SERVERS.NET. 5w6d16h IN A        193.0.14.129
L.ROOT-SERVERS.NET. 5w6d16h IN A        198.32.64.12
M.ROOT-SERVERS.NET. 5w6d16h IN A        202.12.27.33
A.ROOT-SERVERS.NET. 5w6d16h IN A        198.41.0.4
H.ROOT-SERVERS.NET. 5w6d16h IN A        128.63.2.53
B.ROOT-SERVERS.NET. 5w6d16h IN A        128.9.0.107
C.ROOT-SERVERS.NET. 5w6d16h IN A        192.33.4.12
```

```
D.ROOT-SERVERS.NET.  5w6d16h IN A        128.8.10.90
E.ROOT-SERVERS.NET.  5w6d16h IN A        192.203.230.10
I.ROOT-SERVERS.NET.  5w6d16h IN A        192.36.148.17
F.ROOT-SERVERS.NET.  5w6d16h IN A        192.5.5.241
```

VELICE DŮLEŽITÉ: V některých verzích tohoto dokumentu jsou výpisy programů uvedeny s prázdnými znaky nebo tabulátory před prvním znakem na řádku. Tak by to ve skutečných souborech nemělo být. Jestliže tedy soubory využíváte z tohoto dokumentu, **smažte všechny mezery na začátcích řádků**.

Nezapomeňte na to, co jsem říkal o mezerách na začátcích řádků!

Soubor popisuje světové kořenové DNS-servery. Ty se během doby mění a *musí* být udržovány. Jejich aktualizaci probírám v 6. části.

V `named.conf` následuje řádek `primary`. Jeho použití vysvětlím později, prozatím stačí v adresáři `pz` vytvořit soubor `127.0.0`:

```
@               IN      SOA     nslinux.bogus. hostmaster.linux.bogus. (
                                1       ; Serial
                                8H      ; Refresh
                                2H      ; Retry
                                1W      ; Expire
                                1D)     ; Minimum TTL
                        NS      ns.linux.bogus.

1                       PTR     localhost.
```

Pak potřebujete `/etc/resolv.conf`, který vypadá přibližně takto:

```
search subdomain.your-domain.edu your-domain.edu
nameserver 127.0.0.1
```

Řádek „`search`“ určuje, ve kterých doménách budou vyhledány názvy hostitelů, ke kterým se chcete připojit. Řádek „`nameserver`“ určuje adresu vašeho DNS-serveru - zde vašeho stroje, protože na něm spouštíte `named` (správně je 127.0.0.1, nezávisle na případných dalších adresách vašeho stroje). Jestliže hodláte vypsat několik DNS-serverů, pro každý z nich vložte jeden řádek „`nameserver`“ (poznámka: `named` tento soubor nikdy nečte, ale resolver, který používá `named`, jej čte).

Aby bylo jasné, co tento soubor dělá: Jestliže se klient pokusí vyhledat **foo**, pak se nejprve vyzkouší **foo.poddoména.vaše-doména.edu**, potom **foo.vaše-doména.edu** a nakonec **foo**. Jestli-

že se klient pokouší vyhledat **sunsite.unc.edu**, vyzkouší se nejprve **sunsite.unc.edu.poddoména.vaše-doména.edu** (ano, je to hloupé, ale tak to funguje), potom **sunsite.unc.edu.vaše-doména.edu** a nakonec **sunsite.unc.edu**. Na vyhledávací řádek tedy raději nedávejte příliš mnoho domén, vyhledávání by mohlo trvat dlouho.

Příklad předpokládá, že patříte do domény **poddoména.vaše-doména.edu**, váš stroj se tedy nazývá **váš-stroj.poddoména.vaše-doména.edu**. Vyhledávací řádek by neměl obsahovat vaši TLD (doménu nejvyšší úrovně, zde „edu"). Jestliže se často připojujete na hostitele v jiné doméně, můžete tuto doménu přidat do vyhledávacího řádku:

```
search subdomain.your-domain.edu your-domain.edu other-domain.com
```

A tak dále. Přirozeně, že zde použijete funkční názvy domén. Povšimněte si, že za názvy domén chybí tečky.

Poté musíte upravit buď `/etc/nsswitch.conf`, nebo `/etc/host.conf` (v závislosti na verzi libc). Jestliže máte `nsswitch.conf`, budeme jej upravovat, jestliže jej nemáte, budeme upravovat `host.conf`.

/`etc/nsswitch.conf` Tohle je dlouhý soubor, který určuje, odkud se vezmou různé datové typy, ze kterého souboru nebo databáze. Obsahuje nahoře obvykle užitečné komentáře, které si raději hned přečtěte. Poté nalezněte řádek, začínající „`hosts:`", měl by vypadat takto:

```
hosts: files dns
```

Jestliže zde není žádný řádek, který by začínal „`hosts`**:**", pak si jej vezměte odtud a vložte jej tam. Sděluje, že program by se měl dívat nejprve do souboru `/etc/hosts`, potom podle `resolv.conf` kontrolovat DNS.

`/etc/host.conf` Obsahuje pravděpodobně několik řádků, jeden by měl začínat **order** a vypadat takto:

```
order hosts,bind
```

Jestliže zde není žádný řádek „`order`", přidejte jej. Sděluje, že rozhodovací rutiny jména hostitele budou hledat nejprve v `/etc/hosts`, poté požádají DNS-server (v `resolv.conf` jste sdělili, že je to 127.0.0.1). Tyto dva poslední soubory jsou ve většině linuxových distribucích dokumentovány v manuálové stránce `resolv`(8) (proveďte „`man 8 resolv`"). Podle mého názoru je tato manuálová stránka celkem čitelná a měl by si ji přečíst každý, zejména správci DNS. Přečtěte si ji hned, když to budete odkládat, nikdy se k tomu nedostanete.

3.1 Spuštění named

Takže nastal čas spustit named. Jestliže používáte připojení přes modem, tak se nejprve připojte. Zadejte „`ndc start`“ a stiskněte Enter (bez voleb). Jestli to zlobí, zkuste místo toho „`/usr/sbin/ndc start`“. Pokud to pořád zlobí, přejděte k 8. části. Nyní můžete otestovat nastavení. Jestliže si při spuštění `named` (proveďte `tail -f /var/log/messages`) zobrazíte váš soubor se zprávami (obvykle se nazývá `/var/adm/messages`, ale může být i v adresáři `/var/log` nebo v souboru `syslog`), měl by se objevit podobný výpis:

(některé řádky jsou rozloženy do dvou)

```
Feb 15 01:26:17 roke named[6091]: starting. named 8.1.1 Sat Feb 14 \
00:18:20 MET 1998 ^Ijanl@roke.uio.no:/var/tmp/bind-8.1.1/src/bin/named
Feb 15 01:26:17 roke named[6091]: cache zone ""\
  (IN) loaded (serial 0)
Feb 15 01:26:17 roke named[6091]: master zone "0.0.127.in-addr.arpa" \
  (IN) loaded (serial 1)
Feb 15 01:26:17 roke named[6091]: listening [127.0.0.1].53 (lo)
Feb 15 01:26:17 roke named[6091]:\
  listening [129.240.230.92].53 (ippp0)
Feb 15 01:26:17 roke named[6091]:\
  Forwarding source address is [0.0.0.0].1040
Feb 15 01:26:17 roke named[6092]: Ready to answer queries.
```

Jestliže jsou zde nějaké zprávy o chybách, pak ta chyba někde musí být. `Named` pojmenuje soubor, ve kterém ji naleznete (doufám, že jeden z `named.conf` a `root.hints`). Pomocí `kill` zrušte `named`, vraťte se zpět a soubor zkontrolujte.

Nyní nastal čas spustit `nslookup` a vyzkoušet výsledek vašeho snažení.

```
$ nslookup
Default Server: localhost
Address: 127.0.0.1
>
```

Jestli dostanete uvedený výsledek, tak to funguje. Doufejme. Pokud se objeví něco jiného, vraťte se a všechno zkontrolujte. Pokaždé, když změníte soubor `named.conf`, musíte pomocí příkazu `ndc restart named` znovu spustit.

Nyní můžete zadávat dotazy. Zkuste vyhledávat některé blízké stroje. Pro mě je to **pat.uio.no** (na univerzitě v Oslo):

```
> pat.uio.no
Server: localhost
Address: 127.0.0.1

Name: pat.uio.no
Address: 129.240.130.16
```

`Nslookup` nyní požádal váš `named` o nalezení stroje **pat.uio.no**. Poté kontaktoval jeden z DNS-serverů, vyjmenovaných ve vašem souboru `root.hints`, a požádal o odpověď odtud. Výsledek můžete dostat až za chvíli, protože se prohledávají všechny domény, vyjmenované v `/etc/resolv.conf`.

Jestliže se na to stejné dotážete ještě jednou, dostanete následující:

```
> pat.uio.no
Server: localhost
Address: 127.0.0.1

Non-authoritative answer:
Name: pat.uio.no
Address: 129.240.2.50
```

Povšimněte si řádku „`Non-authoritative answer`**:**“, který se nám nyní objevil. Znamená to, že `named` se pro odpověď neodebral na síť, ale jen do své vyrovnávací paměti, kde ji také našel. Ale tato informace *by mohla* být stará. Proto jste na tohle (velmi malé) nebezpečí upozorněni sdělením „`Non-authoritative answer:`“ (neautorizovaná odpověď). Když vám to `nslookup` sdělí při druhém dotazu na stejnou lokaci, jedná se o znamení, že `named` funguje a ukládá si informace do vyrovnávací paměti. `Nslookup` ukončíte příkazem „**exit**“.

Nyní víte, jak nastavit `named` s vyrovnávací pamětí. Dejte si na oslavu jedno pivo, mléko nebo to, čemu dáváte přednost.

4 Jednoduchá doména

Jak nastavit vlastní doménu

4.1 Nejprve ale trochu suché teorie

Předtím, než *opravdu* začneme tuto část, vám předložím teorii o funkci DNS. A vy si ji pročtete, protože je pro vás poučná. Jestli se vám „nechce“, tak ji alespoň v rychlosti projděte. Ale nezapomeňte se zastavit u obsahu souboru `named.conf`.

DNS je hierarchický systém. Jeho vrchol se píše „.“ a vyslovuje „root“. Pod „.“ je mnoho domén nejvyšší úrovně (TLD). Nejznámější jsou ORG, COM, EDU a NET, ale je zde mnoho dalších.

Při vyhledávání stroje se dotaz zpracovává do hierarchie rekurzivně od vrcholu. Jestliže hodláte nalézt adresu pro **prep.ai.mit.edu**, musí váš DNS-server nalézt DNS-server, který obsluhuje doménu edu. Požádá .server (kořenový server je znám, od toho je zde soubor `root.hints`). Kořenový server poskytne seznam serverů edu:

```
$ nslookup
Default Server: localhost
Address: 127.0.0.1
```

Začátek dotazů na kořenový server:

```
> server c.root-servers.net.
Default Server: c.root-servers.net
Address: 192.33.4.12
```

Nastavení typu dotazu na NS (záznamy DNS serverů):

```
> set q=ns
```

Dotaz na edu:

```
> edu.
```

Tečka je zde důležitá - sděluje serveru, že se ptáme na edu přímo pod .(tím se vyhledávání zjednoduší).

```
edu    nameserver = A.ROOT-SERVERS.NET
edu    nameserver = H.ROOT-SERVERS.NET
```

```
edu    nameserver = B.ROOT-SERVERS.NET
edu    nameserver = C.ROOT-SERVERS.NET
edu    nameserver = D.ROOT-SERVERS.NET
edu    nameserver = E.ROOT-SERVERS.NET
edu    nameserver = I.ROOT-SERVERS.NET
edu    nameserver = F.ROOT-SERVERS.NET
edu    nameserver = G.ROOT-SERVERS.NET
A.ROOT-SERVERS.NET  internet address = 198.41.0.4
H.ROOT-SERVERS.NET  internet address = 128.63.2.53
B.ROOT-SERVERS.NET  internet address = 128.9.0.107
C.ROOT-SERVERS.NET  internet address = 192.33.4.12
D.ROOT-SERVERS.NET  internet address = 128.8.10.90
E.ROOT-SERVERS.NET  internet address = 192.203.230.10
I.ROOT-SERVERS.NET  internet address = 192.36.148.17
F.ROOT-SERVERS.NET  internet address = 192.5.5.241
G.ROOT-SERVERS.NET  internet address = 192.112.36.4
```

Tohle nám říká, že ***.root-servers.net** obsluhuje **edu**, takže můžeme požádat **c**. Nyní chceme vědět, kdo obsluhuje následující úroveň názvu domény – **mit.edu**:

```
> mit.edu.
Server: c.root-servers.net
Address: 192.33.4.12

Non-authoritative answer:
mit.edu nameserver = W20NS.mit.edu
mit.edu nameserver = BITSY.mit.edu
mit.edu nameserver = STRAWB.mit.edu

Authoritative answers can be found from:
W20NS.mit.edu internet address = 18.70.0.160
BITSY.mit.edu internet address = 18.72.0.3
STRAWB.mit.edu internet address = 18.71.0.151
```

Takže **mit** obsluhují **steawb**, **w20ns** a **bitsy**. Vyberte si jeden a pokračujte s **ai.mit.edu**:

```
> server W20NS.mit.edu.
```

U názvů hostitelů nezáleží na velkých a malých písmenech, zde jsou přímo okopírovány (pomocí paste) z obrazovky.

```
Server: W20NS.mit.edu
Address: 18.70.0.160

> ai.mit.edu.
Server: W20NS.mit.edu
Address: 18.70.0.160

Non-authoritative answer:
ai.mit.edu    nameserver = ALPHA-BITS.AI.MIT.EDU
ai.mit.edu    nameserver = GRAPE-NUTS.AI.MIT.EDU
ai.mit.edu    nameserver = TRIX.AI.MIT.EDU
ai.mit.edu    nameserver = MUESLI.AI.MIT.EDU
ai.mit.edu    nameserver = LIFE.AI.MIT.EDU
ai.mit.edu    nameserver = BEET-CHEX.AI.MIT.EDU
ai.mit.edu    nameserver = MINI-WHEATS.AI.MIT.EDU
ai.mit.edu    nameserver = COUNT-CHOCULA.AI.MIT.EDU
ai.mit.edu    nameserver = MINTAKA.LCS.MIT.EDU

Authoritative answers can be found from:
AI.MIT.EDU    nameserver = ALPHA-BITS.AI.MIT.EDU
AI.MIT.EDU    nameserver = GRAPE-NUTS.AI.MIT.EDU
AI.MIT.EDU    nameserver = TRIX.AI.MIT.EDU
AI.MIT.EDU    nameserver = MUESLI.AI.MIT.EDU
AI.MIT.EDU    nameserver = LIFE.AI.MIT.EDU
AI.MIT.EDU    nameserver = BEET-CHEX.AI.MIT.EDU
AI.MIT.EDU    nameserver = MINI-WHEATS.AI.MIT.EDU
AI.MIT.EDU    nameserver = COUNT-CHOCULA.AI.MIT.EDU
AI.MIT.EDU    nameserver = MINTAKA.LCS.MIT.EDU
ALPHA-BITS.AI.MIT.EDU      internet address = 128.52.32.5
GRAPE-NUTS.AI.MIT.EDU      internet address = 128.52.36.4
TRIX.AI.MIT.EDU            internet address = 128.52.37.6
MUESLI.AI.MIT.EDU          internet address = 128.52.39.7
LIFE.AI.MIT.EDU            internet address = 128.52.32.80
BEET-CHEX.AI.MIT.EDU       internet address = 128.52.32.22
```

```
MINI-WHEATS.AI.MIT.EDU     internet address = 128.52.54.11
COUNT-CHOCULA.AI.MIT.EDU   internet address = 128.52.38.22
MINTAKA.LCS.MIT.EDU        internet address = 18.26.0.36
```

Takže DNS-server pro **ai.mit.edu** je **muesli.ai.mit.edu**:

```
> server MUESLI.AI.MIT.EDU
Default Server: MUESLI.AI.MIT.EDU
Address: 128.52.39.7
```

Nyní změním typ požadavku (query). Nalezli jsme DNS-server, takže se ho zeptáme na vše, co ví o **prep.ai.mit.edu**.

```
> set q=any
> prep.ai.mit.edu.
Server: MUESLI.AI.MIT.EDU
Address: 128.52.39.7

prep.ai.mit.edu CPU = dec/decstation-5000.25 OS = unix
prep.ai.mit.edu
 inet address = 18.159.0.42, protocol = tcp
 ftp telnet smtp finger
prep.ai.mit.edu preference = 1, mail exchanger = gnu-life.ai.mit.edu
prep.ai.mit.edu internet address = 18.159.0.42
ai.mit.edu    nameserver = beet-chex.ai.mit.edu
ai.mit.edu    nameserver = alpha-bits.ai.mit.edu
ai.mit.edu    nameserver = mini-wheats.ai.mit.edu
ai.mit.edu    nameserver = trix.ai.mit.edu
ai.mit.edu    nameserver = muesli.ai.mit.edu
ai.mit.edu    nameserver = count-chocula.ai.mit.edu
ai.mit.edu    nameserver = mintaka.lcs.mit.edu
ai.mit.edu    nameserver = life.ai.mit.edu
gnu-life.ai.mit.edu        internet address = 128.52.32.60
beet-chex.ai.mit.edu       internet address = 128.52.32.22
alpha-bits.ai.mit.edu      internet address = 128.52.32.5
mini-wheats.ai.mit.edu     internet address = 128.52.54.11
trix.ai.mit.edu            internet address = 128.52.37.6
```

```
muesli.ai.mit.edu          internet address = 128.52.39.7
count-chocula.ai.mit.edu   internet address = 128.52.38.22
mintaka.lcs.mit.edu        internet address = 18.26.0.36
life.ai.mit.edu            internet address = 128.52.32.80
```

Takže počínaje u „.“ jsme postupně nalezli všechny DNS-servery pro následující úrovně názvu domény. Pokud byste místo použití všech ostatních serverů použili vlastní DNS-server, váš `named` by si samozřejmě všechny získané informace ukládal do vyrovnávací paměti a případný další dotaz na to stejné by už zodpověděl sám.

Méně diskutovanou, ale stejně důležitou doménou je **in-addr.arpa**. Je to „stejná“ doména jako ostatní. Umožňuje získat názvy hostitelů podle adres. Zde je důležité si povšimnout, že IP-adresy jsou v doméně `in-addr.arpa` napsány obráceně. Jestliže máte adresu stroje 192.128.52.43, named pokračuje stejně jako u předchozího příkladu **prep.ai.mit.edu**. Nalezne servery **arpa.**, nalezne servery **in-addr.arpa.**, nalezne servery **192.in-addr.arpa.**, nalezne servery **128.192.in-addr.arpa.**, nalezne servery **52.128.192.in-addr.arpa.**. Nalezne požadovaný záznam pro **43.52.128.192.in-addr.arpa.** Jasné? (odpovězte „Jasné!“) Tak dva roky se vám obrácené pořadí čísel může ještě zdát matoucí.

Teď jsem lhal. DNS sice nepracuje tak, jak jsem říkal, ale zato dost podobně.

4.2 Naše vlastní doména

Nyní k definici vaší vlastní domény. Chystáme se vytvořit doménu *linux.bogus* a definovat v ní stroje. Využívám zde fiktivní (bogus) název domény, aby bylo jasné, že tam nikoho nevyrušíme.

Ještě něco, než začneme: Ve jménech hostitelů nejsou povoleny všechny znaky. Jsme omezeni na znaky anglické abecedy (a-z), číslice (0-9) a pomlčky „-“. Držte se toho. Pro DNS se nerozlišují velká a malá písmenka. Takže **pat.uio.no** je to stejné, jako **Pat.UiO.No**.

Tuto část jsme již začali s tímto řádkem v `named.conf`:

```
zone "0.0.127.in-addr.arpa" {
 type master;
 file "pz/127.0.0";
};
```

Povšimněte si, že zde na koncích názvů domén není „.“. To znamená, že nyní budeme definovat zónu **0.0.127.in-addr.arpa**, pro kterou jsme hlavním serverem a která je uložena v souboru `pz/127.0.0`. Tento soubor už jsme nastavili, vypadá takto:

```
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                        1       ; Serial
                        8H      ; Refresh
                        2H      ; Retry
                        1W      ; Expire
                        1D)     ; Minimum TTL
                NS      ns.linux.bogus.
1               PTR     localhost.
```

Povšimněte si zde „.“ na konci všech plných názvů domén, což je v kontrastu se souborem `named.conf`. Někteří lidé rádi začínají každý zónový soubor direktivou `$ORIGIN`, ale je to zbytečné. Původ (kam náleží v hierarchii DNS) zónového souboru je určen na zónové řádce v souboru `named.conf`, v tomto případě to je **0.0.127.in-addr.arpa.**

Tento „zónový soubor“ obsahuje 3 „zdrojové záznamy“ (RR): SOA, NS a PTR. SOA je zkratkou pro začátek autority. „@“ je speciální označení, znamenající počátek, a protože „doménový“ sloupec pro tento soubor říká **0.0.127.in-addr.arpa**, první řádek ve skutečnosti znamená

```
0.0.127.in-addr.arpa. IN SOA ...
```

NS je zdrojový záznam pro DNS-server. Na začátku tohoto řádku není žádné „@“, je to *implicitní*, protože poslední řádek začínal na „@“. Tím se ušetří trocha psaní. Takže NS-řádek skutečně vypadá takto:

```
0.0.127.in-addr.arpa. IN NS ns.linux.bogus
```

Sděluje DNS, který stroj je DNS-serverem domény **0.0.127.in-addr.arpa**. Je to **ns.linux.bogus**. „ns“ je obecný název pro DNS-servery, ale název může být i jiný, stejně jako u webových serverů, které by měly být pojmenovány www.*něco*, ale často jsou pojmenovány jinak.

A konečně záznam PTR sděluje, že hostitel na adrese 1 v podsíti **0.0.127.in-addr.arpa** (127.0.0.1) je pojmenován **localhost**.

Záznam SOA je úvodem ke *všem* zónovým souborům a v každém zónovém souboru by měl být právě jeden - jako první záznam. Popisuje zónu, ze které pochází (stroj, pojmenovaný **ns.linux.bogus**), která je zodpovědná za jeho obsah (**hostmaster@linux.bogus**), jaké verze je tento zónový soubor (serial: 1) a další věci, které mají co do činění se sekundárními DNS-servery s vyrovnávací pamětí. Pro zbylá pole (refresh, retry, expire a minimum) použijte čísla z našeho dokumentu a měli byste být bez problémů.

Nyní znovu spusťte `named` (příkaz je `ndc restart`) a to co jste vytvořili otestujte pomocí `nslookup`:

```
$ nslookup

Default Server: localhost
Address: 127.0.0.1

> 127.0.0.1
Server: localhost
Address: 127.0.0.1

Name: localhost
Address: 127.0.0.1
```

Takže z 127.0.0.1 se získal **localhost**, to je dobré. Nyní pro náš hlavní úkol (doména **linux.bogus**) vložte do `named.conf` novou „zónovou“ část:

```
zone "linux.bogus" {
notify no;
type master;
file "pz/linux.bogus";
};
```

Povšimněte si, že na konci názvu domény v souboru `named.conf` opět chybí „.“.

Do zónového souboru `linux.bogus` vložíme některá zcela fiktivní data:

```
;
; Zone file for linux.bogus
;
; The full zone file
;
@       IN      SOA     ns.linux.bogus.      hostmaster.linux.bogus. (
                        199802151            ; serial, todays date +
                                               todays serial #
                        8H                   ; refresh, seconds
                        2H                   ; retry, seconds
                        1W                   ; expire, seconds
                        1D )                 ; minimum, seconds
;
```

```
                NS      ns                      ; Inet Address of DNS server
                MX      10 mail.linux.bogus ; Primary Mail Exchanger
                MX      20 mail.friend.bogus. ; Secondary Mail Exchanger
;
localhost       A       127.0.0.1
ns              A       192.168.196.2
mail            A       192.168.196.4
```

K záznamu SOA musí být poznamenány ještě dvě věci. `ns.linux.bogus` musí být skutečným strojem s A-záznamem. Pro stroj, zmíněný v záznamu SOA, není povoleno mít záznam CNAME. Jeho jméno nemusí být „ns“ a může to být jakýkoliv povolený název hostitele. Dále hostmaster.linux.bogus by měl být čten jako **hostmaster@linux.bogus** a měl by to být poštovní alias nebo schránka, kde by měla osoba(y), spravující DNS, často číst poštu. Jakákoliv pošta, týkající se domény, je odesílána na zde vypsanou adresu. Název nemusí být „hostmaster“, ale nějaká jiná povolená e-mailová adresa. Ale i ten „hostmaster“ by *měl být* funkční.

V tomto souboru je jeden nový typ zdrojového záznamu, MX neboli Mail eXchanger (poštovní server). Sděluje poštovním systémům kam mají posílat poštu, určenou na **někdo@linux.bogus**, jmenovitě na **mail.linux.bogus** nebo **mail.friend.bogus**. Číslo před každým názvem stroje je priorita daného MX. Na MX s nejnižším číslem (10) by měla být pošta odesílána primárně. Jestliže to není možné, pošta je odeslána na MX s druhým nejvyšším číslem (sekundární), zde **mail.friend.bogus** s prioritou 20.

Spuštěním `ndc restart` znovu spusťte `named`. Výsledky zkontrolujte pomocí `nslookup`:

```
$ nslookup
> set q=any
> linux.bogus
Server: localhost
Address: 127.0.0.1

linux.bogus
       origin = ns.linux.bogus
       mail addr = hostmaster.linux.bogus
       serial = 199802151
       refresh = 28800 (8 hours)
       retry = 7200 (2 hours)
       expire = 604800 (7 days)
       minimum ttl = 86400 (1 day)
```

```
linux.bogus   nameserver = ns.linux.bogus
linux.bogus   preference = 10, mail exchanger = \
    mail.linux.bogus.linux.bogus
linux.bogus   preference = 20, mail exchanger = mail.friend.bogus
linux.bogus   nameserver = ns.linux.bogus
ns.linux.bogus      internet address = 192.168.196.2
mail.linux.bogus    internet address = 192.168.196.4
```

Po podrobném prozkoumání objevíte chybu. Řádek

```
linux.bogus preference = 10, mail exchanger = \
    mail.linux.bogus.linux.bogus
```

je celý špatně. Měl by vypadat

```
linux.bogus preference = 10, mail exchanger = mail.linux.bogus
```

Tu chybu jsem udělal schválně, abyste se mohli poučit. Když se podíváte do zónového souboru, zjistíte, že v řádku

```
MX 10 mail.linux.bogus ; Primary Mail Exchanger
```

chybí tečka. Jestliže název stroje nekončí v zónovém souboru tečkou, na jeho konec je přidán počátek, takže máme dvojitý **linux.bogus.linux.bogus**. Proto správně je buď

```
MX 10 mail.linux.bogus. ; Primary Mail Exchanger
```

nebo

```
MX 10 mail ; Primary Mail Exchanger
```

Já dávám přednost druhé formě, je zde méně psaní. Existují úzkoprsí uživatelé, kteří teď nesouhlasí, ale také uživatelé, kteří souhlasí. V zónovém souboru by měla být doména buď vypsána a zakončena „.“, nebo by neměla být vůbec zmíněna a brala by se implicitně z počátku.

Musím zde znovu zdůraznit, že v `named.conf` by *neměly* být za názvy domén „.“. Nedokážete si představit, jak často tyto přebývající nebo chybějící tečky komplikují život.

Takže zde je nový zónový soubor s některými informacemi navíc:

```
;
; Zone file for linux.bogus
;
; The full zone file
;
@       IN      SOA     ns.linux.bogus.      hostmaster.linux.bogus. (
                        199802151            ; serial, todays date +
                                               todays serial #
                        8H                   ; refresh, seconds
                        2H                   ; retry, seconds
                        1W                   ; expire, seconds
                        1D )                 ; minimum, seconds
;
                TXT     "Linux.Bogus, your DNS consultants"
                NS      ns              ; Inet Address of DNS server
                NS      ns.friend.bogus.
                MX      10 mail         ; Primary Mail Exchanger
                MX      20 mail.friend.bogus. ; Secondary Mail Exchanger

localhost       A       127.0.0.1

gw              A 1     92.168.196.1
                HINFO   "Cisco" "IOS"
                TXT     "The router"

ns               A      192.168.196.2
                MX      10 mail
                MX      20 mail.friend.bogus.
                HINFO   "Pentium" "Linux 2.0"
www             CNAME   ns

donald          A       192.168.196.3
                MX      10 mail
                MX      20 mail.friend.bogus.
                HINFO   "i486" "Linux 2.0"
                TXT     "DEK"
```

```
mail            A       192.168.196.4
                MX      10 mail
                MX      20 mail.friend.bogus.
                HINFO   "386sx" "Linux 1.2"

ftp             A       192.168.196.5
                MX      10 mail
                MX      20 mail.friend.bogus.
                HINFO   "P6" "Linux 2.1.86"
```

Nalezneme zde množství nových zdrojových záznamů: HINFO (informace o hostiteli) má dvě části a je zvykem je oddělovat. První částí je hardware nebo procesor stroje a druhou částí je software nebo operační systém stroje. Stroj, nazvaný „ns“, má procesor (CPU) Pentium a používá Linux 2.0. CNAME (kanonické jméno) je způsob pojmenování stroje více názvy. Takže www je aliasem pro ns.

Používání záznamu CNAME je trochu kontroverzní. Bezpečné pravidlo ale říká, že záznamy MX, CNAME nebo SOA by se *neměly* odkazovat na záznam CNAME, měly by se odkazovat pouze na něco se záznamem A. Takže následující by bylo špatně

```
foobar          CNAME   www                     ; NO!
```

ale toto by bylo správně

```
foobar          CNAME   ns                      ; Yes!
```

Pro adresy elektronické pošty je také bezpečné předpokládat, že CNAME není platný název hostitele:

webmaster@www.linux.bogus je podle předchozího nastavení neplatnou adresou elektronické pošty. Můžete předpokládat, že pouze málo poštovních správců „zvenku“ bude toto pravidlo akceptovat (i když vám funguje dobře). Způsob, jakým se situace vyřeší, je použití A-záznamů (a snad i dalších, jako jsou MX):

```
www             A       192.168.196.2
```

V určitých kruzích se použití CNAME *nedoporučuje*. Takže CNAME neberte *příliš* vážně. Ale tento dokument a množství hostitelů se tímto pravidlem neřídí.

Spuštěním `ndc reload` nahrajte novou databázi. Tím přinutíte `named`, aby svoje soubory znovu načetl.

```
$ nslookup
Default Server: localhost
Address: 127.0.0.1
> ls -d linux.bogus
```

To znamená, že všechny záznamy by měly být vypsány. Výsledek je tento:

```
[localhost]
$ORIGIN linux.bogus.
@             1D IN SOA     ns hostmaster (
                            199802151     ; serial
                            8H            ; refresh
                            2H            ; retry
                            1W            ; expiry
                            1D )          ; minimum

              1D IN NS      ns
              1D IN NS      ns.friend.bogus.
              1D IN TXT     "Linux.Bogus, your DNS consultants"
              1D IN MX      10 mail
              1D IN MX      20 mail.friend.bogus.
gw            1D IN A       192.168.196.1
              1D IN HINFO   "Cisco" "IOS"
              1D IN TXT     "The router"
mail          1D IN A       192.168.196.4
              1D IN MX      10 mail
              1D IN MX      20 mail.friend.bogus.
              1D IN HINFO   "386sx" "Linux 1.0.9"
localhost     1D IN A       127.0.0.1
www           1D IN CNAME   ns
donald        1D IN A       192.168.196.3
              1D IN MX      10 mail
              1D IN MX      20 mail.friend.bogus.
              1D IN HINFO   "i486" "Linux 1.2"
              1D IN TXT     "DEK"
ftp           1D IN A       192.168.196.5
              1D IN MX      10 mail
              1D IN MX      20 mail.friend.bogus.
```

```
                1D IN HINFO   "P6" "Linux 1.3.59"
ns              1D IN A       192.168.196.2
                1D IN MX      10 mail
                1D IN MX      20 mail.friend.bogus.
                1D IN HINFO   "Pentium" "Linux 1.2"
@               1D IN SOA     ns hostmaster (
                              199802151     ; serial
                              8H      ; refresh
                              2H      ; retry
                              1W      ; expiry
                              1D )    ; minimum
```

Toto je správné. Jak je vidět, vypadá to spíše jako samotný zónový soubor. Podívejme se, co se zde říká o samotném www:

```
> set q=any
> www.linux.bogus.
Server: localhost
Address: 127.0.0.1

www.linux.bogus canonical name = ns.linux.bogus
linux.bogus   nameserver = ns.linux.bogus
linux.bogus   nameserver = ns.friend.bogus
ns.linux.bogus internet address = 192.168.196.2
```

Jinými slovy **www.linux.bogus** má pravý název **ns.linux.bogus** a dává vám některé informace o ns, postačující k připojení.

Nyní máme polovinu za sebou.

4.3 Reverzní zóna

Nyní mohou programy převádět názvy z linux.bogus na adresy, ke kterým se mohou připojit. Nutná je ale také reverzní zóna, která DNS umožňuje převod adresy na název. Tento název je využíván mnoha servery různých druhů (FTP, IRC, WWW a další), aby se rozhodly, jestli s vámi budou mluvit nebo ne, a když se rozhodnou kladně, tak se ještě musí určit, jaká obdržíte práva. Pro plný přístup ke všem službám Internetu je nutná reverzní zóna.

Následující vložte do `named.conf`:

```
zone "196.168.192.in-addr.arpa" {
       notify no;
       type master;
       file "pz/192.168.196";
       };
```

Je to stejná situace jako u **0.0.127.in-addr.arpa** a obsahy jsou podobné:

```
@       IN      SOA     ns.linux.bogus. hostmaster.linux.bogus. (
                        199802151  ; Serial, todays date + todays serial
                        8H              ; Refresh
                        2H              ; Retry
                        1W              ; Expire
                        1D)             ; Minimum TTL
                NS      ns.linux.bogus.

1               PTR     gw.linux.bogus.
2               PTR     ns.linux.bogus.
3               PTR     donald.linux.bogus.
4               PTR     mail.linux.bogus.
5               PTR     donald.linux.bogus.
```

Nyní znovu spusťte named (`ndc restart`) a znovu otestujte výsledky vaší práce pomocí nslookup:

```
> 192.168.196.4
Server: localhost
Address: 127.0.0.1

Name: mail.linux.bogus
Address: 192.168.196.4
```

vypadá to v pořádku, takže vyzkoušejte úplně všechno:

```
> ls -d 196.168.192.in-addr.arpa
[localhost]
$ORIGIN 196.168.192.in-addr.arpa.
@               1D IN SOA       ns.linux.bogus. hostmaster.linux.bogus. (
                                199802151       ; serial
                                8H              ; refresh
```

```
                                2H              ; retry
                                1W              ; expiry
                                1D )            ; minimum

                    1D  IN NS       ns.linux.bogus.
1                   1D  IN PTR      gw.linux.bogus.
2                   1D  IN PTR      ns.linux.bogus.
3                   1D  IN PTR      donald.linux.bogus.
4                   1D  IN PTR      mail.linux.bogus.
5                   1D  IN PTR      donald.linux.bogus.
@                   1D  IN SOA      ns.linux.bogus. hostmaster.linux.bogus. (
                                199802151       ; serial
                                8H              ; refresh
                                2H              ; retry
                                1W              ; expiry
                                1D )            ; minimum
```

Opět správně!

Tady bych měl doplnit ještě několik věcí. IP-čísla, která jsem v příkladu využil, jsou vyňata z bloku „soukromých sítí“. Tato čísla není povoleno na Internetu veřejně používat. Proto je jejich použití v našem příkladu bezpečné. Další záležitostí je řádek `notify no`**;** `Named` z něho zjistí, že při aktualizaci jednoho ze svých zónových souborů nemá upozorňovat svoje sekundární servery. V `bind-8` může `named` při aktualizaci zóny upozornit další servery, vypsané v záznamech NS. To je vhodné pro pravidelné využívání, ale pro soukromé experimenty se zónami by tato funkce měla být vypnuta. Přece nechceme, aby náš experiment zamořil Internet.

A samozřejmě musím dodat, že tato doména a její adresy jsou všechny smyšlené. Příklad skutečné domény bude v následující části.

5 Příklad skutečné domény

Kde nalezneme některé soubory *skutečných* domén

Uživatelé navrhli, abych vedle ukázkového příkladu použil příklad skutečné funkční domény.

Příklad jsem použil se souhlasem Davida Bullocka z LAND-5. Tyto soubory byly aktuální 24. 9. 1996 a potom byly mnou upraveny, aby odpovídaly omezením `bind-8` a využívaly rozšíření. Takže zde máte určitý rozdíl oproti současné situaci na DNS-serverech LAND-5.

5.1 /etc/named.conf (nebo /var/named/named.conf)

Zde nalezneme základní vazby pro dvě reverzní zóny, které jsou nutné: síť 127.0.0 a podsíť 206.6.177 v LAND-5. A také základní vazbu pro přesměrovací zónu `land-5.com`. Povšimněte si také, že místo hromadění souborů v adresáři `pz` (jak to dělám já v tomto dokumentu) jsou soubory v adresáři `zone`.

```
// Boot file for LAND-5 DNS server

options {
       directory "/var/named";
};

zone "." {
       type hint;
       file "root.hints";
};

zone "0.0.127.in-addr.arpa" {
       type master;
       file "zone/127.0.0";
};

zone "land-5.com" {
       type master;
       file "zone/land-5.com";
};

zone "177.6.206.in-addr.arpa" {
       type master;
       file "zone/206.6.177";
};
```

Jestliže toto vložíte do vašeho souboru `named.conf`, abyste si s tím pohráli, *PROSÍM* použijte pro obě zóny land-5 v zónových částech `notify no`**;** - vyhnete se tak nehodám.

5.2 /var/named/root.hints

Uvědomte si, že tento soubor je dynamický a ten, který jsem zde popsal, je již zastaralý. Pro vás bude lepší využít nějaký novější, vytvořený pomocí `dig`, jak je výše popsáno.

```
; <<>> DiG 8.1 <<>> @A.ROOT-SERVERS.NET. =
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10
;; flags: qr aa rd; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 13
;; QUERY SECTION:
;; ., type = NS, class = IN

;; ANSWER SECTION:
.                    6D IN NS            G.ROOT-SERVERS.NET.
.                    6D IN NS            J.ROOT-SERVERS.NET.
.                    6D IN NS            K.ROOT-SERVERS.NET.
.                    6D IN NS            L.ROOT-SERVERS.NET.
.                    6D IN NS            M.ROOT-SERVERS.NET.
.                    6D IN NS            A.ROOT-SERVERS.NET.
.                    6D IN NS            H.ROOT-SERVERS.NET.
.                    6D IN NS            B.ROOT-SERVERS.NET.
.                    6D IN NS            C.ROOT-SERVERS.NET.
.                    6D IN NS            D.ROOT-SERVERS.NET.
.                    6D IN NS            E.ROOT-SERVERS.NET.
.                    6D IN NS            I.ROOT-SERVERS.NET.
.                    6D IN NS            F.ROOT-SERVERS.NET.

;; ADDITIONAL SECTION:
G.ROOT-SERVERS.NET.  5w6d16h IN A         192.112.36.4
J.ROOT-SERVERS.NET.  5w6d16h IN A         198.41.0.10
K.ROOT-SERVERS.NET.  5w6d16h IN A         193.0.14.129
L.ROOT-SERVERS.NET.  5w6d16h IN A         198.32.64.12
M.ROOT-SERVERS.NET.  5w6d16h IN A         202.12.27.33
A.ROOT-SERVERS.NET.  5w6d16h IN A         198.41.0.4
H.ROOT-SERVERS.NET.  5w6d16h IN A         128.63.2.53
B.ROOT-SERVERS.NET.  5w6d16h IN A         128.9.0.107
C.ROOT-SERVERS.NET.  5w6d16h IN A         192.33.4.12
```

```
D.ROOT-SERVERS.NET.  5w6d16h IN A        128.8.10.90
E.ROOT-SERVERS.NET.  5w6d16h IN A        192.203.230.10
I.ROOT-SERVERS.NET.  5w6d16h IN A        192.36.148.17
F.ROOT-SERVERS.NET.  5w6d16h IN A        192.5.5.241

;; Total query time: 215 msec
;; FROM: roke.uio.no to SERVER: A.ROOT-SERVERS.NET. 198.41.0.4
;; WHEN: Sun Feb 15 01:22:51 1998
;; MSG SIZE sent: 17 rcvd: 436
```

5.3 /var/named/zone/127.0.0

Pouze základy – klasický záznam SOA a záznam, který mapuje 127.0.0 na **localhost**. Nutné jsou oba. V tomto souboru by nemělo být nic dalšího. Tyto soubory nemusí být nikdy aktualizovány, pokud se nezmění adresy vašeho DNS-serveru nebo hostmastera.

```
@               IN      SOA     land-5.com. root.land-5.com. (
                                199609203     ; Serial
                                28800         ; Refresh
                                7200          ; Retry
                                604800        ; Expire
                                86400)        ; Minimum TTL
                        NS      land-5.com.

1                       PTR     localhost.
```

5.4 /var/named/zone/land-5.com

Zde vidíme povinný záznam SOA, požadovaný záznamy NS. Je vidět, že má sekundární DNS-server na `ns2.psi.net`. Tak to má být - jako zálohu *vždy* používat sekundární server na jiném hostiteli. Je zde také patrné, že jako hlavní hostitel je zde land-5, který se stará o mnoho různých internetových služeb. Využívá k tomu CNAME (alternativou je využití záznamů A).

Jak je vidět ze záznamu SOA, zónové soubory pochází z land-5.com, kontaktní osobou je **root@land-5.com. hostmaster** je další často používanou adresou kontaktní osoby. Sériové číslo je v upravitelném formátu yyyymmdd (rok, měsíc, den) s připojenými dnešními sériovými čísly; 20. 9. 1996 se jedná pravděpodobně o šestou verzi zónového souboru. Uvědomte si, že

sériové číslo se *musí* stále zvyšovat, zde máme pouze *jednu* číslici pro dnešní sériová čísla, takže po 9 editacích souboru se může v jeho editacích pokračovat až následující den. Zvažte využití dvou číslic.

```
@       IN      SOA l  and-5.com. root.land-5.com. (
                       199609206     ; serial, todays date
                                       + todays serial #
                       8H            ; refresh, seconds
                       2H            ; retry, seconds
                       1W            ; expire, seconds
                       1D )          ; minimum, seconds
                NS     land-5.com.
                NS     ns2.psi.net.
                MX 10  land-5.com.   ; Primary Mail Exchanger

localhost       A      127.0.0.1

router          A      206.6.177.1

land-5.com.     A      206.6.177.2
ns              A      206.6.177.3
www             A      207.159.141.192

ftp             CNAME  land-5.com.
mail            CNAME  land-5.com.
news            CNAME  land-5.com.

funn            A      206.6.177.2
@               TXT    "LAND-5 Corporation"

;
;       Workstations
;
ws-177200       A      206.6.177.200
                MX     10 land-5.com. ; Primary Mail Host
ws-177201       A      206.6.177.201
                MX     10 land-5.com. ; Primary Mail Host
```

```
ws-177202     A      206.6.177.202
              MX     10 land-5.com.       ; Primary Mail Host
ws-177203     A      206.6.177.203
              MX     10 land-5.com.       ; Primary Mail Host
ws-177204     A      206.6.177.204
              MX     10 land-5.com.       ; Primary Mail Host
ws-177205     A      206.6.177.205
              MX     10 land-5.com.       ; Primary Mail Host
; {Many repetitive definitions deleted - SNIP}
ws-177250     A      206.6.177.250
              MX     10 land-5.com.       ; Primary Mail Host
ws-177251     A      206.6.177.251
              MX     10 land-5.com.       ; Primary Mail Host
ws-177252     A      206.6.177.252
              MX     10 land-5.com.       ; Primary Mail Host
ws-177253     A      206.6.177.253
              MX     10 land-5.com.       ; Primary Mail Host
ws-177254     A      206.6.177.254
              MX     10 land-5.com.       ; Primary Mail Host
```

Jestliže otestujete nameserver z land-5, zjistíte, že názvy hostitelů mají formát **ws**_*číslo*. Od pozdějších verzí `bind` 4 začal named omezovat znaky, které mohou být použity v názvech lokací. Takže v `bind-8` by všechno nefungovalo a já jsem pomlčky „-“ nahradil podtržítky „_“.

Dále za zmínku stojí to, že pracovní stanice nemají jednotlivé názvy, ale spíše předpony, následované posledními dvěma částmi IP-adres. Použití takové konvence značně zjednoduší údržbu, ale je trochu neosobní a může být zdrojem odporu vašich uživatelů.

Zřejmě je také **funn.land-5.com** aliasem pro land-5.com, ale s použitím záznamu A, ne CNAME.

5.5 /var/named/zone/206.6.177

Tento soubor ještě později okomentuji.

```
@             IN     SOA    land-5.com. root.land-5.com. (
                            199609206     ; Serial
                            28800         ; Refresh
                            7200          ; Retry
```

```
 6                          04800         ; Expire
                            86400)        ; Minimum TTL
                            NS     land-5.com.
                            NS     ns2.psi.net.
;
;       Servers
;
1               PTR    router.land-5.com.
2               PTR    land-5.com.
2               PTR  funn.land-5.com.
;
;       Workstations
;
200             PTR    ws-177200.land-5.com.
201             PTR    ws-177201.land-5.com.
202             PTR    ws-177202.land-5.com.
203             PTR    ws-177203.land-5.com.
204             PTR    ws-177204.land-5.com.
205             PTR    ws-177205.land-5.com.
; {Many repetitive definitions deleted - SNIP}
250             PTR    ws-177250.land-5.com.
251             PTR    ws-177251.land-5.com.
252             PTR    ws-177252.land-5.com.
253             PTR    ws-177253.land-5.com.
254             PTR    ws-177254.land-5.com.
```

Reverzní zóna je částí konfigurace, která způsobuje nejvíce neštěstí. Používá se k nalezení názvu hostitele, když máte IP-adresu stroje. Příklad: jste IRC-server a přijímáte připojení z IRC-klientů. Jste ale norský IRC-server a chcete přijímat připojení pouze z Norska a dalších skandinávských zemí. Když obdržíte od klienta připojení, knihovna C vám může sdělit IP-adresu připojovaného stroje, protože IP-adresa klienta je obsažena ve všech paketech, posílaných po síti. Nyní můžete volat funkci, nazvanou gethostbyaddr, která vyhledá název hostitele podle IP-adresy. Gethostbyaddr požádá DNS-server, který stroj vyhledá. Předpokládejme, že klient se připojuje z **ws-177200.land-5.com**. IP-adresa, kterou knihovna C předá IRC-serveru, je **206.6.177.200**. Abychom nalezli název stroje, musíme nalézt **200.177.6.206.in-addr.arpa**. DNS server nejprve nalezne servery arpa, potom servery in-addr.arpa, pak bude pokračovat obrácenou cestou přes 206, poté 6, až nakonec nalezne server pro zónu 177.6.206.in-addr.arpa na land-5. Odtud získá odpověď, že pro **200.177.6.206.in-addr.arpa** máme záznam „`PTR ws-`

177200.land-5.com“, což znamená, že název, který odpovídá 206.6.177.200, je **ws-177200.land-5.com**. Stejně jako u příkladu vyhledávání **prep.ai.mit.edu** je toto vysvětlení zjednodušené.

Vraťme se k příkladu IRC-serveru. IRC-server přijímá připojení pouze ze skandinávských zemí (*.no, *.se, *.dk), přičemž název ws-177200.land-5.com zjevně neodpovídá podmínce a server spojení nepovolí. Kdyby zde *nebylo* žádné zpětné mapování 206.6.177.200 přes zónu in-addr.arpa, server by vůbec nebyl schopen nalézt název. Musel by 206.6.177.200 srovnávat s *.no, *.se a *.dk, takže by nic neodpovídalo.

Někteří lidé vám řeknou, že zpětná vyhledávací mapování jsou důležitá pouze pro servery nebo nejsou důležitá vůbec. Ne tak zcela: Množství serverů FTP, news, IRC a dokonce HTTP (WWW) *nepřijme* připojení ze stroje, ke kterému nejsou schopny nalézt název. Takže zpětné mapování je u strojů v podstatě *povinné*.

6 Údržba

Udržujte v chodu

U named musíte kromě udržování chodu zajistit ještě jeden aspekt údržby. Jedná se o aktualizace souboru root.hints. Nejjednodušším způsobem je použití dig, nejprve jej spusťte bez argumentů, získáte root.hints podle vlastního serveru. Poté požádejte jeden z vypsaných kořenových serverů pomocí dig @rootserver. Zjistíte, že výstup vypadá jako soubor root.hints. Uložte jej do souboru (dig @e.root-servers.net .ns >root.hints.new) a nahraďte jím starý root.hints.

Po nahrazení souboru nezapomeňte znovu spustit named.

Al Longyear mi poslal tento skript, kterým se aktualizuje root.hints. Skript předpokládá, že máte funkční poštu a že je definován poštovní alias „hostmaster“. Aby skript odpovídal vašemu nastavení, musíte jej ještě upravit.

```
#!/bin/sh
#
# Update the nameserver cache information file once per month.
# This is run automatically by a cron entry.
#
(
echo "To: hostmaster <hostmaster>"
echo "From: system <root>"
```

```
echo "Subject: Automatic update of the named.conf file"
echo

export PATH=/sbin:/usr/sbin:/bin:/usr/bin:
cd /var/named

dig @rs.internic.net . ns >root.hints.new

echo "The named.conf file has been updated to contain the following
information:"
echo
cat root.hints.new

chown root.root root.hints.new
chmod 444 root.hints.new
rm -f root.hints.old
mv root.hints root.hints.old
mv root.hints.new root.hints
ndc restart
echo
echo "The nameserver has been restarted to ensure that the update is
    complete."
echo "The previous root.hints file is now called

/var/named/root.hints.old."
) 2>&1 | /usr/lib/sendmail -t
exit 0
```

Někteří z vás možná zjistili, že soubor `root.hints` je přes FTP k dispozici na Internicu. Prosím, *nepoužívejte* FTP k aktualizaci `root.hints`. Výše popsaná metoda je pro síť daleko přijatelnější.

7 Převod z verze 4 na verzi 8

Toto původně byla část, zabývající se použitím `bind` verze 8, kterou napsal David E. Smith (**dave@bureau42.ml.org**). Trochu jsem ji upravil, aby odpovídala svému novému názvu.

Není zde moc co dodat. Kromě použití `named.conf` místo `named.boot` je všechno stejné. A `bind8` se dodává s perlovým skriptem, který převádí soubory starých verzí na nové. Ukázkový `named.boot` (stará verze) pro DNS-servery pouze s vyrovnávací pamětí:

```
directory /var/named
cache .  root.hints
primary 0.0.127.IN-ADDR.ARPA 127.0.0.zone
primary localhost  localhost.zone
```

Na příkazovém řádku zadejte v adresáři `bind8/src/bin/named` (*zde se předpokládá, že máte zdrojovou distribuci, jestliže máte binární - skript někde je, ale já nevím kde - ed.*):

```
./named-bootconf.pl < named.boot > named.conf
```

Čímž se vytvoří `named.conf`:

```
// generated by named-bootconf.pl

options {
       directory "/var/named";
};

zone "." {
       type hint;
       file "root.hints";
};

zone "0.0.127.IN-ADDR.ARPA" {
       type master;
       file "127.0.0.zone";
};

zone "localhost" {
       type master;
       file "localhost.zone";
};
```

Funguje to pro všechno, co může jít do souboru `named.conf`, ačkoliv se nepřidávají všechna nová rozšíření a konfigurační volby, které `bind8` umožňuje. Zde následuje úplnější `named.conf`, který provádí to stejné, ale trochu efektivněji.

```
// This is a configuration file for named (from BIND 8.1 or later).
// It would normally be installed as /etc/named.conf.
// The only change made from the 'stock' named.conf (aside from this
// comment :) is that the directory line was uncommented, since I
// already had the zone files in /var/named.

options {
       directory "/var/named";
       check-names master warn;  /* default. */
       datasize 20M;
};

zone "localhost" IN {
       type master;
       file "localhost.zone";
       check-names fail;
       allow-update { none; };
       allow-transfer { any; };
};

zone "0.0.127.in-addr.arpa" IN {
       type master;
       file "127.0.0.zone";
       check-names fail;
       allow-update { none; };
       allow-transfer { any; };
};

zone "." IN {
       type hint;
       file "root.hints";
};
```

Vše je obsaženo v `bind8/src/bin/named/test`, kde jsou i kopie zónových souborů, které můžete ihned využít.

Formáty zónových souborů a souborů `root.hints` jsou shodné, stejně jako příkazy pro jejich aktualizace.

8 FAQ

V této části je seznam některých často pokládaných otázek (FAQ), které se vztahují k DNS a tomuto dokumentu. A jsou zde také odpovědi. Předtím, než mi napíšete, si ještě přečtěte tuto část.

1. Můj `named` požaduje soubor `named.boot`.

Čtete špatný dokument. Přečtěte si prosím starou verzi tohoto dokumentu, která pokrývá bind 4. Naleznete ji na `http://www.math.uio.no/janl/DNS/`

2. Jak mám DNS používat ze vnitřku firewallu?

Několik rad: „`forwarders`“, „`slave`“ a podívejte se na seznam literatury na konci tohoto dokumentu.

3. Jak DNS přinutit k cyklickému procházení adres, které jsou k dispozici pro nějakou službu (řekněme například www.busy.site), abychom získali efekt rovnoměrného nahrávání?

Vytvořte pro www.busy.site několik A-záznamů a používejte `bind` 4.9.3 nebo pozdější. `Bind` cyklicky obslouží odpovědi. Tohle *nebude* fungovat ve starších verzích `bind`.

4. Chci DNS nastavit na (uzavřeném) intranetu. Co mám dělat?

Zbavte se souboru `root.hints` a vytvořte zónové soubory. Znamená to také, že nemusíte stále tento soubor aktualizovat soubory hints.

5. Jak mám nastavit sekundární (podřízený) DNS-server?

Jestliže má primární server adresu 127.0.0.1, pak na svém sekundárním vložíte do souboru `named.conf` tento řádek:

```
zone "linux.bogus" {
type slave;
file "sz/linux.bogus";
masters { 127.0.0.1; };
};
```

Na seznamu `masters` můžete vypsat několik alternativních hlavních serverů, ze kterých je možné kopírovat zónu. Oddělujte je „;“ (středníkem).

6. Chci, aby `bind` běžel, i když jsem ze sítě odpojen.

Od Iana Clarka **<ic@deakin.edu.au>** jsem obdržel dopis, ve kterém popisuje způsob, který k tomu používá:

`Named` spouštím na svém „maškarádujícím" stroji. Mám dva soubory `root.hints` – jeden se jmenuje `root.hints.real` (obsahuje pravé názvy kořenových serverů) a druhý se jmenuje `root.hints.fake` a obsahuje...

```
----
; root.hints.fake
; tento soubor neobsahuje žádné informace
----
```

Když se odpojuji, kopíruji soubor `root.hints.fake` na `root.hints` a znovu spouštím `named`.

Když se připojuji, kopíruji soubor `root.hints.real` na `root.hints` a znovu spouštím `named`.

Tohle provádím přes `ip-down` (resp. `ip-up`).

Jakmile poprvé požaduji název domény, když nejsem připojen, `named` nemá k dispozici žádné podrobnosti, takže vypustí následující zprávu..

```
Jan 28 20:10:11 hazchem named[10147]: No root nameserver for
class IN
```

s čímž dokáži vyžít.

U mě vše spolehlivě funguje. DNS-server mohu pro lokální stroje využívat bez pauz při neúspěšném vyhledávání externích názvů domén (když nejsem připojen k Internetu). Když jsem připojen k Internetu, vyhledávání probíhá normálně.

7. Kam si DNS-server s vyrovnávací pamětí ukládá vyrovnávací paměť? Je zde nějaký způsob, jakým bych mohl kontrolovat její velikost?

Vyrovnávací paměť je celá uložena v paměti, nikdy se *neukládá* na disk. Kdykoliv ukončíte `named`, vyrovnávací paměť je ztracena. Vyrovnávací paměť *není* možné žádným způsobem kontrolovat. `Named` ji spravuje podle určitých jednoduchých pravidel a tak to má být. Vyrovnávací paměť nebo její velikost nemůžete z žádného důvodu ovládat. Jestli k tomu máte přesto chuť, musíte si programově upravit celý `named`. To samozřejmě příliš nedoporučujeme.

8. Ukládá `named` svoji vyrovnávací paměť mezi jednotlivými restarty? Mohu ho přinutit, aby ji uložil?

Ne, `named` při svém zániku vyrovnávací paměť *neukládá*. To znamená, že vyrovnávací paměť se musí při každém restartu znovu vytvořit. Neexistuje *žádný* způsob, jak `named` přinutit uložit svoji vyrovnávací paměť do souboru. Jestli k tomu máte přesto chuť, musíte si opět upravit celý `named`. To ale samozřejmě příliš nedoporučujeme.

9 Jak se stát lepším správcem DNS

Dokumentace a nástroje

Vhodná dokumentace existuje. Na síti i v tištěné podobě. K tomu, aby se z obyčejného správce DNS stal správce lepší, je nutné si část této dokumentace pročíst. Tištěná dokumentace, to je zejména kniha *DNS and BIND* od C. Liu a P. Albitze, vydaná v O'Reilly & Associates, Sebastopol, CA, ISBN 0-937175-82-X. Četl jsem ji, je skvělá. O DNS pojednává také část knihy *TCP/IP Network Administration* od Craiga Hunta, vydaná v O'Reilly..., ISBN 0-937175-82-X. Další nutnou koupí pro dobrou správu DNS (nebo spíše něčeho jiného) je *Zen and the Art of Motorcycle Maintenance* od Roberta M. Prisiga :-). K dispozici pod ISBN 0688052304 a další.

Na síti naleznete další zajímavosti na `http://www.dns.net/dnsrd/`, `http://www.isc.org/bind.html`; jsou to FAQ, referenční příručka (BOG) a definice protokolů spolu s úpravami DNS (většinu z toho, plus některá níže zmíněná RFC jsou obsažena v distribuci `bindu`). Většinu z toho jsem nečetl, ale já také nejsem žádný skvělý správce. Na druhou stranu Arnt Gulbrandsen četl BOG a líbilo se mu to :-). K DNS se vztahuje skupina news – comp.protocols.tcp-ip.domains. Navíc k DNS existuje množství RFC, z nichž nejdůležitější jsou pravděpodobně tato:

RFC 2052

A. Gulbrandsen, P. Vixie, *A DNS RR for specifying the location of services (DNS SRV)*, October 1996

RFC 1918

Y. Rekhter, R. Moskowitz, D. Karrenberg, G. de Groot, E. Lear,

Address Allocation for Private Internets, 02/29/1996.

RFC 1912

D. Barr, *Common DNS Operational and Configuration Errors*, 02/28/1996.

RFC 1912 Errors

B. Barr, Errors in RFC 1912, k dispozici také na `http://www.cis.ohio-state.edu/~barr/rfc1912-errors.html`

RFC 1713

A. Romao, *Tools for DNS debugging*, 11/03/1994.

RFC 1712

C. Farrell, M. Schulze, S. Pleitner, D. Baldoni, *DNS Encoding of Geographical Location*, 11/01/1994.

RFC 1183

R. Ullmann, P. Mockapetris, L. Mamakos, C. Everhart, *New DNS RR Definitions*, 10/08/1990.

RFC 1035

P. Mockapetris, *Domain names – implementation and specification*, 11/01/1987.

RFC 1034

P. Mockapetris, *Domain names – concepts and facilities*, 11/01/1987.

RFC 1033

M. Lottor, *Domain administrators operations guide*, 11/01/1987.

RFC 1032

M. Stahl, *Domain administrators guide*, 11/01/1987.

RFC 974

C. Partridge, *Mail routing and the domain system*, 01/01/1986.

,

Kapitola

5

Jádro Linuxu

Brian Ward, **bri@blah.math.tu-graz.ac.at** verze 0.80, 26. května 1997

Podrobný průvodce ke konfiguraci, kompilaci, aktualizaci a odstraňování problémů u jádra na systémech, založených na procesorech ix86.

1 Úvod

Měli byste vůbec číst tento dokument? Dobrá, tak si projděte následující příklady a zjistěte, jestli vám alespoň jeden neodpovídá:

- „Hanba! Tenhle wizzo-46.5.6 tvrdí, že potřebuje jádro verze 1.8.193 a já mám stále jen verzi 1.0.9!“
- V jednom z novějších jader je ovladač zařízení, které se chystáte připojit.
- Nevíte, jak se kompiluje jádro.
- „Je to, co se píše v `README`, *opravdu* všechno?“
- Zkusili jste to, ale nefunguje to.
- Musíte nějak uspokojit lidi, kteří po vás žádají instalaci jádra.

1.1 Nejprve si přečtěte tohle! (myslím to vážně)

Některé z příkladů tohoto dokumentu předpokládají, že máte GNU `tar`, `find` a `xargs`. To je standard; zde by neměly nastat problémy. Předpokládám také, že na svém systému znáte strukturu systému souborů; pokud neznáte, měli byste si zapsat na papír výstup příkazu `mount` během normální funkce systému (nebo výpis `/etc/fstab`, jestliže jej můžete číst). Tato informace je důležitá a pokud nerozdělíte jinak disk, nepřidáte další, nepřeinstalujete systém nebo něco podobného, tak se nemění.

Poslední verzí jádra byla v době psaní dokumentu verze 2.0.30, což znamená, že odkazy a příklady se vztahují právě k ní. I když se snažím dokument pojmout nezávisle na verzích, jádro se stále vyvíjí, takže pokud budete mít nějaké novější, mohou se objevit odlišnosti. Větší problémy by se objevit neměly, ale k určitému zmatení by dojít mohlo.

Existují dvě verze zdrojových textů jádra Linuxu - „produkční“ a „vývojová“. Produkční verze začínají u 1.0.x a jsou číslovány sudě; 1.0.x je produkční, 1.2.x je produkční, stejně jako 2.0.x. Tato jádra jsou v době svého vydání považována za nejstabilnější a bez chyb. Vývojová jádra (1.1.x, 1.1.3 atd.) jsou považována za zkušební, určená lidem, kteří by je mohli testovat a odhalovat možné chyby. Takže varováni jste již byli.

1.2 Poznámka ke stylu

Text, který vypadá takto, je buď něco, co se objeví na vaší obrazovce, název souboru nebo něco, co je možné přímo zadat (příkazy, volby). Jestliže si ale tento text prohlížíte v čistě textové podobě, rozdíl samozřejmě nepoznáte. Příkazy a další vstupy jsou často v uvozovkách („ “), přičemž zde se objevuje klasický problém s tečkou. Když se položka v uvozovkách objeví na konci věty, v Americe se tečka píše ještě před poslední apostrof. Pokud zde tedy budu chtít naznačit, abyste napsali „`make config`“, napíši „`make config`“, ne „`make config`.“

2 Důležité otázky a odpovědi

2.1 Mimochodem, co vlastně jádro dělá?

Unixové jádro funguje jako prostředník mezi vašimi programy a hardwarem. Zejména provádí správu paměti všech běžících programů (procesů) a zaručuje, že všechny dostanou patřičný (nebo nepatřičný) podíl cyklu procesoru. Navíc poskytuje krásné, snadno propojitelné rozhraní programů pro komunikaci s hardwarem.

K operacím jádra náleží samozřejmě mnohem více, ale tyto základní funkce jsou nejdůležitější, které je potřeba znát.

2.2 Proč bych měl chtít aktualizovat jádro?

Novější jádra obecně nabízí schopnost komunikace s větším množstvím typů hardwaru (to znamená, že mají více ovladačů zařízení), mohou mít lepší správu procesů, mohou být rychlejší, stabilnější a upravují chyby starých verzí jader. Většině lidí stačí jako důvod pro aktualizaci nové ovladače zařízení a nápravy chyb.

2.3 Jaký druh hardwaru novější jádra podporují?

Viz dokument o hardware. Kromě toho můžete nahlédnout do souboru „`config.in`“ ve zdrojovém textu Linuxu nebo si prostě vyzkoušíte „`make config`“. Tak se zobrazí veškerý hardware, podporovaný standardní distribucí jádra, ale ne všechen, který podporuje Linux; množství běžných ovladačů zařízení (například PCMCIA) je dodáváno odděleně ve formě modulů, určených k nahrání.

2.4 Jakou verzi gcc a libc potřebuji?

Verzi gcc doporučuje Linus v souboru `README`, dodávaném s linuxovým zdrojovým textem. Jestliže tuto verzi nemáte, dokumentace v doporučené verzi gcc by vám měla sdělit, jestli potřebujete aktualizovat libc. Postup není obtížný, ale je důležité přesně sledovat instrukce.

2.5 Co je to modul, určený k nahrání?

Existují části kódu jádra, které nejsou přilinkovány (obsaženy) přímo v jádru. Je možné je kompilovat odděleně a téměř kdykoliv je přidat nebo odstranit z běžícího jádra. Vzhledem k pružnosti se nyní jedná o upřednostňovaný způsob kódování určitých funkcí jádra. Mezi moduly, určenými k nahrání, nalezneme mnohé oblíbené ovladače zařízení, jako jsou ovladače PCMCIA a ovladače pásek QIC-80/40.

2.6 Kolik potřebuji místa na disku?

To závisí na vaší konkrétní systémové konfiguraci. Komprimovaný zdrojový text linuxu zabírá ve verzi 2.0.10 téměř 6 MB. Některé servery si jej uchovávají i v nezkomprimované podobě, která zabírá 24 MB. Ale to není vše – ke kompilaci potřebujete mnohem více. To závisí na tom, kolik toho nakonfigurujete do svého jádra. Já mám například na jednom stroji nakonfigurovánu podporu sítě, ovladač 3Com 3C509 a tři systémy souborů, což využívá 30 MB.

Na přidání komprimovaného linuxového zdrojového textu potřebujete při takové konfiguraci 36 MB. Na jiném systému bez podpory síťového adaptéru (ale s ponechanou podporou sítě) a navíc s podporou zvukové karty se požadované místo ještě zvětšuje. Nové jádro také vždy zabírá více místa než staré, takže pokud máte dostatečné možnosti v hardwaru, zajistěte si dostatek diskové kapacity (při dnešních cenách se vyplatí i mít i větší rezervu).

2.7 Jak dlouho to potrvá?

Pro většinu lidí je odpověď „dost dlouho“. Rychlost vašeho systému a množství paměti, které máte, rychlost přímo určí, ale ještě zde může mít menší vliv množství položek, konfigurovaných v jádru. Na 486DX4/100 s 16 MB RAM s jádrem v1.2 s pěti systémy souborů, podporou sítě a ovladači zvukových karet to zabere kolem 20 minut. Na 386DX/40 s 8 MB RAM a při podobné konfiguraci to bude téměř 1,5 hodiny. Zde se doporučuje trocha kávy, televize, pletení nebo jakákoliv jiná zábava, prováděná po dobu kompilace jádra. Jestliže máte opravdu pomalý stroj, můžete si jádro nechat kompilovat u někoho jiného (s rychlým strojem).

3 Jak skutečně nakonfigurovat jádro

3.1 Získání zdrojového textu

Zdrojový text je možné získat na anonymním ftp-serveru **ftp.funet.fi** v adresáři `/pub/Linux/PEOPLE/Linus`, na jeho mirrorech nebo jiných serverech. Označení je většinou `linux-x.y.z.tar.gz`, kde `x.y.z` je číslo verze. Novější (lepší?) verze a patche jsou v podadresářích, jako je „`v1.1`“ a „`v1.2`“. Nejvyšší číslo znamená poslední verzi a obvykle se jedná o testovací, takže pokud vás děsí alfa a beta verze, držte se raději těch základních.

Místo **ftp.funet.fi*** *silně* doporučuji mirrory. Zde je krátký seznam těch základních:

USA:	`sunsite.unc.edu:/pub/Linux/kernel`
USA:	`tsx-11.mit.edu:/pub/linux/sources/system`
Velká Británie:	`sunsite.doc.ic.ac.uk:/pub/unix/Linux/sunsite.unc-mirror/kernel`
Rakousko:	`ftp.univie.ac.at:/systems/linux/sunsite/kernel`
Německo:	`ftp.Germany.EU.net:/pub/os/Linux/Local.EUnet/Kernel/Linus`

* Poznámka korektora: Primárním zdrojem pro zdrojové texty operačního systému Linux je nyní server ftp.kernel.org.

Německo:	`sunsite.informatik.rwth-aachen.de:/pub/Linux/PEOPLE/Linus`
Francie:	`ftp.ibp.fr:/pub/linux/sources/system/patches`
Austrálie:	`sunsite.anu.edu.au:/pub/linux/kernel`

Obecně bývá nejlepším místem mirror **sunsite.unc.edu**. Soubor `/pub/Linux/MIRRORS` obsahuje seznam jeho známých mirrorů. Jestliže nemáte možnost používat ftp, na comp.os.linux.announce je pravidelně zasílán seznam systémů BBS, kde je možné Linux získat.

Jestliže hledáte obecné informace o Linuxu a jeho distribuci, je zde `http://www.linux.org`.

3.2 Rozbalení zdrojového textu

Přihlaste se jako „**root**“ (nebo použijte **su**) a proveďte `cd /usr/src`. Jestliže jste instalovali zdrojový text jádra již při instalaci Linuxu (což je běžné), měl by zde být adresář „`linux`“, který obsahuje celý starý zdrojový strom. Jestliže máte dostatek místa na disku a chcete postupovat bezpečně, tento adresář zachovejte. Dobrý nápad je zjistit, kterou verzi váš systém nyní používá, a podle toho adresář přejmenovat. Příkaz „`uname -r`“ vypíše verzi aktuálního jádra. Proto pokud „`uname -r`“ hlásí „**1.0.9**“, přejmenujete (pomocí „`mv`“) „`linux`“ na „`linux-1.0.9`“. Jestliže se nehodláte obtěžovat, prostě zrušte celý adresář. V každém případě nesmí být v **/usr/src** před rozbalením plného zdrojového kódu žádný adresář „`linux`“.

Nyní v **/usr/src** rozbalte zdrojový text pomocí „`tar zxpvf linux-x.y.z.tar.gz`“ (jestliže máte jen soubor `.tar` bez koncovky `.gz`, stačí „`tar xpvf linux.x.y.z.tar`“). Obsah archivního souboru se rozbalí. Poté se v `/usr/src` objeví nový adresář „`linux`“. Proveďte `cd linux` a projděte si soubor `README`. Měla by zde být část s nadpisem „`INSTALLING the kernel`“. Proveďte příslušné instrukce – vytvoření symbolických odkazů*, odstranění souborů `.o` atd.

3.3 Konfigurace jádra

Poznámka: Část je zde jen opakováním podobné části ze souboru „`README`“.

Příkaz „`make config`“, provedený v `/usr/src/linux` spustí skript, který vám položí mnohé otázky. Požaduje bash, takže ověřte, že bash je `/bin/bash, /bin/sh` nebo `$BASH`.

K „**make config**“ existují určité alternativy a vám se mohou jevit i snazší a pohodlnější. Pro ty, kteří používají X-Window System, je zde „**make xconfig**“ (máte li instalováno Tk). „**make menuconfig**“ je pro ty, co nemají rádi ano/ne a raději používají textová menu. Tato rozhraní mají jednu výhodu – pokud se při volbě zmýlíte, není problém se vrátit a opravit.

* Poznámka korektora: Novější verze jádra Linuxu již tyto odkazy vytvářejí samy.

Připravte se na odpovědi na otázky, většinou „y“ (ano) nebo „n“ (ne). Ovladače zařízení mají většinou volbu „m“. Znamená to „modul“, což znamená kompilaci ne do jádra, ale jako modul, určený k nahrání. Trochu méně vážné vysvětlení pak zní jako „možná“. Některé jasnější a méně kritické volby se zde nepopisují; stručný popis několika dalších viz část „Další konfigurační volby“.

V 2.0.x a pozdějších je volba „?“, která nabízí stručný popis konfiguračních parametrů. Tato informace je nejaktuálnější.

3.3.1 Matematická emulace v jádru

Jestliže nemáte matematický koprocesor (máte samotné 386 nebo 486SX), musíte zde odpovědět „y“. Pokud koprocesor máte a odpovíte „y“, příliš se nevzrušujte – koprocesor je stejně využíván a emulace ignorována. Jediný následek je zvětšení jádra (obsadí více RAM). Slyšel jsem, že emulace je pomalá; ačkoliv to by nám zde až tak vadit nemuselo, to se projeví až ve špatných výkonech X-Window System.

3.3.2 Podpora normálního (MFM/RLL) disku a IDE disku/cdrom

Pravděpodobně je potřebujete podporovat; to znamená, že jádro bude podporovat standardní PC pevné disky, které má většina lidí. Tento ovladač nezahrnuje ovladače SCSI; ty přijdou v konfiguraci později.

Poté budete dotázáni „old disk-only“ (pouze staré disky) a „new IDE drivers“ (nové IDE ovladače). Vyberete si jeden z nich; hlavní rozdíl je v tom, že starý ovladač podporuje pouze dva disky na jednom rozhraní, zatímco nový podporuje druhotné rozhraní a IDE/ATAPI cdrom mechaniky. Nový ovladač zabírá o 4 KB více než starý a je také trochu „vylepšen“ – to znamená, že kromě jiného množství chyb, které obsahuje, může zvýšit vaše výkony, obzvláště pokud vlastníte novější hardware (EIDE).

3.3.3 Podpora sítě

Jestliže je váš stroj na síti, jako je Internet, nebo chcete pro přístup přes modem používat SLIP, PPP, term..., tak v podstatě stačí odpovědět „y“. Ale množství balíků (například systém X-Window System) vyžaduje podporu sítě, i když na opravdové síti nejste, takže stejně musíte říci „y“. Později budete dotázáni, jestli chcete podporovat TCP/IP sítě; opět odpovězte „y“, nejste-li si zcela jisti.

3.3.4 Omezení paměti pod 16 MB

Existují chybné ovladače 386 DMA, které mají problémy s adresací čehokoliv nad 16 MB RAM; jestliže jste tento (vzácný) případ, musíte odpovědět „y“.

3.3.5 System V IPC

Jednu z nejlepších definicí IPC (meziprocesorová komunikace) nalezneme v perlovém slovníku. Pak není překvapivé, že programátoři v Perlu mnoho balíků (nevyjímaje například DOOM) takto nechávají procesy hovořit mezi sebou. Takže zde odpovídat „n“ není zrovna moudré, pokud ovšem přesně nevíte, co děláte.

3.3.6 Typ procesoru (386, 486, Pentium, PPro)

(ve starších jádrech: optimalizace pro 486 využijete s přepínačem -m486)

Většinou se zde kompilace prováděla s optimalizací pro určitý procesor; jádra běžela i na jiných procesorech, ale byla o něco větší. V novějších jádrech už to tak není, takže byste měli zadat procesor, pro který kompilaci provádíte. Jádro „386“ bude fungovat na všech strojích.

3.3.7 Podpora SCSI

Jestliže máte SCSI zařízení, odpovězte „y“. Budete požádáni o další informace, jako je podpora CD-ROM, disků a druh použitého SCSI adaptéru. Více podrobností viz dokument SCSI-HOWTO.

3.3.8 Podpora síťových zařízení

Jestliže máte síťovou kartu nebo byste rádi pro připojení k Internetu použili SLIP, PPP nebo adaptér na paralelní port, odpovězte „y“. Konfigurační skript se vás dotáže na druh karty a použitý protokol.

3.3.9 Systémy souborů

Konfigurační skript se vás pak zeptá, jestli si přejete podporovat následující systémy souborů:

Standard (minix) - novější distribuce systémy souborů minix nevytváří a množství lidí je nevyužívá, ale přesto se jejich nakonfigurování může hodit. Využívají je některé programy na „záchranu disku“ a dále množství disket, pro které je tento systém výhodnější.

Extended fs – tohle byla první verze rozšířeného systému souborů, který se ale už moc nepoužívá. Jestliže jej znáte, možná jej budete potřebovat, jestliže nevíte, tak ho ani nepotřebujete.

Second extended – V nových distribucích se používá masově. Jednu z nich pravděpodobně máte a musíte odpovědět „y“.

xiafs – svého času nebyl neznámý, ale dnes neznám nikoho, kdo by jej používal.

msdos – jestliže hodláte používat svoje dosové disky nebo diskety, odpovězte „y“.

umsdos – tento systém rozšiřuje systém souborů MS-DOS o obvyklé unixové věci, jako jsou dlouhé názvy souborů. Není nutný pro lidi (jako jsem já), kteří „nedělají v MS-DOSu“.

`/proc` – další z velkých věcí od dob sušeného mléka (myšlenka byla podle mého předpokladu hanebně ukradena z Bellových laboratoří). Nejedná se o systém souborů na disku; je to systém souborů jako rozhraní mezi jádrem a procesy. Využívají jej mnohé vypisovače procesů (jako je „`ps`“). Někdy si zkuste „`cat /proc/meminfo`“ nebo „`cat /proc/devices`“. Některé příkazové interprety (jmenovitě rc) používají pro I/O `/proc/self/fd` (na jiných systémech známý jako `/dev/fd`). Zde musíte skoro jistě odpovědět „`y`“; závisí na tom mnoho důležitých nástrojů Linuxu.

NFS – jestliže váš stroj žije na síti a vy chcete využívat systémy souborů z jiných systémů pod NFS, odpovězte „`y`“.

ISO9660 – je na většině CD-ROM. Jestliže máte mechaniku CD-ROM a chcete ji pod Linuxem používat, odpovězte „`y`“.

OS/2 HPFS – v době psaní tohoto dokumentu se jedná o systém souborů pro samotné čtení OS/2 HPFS.

System V and Coherent – System V a Coherent jsou další varianty na PC Unix.

Ale já nevím, které systémy souborů potřebuji! Dobrá, zadejte „`mount`“. Výstup by měl vypadat asi takto:

```
blah# mount
/dev/hda1 on / type ext2 (defaults)
/dev/hda3 on /usr type ext2 (defaults)
none on /proc type proc (defaults)
/dev/fd0 on /mnt type msdos (defaults)
```

Podívejte se na každý řádek; slovo vedle „**type**“ je typ systému souborů. V tomto příkladu jsou můj `/` a `/usr` systém souborů typu second extended, využívám `/proc` a je zde připojena disketová mechanika, využívající dosovský systém souborů.

Jestliže máte právě nastaveno `/proc`, můžete zkusit „`cat /proc/filesystems`“; vypíší se systémy souborů, aktuálně ve vašem jádru používané.

Nakonfigurováním zřídkakdy používaných, nekritických systémů souborů můžete jádro zahltit; jak se tomu vyhnout zjistíte níže, v části o modulech, a později se také dozvíte, proč je zahlcené jádro nežádoucí.

3.3.10 Znaková zařízení

Zde se nastavují ovladače pro tiskárnu (paralelní), busmouse, myš PS/2 (množství notebooků používá tento protokol pro svoje trackbally), některé páskové ovladače a další podobná „znaková“ zařízení. Když je to nutné, odpovězte „y“.

Poznámka: Selection je program, který umožňuje využití myši mimo X-Window System k operacím cut&paste mezi virtuálními konzolami. Je sice pěkné, že máte sériovou myš, protože v X-Window System funguje dobře, ale ještě je nutné umožnit provádění určitých speciálních triků. Kdysi byla podpora selection volitelná, dnes je již standardní.

Poznámka 2: Selection se nyní považuje za zastaralý. Novější program má název „gpm“. Dokáže více věcí, jako je překlad protokolů myší, obsloužení více myší...

3.3.11 Zvuková karta

Jestli silně toužíte po zvuku, odpovězte „y“ a později se vás další konfigurační program zeptá na všechno o vaší zvukové kartě. Poznámka ke konfiguraci zvukové karty: když máte rozhodnout, jestli se má instalovat plná verze ovladače, odpovězte „n“ a ušetříte část paměti jádra tím, že vyberete jen to, co opravdu potřebujete. Jestliže máte zvukovou kartu, doporučuji vám k pročtení dokument Sound-HOWTO.

3.3.12 Další konfigurační volby

Všechny konfigurační volby zde vypsány nejsou, protože se velmi často mění nebo mají význam zcela zřejmý již podle názvu (například 3Com 3C509 support je podpora ovladače dané ethernetové karty). Podrobný seznam všech voleb (a způsob jejich začlenění ve skriptu `Configure`) od Axela Boldta (axel@uni-paderborn.de) je k mání na adrese s následujícím URL:

http://math-www.uni-paderborn.de/~axel/config_help.html

nebo přes anonymní FTP na:

`ftp://sunsite.unc.edu/pub/Linux/kernel/config/krnl_cnfg_hlp.x.yz.tgz`

kde **x.yz** je číslo verze.

Pro pozdější jádra (od verze 2.0.x) je to začleněno do zdrojového stromu.

3.3.13 Úprava jádra

V `README` Linus píše:

Některé konfigurační volby (označené jako „kernel hacking“) mají obvykle za následek větší nebo pomalejší jádro (nebo obojí) a mohou zvýšit nestabilitu jádra – povolením některých rutin, určených k odstraňování problémových částí kódu jádra (kmalloc()). U „produkčního“ jádra je tedy lepší na tyto dotazy odpovědět „`n`“.

3.4 Co teď? (soubor Makefile)

Po provedení `make config` vám hlášení potvrdí ukončenou konfiguraci jádra, doporučí kontrolu nejvyššího `Makefile` s další konfigurací atd.

Takže se podívejte do **Makefile**. Nezměníte zde pravděpodobně nic, ale za podívání nic nedáte. Jakmile je nové jádro připraveno, můžete jeho volby změnit pomocí příkazu „`rdev`“.

4 Kompilace jádra

4.1 Čištění a závislosti

Jakmile se ukončí konfigurační skript, doporučí vám také „`make dep`“ a (možná) „`clean`“. Takže proveďte „`make dep`“. Tím zajistíte, že všechny závislosti, budou registrovány. Nemělo by to trvat dlouho, pokud nemáte opravdu pomalý počítač. U starších verzí jádra musíte po ukončení provést „`make clean`“. Tím se odstraní všechny objektové soubory a některé další věci, které stará verze po sobě zanechává. V každém případě na tento krok *nezapomeňte*, pokud se chystáte jádro překompilovat.

4.2 Doba kompilace

Po `dep` a `clean` můžete provést „`make zImage`“ nebo „`make zdisk`“ (tohle je ta část, která trvá tak dlouho). Jádro se kompiluje pomocí „`make zImage`“. V `arch/i386/boot` zůstane soubor „`zImage`“ (mimo jiné). Tohle je nové komprimované jádro. Stejnou věc provádí „`make zdisk`“, ale nový `zImage` ukládá na disketu, kterou musíte mít v mechanice „A:“. Volba „`zdisk`“ je vhodná pro testování nových jader; jestliže dojde k nějakému selhání, prostě vyjměte disketu z mechaniky a proveďte restart se starým jádrem. Jedná se také o šikovný způsob zavedení systému, pokud nechtěně smažete jádro (nebo něco stejně nepatřičného). Disketu můžete také využít při instalaci systému u někoho jiného – prostě mu předáte její obsah.

Všechna sudá pozdější jádra jsou komprimována (proto „`z`“ před jmény. Komprimované jádro se při spuštění automaticky dekomprimuje.

4.3 Další možnosti pro make

Volba „`make mrproper`“ provede mnohem rozsáhlejší „`clean`“. Někdy je nezbytná; můžete ji provést při každé úpravě. Tato volba také smaže váš konfigurační soubor, takže si jej (`.config`) můžete zálohovat, může-li být nějak užitečný.

Volba „`make oldconfig`“ se pokusí jádro konfigurovat podle starého konfiguračního souboru; který použije během procesu „`make config`“. Jestliže jste před tím ještě nikdy nekompilovali jádro nebo nemáte starý konfigurační soubor, nepoužívejte tuto volbu, protože byste se připravili o možnost úpravy implicitní konfigurace.

O „`make modules`“ pojednává část o modulech.

4.4 Instalace jádra

Jakmile máte nové jádro, které vypadá, že bude fungovat požadovaným způsobem, nastal čas jej nainstalovat. Většina lidí k tomu užívá LILO (Linux Loader). Volba „`make zlilo`“ nainstaluje jádro, spustí na něm LILO a připraví vás na zavedení systému, ALE POUZE v případě, že LILO je na vašem systému nakonfigurováno následujícím způsobem: jádro je `/vmlinuz`, LILO je v `sbin` a vaše lilo konfigurace LILO (`/etc/lilo.conf`) s tím souhlasí.

Jinak musíte LILO použít přímo. Tento balík se velice snadno instaluje a ovládá, ale často mate uživatele svým konfiguračním souborem. Podívejte se do něj (buď `/etc/lilo/config` u starších verzí nebo `/etc/lilo.conf` u novějších) a zjistěte aktuální nastavení. Konfigurační soubor vypadá asi takto

```
image = /vmlinuz
       label = Linux
       root = /dev/hda1
       ...
```

Řádek „`image =`“ je nastaven na aktuálně instalované jádro. Většina lidí používá `/vmlinuz`. Řádek „`label`“ se pro LILO využívá k určení, které jádro nebo operační systém se má zavádět, a „`root`“ je kořenovým adresářem příslušného operačního systému. Vytvořte záložní kopii vašeho starého jádra a okopírujte `zImage`, který jste právě vyrobili, na správné místo (jestliže používáte „`/vmlinuz`“, napíšete „`cp zImage /vmlinuz`“). Potom znovu spusťte LILO – na novějších systémech stačí spustit „`lilo`“, ale na starších budete možná muset provést `/etc/lilo/install` nebo dokonce `/etc/lilo/lilo -C /etc/lilo/config`.

Jestliže se chcete o konfigurování LILO dozvědět více nebo jestliže LILO nemáte, sežeňte si z vašeho oblíbeného ftp-serveru nejnovější verzi a řiďte se instrukcemi.

Při zavádění systému z jiného místa než obvykle (další způsob záchrany při pokažení nového jádra) okopírujte v konfiguračním souboru pro LILO řádky pod řádkem „**image = xxx**“ (a včetně) a změňte „`image = xxx`“ na „`image = yyy`“, kde „`yyy`“ je plná cesta k souboru, který jste uložili jako záložní jádro. Poté změňte „`label = zzz`“ na „`label = linux-backup`“ a znovu spusťte `lilo`. Do konfiguračního souboru možná ještě vložíte řádek „`delay=x`“, kde x je čas v setinách sekundy, který pro LILO určuje, jak dlouho má před zavedením systému čekat, abyste mohli zavedení ještě přerušit (například klávesou Shift) a zadat označení záložního obrazu jádra (kdyby například nebylo všechno v pořádku).

5 Úprava jádra

5.1 Aplikace záplat (patch)

Postupné aktualizace jádra se dodávají jako záplaty (patch). Jestliže máte například verzi 1.1.45 a existuje pro ni záplata „`patch46.gz`“, můžete aplikací této záplaty přejít na verzi 1.1.46. Nejprve si můžete vytvořit záložní kopii zdrojového stromu („`make clean`“ a poté „`cd /usr/src; tar zcvf old-tree.tar.gz linux`“ vám vytvoří komprimovaný archiv tar).

Takže když budeme pokračovat v předchozím příkladu, předpokládejme, že „`patch46.gz`“ máte v `/usr/src`. Proveďte `cd /usr/src` a „`zcat patch46.gz | patch -p0`“ (nebo „`patch -p0 < patch46`“, není-li záplata komprimována). Všechno vám pak na obrazovce proběhne (nebo projde, pokud máte pomalý systém) a sdělí, že se o něco úspěšně nebo neúspěšně pokouší. Většinou všechno proběhne příliš rychle, než aby se to dalo vnímat, takže použijte k `patch` přepínač **-s**, který zredukuje hlášení pouze na chybová (sice nebudete vnímat změny, prováděné vaším počítačem, ale možná je to tak lepší). Při hledání částí, které by nemusely proběhnout v pořádku, proveďte `cd /usr/src/linux` a zaměřte se na soubory s koncovkou `.rej`. Některé verze `patch` používají koncovku **#**. K vyhledání můžete využít příkaz „`find`“;

```
find . -name '*.rej' -print
```

Na standardní výstup se vytisknou všechny soubory s koncovkou `.rej`, které jsou v aktuálním adresáři nebo jeho podadresářích.

Jestliže vše proběhlo v pořádku, proveďte podle 3. a 4. části „`make clean`“, „`config`“ a „`dep`“.

Příkaz `patch` má několik voleb. Jak jsme již naznačili, `patch -s` odfiltruje všechna hlášení kromě chybových. Jestliže máte zdrojové texty jádra na jiném místě než je

/usr/src/linux, patch -p1 (v požadovaném adresáři) se provede bez problémů. Další volby patch jsou přesně dokumentovány na manuálové stránce.

5.2 Jestliže je něco špatně

(**Poznámka:** Tato část se týká zejména starých jader.)

Nejčastějším problém se objevil, když patch upravil soubor „config.in“, ale ne správně, protože jste si jej předtím upravili podle svého stroje. Dnes je již tento problém vyřešen, ale u starších verzí se s ním ještě můžete setkat. Nápravu provádíte po nahlédnutí do souboru config.in.rej a zjištění, co zbylo z původní podoby. Změny jsou většinou na začátku řádku označeny „+“ a „-“. Podívejte se na okolní řádky a vzpomeňte si, jestli byly nastaveny na „y“ nebo „n“. Nyní editujte config.in a na příslušných místech změňte „y“ na „n“ a obráceně. Proveďte

```
patch -p0 < config.in.rej
```

a jestliže se příkaz provede bez chyb, můžete v konfigurování a kompilaci pokračovat. Soubor config.in.rej vám zůstane, ale můžete ho smazat.

Jestli se objeví další problémy, může to být díky tomu, že máte špatnou verzi záplaty. Jestliže se objeví „previously applied patch detected: Assume -R?“, snažíte se pravděpodobně provést starší úpravu na novější verzi, než právě máte; jestliže přesto odpovíte „y“, dojde k pokusu o degradaci vašeho zdrojového textu, která většinou neprojde. Nezbude vám než si opatřit novější zdrojový strom.

Vrácení záplaty provedete z původní úpravy pomocí „patch -R“.

Nejvhodnější postup při selhání záplat je začít úplně znovu s čistým zdrojovým stromem (například z jednoho ze souborů linux-x.y.z.tar.gz).

5.3 Odstranění souborů .orig

Po pár úpravách se začnou hromadit soubory .orig. Například strom 1.1.51 byl naposledy promazán v dobách 1.1.48. Odstraněním souborů .orig jsem získal více než 0,5 MB volného místa.

```
find . -name '*.orig' -exec rm -f {} ';'
```

by mělo potřebné promazání provést. Verze patch, které pro .rej používají koncovku #, používají místo .orig tildu (~). Existují ale i lepší způsoby smazání souborů .orig, které závisí na GNU xargs:

```
find . -name '*.orig' | xargs rm
```

nebo ještě „jistá, ale trochu delší“ metoda:

```
find . -name '*.orig' -print0 | xargs --null rm --
```

5.4 Další záplaty

Existují další záplaty (říkám jim „nestandardní“), odlišné od záplat, distribuovaných Linusem. Jestliže je využijete, nemusí fungovat správně. Potom je musíte vrátit, opravit zdrojový text nebo záplatu, instalovat nový zdrojový text nebo tohle všechno současně. Tohle vás může dost potrápit, takže pokud nechcete všechno opravovat sami, vraťte záplaty zpět nebo instalujte nový strom. Pak se teprve podívejte, jestli vám nestandardní záplaty fungují. Jestliže nefungují, máte možnost buď laborovat se starým jádrem, nebo si počkejte na novou verzi záplaty.

Jak běžné jsou záplaty nedodávané standardně? Pravděpodobně o nich uslyšíte. Jednu jsem používal pro svoje virtuální konzoly, protože nemám rád blikající kurzory (tato úprava byla zatím pro nové verze jádra vždy obnovována). Protože se ale nové ovladače zařízení vyvíjí jako moduly, snižuje se i rozsah použití „nestandardních“ záplat.

6 Dodatečné balíky

Vaše linuxové jádro má množství funkcí, které nejsou vysvětleny v samotném zdrojovém textu jádra; tyto funkce se většinou zajišťují pomocí externích balíků. Nyní následuje výpis těch nejběžnějších.

6.1 kbd

Linuxová konzola má pravděpodobně více funkcí než kolik by potřebovala. Mimo jiné je zde možnost přepínat fonty, přemapovat vaši klávesnici, přepínat videomódy (v novějších jádrech) atd. Balík `kbd` obsahuje programy, které uživatelům tohle všechno zpřístupní, a navíc mnoho fontů a rozložení kláves pro téměř jakoukoliv klávesnici. Balík je k dispozici na stejných místech jako zdrojové texty jádra.

6.2 util-linux

Rik Faith (**faith@cs.unc.edu**) složil velkou sbírku linuxových utilit, která se nazývá `util-linux`. Nyní ji udržuje Nicolai Langfeldt (**util-linux@math.uio.no**). K dispozici je přes anonymní FTP ze **sunsite.unc.edu** v `/pub/Linux/system/misc`. Obsahuje programy, jako jsou `setterm`, `rdev` a `ctrlaltdel`, které souvisí s jádrem. Jak říká Rik, *neinstalujte bezmyšlenkovitě*. Nepotřebujete instalovat celý balík, mohli byste si způsobit vážné problémy.

6.3 hdparm

Stejně jako u mnoha jiných balíků se i zde původně jednalo o záplatu jádra a podpůrné programy. Záplaty se dostaly do oficiálního jádra a programy na optimalizaci a různé nastavování vašeho pevného disku jsou dodávány odděleně.

6.4 gpm

`Gpm` je zkratkou pro víceúčelovou myš (General Purpose Mouse). Tento program umožňuje přenos textu mezi virtuálními konzolami (pomocí cut&paste) a další věci, prováděné různými typy myší.

7 Některá úskalí

7.1 make clean

Jestliže vaše nové jádro provádí po aktualizaci opravdu podivné věci, je možné, že jste před kompilací nového jádra zapomněli na `make clean`. Příznaky mohou být různé – od pádu celého systému přes problémy se vstupními a výstupními zařízeními až po nevyrovnané výkony. Nezapomeňte také na `make dep`.

7.2 Velká nebo pomalá jádra

Jestliže vaše jádro spotřebovává příliš mnoho paměti, je velké nebo se kompiluje velmi dlouho i na vašem novém 786DX6/440, máte asi nakonfigurováno příliš mnoho nepotřebných věcí (ovladačů zařízení, systémů souborů atd.).

Kolik paměti jádro využívá zjistíte, když vezmete celkovou velikost paměti vašeho počítače a odečtete od ní množství „celkové paměti“ z `/proc/meminfo` nebo z výstupu příkazu „`free`“. K výsledku se dopracujete i pomocí „`dmesg`“ (nebo nahlédnutím do log-souboru jádra, ať už je na systému kdekoliv). Bude zde řádek, který bude vypadat asi takto:

```
Memory: 15124k/16384k available (552k kernel code,
  384k reserved, 324k data)
```

Moje 386-ka (která má nakonfigurováno trochu méně zbytečností) říká:

```
Memory: 7000k/8192k available (496k kernel code,
  384k reserved, 312k data)
```

Když „prostě musíte“ mít velké jádro, ale systém vám to neumožní, můžete zkusit „`make bzImage`“. V takovém případě možná budete muset nainstalovat novou verzi LILO.

7.3 Jádro se nezkompiluje

Jestliže se nezkompiluje, bude to proto, že záplata se nepovedla nebo je jádro poškozené. Můžete mít také nesprávnou nebo poškozenou (například s chybou v souborech include) verzi gcc. Zkontrolujte, jestli jsou správně nastaveny symbolické odkazy, které Linus popisuje v `README`. Jestliže se standardní jádro nekompiluje, obecně je to proto, že je něco v systému opravdu špatně. Nutná bude nová instalace určitých nástrojů.

Nebo také kompilujete jádro 1.2.x kompilátorem ELF (gcc 2.6.3 a vyšší). Jestliže se vám během kompilace objeví spousta hlášení typu `to-a-to undefined` (není nadefinováno), je problém možná na vaší straně. Náprava je ve většině případů velice jednoduchá. Tyto řádky přidejte na začátek `arch/i386/Makefile`:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Potom opět proveďte `make dep` a `zImage`.

Ve vzácných případech se může gcc zhroutit díky problémům hardwaru. Chybové hlášení bude pak vypadat asi takto: „`xxx exited with signal 15`“ (xxx ukončeno signálem 15). To většinou vypadá dost záhadně. Ani bych se o tom nezmínil, ale jednou se mi to stalo – měl jsem špatnou cache na základní desce a kompilátor by občas sahal do prázdna. Jestliže se objeví problémy, pokuste se nejprve o reinstalaci `gcc`. Obávat se můžete začít až když se vám jádro bude dobře kompilovat s vypnutou externí cache, se zmenšenou RAM atd.

Lidé se většinou při hlášení problémů hardwaru vylekají. Tyto problémy zde řešit nebudu, k tomu je zde FAQ – na adrese `http://www.bitwizard.nl/sig11/`.

7.4 Nová verze jádra se nespustí

Nespustili jste LILO nebo není správně nakonfigurováno. Jednou mě „dostala“ jedna věc – problém v konfiguračním souboru; bylo tam „`boot = /dev/hda1`“ místo „`boot = /dev/hda`“ (tohle může být zpočátku velmi mrzuté, ale jakmile vám konfigurační soubor funguje, nemusíte to měnit).

7.5 Zapomněli jste spustit LILO nebo se systém vůbec nezavede

Jéje! Tady je nejlepší zavést systém z diskety a připravit další zaváděcí disketu (například „`make zdisk`“). Musíte vědět, kde je váš kořenový (`/`) systém souborů a jakého je typu (second extended, minix atd.). V následujícím příkladu musíte také vědět, na jakém systému souborů je váš zdrojový strom `/usr/src/linux`, jakého je typu a kam se normálně připojuje (`mount`).

V našem příkladu je v oblasti `/dev/hda1` a systém souborů, na kterém je `/usr/src/linux`, je `/dev/hda3`, normálně připojovaný jako `/usr`. Oba dva systémy souborů jsou second extended. Funkční obraz jádra v `/usr/src/linux/arch/i386/boot` se nazývá `zImage`.

Vycházíme zde z toho, že když je `zImage` funkční, je možné jej použít pro další disketu. Další možnost, která může a nemusí být lepší (záleží na problému vašeho systému), bude následovat po příkladu.

Nejprve zaveďte systém ze zaváděcí nebo záchranné diskety a připojte systém souborů, který obsahuje obraz funkčního jádra:

```
mkdir /mnt
mount -t ext2 /dev/hda3 /mnt
```

Jestliže vám `mkdir` sdělí, že adresář již existuje, ignorujte jej. Nyní proveďte `cd` na místo, kde byl obraz funkčního jádra. Povšimněte si, že

```
/mnt + /usr/src/linux/arch/i386/boot - /usr =
  /mnt/src/linux/arch/i386/boot
```

Do disketové mechaniky (ne do zaváděcího nebo kořenového disku!) vložte naformátovanou disketu, okopírujte na ni obraz a nakonfigurujte jej pro váš kořenový systém souborů:

```
cd /mnt/src/linux/arch/i386/boot
dd if=zImage of=/dev/fd0
rdev /dev/fd0 /dev/hda1
```

Proveďte `cd /` a odpojte normální systém souborů `/usr`:

```
cd /
umount /mnt
```

Nyní byste měli být schopni z této diskety normálně zavést systém. Nezapomeňte po zavedení systému spustit `lilo` (nebo to, díky čemu jste se dostali do problémů)!

Jak už jsme naznačili, je zde ještě jedna alternativa. Jestliže máte funkční obraz jádra v / (například `/vmlinuz`), můžete jej využít pro zaváděcí disk. Když si vezmeme předchozí příklad a to, že můj obraz jádra je `/vmlinuz`, proveďte oproti příkladu tyto změny: změňte `/dev/hda3` na `/dev/hda1` (systém souborů /), `/mnt/src/linux` na `/mnt` a `if=zImage` na `if=vmlinuz`.

Poznámka: Vysvětlení jak odvodit `/mnt/src/linux` může být ignorováno.

Použití LILO na velkých mechanikách (discích s více než 1 024 cylindry) může způsobovat problémy. V takovém případě doporučujeme pročíst dokumentaci LILO.

7.6 Objeví se 'warning: bdflush not running'

Toto může být velký problém. Počínaje verzí jádra od 1.0 (kolem 20. dubna 1994) byl aktualizován/nahrazen program „`update`“, který pravidelně vyprazdňuje buffery systému souborů. Sežeňte si zdrojové texty k „`bdflush`“ (měli byste je nalézt tam, kde jste si opatřili zdrojový text jádra) a instalujte jej (přitom asi budete chtít systém spouštět pod starým jádrem). Sám se nainstaluje jako „`update`“ a po novém zavedení systému by si už nové jádro nemělo stěžovat.

7.7 Objeví se něco o nenadefinovaných symbolech a kompilace neproběhne

Pravděpodobně máte kompilátor ELF (`gcc` 2.6.3 a vyšší) a zdrojový text jádra 1.2.x (nebo starší). V takové situaci se většinou postupuje přidáním těchto tří řádků na začátek `arch/i386/Makefile`:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Tak se bude jádro verze 1.2.x kompilovat s knihovnami `a.out`.

7.8 Nemohu zprovoznit svoji mechaniku IDE/ATAPI CD-ROM

Je zvláštní, že mnoha lidem nefungují jejich mechaniky ATAPI, což je pravděpodobně tím, že je zde množství faktorů, které to mohou zapříčinit.

Jestliže máte mechaniku CD-ROM jako jediné zařízení na některém rozhraní IDE, musíte ji nastavit jako „master“ nebo „single“. To bývá nejčastější chyba.

Firma Creative Labs někdy na své zvukové karty umisťuje rozhraní IDE. Tak vzniká zajímavý problém – někteří lidé mají jen jedno rozhraní, ale někteří mají na základní desce zabudována dvě rozhraní IDE (obvykle na IRQ 15), takže je nutné na zvukové kartě nastavit třetí port IDE (IRQ 11, jak mi bylo řečeno).

Tím vzniká problém u verze Linuxu 1.2.x, která nepodporuje třetí rozhraní IDE (podpora začíná až někdy od 1.3.x, ale takový už je vývoj, s tím se musíte smířit). Při řešení máte několik možností.

Jestliže máte druhý port IDE, je možné, že jej ještě nevyužíváte nebo na něm ještě nemáte dvě zařízení. Odpojte mechaniku ATAPI ze zvukové karty a dejte ji na druhé rozhraní. Pak můžete odstavit rozhraní ze zvukové karty, čímž mimochodem ušetříte jedno IRQ.

Jestliže nemáte druhé rozhraní, nastavte rozhraní zvukové karty (samozřejmě, že ne to zvukové) jako IRQ 15 – tedy druhé rozhraní. To by mělo fungovat.

Jestliže je připojení mechaniky na „třetím“ rozhraní prostě nutné nebo máte ještě jiné problémy, obstarejte si jádro novější a pročtěte si `drivers/block/README/ide`. Je zde mnohem více informací.

7.9 Objevují se podivná hlášení o zastaralých směrovacích požadavcích

Obstarejte si novou verzi programu `route` a dalších programů, které se používají při směrování. Změnil se `/usr/include/linux/route.h` (což je ve skutečnosti soubor v `/usr/src/linux`).

7.10 Ve verzi 1.2.0 nefunguje firewall

Přejděte na verzi alespoň 1.2.1.

7.11 „Not a compressed kernel image file“

Nepoužívejte jako obraz zaváděcího disku soubor `vmlinux`, vytvořený v `/usr/src/linux`; tím správným je `[..]/arch/i386/boot/zImage`.

7.12 Problémy s konzolovým terminálem po přechodu na 1.3.x

V `/etc/termcap` v záznamu `console` změňte slovo `dumb` na `linux`. Můžete vytvořit i nový záznam v `terminfo`.

7.13 Po aktualizaci jádra se nic nekompiluje

Zdrojový text jádra linuxu obsahuje množství souborů include (končí na .h), které jsou standardně požadovány v `/usr/include`. Většinou se odkazují takto (`xyzzy.h` je něco z `/usr/include/linux`):

```
#include <linux/xyzzy.h>
```

Normálně je v `/usr/include` odkaz do adresáře `include/linux` vašeho zdrojového textu jádra (většinou `/usr/src/linux/include/linux`), nazývaný `linux`. Jestliže tento odkaz chybí nebo ukazuje na špatné místo, většina věcí se nebude kompilovat. Jestliže jste si řekli, že vám zdrojový text jádra zabíral na disku příliš mnoho místa a smazali jste jej, je to asi tím. Dalším problémem mohou být přístupová práva k souborům; jestliže má váš **root** umask, neumožňující ostatním uživatelům implicitně vidět jeho soubory a vy jste si vzali zdrojový text jádra bez volby **p** (zachovat mód souboru), tak ostatní uživatelé také nebudou schopni využívat C-kompilátor. I když je možné k nápravě použít příkaz `chmod`, jednodušší asi bude znovu získat soubory include. To můžete provést stejným způsobem, jako u celého zdrojového textu na začátku, pouze s přidaným argumentem:

```
blah# tar zxvpf linux.x.y.z.tar.gz linux/include
```

Poznámka: Jestliže zde není, „`make config`“ znovu odkaz `/usr/include/linux` vytvoří.

7.14 Zvýšení limitů

Následujících pár *vzorových* příkazů může pomoci těm, kteří chtějí zvýšit některé dolní limity, uvalené jádrem:

```
echo 4096 > /proc/sys/kernel/file-max
echo 12288 > /proc/sys/kernel/inode-max
echo 300 400 500 > /proc/sys/vm/freepages
```

8 Poznámka pro přechod k verzi 2.0.x

Jádro verze 2.0.x představilo pár změn ve své instalaci. Soubor `Documentation/Changes` ve zdrojovém stromu 2.0.x obsahuje informace, které byste měli znát při přechodu na verzi 2.0.x. Bude nutné aktualizovat několik základních balíků, jako jsou gcc, libc a SysVInit, a snad také upravit některé systémové soubory, takže s tím počítejte a nezpanikařte.

9 Moduly

Moduly jádra, určené k nahrání, mohou ušetřit paměť a usnadnit konfiguraci. Moduly zahrnují již i systémy souborů, ovladače ethernetových karet, páskové ovladače, ovladače tiskáren a další.*

9.1 Instalace modulových utilit

Modulární utility jsou k dispozici ze stejného místa, jako jádro, a sice pod názvem `modules-x.y.z.tar.gz`; vyberte si nejvyšší číslo `x.y.z`, které je menší nebo rovno verzi vašeho aktuálního jádra. Rozbalte jej pomocí „`tar zxvf modules-x.y.z.tar.gz`“, proveďte `cd` do adresáře, který vytvoří (`modules-x.y.z`), nahlédněte do `README` a proveďte instalační instrukce zde popsané (samé jednoduché, jako je `make install`). Nyní byste měli mít v `/sbin` programy `insmod`, `rmmod`, `ksyms`, `lsmod`, `genksyms`, `modprobe` a `depmod`. Jestli chcete, otestujte utility pomocí příkladu ovladače „hw“ v `insmod`; podrobnosti ve stejném adresáři v souboru `INSTALL`.

Moduly vkládá do běžícího jádra `insmod`. Moduly mají obvykle koncovku `.o`; příklad ovladače (viz výše) se nazývá `drv_hello.o`, takže aby se použil, musí se napsat „`insmod drv_hello.o`“. Výpis modulů, právě používaných jádrem, získáte pomocí `lsmod`. Výstup vypadá takto:

```
blah# lsmod
Module: #pages: Used by:
drv_hello 1
```

„`drv_hello`“ je název modulu, využívá jednu stránku (4KB) paměti a v této chvíli na něm nezávisí žádné další moduly. Tento modul odstraníte pomocí „`rmmod drv_hello`“. Po-

* Poznámka korektora: Nyní se tyto pomocné programy nazývají `modutils`.

všimněte si, že `rmmod` požaduje *module name* (název modulu), ne název souboru; získáte jej ze seznamu **lsmod**. Účely dalších modulových utilit jsou dokumentovány na jejich manuálových stránkách.

9.2 Moduly, dodávané s jádrem

Od verze 2.0.30 je skoro všechno k dispozici ve formě modulů. Při využívání modulů si nejprve uvědomte, jestli jste je nenakonfigurovali přímo do jádra; to znamená neříkat během „`make config`“ na patřičných místech „`y`“. Zkompilujte nové jádro a zaveďte s ním systém. Proveďte `cd /usr/src/linux` a „`make modules`“. Tím se kompilují všechny moduly, které jste neurčili v konfiguraci jádra, a v `/usr/src/linux/modules` se k nim přidají odkazy. Můžete je využívat přímo v tomto adresáři nebo můžete spustit „`make modules_install`“, čímž se instalují do `/lib/modules/x.y.z`, kde `x.y.z` je verzí jádra.

Toto je velice šikovné, obzvláště u systémů souborů. Systémy souborů minix nebo msdos třeba příliš často nepoužíváte. Když mám například dosovou disketu, použiji `insmod /usr/src/linux/modules/msdos.o` a po skončení `rmmod msdos`. Během normálního běhu jádra se tak ušetří kolem 50 KB RAM. Ještě malá poznámka, týkající se systému souborů minix: pro použití na „záchranných“ discích byste jej měli *vždy* konfigurovat přímo do jádra.

10 Další konfigurační volby

Tato část obsahuje popisy vybraných konfiguračních voleb (v `make config`), které nejsou vypsány v části o konfiguraci. Není zde vypsána většina ovladačů zařízení.

10.1 Celkové nastavení

Normal floppy disk support – podpora normálních disket. Možná je dobré si pročíst soubor `drivers/block/README.fd` **–** zejména pro uživatele IBM Thinkpad.

XT harddisk support – pokud byste chtěli používat ještě 8bitový XT řadič disků, který leží zaprášený někde v koutě.

PCI bios support – pokud máte PCI, můžete to zkusit; ale pozor, některé starší základní desky PCI mohou s touto volbou zlobit. Více informací o sběrnicích PCI pod Linuxem viz PCI-HOWTO.

Kernel support for ELF binaries – ELF je snahou o umožnění použití binárních souborů mezi architekturami a operačními systémy; Linux, jak se zdá, tento trend podporuje, takže volba je pro vás žádoucí.

Set version information on all symbols for modules – dříve byly moduly jádra překompilovány s každým novým jádrem. Jestliže odpovíte „`y`“, bude možné používat moduly, kompilované pod jinou verzí. Více podrobností viz `README.modules`.

10.2 Volby pro sítě

Volby pro sítě jsou popsány v příslušném dokumentu (NET-3-HOWTO).

11 Tipy a triky

11.1 Přesměrování výstupu příkazů make nebo patch

Jestliže potřebujete záznam toho, co provádí příkazy „`make`“ a „`patch`“, můžete výstup přesměrovat do souboru. Nejprve zjistěte, který příkazový interpret využíváte: „`grep root /etc/passwd`“ a hledejte něco jako „`/bin/csh`“.

Pokud používáte sh nebo bash, kopie výstupu „**příkazu**“ se umístí do „**výstupního souboru**“ takto:

```
(příkaz) 2>&1 | tee (výstupní soubor)
```

U `csh` nebo `tcsh` takto:

```
(příkaz) |& tee (výstupní soubor)
```

Pro `rc` (poznámka: `rc` pravděpodobně nepoužíváte) takto:

```
(příkaz) >[2=1] | tee (výstupní soubor)
```

11.2 Podmíněná instalace jádra

Kromě využití disket je zde ještě několik metod testování nového jádra bez ovlivnění starého. Na rozdíl od jiných unixových záležitostí je LILO schopné zavést systém z libovolného místa disku (jestliže máte velký disk – asi 500 MB a více – pročtěte si raději dokumentaci k LILO, protože zde mohou vyvstat problémy). Takže pokud na konec konfiguračního souboru LILO přidáte něco takového:

```
image = /usr/src/linux/arch/i386/boot/zImage
       label = new_kernel
```

můžete si (samozřejmě po spuštění `lilo`) vybrat spuštění nově kompilovaného jádra bez narušení starého `/vmlinuz`. V LILO je nejsnazší způsob zavedení nového jádra stisk klávesy Shift při zavádění systému (když je na obrazovce **LILO** a nic jiného), získáte tak prompt a zadáte „**new_kernel**“.

Jestliže si chcete v systému ponechat několik různých zdrojových stromů jader (takhle obsadíte *hodně* místa na disku, pozor), obvykle se pojmenovávají `/usr/src/linux-x.y.z`, kde `x.y.z` je verze jádra. Zdrojový strom si pak můžete „vybrat“ pomocí symbolického odkazu; například „`ln -sf linux-1.2.1 /usr/src/linux`“ nastaví jako aktuální strom 1.2.2. Před vytvořením takového symbolickho odkazu se ujistěte, že poslední argument u **ln** není skutečný adresář (staré symbolické odkazy jsou dobré); výsledek by nebyl takový, jaký byste očekávali.

11.3 Aktualizace jádra

Russel Nelson (**nelson@crynwr.com**) shrnul změny v nových verzích jádra. Jsou popsány stručně a před aktualizací do nich můžete nahlédnout. K dispozici jsou na anonymním FTP serveru na **ftp.emlist.com** v adresáři `pub/kchanges` nebo na URL

`http://www.crynwr.com/kchanges`*

12 Další dokumenty této série, které se mohou hodit

- Sound-HOWTO: zvukové karty a utility
- SCSI-HOWTO: vše o řadičích a ovladačích SCSI
- NET-3-HOWTO: sítě
- PPP-HOWTO: PPP sítě obecně
- PCMCIA-HOWTO: o ovladačích pro váš notebook
- ELF-HOWTO: co je ELF, převody..
- Hardware-HOWTO: přehled podporovaného hardwaru
- Module-HOWTO: něco k modulům jádra
- Kerneld-HOWTO: o `kerneld`
- BogoMips-HOWTO: když nevíte o co se jedná

* Poznámka korektora: Seznamy změn v novějších jádrech naleznete na adrese `http://www.linuxhq.com`.

13 Závěr

13.1 Autor

Autorem a správcem tohoto dokumentu je Brian Ward (**bri@blah.math.tu-graz.ac.at**). Posílejte mi prosím jakékoliv komentáře, dodatky a opravy (nejdůležitější jsou pro mě opravy).

Na jednom z těchto URL se můžete podívat na moji „domovskou stránku“:

`http://www.math.psu.edu/ward/`

`http://blah.math.tu-graz.ac.at/~bri/`

I když se snažím pozorně pročítat veškerou poštu, uvědomte si prosím, že jí každý den dostávám *opravdu* hodně, takže na vás může přijít řada až za dlouho. Tohle platí zejména v případě, kdy se na mne s něčím obracíte. Snažte se proto být struční a srozumitelní. Jestliže píšete o nefunkčním hardwaru (nebo něčem podobném), musím vědět, jaká je vaše hardwarová konfigurace. Když píšete o chybě, nepište jen: „Zkusil jsem to, ale objevila se chyba.“ Musím vědět, o jakou chybu se jedná. Měl bych také vědět, jakou verzi jádra, `gcc` a libc používáte. Když mi napíšete jen, že používáte tu a tu distribuci, moc mi to neřekne. Nevadí mi, že mi pokládáte jednoduché dotazy; uvědomte si, že když se nikdy nezeptáte, nikdo vám také neodpoví! Rád bych poděkoval všem za jejich ohlasy.

Jestliže jste mi napsali a odpověď v rozumné době (tři týdny a více) nepřišla, možná jsem omylem vaši zprávu smazal nebo něco podobného (znáte to). Pak zkuste dotaz poslat ještě jednou.

Dostávám spoustu dopisů, které se týkají problémů s hardwarem. V pořádku, ale uvědomte si, že neznám všechen hardware světa a nevím, jak moc vám mohu být nápomocen; já osobně používám stroje s disky IDE a SCSI, CD-ROM mechaniky SCSI, ethernetové karty 3Com a WD, sériové myši, základní desky PCI, řadiče NCR 810 SCSI, procesory AMD 386DX40 w/Cyrix, AMD 5x86, AMD 486DX4 a Intel 486DX4 (takže tohle všechno používám a znám já, ale neberte to jako doporučení pro vás). Pokud chcete, klidně se ptejte.)

Verze 0.1 byla napsána 3. října 1994. Tento dokument je k dispozici ve formátech SGML, PostScript, TeX, roff a v čistém textu.

13.2 Co ještě chybí

Část „Tipy a triky“ je příliš krátká. Chtěl bych ji rozšířit o další nápady dalších lidí.

To stejné platí o „dalších balících“.

Potřebuji také více informací o odladňování a odstraňování havárií.

13.3 Příspěvky

Použil jsem malou část Linusova `README` (volby pro úpravy jádra). Díky!

uc@brian.lunetix.de (Ulrich Callmeier): `patch` -s a `xargs`.

quinlan@yggdrasil.com (Daniel Quinlan): úpravy a doplnění v mnoha částech.

nat@nataa.fr.eu.org (Nat Makarevitch): `mrproper`, `tar -p`, mnoho dalších věcí.

boldt@math.ucsb.edu (Axel Boldt): sestavil na síti popisy voleb konfigurace jádra a tento seznam mi poskytl.

lembark@wrkhors.psyber.com (Steve Lembark): návrh na využití více možností zavádění systému.

kbriggs@earwax.pd.uwa.edu.au (Keith Briggs): pár oprav a návrhů.

rmcguire@freenet.columbus.oh.us (Ryan McGuire): dodatky možností u `make`.

dumas@excalibur.ibp.fr (Eric Dumas): překlad do francouzštiny.

simazaki@ab11.yamanashi.ac.jp (Yasutada Shimazaki): překlad do japonštiny.

jjamor@lml.ls.fi.upm.es (Juan Jose Amor Iglesias): překlad do španělštiny.

mva@sbbs.se (Martin Wahlen): překlad do švédštiny.

jzp1218@stud.u-szeged.hu (Zoltan Vamosi): překlad do maďarštiny.

bart@mat.uni.torun.pl (Bartosz Maruszewski): překlad do polštiny.

donahue@tiber.nist.gov (Michael J Donahue): překlepy.

rms@gnu.ai.mit.edu (Richard Stallman): poznámka k „volné“ šiřitelnosti dokumentu.

dak@Pool.Informatik.RWTH-Aachen.DE (David Kastrup): NFS-záležitosti.

esr@snark.thyrsus.com (Eric Raymond): různé úpravy.

13.4 Autorská práva, licence a všechny tyhle záležitosti

Kapitola

6

Linux IPX

Terry Dawson, **terry@perf.no.itg.telstra.com.au** v2.2, 29. března 1997

Cílem tohoto dokumentu je popsat způsob získání, instalace a konfigurace různých nástrojů, které jsou dostupné pro operační systém Linux a využívají podporu jádra Linuxu protokolu IPX.

1 Úvod

Toto je dokument Linux IPX-HOWTO. V kombinaci s tímto dokumentem byste si měli přečíst i dokument NET-3-HOWTO.

1.1 Změny oproti předchozím verzím

Dodatky:

Doplněny některé informace o typu rámce

- Opravy/aktualizace:
- Pro síťové `IPX-adresy` v souboru `/etc/ppp/options` je vyžadováno `0x`.
- Aktualizace verzí a umístění.
- Některé změny v abstraktním tisku a administrativních nástrojích.

1.2 Úvod

Jádro Linuxu má ve srovnání s jinými operačními systémy na bázi Unixu zcela novou implementaci síťové podpory. Schopnost nového přístupu k vývoji síťového softwaru jádra přispěla k tomu, že jádro Linuxu nabízí zabudovanou podporu i pro řadu jiných protokolů něž TCP/IP. Protokol IPX je jedním z nich.

Jádro Linuxu podporuje pouze protokol IPX. Zatím neobsahuje podporu protokolů typu IPX/RIP, SAP nebo NCP. Tyto protokoly podporuje jiný software, o němž bude zmínka dále.

Podpora protokolu IPX je dílem Alana Coxe **<alan@lxorguk.ukuu.org.uk>** a značně ji vylepšil Greg Page **<greg@caldera.com>**.

2 Záruka

Nevím a ani nemohu vědět vše o síťovém softwaru operačního systému Linux. Proto vás předem upozorňuji, že tento dokument může obsahovat chyby. Přečtěte si prosím soubory `README`, které jsou dodávány společně s programy popisovanými v tomto dokumentu, kde najdete podrobnější a přesnější informace. Budu se snažit, aby tento dokument obsahoval co nejméně chyb a byl co nejaktuálnější. Verze programů, o kterých bude řeč, byly aktuální v době psaní tohoto dokumentu.

Já ani autoři softwaru zmiňovaného v tomto dokumentu neposkytujeme žádnou ochranu proti vašim akcím. I v případě, že nakonfigurujete tento software podle popisu v tomto dokumentu a objeví se nějaké síťové problémy, nesete odpovědnost jenom vy. Toto varování zde uvádím proto, že návrh a konfigurace sítě IPX není vždy jednoduchá záležitost a špatný návrh nebo konfigurace vaší sítě mohou někdy vést k vzájemné interakci s ostatními routery a souborovými servery. Toto varování zde uvádím také proto, že mě jeden nešťastný člověk požádal, abych toto čtení uvedl trošku drsnějším způsobem.

3 Příbuzná dokumentace

V tomto dokumentu se předpokládá, že umíte sestavit jádro Linuxu s příslušně zvolenými síťovými volbami a že umíte používat základní síťové nástroje, jako jsou `ifconfig` a `route`. Pokud tomu tak není, měli byste si společně s tímto dokumentem přečíst i dokument NET-3-HOWTO, který se touto problematikou zabývá.

K dalším dokumentům HOWTO, které by pro vás mohly být užitečné, patří:

- Dokument *Ethernet-HOWTO*, který popisuje podrobnosti konfigurace ethernetového zařízení pro systém Linux.
- Dokument *PPP-HOWTO*, protože od verze 2.2.0d existuje pro implementaci protokolu PPP podpora IPX.

3.1 Nové verze tohoto dokumentu

Je-li vaše kopie tohoto dokumentu starší než dva měsíce, pak vám doporučuji stáhnout si novější verzi. Síťová podpora operačního systému Linux se velmi rychle mění, jsou přidávány nová vylepšení a funkce, takže se poměrně často mění i tento dokument. Poslední verzi tohoto dokument vždy získáte prostřednictvím anonymní služby FTP na adrese:

sunsite.unc.edu

v adresáři `/pub/Linux/docs/HOWTO/IPX-HOWTO`

nebo

`/pub/Linux/docs/HOWTO/other-formats/IPX-HOWTO{-html.tar,ps,dvi}.gz`

Prostřednictvím webu je tento dokument k dispozici na serveru `http://sunsite.unc.edu/LDP/linux.html`, na internetové stránce IPX-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/IPX-HOWTO.html` nebo na mé adrese **<terry@perf.no.itg.telstra.com.au>**. Občas může být také zaslán do diskusních skupin *comp.os.linux.networking, comp.os.linux.answers a news.answers*.

3.2 Zpětná vazba

Jakékoliv komentáře, aktualizace nebo návrhy mi prosím pošlete na adresu **<terry@perf.no.itg.telstra.com.au>**. Čím dříve dostanu vaše připomínky, tím dříve budu moci aktualizovat a opravit tento dokument. Pokud v něm najdete nějaké chyby, pak mi prosím neprodleně napište, protože v současné době čtu diskusní skupiny jen zřídka.

3.3 Podpora konferencí

Diskusi o různých softwarových IPX-balících popisovaných v tomto dokumentu je také věnována jedna konference. Přihlásit se do ní můžete tak, že pošlete zprávu na adresu **listserv@sh.cvut.cz** a do těla zprávy vložíte řetězec „`add linware`“. Příspěvky do konference pak zasílejte na adresu **linware@sh.cvut.cz.**

Tato konference je archivována na WWW-stránce `http://www.kin.vslib.cz/hypermail/linware/`.

4 Některé termíny používané v tomto dokumentu

V tomto dokumentu se často setkáte s termíny **klient** a **server**. Normálně se jedná o poměrně specifické termíny, ale v tomto dokumentu jsem jejich definice trošku zevšeobecnil, takže zde mají následující význam:

klient Počítač nebo program, který zahajuje nějakou akci nebo spojení za účelem získání nějaké služby nebo dat.

server Počítač nebo program, který přijímá příchozí spojení od více počítačů a poskytuje jim služby nebo data.

Obě tyto definice sice nejsou příliš spolehlivé, ale jsou prostředkem, jak rozlišit dva konce systémů peer-to-peer, jako je *SLIP* nebo *PPP*, které ve skutečnosti nemají klienty ani servery.

Mezi další termíny, s nimiž se zde setkáte, patří:

Bindery *Bindery* je specializovaná databáze uchovávající síťové konfigurační informace na novellovském souborovém serveru. Klienti NetWare mohou této databázi posílat dotazy a získávat informace o dostupných službách, směrování a informace o uživatelích.

Typ rámce (Frame Type) Je termín používaný k popisu skutečného protokolu použitého pro přenos datagramů IPX (a IP) přes ethernetové síťové segmenty. Existují čtyři běžné typy. Jsou to:

Ethernet_II Jedná se o vylepšenou verzi původního ethernetového standardu DIX. Novellu bylo přiděleno id původního protokolu a to znamená, že IPX a IP mohou v prostředí Ethernet_II poměrně dobře koexistovat. Tento typ rámce se v novellovských prostředích běžně používá a je dobrou volbou.

802.3 Jedná se o protokol podle standardu I.E.E.E, který definuje mechanismus CSMA/CD (Carrier Sense Multiple Access with Collision Detecion). Je založen na původním standardu DIX Ethernet, ale obsahuje důležité úpravy – typové pole (id protokolu) bylo převedeno na délkové pole. To proto, že IPX by zde neměl být pouštěn. Standard IEEE 802.3 byl navržen **pouze** pro přenos rámců 802.3, existují však implementace, které ho využívají k přímému přenosu IPX-rámců. Pokud budete spolupracovat se sítí, která ho již používá, pak se tomuto standardu vyhněte.

802.2 Tento I.E.E.E protokol definuje sadu Logical Link Control procedur. Poskytuje zjednodušený způsob povolování soužití různých protokolů, ale v tomto ohledu je poměrně omezený. Firma Novell používá neoficiální Service Address Point (jako id protokolu), ale protože je používá skoro každý, nebyl to zatím velký problém.

SNAP SNAP znamená Sub Network Access Protocol. Tento protokol je navržen pro funkci nad protokoly 802.3 a 802.2. Rozšiřuje mezi-protokolovou kompatibilitu 802.2 a poskytuje určité měřítko kompatibility s existujícími typy rámců Ethernet a Ethernet_II.

IPX Internet Packet eXchange je protokol, s jehož pomocí poskytuje společnost Novell mezisíťovou podporu svého produktu NetWare(tm). Protokol IPX je funkcí podobný protokolu IP, který používají sítě TCP/IP.

Síťová IPX adresa
Toto číslo slouží k jedinečné identifikaci konkrétní IPX-sítě. Tato adresa je obvykle zapisována hexadecimálně. Příklad by vypadal asi takto: `0x23a91002`.

Interní IPX síť
Jedná se o virtuální IPX-síť. Virtuální je proto, že nekoresponduje s fyzickou sítí. Slouží k jedinečné identifikaci a adresaci konkrétního IPX-hostitele. Tento způsob je obecně použitelný pouze pro IPX-hostitele, kteří figurují ve více fyzických IPX-sítích, kam patří například souborové servery. Adresa je zakódována stejným způsobem, jako pro fyzickou IPX-síť.

RIP Routing Information Protocol je protokol používaný k automatickému šíření síťových tras (route) v síti IPX. Jeho funkce je podobná protokolu RIP používanému v sítích TCP/IP.

NCP NetWare Core Protocol je protokol síťového souborového systému, který vyvinula společnost Novell Corporation pro svůj produkt NetWare (tm). Funkce protokolu NCP je podobná systému NFS používanému v sítích TCP/IP.

SAP Service Advertisement Protocol je protokol, který navrhla společnost Novell Corporation. Slouží k propagování síťových služeb v prostředí NetWare(tm).

Hardwarová adresa
Je číslo, které slouží k jedinečné identifikaci hostitele ve fyzické síti na úrovni přístupu k médiu. Příkladem hardwarové adresy je *ethernetová adresa*. Ethernetová adresa je obecně tvořena šesti hexadecimálními hodnotami oddělenými znaky dvojtečka, tj: `00:60:8C:C3:3C:0F`.

trasa (route) *Trasa* je cesta, po které vaše pakety putují sítí, než dorazí ke svému cíli.

5 Soubory IPX v souborovém systému /proc

Protokol IPX má v Linuxu několik podpůrných souborů, které jsou umístěny v adresáři `/proc`. Jsou to:

`/proc/net/ipx_interface`
: Tento soubor obsahuje informace o rozhraních IPX konfigurovaných ve vašem počítači. Lze je konfigurovat buď manuálně pomocí nějakého příkazu, nebo je může systém detekovat a nakonfigurovat automaticky.

`/proc/net/ipx_route`
: Tento soubor obsahuje seznam tras existujících ve směrovací tabulce protokolu IPX. Tyto trasy lze přidávat manuálně pomocí příkazu nebo automaticky pomocí směrovacího IPX démona.

`/proc/net/ipx`
: Tento soubor obsahuje seznam IPX soketů, které jsou aktuálně otevřeny.

6 Nástroje IPX Grega Page

Greg Page **<greg@caldera.com>** ze společnosti Caldera Incorporated napsal sadu konfiguračních nástrojů protokolu IPX a zdokonalil podporu jádra Linuxu pro tento protokol.

Díky těmto vylepšením může být Linux nakonfigurován jako plnohodnotný IPX-most nebo router. Zdokonalená podpora protokolu IPX byla zabudována také do jádra většiny hlavních distribucí, takže ji zřejmě již máte.

Nástroje pro konfiguraci sítě umožňují nastavit síťová zařízení tak, aby podporovala protokol IPX a umožňovala konfigurovat IPX-směrování a další prostředky pod systémem Linux. Síťové nástroje podporující protokol IPX jsou dostupné na adrese `ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/ipx.tgz`.

6.1 Nástroje IPX podrobněji

`ipx_interface`
: Tento příkaz slouží k manuálnímu přidávání, rušení a kontrole podpory protokolu IPX u existujících síťových zařízení. Normálně by bylo síťové zařízení ethernetovým zařízením, například `eth0`. Alespoň jedno rozhraní protokolu IPX musí být označeno jako *primární* a o to se postará pří-

volba -p. Chcete-li například ethernetovému zařízení eth0 umožnit podporu IPX-rozhraní pomocí typu rámce IEEE 802.2 a síťové IPX-adresy 0x39ab0222, pak použijete následující zápis:

```
# ipx_interface add -p eth0 802.2 0x39ab0222
```

Pokud se při spuštění tohoto programu objeví chyba a náhodou ještě nemáte nakonfigurován protokol TCP/IP, budete muset manuálně spustit rozhraní eth0 pomocí příkazu:

```
# ifconfig eth0 up
```

ipx_configure Tento příkaz povoluje nebo zakazuje automatické nastavení konfigurace rozhraní a primárního rozhraní.

```
--auto_interface
```

umožňuje nastavit, zda mají být nová síťová zařízení automaticky nakonfigurována jako IPX-zařízení.

```
--auto_primary
```

umožňuje nastavit, zda má software protokolu IPX automaticky zvolit primární zařízení.

Typickým příkladem je povolení jak automatické konfigurace rozhraní, tak i automatického nastavení primárního rozhraní pomocí následujícího příkazu:

```
# ipx_configure --auto_interface=on --auto_primary=on
```

ipx_internal_net

Tento příkaz umožňuje nastavit nebo zrušit interní síťovou adresu. Interní síťová adresa je volitelná, ale je-li nastavena, bude vždy používána jako primární adresa. síťovou adresu IPX ab000000 na IPX-uzlu 1 nastavíte následujícím způsobem:

```
# ipx_internal_net add 0xab000000 1
```

ipx_route Tento příkaz umožňuje manuální úpravy ve směrovací tabulce protokolu IPX. Chcete-li například přidat trasu do IPX-sítě 0x39ab0222 prostřednictvím routeru s uzlem číslo 0x00608CC33C0F v IPX-síti 0x39ab0108, použijte následující příkaz:

```
# ipx_route add 0x39ab0222 0x39ab0108 0x00608CC33C0F
```

7 Konfigurace linuxového počítače jako IPX-routeru

Pokud máte hodně IPX-segmentů, které chcete vzájemně propojit do sítě, potřebujete služby routeru. V prostředí sítě Novell existují dvě informace, které je třeba předat. Jedná se o síťovou směrovací informaci šířenou prostřednictvím protokolu Novell RIP a informaci o oznamování služeb šířenou prostřednictvím protokolu Novell SAP. Každý router, aby byl použitelný ve většině situací, musí podporovat oba tyto protokoly.

Operační systém Linux má zabudovanou podporu pro oba tyto protokoly a poměrně snadno se může fungovat jako plnohodnotný novellovský router.

Podpora jádra Linuxu protokolu IPX ve skutečnosti řídí doručování IPX-paketů po rozhraních, dělá to však podle pravidel zakódovaných do směrovací tabulky protokolu IPX. Linux potřebuje k implementaci protokolů Novell RIP a SAP nějaký program, který zajistí správné vytvoření směrovací tabulky protokolu IPX a její pravidelnou aktualizaci, aby odrážela stav sítě.

Volker Lendecke **<lendecke@namu01.gwdg.de>** vytvořil směrovacího démona, který to udělá za vás.

Najdete ho na adrese:

`ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/ipxripd-0.7.tgz`

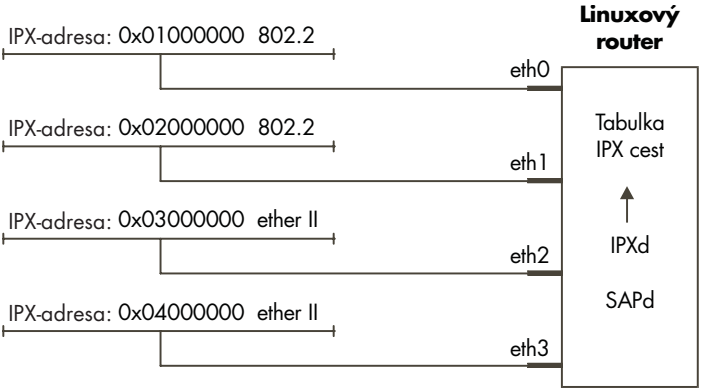
nebo na Volkerově domovském serveru:

`ftp://ftp.gwdg.de/pub/linux/misc/ncpfs/ipxripd-0.7.tgz`

Konfigurace linuxového počítače jakožto routeru je poměrně přímočará. Řiďte se následujícími kroky:

1. Sestavte jádro s podporou protokolů IPX, Ethernet a souborového systému `/proc`.
2. Sežeňte si, zkompilujte a nainstalujte démona `ipxd`.
3. Nastartujte nové jádro a ujistěte se, že byly všechny ethernetové karty správně detekovány a nedochází k žádným hardwarovým konfliktům.
4. Na všech rozhraních povolte za pomoci výše popsaného příkazu `ipx_interface` protokol IPX.
5. Spusťte démona `ipxd`.

Uvažujte následující jednoduchou síť:



Konfigurace výše uvedené sítě by vypadala následovně:

```
# ipx_interface add eth0 802.2 0x0100000000
# ipx_interface add eth1 802.2 0x0200000000
# ipx_interface add eth2 etherii 0x0300000000
# ipx_interface add eth3 etherii 0x0400000000
# ipxd
```

Potom byste měli chvíli počkat a podívat se, zda byl soubor `/proc/net/ipx_route` rozšířen o IPX-routery důležité pro vaši konfiguraci a další informace získané od jiných routerů v síti.

7.1 Potřebuji konfigurovat interní síť?

V síti Novell se setkáte s pojmem interní síť, která zjednodušuje směrování v situacích, kdy je k hostiteli připojeno více než jedno síťové zařízení. To je užitečné v případě souborového serveru připojeného k více sítím, protože tak stačí oznámit jen jedinou trasu, pomocí které je možné serveru dosáhnout, bez ohledu na síť, z níž se o to pokoušíte.

V případě konfigurace, při které neprovozujete souborový server a váš počítač slouží pouze jako IPX-router, je odpověď na nastolenou otázku trošku složitější. Je známo, že konfigurace protokolu IPX/PPP funguje „lépe“, když současně zkonfigurujete i interní síť.

V každém případě je tato činnost jednoduchá, nicméně vyžaduje nové sestavení jádra. Když budete odpovídat na dotazy příkazu make config, musíte na otázku `Full internal IPX network` odpovědět `y`, jak je ukázáno v následujícím výpisu:

```
...
...
Full internal IPX network (CONFIG_IPX_INTERN) [N/y/?] y
...
...
```

Ke konfiguraci interního síťového rozhraní použijte příkaz `ipx_internal_net`, který jsme si popsali dříve ve stati IPX-nástroje. Nejdůležitější je zajistit, aby síťová IPX-adresa, kterou mu přidělíte, byla ve vaší síti jedinečná a nepoužíval ji žádný jiný počítač nebo síť.

8 Konfigurace linuxového počítače jako NCP-klienta

Jste-li uživatelem sítě založené na různých technologiích, jež zahrnují protokol IP i IPX, je pravděpodobné, že někdy bude potřeba, aby váš linuxový počítač přistupoval k datům uloženým na novellovském souborovém serveru ve vaší síti. Firma Novell již dlouho nabízí pro své souborové servery balík NFS-serveru, který to umožňuje, ale jste-li jen malý systém nebo jen malý počet vašich uživatelů má o tuto funkci zájem, pak to nevyváží cenu tohoto komerčního balíku.

Volker Lendecke **<lendecke@namu01.gwdg.de>** napsal modul jádra linuxového souborového systému, který podporuje podmnožinu funkcí protokolu Novell NCP, jež umožňují připojení novellovských svazků k linuxovému souborovému systému bez dodatečných produktů pro souborový server. Volker Lendecke nazval tento balík `ncpfs` a nezbytné informace získal v knize „Netzwerkprogrammierung in C“, jejímiž autory jsou Manfred Hill a Ralf Zessin (další informace o této knize najdete v souboru `README`, který je součástí balíku `ncpfs`).

Tento software způsobí, že Linux emuluje pro souborové služby normální novellovskou pracovní stanici. Obsahuje také malou tiskovou utilitu, která umožňuje tisknout do novellovské tiskové fronty (je zdokumentována ve stati „Tiskový klient“ dále). Balík `ncpfs` spolupracuje s novellovskými souborovými servery verze 3.x a vyšší. Se serverem Novell 2.x fungovat nebude. Klient `ncpfs` také spolupracuje s produkty, které jsou úzce kompatibilní s produkty firmy Novell, ale některé programy, které o sobě tvrdí, že jsou kompatibilní, nesplňují tuto vlastnost úplně. Aby bylo možné používat balík `ncpfs` se souborovými servery Novell 4.x, musí být souborový server nakonfigurován pro režim bindery emulation, protože `ncpfs` zatím nepodporuje NDS.

8.1 Získání balíku ncpfs

Poslední verze balíku ncpfs byla navržena pro verzi jádra 1.2.13 nebo verze vyšší než 1.3.71 (sem patří i verze 2.x.x). Pokud používáte jádro, které nespadá do žádné z těchto dvou kategorií, pak budete muset provést jeho upgrade. Podrobný popis najdete v dokumentu *Kernel-HOWTO*.

Balík ncpfs získáte prostřednictvím anonymní služby FTP z Volkerova domovského systému `ftp://ftp.gwdg.de/pub/linux/misc/ncpfs/` nebo na adrese `ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs`, případně jeho zrcadlech. Aktuální verze měla v době psaní tohoto dokumentu označení `ncpfs-2.0.10.tgz`.

8.2 Sestavení ncpfs pro jádro verze 1.2.13

Sestavení jádra s podporou Ethernetu a protokolu IPX

Nejprve se musíte přesvědčit, že bylo vaše jádro sestaveno se zabudovanou podporou protokolu IPX. Ve verzi 1.2.13 je třeba jen kladně odpovědět na dotaz „`The IPX protocol`“, jak je ilustrováno v následujícím výpisu:

```
...
...
Assume subnets are local (CONFIG_INET_SNARL) [y]
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
The IPX protocol (CONFIG_IPX) [n] y
*
* SCSI support
...
...
```

Také je třeba zajistit zařazení příslušného ovladače pro vaši ethernetovou kartu. Pokud nevíte, jak na to, pak si přečtěte dokument *Ethernet-HOWTO*.

Pak můžete přistoupit k sestavení jádra. Nezapomeňte po skončení spustit příkaz `lilo`, který nainstaluje bootovací manažer.

Rozbalení balíku ncpfs

```
# cd /usr/src
# tar xvfz ncpfs-2.0.10.tgz
# cd ncpfs
```

Kontrola souboru Makefile

Hodláte-li používat program `kerneld` k automatickému zavedení modulu jádra `ncpfs`, pak musíte zrušit komentář na řádku v souboru `Makefile`, který odkazuje na `KERNELD`. Pokud si nejste jistí, co to znamená, měli byste si přečíst dokument *Kernel-HOWTO*, který vás seznámí s konfigurací modulů jádra.

Kompilace softwaru ncpfs

Celý balík by pak mělo být možné přeložit bez nutnosti další konfigurace:

```
# make
```

Zkopírování IPX-nástrojů na příslušné místo

Po skončení příkazu `make` by měly být všechny potřebné nástroje umístěny v adresáři `ncpfs/bin`. Pomocí příkazu

```
# make install
```

je nainstalujete do adresářů, které zvolil Volker Lendecke. Pokud používáte systém založený na ELF, pak bude nutné znovu spustit příkaz „`ldconfig -v`“, který zajistí dostupnost sdílené knihovny.

Zkopírování modulu ncpfs.o na příslušné místo

Po sestavení jádra verze `1.2.*` najdete v adresáři `ncpfs/bin` soubor `ncpfs.o`. To je modul jádra `ncpfs`. Tento soubor byste měli zkopírovat na jiné užitečnější místo. V mé distribuci *Debian* jsem ho zkopíroval do adresáře `/lib/modules/1.2.13/fs` a do souboru `/etc/modules` jsem automaticky přidal záznam `ncpfs`, aby byl tento modul automaticky spouštěn při startu. Používáte-li nějakou jinou distribuci, pak si musíte zjistit, kde uchovává své moduly a modul `ncpfs.o` sem zkopírovat, případně ho zkopírovat do adresáře `/etc`. K manuálnímu natažení modulu použijte příkaz

```
# insmod ncpfs.o
```

8.3 Sestavení ncpfs pro jádra verzí 1.3.71++/2.0

Abyste mohli používat poslední verzi balíku `ncpfs`, musíte mít jádro verze `1.3.71` nebo novější. Sem patří i jádra `2.0.*`.

Budete-li používat jádro verze `1.3.71` nebo novější, pak je balík `ncpfs` součástí standardní distribuce jádra. Stačí jen, když odpovíte `y` na následující dotaz:

```
Networking options  --->
       ...
       ...
       <*> The IPX protocol
       ...
Filesystems  --->
       ...
       ...
       <*> NCP filesystem support (to mount NetWare volumes)
       ...
```

Stále se ovšem budete muset řídit instrukcemi pro sestavení verzí jádra `1.2.*`, takže budete moci zkompilovat nástroje, ale nevznikne tím žádný modul, který byste mohli nainstalovat.

8.4 Konfigurace a používání ncpfs

Konfigurace síťového IPX-softwaru

Existují dva způsoby konfigurace síťového IPX-softwaru. Můžete buď všechny síťové IPX-informace nastavit ručně, anebo můžete nechat volbu rozumných nastavení na softwaru. Druhého případu dosáhnete pomocí následujícího příkazu:

```
# ipx_configure --auto_interface=on --auto_primary=on
```

Toto nastavení by mělo vyhovovat ve většině případů. Pokud tomu tak není, pak si přečtěte výše uvedenou stať „IPX-nástroje“, kde se dozvíte, jak lze tento software nakonfigurovat ručně.

Otestování konfigurace

Po nakonfigurování IPX-sítě by mělo být možné pomocí příkazu `slist` získat seznam všech novellovských souborových serverů ve vaší síti:

```
# slist
```

Pokud příkaz `slist` vypíše zprávu typu: `ncp_connect: Invalid argument`, pak vaše jádro zřejmě nepodporuje protokol IPX. Zkontrolujte, zda jste skutečně spustili příslušné jádro. Při startu systému by se měly objevit zprávy o protokolu IPX a modulu `ncpfs`. Pokud příkaz `slist` nevypíše všechny vaše souborové servery, pak bude zřejmě nutné přistoupit k ruční konfiguraci sítě.

Připojení novellovského svazku

Pracuje-li váš síťový IPX-software správně, měl by jít připojit svazek novellovského souborového serveru k vašemu linuxovému souborovému systému. K tomuto účelu slouží příkaz zajišťující odpojení, který vyžaduje zadání minimálně následujících parametrů:

1. Název souborového serveru.
2. Připojovací id souborového serveru. Je-li vyžadováno heslo, budete potřebovat také je.
3. Přihlašovacího bod, tj. kde chcete, aby došlo k připojení. Měl by to být existující adresář na vašem počítači.

K odpojení připojeného souborového systému NCP slouží ekvivalentní příkaz `ncpmout`. Pokud normálně ukončíte práci s počítačem, dojde ke správnému odpojení souborových systémů NCP, takže pokud použijete příkazy `halt` nebo `shutdown`, nemusíte se o příkaz `ncpunmount` starat.

Jako příklad si ukážeme připojení souborového serveru `ACCT_FS01` s přihlašovacím id `guest` bez hesla k adresáři `/mnt/Accounts`:

```
# ncpmount -S ACCT_FS01 /mnt/Accounts -U guest -n
```

Všimněte si použití volby `-n`, která udává, že k přihlášení není potřeba heslo. Stejné přihlášení s heslem `secret` by vypadalo následovně:

```
# ncpmount -S ACCT_FS01 /mnt/Accounts -U guest -P secret
```

Kontrola připojení

Pokud bylo připojení úspěšné, budou svazky přístupné příslušnému id pod adresářem, který jste uvedli jako bod připojení. Měli byste také být schopni procházet adresářovou strukturu a hledat příslušné soubory. Protože NCP neposkytuje uid ani gid vlastnictví souborů, budou mít všechny soubory nastavena stejná přístupová práva a vlastnictví jako adresář, který je přípojným bodem. Mějte to na paměti při sdílení připojení mezi linuxovými uživateli.

Nastavení automatického připojování

Potřebujete-li mít nějaký svazek NCP trvale připojen, budete muset výše uvedené příkazy zařadit do vašich souborů *rc*, aby byly prováděny automaticky při startu systému. Pokud již vaše distribuce Linuxu nabízí nějaký způsob konfigurace protokolu IPX, jako v případě distribuce Debian, pak vám doporučuji umístit příslušné příkazy do souboru `/etc/rc.local`. Můžete použít například následující syntaxi:

```
#
# Podpora pro souborový systém NCP

/sbin/insmod /lib/modules/1.2.13/fs/ncpfs.o

# konfigurace IPX-sítě
ipx_configure --auto_interface=on --auto_primary=on

# přihlášení na souborový server
ncpmount -S ACCT_FS01 /mnt/Accounts -U guest -n

#
```

Existuje ještě jeden způsob konfigurace připojení NCP, který spočívá ve vytvoření souboru `$HOME/.nwclient`. Tento soubor obsahuje informace o dočasných nebo uživatelských připojeních NCP, ke kterým pravidelně dochází. Můžete do něj umístit podrobnosti jednotlivých připojení, takže již příště nemusíte zadávat všechny detaily znovu.

Formát tohoto souboru je poměrně jednoduchý:

```
# První záznam je preferovaný server,
# který je použit když nespecifikujete server.
#
# Uživatel Terry se přihlašuje k serveru DOCS_FS01
s heslem 'password'
DOCS_FS01/TERRY password
#
# Přihlášení k serveru ACCT_FS01 bez hesla.

ACCT_FS01/GUEST -
```

K aktivaci těchto připojení můžete použít třeba následující příkaz:

```
$ ncpmount /home/terry/docs
```

který připojí server `DOCS_FS`, k němuž se přihlásí jako uživatel `TERRY`, pod adresář `/home/terry/docs`. Tento záznam byl zvolen z toho důvodu, že nebyl zadán žádný souborový server. Pokud bychom použili následující příkaz:

```
$ ncpmount -S ACCT_FS01 /home/terry/docs
```

připojil by systém adresář pro uživatele GUEST na serveru ACCT_FS01.

Poznámka: Aby výše uvedený mechanismus fungoval, musí mít soubor $HOME/nwclient nastavena přístupová práva 0600, čehož dosáhnete pomocí následujícího příkazu:

```
$ chmod 0600 $HOME/.nwclient
```

Chcete-li, aby mohli tento mechanismus využívat i jiní uživatelé než pouze uživatel root, musíte příkaz ncpmount pomocí následujícího příkazu nastavit suid root.

```
# chmod 4755 ncpmount
```

Vyzkoušení utility nsend

Součástí balíku je i utilita pro posílání zpráv novellovským uživatelům. Jmenuje se nsend a používá se následujícím způsobem:

```
# nsend rod hello there
```

Tento příkaz by poslal uživateli, který je přihlášen k vašemu preferovanému souborovému serveru (první server, který je uveden v souboru .nwclient) jako uživatel „rod“ zprávu „hello there“. Pomocí stejné syntaxe, jakou používá příkaz ncpmount, můžete zadat i jiný souborový server.

9 Konfigurace linuxového počítače jako NCP-serveru

K dispozici jsou dva balíky, které umožňují Linuxu simulovat funkce novellovského souborového serveru. Oba dva dovolují sdílet soubory uložené na vašem linuxovém počítači společně s uživateli, kteří používají klientský software Novell NetWare. Uživatelé si mohou připojovat a mapovat souborové systémy, které se jim na jejich počítačích budou jevit jako lokální disky, jako v případě skutečného novellovského souborového serveru. Můžete si vyzkoušet oba dva balíky a vybrat si ten, který bude lépe vyhovovat vašim účelům.

9.1 Balík mars_nwe

Martin Stover **<mstover@freeway.de>** vytvořil balík mars_nwe, díky němuž může Linux poskytovat klientům NetWare jak souborové, tak tiskové služby.

V případě, že vás zajímá původ názvu tohoto balíku, tak jde o zkatku z Martin Stovers NetWare Emulator.

9.1.1 Schopnosti balíku mars_nwe

Balík `mars_nwe` implementuje podmnožinu souborových služeb, disk based bindery a tiskových služeb Novell NCP. Pravděpodobně bude obsahovat chyby, ale v současné době ho používá velké množství lidí a počet chyb se dalšími verzemi neustále snižuje.

9.1.2 Získání balíku mars_nwe

Program mars_nwe získáte na adrese `ftp://ftp.gwdg.de/pub/linux/misc/ncpfs/` nebo `ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/`.

Nejaktuálnější verze v době psaní této knihy byla `mars_nwe-0.98.p18.tgz`.

9.1.3 Sestavení balíku mars_nwe

Sestavení jádra s podporou Ethernetu a protokolu IPX

Jste-li vlastníkem jádra verze 1.2.13, pak stačí, když při jeho kompilaci odpovíte kladně na otázku „`The IPX protocol`“ a záporně na otázku „`Full internal IPX network's`“, jako v následujícím výpisu:

```
...
...
The IPX protocol (CONFIG_IPX) [n] y
...
...
Full internal IPX network (CONFIG_IPX_INTERN) [N/y/?] n
...
...
```

Do novějších verzí jádra byl převzat podobný proces, i když skutečný text výzev se mohl mírně změnit.

Také je třeba zahrnout příslušný ovladač pro vaši ethernetovou kartu. Pokud nevíte, jak na to, pak byste si měli přečíst dokument *Ethernet-HOWTO*.

Pak můžete přistoupit ke kompilaci jádra. Nezapomeňte po skončení spustit příkaz `lilo`, který nainstaluje zaváděcí manažer.

Rozbalení balíku mars_nwe

```
# cd /usr/src
# tar xvfz mars_nwe-0.98.pl3.tgz
```

Kompilace balíku mars_nwe

Kompilace tohoto balíku je velice jednoduchá. Prvním krokem je spuštění programu `make`, čehož výsledkem bude soubor `config.h`. Potom byste měli tento soubor vyhledat a případně upravit. Můžete zde nastavit položky jako instalační adresáře, které se mají použít, a maximální počet relací a svazků, které bude příslušný server podporovat. Skutečně zajímavé položky, na které byste se měli zaměřit, jsou tyto:

```
FILENAME_NW_INI          umístění inicializačního souboru

PATHNAME_PROGS           kde se nacházejí podpůrné programy

PATHNAME_BINDERY         kde budou soubory bindery
PATHNAME_PIDFILES        adresář pro soubory s PID procesů
MAX_CONNECTIONS          maximální počet souběžných spojení
MAX_NW_VOLS              maximální počet svazků, které bude
                         mars-nwe podporovat
MAX_FILE_HANDLES_CONN    maximální počet otevřených souborů
                         na spojení
WITH_NAME_SPACE_CALLS    pokud chcete podporovat klienty ncpfs
INTERNAL_RIP_SAP         chcete, aby mars_nwe poskytoval RIP/SAP
                         směrování?
SHADOW_PWD               používáte stínová hesla?
```

Implicitní hodnoty pravděpodobně budou v pořádku, ale i tak byste je měli zkontrolovat.

Potom pomocí následujících příkazů

```
# make
# make install
```

sestavíte servery a nainstalujete je do příslušných adresářů. Instalační skript také nainstaluje konfigurační soubor `/etc/nwserv.conf`.

Konfigurace serveru

Konfigurace je velice jednoduchá. Stačí upravit soubor `/etc/nwserv.conf`. Formát tohoto souboru vám možná na první pohled bude připadat tajemný, ale ve skutečnosti je dobře srozumitelný. Soubor obsahuje několik jednořádkových konfiguračních záznamů. Každý řádek je vymezen bílým místem a začíná číslem, které udává jeho obsah. Veškeré znaky následující za znakem #“ jsou považovány za komentář a jsou ignorovány. Martin Stover dodává jako součást balíku vzorový konfigurační soubor, ale já vám nyní jako alternativu představím zjednodušený příklad tohoto souboru.

```
# VOLUMES (max. 5)
# Only the SYS volume is compulsory. The directory containing the SYS
# volume must contain the directories: LOGIN, PUBLIC, SYSTEM, MAIL.
# The 'i' option ignores case.
# The 'k' option converts all filenames in NCP requests to lowercase.
# The 'm' option marks the volume as removable useful for cdroms etc.
# The 'r' option set the volume to read-only.
# The 'o' option indicates the volume is a single mounted filesystem.
# The 'P' option allows commands to be used as files.
# The 'O' option allows use of the OS/2 namespace
# The 'N' option allows use of the NFS namespace
# The default is upper case.
# Syntax:
#  1 <Volumename>   <Volumepath>           <Options>

1      SYS           /home/netware/SYS/                    # SYS
1      DATA          /home/netware/DATA/       k           # DATA
1      CDROM         /cdrom                    kmr         # CDROM

# SERVER NAME
# If not set then the Linuxhostname will be converted to upper case
# and used. This is optional, the hostname will be used if this is
# not configured.
# Syntax:
#    2 <Servername>

2   LINUX_FS01

# INTERNAL NETWORK ADDRESS
# The Internal IPX Network Address is a feature that simplifies IPX
# routing for multihomed hosts (hosts that have ports on more than
# one IPX network).
# Syntax:
#    3 <Internal Network Address> [<Node Number>]
# or:
#    3 auto
#
# If you use 'auto' then your host IP address will be used. NOTE:
```

```
# this may be dangerous, please be sure you pick a number unique to
# your network.
# Addresses are 4byte hexadecimal (the leading 0x is required).

3   0x49a01010  1


# NETWORK DEVICE(S)
# This entry configures your IPX network. If you already have your
# IPX network configured then you do not need this. This is the same
# as using ipx_configure/ipx_interface before you start the server.
# Syntax:
#    4 <IPX Network Number> <device_name> <frametype> [<ticks>]
#                  Frame types: ethernet_ii, 802.2, 802.3, SNAP

4  0x39a01010  eth0  802.3  1

# SAVE IPX ROUTES AFTER SERVER IS DOWNED
# Syntax:
#    5 <flag>
# 0 = don't save routes, 1 = do save routes

5 0

# NETWARE VERSION
# Syntax:
#    6 <version>
#      0 = 2.15, 1 = 3.11

6 1
# PASSWORD HANDLING
# Real Novell DOS clients support a feature which encypts your
# password when changing it. You can select whether you want your
# mars server to support this feature or not.
# Syntax
#    7 <flag>
#    <flag> is:
#     0 to force password encryption. (Clients can't change
```

```
#        password)
#      1 force password encryption, allow unencrypted password
#        change.
#      7 allow non-encrypted password but no empty passwords.
#      8 allow non-encrypted password including empty passwords.
#      9 completely unencrypted passwords (doesn't work with OS/2)

7 1

# MINIMAL GID UID rights
# permissions used for attachments with no login. These permissions
# will be used for the files in your primary server attachment.
# Syntax:
#    10 <gid>
#    11 <uid>
#    <gid> <uid> are from /etc/passwd, /etc/groups

10  200
11  201

# SUPERVISOR password
# May be removed after the server is started once. The server will
# encrypt this information into the bindery file after it is run.
# You should avoid using the 'root' user and instead use another
# account to administer the mars fileserver.
#
# This entry is read and encrypted into the server bindery files, so
# it only needs to exist the first time you start the server to
# ensure that the password isn't stolen.
#
# Syntax:
#    12 <Supervisor-Login> <Unix username> [<password>]

12  SUPERVISOR  terry  secret

# USER ACCOUNTS
# This associates NetWare logins with unix accounts. Password are
# optional.
```

```
# Syntax:
#    13 <User Login> <Unix Username> [<password>]

13  MARTIN martin
13  TERRY  terry

# LAZY SYSTEM ADMIN CONFIGURATION
# If you have a large numbers of users and could not be bothered
  using
# type 13 individual user mappings, you can automatically map
# mars_nwe
# logins to Linuxuser names. BUT, there is currently no means of
# making use of the Linuxlogin password so all users configured this
# way are will use the single password supplied here. My
# recommendation is not to do this unless security is absolutely no
# concern to you.
# Syntax:
#    15 <flag> <common-password>
#    <flag> is:     0 - don't automatically map users.
#                   1 - do automatically map users not configured
#                       above.
#                   99 - automatically map every user in this way.

15  0  duzzenmatta

# SANITY CHECKING
# mars_nwe will automatically ensure that certain directories exist
# if you set this flag.
# Syntax:
#    16 <flag>
#    <flag> is 0 for no, don't, or 1 for yes, do.

16  0
# PRINT QUEUES
# This associates NetWare printers with unix printers. The queue
# directories must be created manually before printing is attempted.
# The queue directories are NOT lpd queues.
# Syntax:
```

```
#     21 <queue_name> <queue_directory> <unix_print_cmd>

21  EPSON  SYS:/PRINT/EPSON lpr -h
21  LASER  SYS:/PRINT/LASER lpr -Plaser

# DEBUG FLAGS
# These are not normally needed, but may be useful if are you
# debugging a problem.
# Syntax:
#     <debug_item> <debug_flag>
#
#     100 = IPX KERNEL
#     101 = NWSERV
#     102 = NCPSERV
#     103 = NWCONN
#     104 = start NWCLIENT
#     105 = NWBIND
#     106 = NWROUTED
#       0 = disable debug, 1 = enable debug

100 0
101 0
102 0
103 0
104 0
105 0
106 0

# RUN NWSERV IN BACKGROUND AND USE LOGFILE
# Syntax:
#     200 <flag>
#             0 = run NWSERV in foreground and don't use logfile
#             1 = run NWSERV in background and use logfile

200  1

# LOGFILE NAME
# Syntax:
```

```
#     201 <logfile>

201   /tmp/nw.log

# APPEND LOG OR OVERWRITE
# Syntax:
#     202 <flag>
#         0 = append to existing logfile
#         1 = overwrite existing logfile

202   1

# SERVER DOWN TIME
# This item sets the time after a SERVER DOWN is issued that the
# server really goes down.
# Syntax:
#     210 <time>
#         in seconds. (defaults 10)

210   10

# ROUTING BROADCAST INTERVAL
# The time is seconds between server broadcasts
# Syntax:
#     211 <time>
#         in seconds. (defaults 60)

211   60

# ROUTING LOGGING INTERVAL
# Set how many broadcasts take place before logging of routing
# information occurs.
# Syntax:
#     300   <number>

300   5

# ROUTING LOGFILE
```

```
# Set the name of the routing logfile
# Syntax:
#     301 <filename>

301  /tmp/nw.routes

# ROUTING APPEND/OVERWRITE
# Set whether you want to append to an existing log file or
# overwrite it.
# Syntax:
#     302 <flag>
#         <flag> is 0 for append, 1 for create/overwrite

302  1

# WATCHDOG TIMING
# Set the timing for watchdog messages that ensure the network is
# still alive.
# Syntax:
#     310 <value>
#         <value> =    0 - always send watchdogs
#                    < 0 - (-ve) for disable watchdogs
#                    > 0 - send watchdogs when network traffic
#                          drops below 'n' ticks

310  7

# STATION FILE
# Set the filename for the stations file which determine which
# machines this fileserver will act as the primary fileserver for.
# The syntax of this file is described in the 'examples' directory
# of the source code.
# Syntax:
#     400 <filename>

400  /etc/nwserv.stations

# GET NEAREST FILESERVER HANDLING
```

```
# Set how SAP Get Nearest Fileserver Requests are handled.
# Syntax:
#    401 <flag>
#        <flag> is: 0 - disable 'Get Nearest Fileserver' requests.
#                   1 - The 'stations' file lists stations to be
                       excluded.
#                   2 - The 'stations' file lists stations to be
                       included.

401  2
```

Spuštění serveru

Pokud jste nastavili server tak, aby čekal na externí programy, které nastaví vaši síť a/nebo budou poskytovat směrovací funkce, pak byste měli tyto programy spustit ještě před spuštěním serveru. Za předpokladu, že jste nastavili server tak, aby sám nakonfiguroval rozhraní a poskytoval směrovací služby, stačí spustit pouze následující příkaz:

```
# nwserv
```

Otestování serveru

Abyste server otestovali, měli byste se nejprve zkusit připojit a přihlásit do vaší sítě z klienta NetWare. Pak nastavte `CAPTURE` z klienta a vyzkoušejte tisk. Budete-li v obou případech úspěšní, pak server funguje správně.

9.2 Balík lwared

Aleš Dryák **<A.Dryak@sh.cvut.cz>** vytvořil balík `lwared`, díky němuž může linuxový server fungovat jako souborový server využívající protokol NCP.

Aleš dal tomuto balíku název *lwared*, protože je to zkratka *Lin Ware Daemon*.

9.2.1 Kompatibilita balíku lwared

Server `lwared` je schopen poskytovat část služeb protokolu Novell NCP. Umí posílat zprávy, ale neposkytuje žádné funkce týkající se tisku. V současné době příliš nefunguje s klienty Windows95 nebo Windows NT. Server `lwared` spoléhá při vytvoření a aktuaizaci IPX-směrování a tabulek SAP na externí programy. Nevychovaní klienti mohou způsobit jeho pád. Je také důležité upozornit, že součástí balíku nejsou utility pro převod názvů souborů.

Server funguje pod příkazovými interprety NETX a VLM NetWare.

9.2.2 Získání balíku lwared

Balík `lwared` lze přeložit pod jádrem verze `1.2.0`, doporučuji však verzi `1.2.13`, protože v tom případě nejsou nutné žádné záplaty. S příchodem verze `1.3.*` došlo k určitým změnám protokolu IPX, takže pro správnou funkci tohoto balíku nyní potřebujete záplaty. Příslušné záplaty jsou součástí nových verzí jádra, takže i když musíte používat alfa verzi, bude vám `lwared` fungovat správně.

Balík lwared získáte pomocí anonymní služby FTP na adrese: *f*`tp://klokan.sh.cvut.cz/pub/linux/linware/`

nebo na

`ftp://sunsite.unc.edu/pub/Linux/system/network/demons` nebo na jejich zrcadlech. Aktuální verze v době psaní knihy nesla označení `lwared-0.95.tar.gz`.

9.2.3 Sestavení balíku lwared

Rozbalení balíku lwared

Například:

```
# cd /usr/src
# tar xvpfz lwared-0.95.tar.gz
```

Sestavení jádra s podporou Ethernetu a protokolu IPX

Pokud používáte alfa verzi `1.3.*`, pak by to měla být verze `1.3.17` nebo novější, protože již obsahuje příslušné opravy chyb. Opravy verzí starší než `1.3.17` vyžadují manuální instalaci (*některé informace o tomto zákroku najdete v souboru* `INSTALL`*, který je součástí balíku* `lwared`). Při instalaci oprav jádra verze 1.3.17 nebo vyšší zadejte následující příkaz:

```
# make patch
```

Po provedení nezbytných úprav je třeba se přesvědčit, že jste jádro přeložili s podporou protokolu IPX. Ve verzi jádra `1.2.13` je třeba pouze odpovědět kladně (`Y`) na dotaz „`The IPX protocol`“, jako v následujícím výpise:

```
...
...
Assume subnets are local (CONFIG_INET_SNARL) [y]
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
The IPX protocol (CONFIG_IPX) [n] y
*
* SCSI support
...
...
```

Do novějších verzích jádra byl převzat podobný proces, i když skutečný text výzev se mohl mírně změnit.

Také je třeba zahrnout příslušný ovladač pro vaši ethernetovou kartu. Pokud nevíte, jak na to, pak byste si měli přečíst dokument *Ethernet-HOWTO*.

Pak můžete přistoupit ke kompilaci jádra. Nezapomeňte po skončení spustit příkaz `lilo`, který nainstaluje bootovací manažer.

Kompilace a instalace balíku lwared

Před kompilací balíku `lwared` byste měli nejprve, je-li to nutné, upravit soubor `server/config.h`. Tento soubor obsahuje různá nastavení, která budou řídit způsob chování serveru při běhu. Implicitní hodnoty jsou nastaveny rozumně, i když bude možná dobré zkontrolovat, zda adresáře určené pro log soubory a konfigurační soubory vyhovují vašemu systému.

```
# make depend
# make
# make install
```

Zjistil jsem, že si příkaz „`make depend`“ stěžoval na to, že v mém systému nenašel soubor `float.h`, i přesto však fungoval. Také jsem při pokusu o kompilaci pomocí překladače `gcc 2.6.3` zjistil, že v souboru `lib/ipxkern.c` je třeba změnit řádek

```
#include <net/route.h>
```

na

```
#include <net/if_route.h>
```

protože tento soubor někdy změnil svůj název.

Příkaz „`make install`“ se pokusí nainstalovat server a směrovacího démona do adresáře `/usr/sbin`, program `lwpasswd` do adresáře `/usr/bin`, pomocné programy protokolu IPX budou nainstalovány do adresáře `/sbin` a v neposlední řadě manuálové stránky přijdou do adresáře `/usr/man`. Pokud vám některé z těchto umístění nevyhovuje, pak budete muset změnit cílové adresáře v souboru `Makefile`.

9.2.4 Konfigurace a používání balíku lwared

Nyní trochu legrace.

Konfigurace sítě IPX

Nejprve je nutné nastavit ethernetová rozhraní tak, aby podporovala IPX-sítě, které bude podporovat váš server. K tomu budete potřebovat znát síťové IPX-adresy každého vašeho LAN-segmentu, které ethernetové zařízení (`eth0`, `eth1` atd.) je na kterém segmentu, jaký typ rámce (`802.3`, `EtherII` atd.) každý LAN-segment používá a které interní síťové adresy má váš server používat (to je nutné zejména v případě, že bude váš server obsluhovat více než jeden LAN-segment). Konfigurace serveru, který je na dvou různých segmentech se síťovými IPX-adresami `0x23a91300` a `0x23a91301` a interní síťovou adresou 0xbdefaced, by mohla vypadat následovně:

```
# ipx_internal_net add BDEFACED 1
# ipx_interface add eth0 802.3 23a91300
# ipx_interface add eth1 etherii 23a91301
```

Spuštění směrovacích démonů

Vlastní software jádra ve skutečnosti provádí doručování IPX-paketů stejně jako v případě protokolu IP, ale jádro potřebuje ke správě aktualizací směrovací tabulky dodatečné programy. V případě protokolu IPX jsou potřeba dva démoni, z nichž oba jsou dodáváni s balíkem `lwared`. Démon `ipxripd` spravuje IPX směrovací informace a démon `ipxsapd` řídí SAP-informace. Při spouštění těchto dvou démonů je pouze třeba zadat umístění jejich log-souborů:

```
# ipxripd /var/adm/ipxrip
# ipxsapd /var/adm/ipxsap
```

Konfigurace serveru lwared

K tomu, aby se mohl uživatel přihlásit k serveru `lwared`, je třeba ručně nastavit dva soubory. Jsou to:

/etc/lwpasswd

V tomto souboru uchovává server LinWare informace o uživatelských účtech. Program lwpasswd je udržuje aktuální. V nejjednodušší podobě vypadá soubor /etc/lwpasswd následovně:

```
ales:
terryd:
guest:
```

Jeho formát odpovídá seznamu přihlašovacích id následovaných znakem „:“ a zašifrovaným heslem. Platí zde tato dvě důležitá pravidla: Pokud zde není uvedeno žádné zašifrované heslo, je příslušný učet bez hesla. Uživatelé serveru LinWare musí mít linuxové účty, to znamená, že každý uživatel, který se objeví v souboru /etc/lwpasswd, musí mít svůj záznam také v souboru /etc/passwd a jedině uživatel root může změnit heslo jiného uživatele serveru LinWare. Pokud jste přihlášeni jako root, můžete podle následujícího příkladu změnit heslo některého uživatele serveru LinWare:

```
# lwpasswd rodg
Changing password for RODG
Enter new password:
Re-type new password:
Password changed.
```

/etc/lwvtab

Tento soubor LinWare definuje tabulku svazků a uchovává informace o tom, které adresáře mají být přístupné uživatelům serveru LinWare (podobnou funkci má soubor /etc/exports v systému NFS). Následuje jednoduchý příklad formátu tohoto souboru:

```
SYS          /lwfs/sys
DATA                /lwfs/data

HOME         /home
```

Formát je jednoduchý: Název svazku následovaný bílým místem a exportovaným linuxovým adresářem. Soubor musí obsahovat **minimálně** záznam pro svazek SYS, aby se mohl server spustit. Chcete-li, aby i uživatelé DOSu mohli využívat server LinWare jako svůj primární server, musíte v adresáři, který exportujete jako svazek SYS, vytvořit standardní adresářovou strukturu svazku SYS. Protože jsou tyto soubory vlastnictvím firmy Novell, měli byste mít příslušnou licenci. Budou-li vaši uživatelé používat novellovský souborový server jako svůj primární server, pak tato licence není nutná.

Spuštění serveru lwared

```
# lwared
```

Je to takřka antiklimax, není-liž pravda? Dobře, takže jste dostali otázku, mám pravdu? Jaký je název oznamovaného serveru? Pokud jste server spustili výše uvedeným způsobem, bude inzerovaný název serveru LinWare záviset na tom, co vrátí linuxový program `hostname`. Pokud chcete používat jiný název, můžete ho předat serveru při jeho spuštění. Například:

```
# lwared -nlinux00
```

spustí server s názvem `linux00`.

Otestování serveru lwared

Ze všeho nejdříve vyzkoušejte, zda se váš server LinWare objeví v seznamu, který vrátí program *slist* spuštěný z dosového klienta ve vaší síti. Program `slist` je uložen na svazku `SYS` novellovského souborového serveru, takže ho musíte spustit z počítače, který je k němu již přihlášen. Pokud neuspějete, pak zkontrolujte, zda běží současně programy `ipxsapd` i `lwared`. Po úspěšném provedení příkazu `slist` se zkuste přihlásit k serveru a připojit nějaký svazek:

```
C:> attach linux00/ales
...
...
C:> map l:=linux00/data:
C:> l:
```

Potom by mělo být možné zacházet s novou mapou stejně jako s kteroukoliv jinou mapou. Přístupová práva k souborům, které budete mít, budou záviset na právech přidělených účtu *linux*, který se shoduje s vaším přihlášením k serveru LinWare.

10 Konfigurace linuxového počítače jako novellovského tiskového klienta

Balík `ncpfs` obsahuje dva malé programy, které umožňují posílat tisk z linuxového počítače na tiskárnu připojenou k novellovskému tiskovému serveru. Příkaz `nprint` umožňuje poslat soubor do tiskové fronty serveru NetWare. Příkaz `pqlist` vypíše seznam dostupných tiskových front na serveru NetWare.

Tyto programy získáte a nainstalujete podle výše uvedených instrukcí ohledně NCP-klienta.

Oba příkazy vyžadují zadání uživatelského jména a hesla, takže si možná budete chtít vytvořit skripty příkazového interpretu, které by vám celý proces tisku ulehčily:

Mohly by vypadat třeba takto:

```
# pqlist -S ACCT_FS01 -U guest -n
# nprint -S ACCT_FS01 -q LASER -U guest -n filename.txt
```

Přihlašovací syntaxe je stejná jako u příkazu `ncpmount`. Ve výše uvedeném příkladu předpokládáme, že na souborovém serveru `ACCT_FS01` existuje účet `guest` bez hesla, dále že existuje tisková fronta `LASER` a uživatel `guest` do ní může posílat tiskové úlohy.

11 Konfigurace linuxového počítače jako novellovského tiskového serveru

Součástí balíku `ncpfs` je program, který udělá z vašeho linuxového počítače tiskový server v síti NetWare. Instrukce, jak ho získat a přeložit, najdete výše ve stati „Klient NetWare“.

11.1 Nezbytné předpoklady

Konfigurace tohoto programu je poměrně jednoduchá, ale počítá, že máte dokončenou konfiguraci tiskárny pod Linuxem. Toto téma je hlouběji rozebíráno v dokumentu Printing-HOWTO.

11.2 Konfigurace

Jakmile máte fungující konfiguraci tiskárny a nainstalovanou utilitu `pserver`, zbývá přidat do `rc` souborů příkazy, které ji spustí.

Příkaz, který použijete, bude záviset na tom, jak chcete aby fungoval, ale v tomto jednoduchém příkladu postačí následující zápis:

```
# pserver -S ACCT_01 -U LASER -P secret -q LASERJET
```

Tento příklad požádá utilitu `pserver`, aby se přihlásila k souborovému serveru `ACCT_01` pod uživatelským jménem `LASER` s heslem `secret` a převzala úlohy z tiskové fronty `LASERJET`. Po přijetí příchozí tiskové úlohy předá tuto tiskovou úlohu pomocí implicitního tiskového příkazu nebo programu `lpr` linuxovému tiskovému démonu.

Pokud byste chtěli, mohli byste k přijímání a tisku tiskových úloh používat jakýkoliv linuxový příkaz. Konkrétní tiskový příkaz umožňuje zadat argument `-c`. Například příkaz:

```
# pserver -S ACCT_01 -U LASER -P secret -q LASERJET
  -c "lpr -Plaserjet"
```

by provedl to samé, jako ten předchozí pouze s tím rozdílem, že by úlohu neposlal implicitnímu příkazu, ale na tiskárnu `laserjet`.

12 Přehled uživatelských a administrativních příkazů balíku ncpfs

Nové verze Volkerova balíku `ncpfs` obsahují paletu uživatelských a administrativních příkazů, které byste mohli využít. Tyto nástroje jsou překládány a instalovány v rámci instalace balíku `ncpfs`, takže pokud jste se neřídili instrukcemi uvedenými ve stati „Klient NetWare“ vedoucími k jejich instalaci, potom tak nyní učiňte.

Podrobné informace najdete v dodávaných manuálových stránkách. Zde uvádíme pouze stručný popis jednotlivých příkazů.

12.1 Uživatelské příkazy

`ncopy` Network Copy – umožňuje efektivní kopírování souborů pomocí funkce NetWare místo kopírování přes síť.

`nprint` Network Print – umožňuje tisk souboru do tiskové fronty NetWare na serveru NetWare.

`nsend` Network Send – umožňuje posílat zprávy jiným uživatelům na serveru NetWare.

`nwbols` List Bindery Objects – umožňuje vypsat seznam obsahu bindery serveru NetWare.

`nwboprops` List Properties of a Bindery Object – zpřístupní vlastnosti objektu NetWare bindery.

`nwbpset` Set Bindery Property – slouží k nastavení vlastností objektu NetWare bindery.

`nwbpvalues` Print NetWare Bindery Objects Property Contents – umožňuje vytisknout obsah vlastnosti objektu NetWare bindery.

`nwfsinfo` Fileserver Information – vytiskne některé souhrnné informace o serveru NetWare.

`nwpasswd` NetWare Password – umožňuje změnit hesla uživatelů serveru NetWare.

`nwrights` NetWare Rights – zobrazí práva přidělená konkrétnímu souboru nebo adresáři.

`nwuserlist` Userlist – vypíše seznam uživatelů aktuálně přihlášených k souborovému serveru NetWare.

`pqlist` Print Queue List – zobrazí obsah tiskové fronty serveru NetWare.

`slist` Server List – vypíše seznam známých serverů NetWare.

12.2 Administrativní nástroje

`nwbocreate` Create a Bindery Object – umožňuje vytvořit objekt NetWare bindery.

`nwborm` Remove Bindery Object – umožňuje smazat objekt NetWare bindery.

`nwbpadd` Add Bindery Property – umožňuje nastavit hodnotu existující vlastnosti objektu NetWare bindery.

`nwbpcreate` Create Bindery Property – umožňuje vytvořit novou vlastnost existujícího objektu NetWare bindery.

`nwbprm` Remove Bindery Property – umožňuje odstranit vlastnost z objektu NetWare bindery.

`nwgrant` Grant Trustee Rights – umožňuje přidělit správcovská (trustee) práva adresáři na souborovém serveru NetWare.

`nwrevoke` Revoke Trustee Rights – umožňuje odstranit správcovská (trustee) práva adresáře na souborovém serveru NetWare.

13 Konfigurace protokolu PPP s podporou protokolu IPX

Nové verze démona pppd pro Linux umožňují přenášet IPX-pakety po sériové PPP-lince. Potřebujete k tomu minimálně verzi ppp-2.2.0d. Informace o tom, kde ji můžete získat, najdete v dokumentu *PPP-HOWTO*. Když budete tohoto démona překládat, musíte se ujistit, že soubor /usr/src/linux/pppd-2.2.0f/pppd/Makefile.linux obsahuje následující dva řádky, které zapínají podporu protokolu IPX.

```
IPX_CHANGE = 1
USE_MS_DNS = 1
```

Volba IPX_CHANGE začlení podporu IPX do protokolu PPP. Volba USE_MS_DNS umožňuje počítačům s Microsoft Windows95 provádět vyhledávání názvů (Name Lookups).

Pokud chcete tuto podporu rozchodit, musíte vědět, jak ji nakonfigurovat.

Je to možné udělat několika způsoby, ale zde si popíšeme dva, o kterých se mi podařily sehnat nějaké informace. Ani jeden z nich jsem nezkoušel, takže považujte tuto stať za experimentální a podaří-li se vám něco rozběhnout, pak mi dejte vědět.

13.1 Konfigurace serveru IPX/PPP

Nejprve je nutné nastavit váš linuxový počítač jako server IP/PPP. Nemějte strach, není to nic obtížného. Znovu stačí, když se budete řídit instrukcemi uvedenými v dokumentu *PPP-HOWTO*. Potom je ještě potřeba provést některé úpravy, aby při stejné konfiguraci fungovala i podpora protokolu IPX.

13.1.1 První kroky

Jedním z prvních kroků je konfigurace vašeho linuxového počítače jako IPX-routeru podle popisu, který uvádím dříve v tomto dokumentu. Nebudete muset používat příkaz ipx_route, protože konfiguraci rozhraní ppp, stejně jako IP, za vás udělá démon pppd. Pokud běží démon ipxd, automaticky rozpozná jakákoliv nová IPX-rozhraní a rozšíří pro ně trasy. Takto budou vaši vytáčení hostitelé po připojení automaticky viditelní pro ostatní počítače.

13.1.2 Návrh

Pokud fungujete jako server, budete zodpovědní za přiřazení síťových adres všem PPP-spojením po jejich navázání. Tento bod je velice důležitý, protože každé PPP-spojení bude IPX-sítí a bude mít jedinečnou síťovou IPX-adresu. To znamená, že se musíte rozhodnout, jak a jaké ad-

resy budete alokovat. Běžně se alokuje jedna síťová IPX-adresa pro každé sériové zařízení, které bude podporovat IPX/PPP. Mohli byste alokovat síťovou IPX-adresu v závislosti na přihlašovacím id uživatele, ale pro to neexistuje žádný konkrétní důvod.

Budu předpokládat, že jste provedli vše potřebné a že budeme používat dvě sériová zařízení (modemy). V tomto vymyšleném příkladu jsem jim přiřadil následující adresy:

zařízení	síťová adresa IPX
ttyS0	0xABCDEF00
ttyS1	0xABCDEF01

13.1.3 Konfigurace démona pppd

V souboru `/etc/ppp/options.ttyS0` uveďte následující záznamy:

```
ipx-network 0xABCDEF00
ipx-node 2:0
ipxcp-accept-remote
```

a v souboru `/etc/ppp/options.ttyS1` tyto:

```
ipx-network 0xABCDEF01
ipx-node 3:0
ipxcp-accept-remote
```

Tím požádáte démona `pppd`, aby po navázání spojení alokoval příslušnou síťovou IPX adresu, nastavil lokální číslo uzlu `2` nebo `3` a nechal vzdálený uzel přepsat uzlem, kterým myslí, že je. Všimněte si, že každá z těchto adres je hexadecimální číslo a že na začátku síťové adresy je vyžadován řetězec `0x`, nikoli však na začátku adresy uzlu.

Tuto informaci lze nastavit také na jiných místech. Pokud máte pouze jeden vytáčecí modem, pak by mohl být tento záznam součástí souboru `/etc/ppp/options`. Nebo lze tuto informaci předat démonu `pppd` na příkazové řádce,

13.1.4 Otestování konfigurace serveru

K otestování konfigurace serveru budete potřebovat funkční konfiguraci klienta. Když volající vytočí číslo, přihlásí se a spustí se démon `pppd`, přidělí serveru síťovou adresu, oznámí klientovi číslo uzlu serveru a sjedná číslo uzlu klienta. Když potom démon `ipxd` detekuje nové rozhraní, měl by být klient schopen navázat IPX-spojení se vzdálenými hostiteli.

13.2 Konfigurace klienta IPX/PPP

Při konfiguraci klienta závisí to, zda nakonfigurujete váš linuxový počítač jako IPX-router na tom, zda máte lokální počítačovou síť, pro kterou chcete, aby tento počítač fungoval jako IPX-router. Máte-li samostatný počítač, který se připojuje k vytáčenému IPX/PPP-serveru, pak nebudete potřebovat spouštět démona `ipxd`, ale máte-li lokální počítačovou síť a chcete, aby mohly IPX-router využívat všechny počítače v této síti, pak musíte nakonfigurovat a spustit démona `ipxd` tak, jak bylo popsáno. Tato konfigurace je mnohem jednodušší, protože nemusíte nastavovat více sériových zařízení.

13.2.1 Konfigurace démona pppd

Nejjednodušší konfigurace je ta, která umožňuje serveru poskytovat veškeré informace o konfiguraci IPX-sítě. Tato konfigurace by byla kompatibilní s výše popsanou konfigurací serveru.

Znovu je třeba přidat nějaké záznamy do souboru `/etc/ppp/options`:

```
ipxcp-accept-network
ipxcp-accept-remote
ipxcp-accept-local
```

Tyto záznamy říkají démonu `pppd`, aby se choval zcela pasivně a přijal všechny detaily konfigurace od serveru. Pro servery, které neposkytují podrobné informace, zde můžete také uvést implicitní hodnoty pomocí záznamů `ipx-network` a `ipx-node`.

13.2.2 Testování klienta IPX/PPP

K otestování klienta budete potřebovat fungující server, se kterým byste se mohli spojit. Po vytočení a spuštění démona pppd by měl program `ifconfig` vypsat podrobnosti konfigurace protokolu IPX na vašem zařízení `ppp0` a měl by fungovat příkaz `ncpmount`.

Nejsem si jistý, zda je pro spojení se vzdálenými souborovými servery nutné ručně přidat IPX- router. Zdá se to pravděpodobné. Budu však vděčen komukoliv, kdo s touto konfigurací pracuje a sdělí mi příslušné informace.

14 IPX-tunel přes protokol IP

Spousta z vás se dostane do situace, kdy máte dvě novellovské lokální počítačové sítě a mezi nimi pouze jediné IP-spojení. Jistě vás napadne, jak budete s tímto uspořádáním hrát deathmatch v hře DOOM pro DOS? Andreas Godzina **<ag@agsc.han.de>** má pro vás odpověď v podobě nástroje zvaného `ipxtunnel`.

Program ipxtunnel poskytuje jakýsi most pro protokol IPX tím, že umožní zapouzdření IPX-paktů do IPXD/IP-datagramů, takže mohou být přenášeny po TCP/IP-spojení. Odposlouchává IPX-pakety a jakmile na nějaký narazí, zabalí ho do TCP/IP datagramu a přesměruje ho na vzdálenou IP-adresu, kterou zadáte. Aby to fungovalo, musí samozřejmě na počítači, na který směřujete zapouzdřený protokol IPX, také běžet stejná verze programu ipxtunnel, jakou používáte vy.

14.1 Získání programu ipxtunnel

Program ipxtunnel získáte na adrese
ftp://sunsite.unc.edu/pub/Linux/system/network/demons.

14.2 Přeložení programu ipxtunnel

Program ipxtunnel přeložíte pomocí následující sekvence příkazů:

```
# cd /usr/src
# tar xvfz .../ipxtunnel.tgz
# cd ipxtunnel
# make
```

14.3 Konfigurace programu ipxtunnel

Konfigurace programu ipxtunnel není vůbec složitá. Řekněme, že počítač vašich přátel se nazývá gau.somewhere.com a váš počítač gim.sw.edu. Program ipxtunnel používá konfigurační soubor jménem /etc/ipxtunnel.conf. Tento soubor umožňuje zadat implicitní UDP-port, který se má použít pro TCP/IP-spojení, kam se mají posílat zapouzdřená data, a určit které z vašich lokálních rozhrání má ipxtunnel odposlouchávat a doručovat na ně IPX-pakety.

Jednoduchý konfigurační soubor vypadá asi takto:

```
#
# Soubor /etc/ipxtunnel.conf pro gim.sw.edu
#
# Používaný UDP port:                          (implicitně 7666)
port 7777
#
# Vzdálený server, kam budete posílat pakety:
remote gau.somewhere.com
```

```
#
# Lokální rozhraní, kde čekáme na IPX pakety: (implicitně eth0)
interface eth0
interface eth1
```

Samozřejmě, že počítač na druhém konci by měl podobný konfigurační soubor, v němž by byl tento počítač uveden jako `remote` hostitel.

14.4 Testování a používání programu ipxtunnel

Program `ipxtunnel` funguje jako IPX-most, takže by IPX-sítě měly být na obou koncích spojení stejné. Andreas Godzina nikdy netestoval svůj program v prostředí, které skutečně podporuje novellovské souborové servery, takže pokud se vám to podaří v reálném prostředí, dejte Andreasovi vědět, zda to funguje.

Běží-li program `ipxtunnel`, měla by jít spustit hra DOOM na obou koncích spojení pracujících v režimu IPX a tyto dva počítače by o sobě měly navzájem vědět.

Andreas Godzina používal tento kód pouze na vysokorychlostních linkách a netvrdí, že na pomalejších bude stejně rychlý. I v tomto případě mu napište, co vám fungovalo a co nikoliv.

15 Komerční podpora protokolu IPX pro Linux

15.1 Network Desktop společnosti Caldera

Společnost Caldera Inc. produkuje distribuci Linuxu, která obsahuje velké množství komerčně podporovaných vylepšení včetně podpory klienta Novell NetWare. Základem je respektovaná distribuce Red Hat a společnost Caldera k ní přidala svůj balík produktů nazvaný „Network Desktop“. Podpora NetWare obsahuje plnohodnotného klienta Novell NetWare založeného na technologii licencované společností Novell Corporation. Tento klient umožňuje úplný přístup k souborovým serverům Novell 3.x a 4.x a disponuje službami typu NDS (NetWare Directory Service) a šifrování RSA.

Mnohem více informací o tomto produktu a jeho získání najdete na WWW-serveru společnosti Caldera `http://www.caldera.com/`.

Pokud pracujete v prostředí NetWare 4.x a/nebo NDS, je pro vás Caldera NetWare Client jediným možným řešením.

Také pokud vlastníte obchodně důležitou aplikaci využívající podpory Novellu v Linuxu, měli byste o produktu společnosti Caldera popřemýšlet.

16 Některé často kladené dotazy

Kde najdu komerčně podporovaný IPX-software pro Linux?

Společnost Caldera Corporation nabízí plně licencovaného a plně podporovaného klienta NetWare 3.x a 4.x. Informace o něm získáte na adrese `http://www.caldera.com/`.

1. **Funguje IPX-software se sítěmi ArcNet/Token Ring/atd?**

   Linuxový IPX-software funguje s rozhraními ArcNet a TokenRing. Neslyšel jsem o nikom, kdo by ho zkoušel s AX.25. Konfigurace je podobná konfiguraci Ethernetu, jen s tou výjimkou, že budete muset tam kde je to nutné nahradit příslušné názvy zařízení na místě „eth0“ a příslušné hardwarové adresy.

2. **Jak nastavím víc než jedno IPX-rozhraní?**

   Pokud máte ve vašem počítači víc než jedno rozhraní, měli byste každé z nich nastavit pomocí příkazu `ipx_interface`. Neměli byste používat konfiguraci „plug and play“.

3. **Jaké mám zvolit IPX-adresy?**

   Sítě IPX jsou podobné, ne však identické se sítěmi IP. Základní rozdíl spočívá ve způsobu, jakým používají adresy. Protokol IPX nepoužívá koncept podsítí, takže druh přiřazení mezi síťovými adresami a sítěmi je odlišný. Pravidla jsou poměrně jednoduchá:

   - Každá síťová IPX-adresa musí být jedinečná v rámci WAN. Sem patří interní síťové adresy (Internal Network Addresses). Spousta organizací, které používají protokol IPX v dálkových sítích, bude mít nějaký druh adresovacího standardu, který musíte dodržovat.
   - Každá adresa hostitele v samostatné síti musí být jedinečná. To znamená, že každý hostitel v každé IPX-síti musí mít přiřazenu jedinečnou adresu. V případě ethernetové sítě to není problém, protože každá karta má jedinečnou adresu. V případě sítí IPX/PPP je třeba přidělit jedinečné adresy všem hostitelům v síti, bez ohledu na to, ke kterému konci jsou připojeni. Hostitelské adresy nemusí být jedinečné v rámci dálkové sítě, protože k jedinečné identifikaci hostitele se používá síťová adresa v kombinaci s hostitelskou adresou.

4. **Jaké typy rámců bych měl použít?**

   Existují rozmanité typy rámců, které lze použít pro protokol IPX. Nejběžnějším jsou popisovány ve stati „Některé termíny používané v tomto dokumentu“ (pod heslem „**Frame Type**“).

   V případě, že instalujete váš počítač do existující sítě, pak musíte použít již zaběhnutý typ rámce, aby mohl váš počítač komunikovat s ostatními hostiteli v síti. Pokud ale zavádíte úplně novou síť, můžete k přenosu protokolu IPX použít kterýkoliv z možných protokolů. Já osobně vám pro přenos protokolu IPX i IP doporučuji typ rámce `Ethernet_II`.

5. **Mým počítačům běžícím pod Windows95 se nepodařila autodetekce typu rámce.**

   Jak se zdá, někdy k tomu může dojít. Mohl bych vám dát spoustu rad, ale místo toho vám poradím použít manuální konfiguraci typu rámce. V každém případě je to zřejmě lepší metoda.

6. **Proč se mi při konfiguraci protokolu IPX objevuje zpráva „`invalid argument`“ (neplatný argument)?**

   Pravděpodobně používáte jádro, které nepodporuje protokol IPX. Proto buď jádro znovu sestavte, nebo dvakrát zkontrolujte, že jste pomocí LILO skutečně nainstalovali a používáte nové jádro.

7. **Proč se mi při konfiguraci protokolu IPX objevuje zpráva „`package not installed`“ (balík nebyl nainstalován)?**

   Pravděpodobně používáte jádro, které nepodporuje protokol IPX. Proto buď jádro znovu sestavte, nebo dvakrát zkontrolujte, že jste pomocí LILO skutečně nainstalovali a používáte nové jádro.

8. **Proč dostávám od démona `pppd` zprávu „`IPX support not in kernel`“(Jádro neobsahuje podporu protokolu IPX)?**

   Pravděpodobně jste přeložili podporu protokolu IPX jako modul a nezajistili jste jeho nahrání před démonem `pppd`.

9. **Jak lze pomocí systému NFS exportovat připojený souborový systém NCP?**

   Abyste mohli exportovat souborový systém NCP pomocí NFS, musíte ho připojit příkazem `ncpmount` s volbou `-V`. Tato volba umožňuje připojit pouze jeden svazek souborového serveru, místo obvyklého připojení všech svazků. Když to uděláte, umožní vám démon NFS exportovat tento souborový systém obvyklým způsobem.

10. Proč v interní síti s balíkem `mars_nwe` nefunguje příkaz `slist`?

Musíte mít povolen požadavek „get nearest server“. To znamená, že záznam 401 v souboru `/etc/nwserv.conf` by měl být nulový, ledaže máte důvod neodpovídat na požadavky „get nearest server“. Pokud jen chcete, aby fungoval příkaz `slist` a neodpovídal na každý požadavek „get nearest server“, zařaďte číslo interní sítě a uzlu do souboru `/etc/nwserv.stations` a nastavte záznam 401 v souboru `/etc/nwserv.conf` na hodnotu 2.

11. Funguje balík `ncpfs` s balíkem `mars_nwe`?

Martinův a Volkerův kód pomalu začínají konvergovat. Poslední verze balíku `mars_nwe` obsahuje volbu, která povoluje spolupráci s balíkem `ncpfs`. V souboru `config.h` balíku `mars_nwe` je třeba povolit volbu `WITH_NAME_SPACE_CALLS`.

12. Existuje nějaký volně šířitelný software pro DOS, který by fungoval s balíkem `mars_nwe`?

Vynalézavá otázka si žádá vynalézavou odpověď. Jsem rád, že jste ji položil, protože Martin Stover má balík, který distribuuje společně s balíkem `mars_nwe` a který nabízí volně šířitelnou dosovskou klientskou podporu pro server `mars_nwe`. Tento balík najdete na stejných místech jako server pod názvem `mars_dosutils-0.01.tgz`. Obsahuje zdrojový kód v jazyce C programů jako jsou `slist.exe`, `login.exe`, `map.exe` atd. Kód je přeložen pomocí překladače Borland(tm) C.

17 Poznámka o autorských právech

Dokument IPX-HOWTO, průvodce softwarovou podporou protokolu IPX pro Linux. Copyright (c) 1995 Terry Dawson.

Tento program je volně šířitelný. Můžete ho šířit a/nebo upravovat v souladu s 2. nebo pozdější verzí licence GNU General Public License publikovanou nadací Free Software Foundation.

Tento program je šířen s nadějí, že bude užitečný, ale BEZ JAKÝCHKOLI ZÁRUK. Dále bez záruk OBCHODOVATELNOSTI nebo ZPŮSOBILOSTI PRO KONKRÉTNÍ ÚČEL. Podrobnosti najdete v General Public License.

Společně s tímto programem byste měli obdržet kopii licence GNU General Public License. Pokud nikoli, pak si o ni napište na adresu:

Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

18 Různé a poděkování

David E. Storey **<dave@tamos.gmu.edu>** a Volker Lendecke **<lendecke@namu01.gwdg.de>** mi výrazně pomohli při získávání informací pro tento dokument. Gilbert Callaghan **<gilbert@pokey.inviso.com>,** David Higgins **<dave@infra.com>** a Chad Robinson **<chadr@brtgate.brttech.com>** přispěli informacemi o konfiguraci protokolu IPX/PPP. Bennie Venter **<bjv@Gil-galad.paradigm-sa.com>** přispěl některými užitečnými informacemi o typech rámců. Christopher Wall **<vergil@idir.net>** mi dal některé užitečné návrhy, jak vylepšit čitelnost a úpravu tohoto dokumentu. Axel Boldt **<boldt@math.ucsb.edu>** přispěl některými užitečnými nápady a jako zpětná vazba. Erik D. Olson **<eriko@wrq.com>** poskytl zpětnou vazbu a informace ohledně konfigurace protokolu PPP pro protokol IPX. Brian King **<root@brian.library.dal.ca>** přispěl dotazem do sekce FAQ (Často kladené dotazy).

„NetWare“ je registrovaná obchodní značka společnosti Novell Corporation `http://www.novell.com/`. „Caldera“ je registrovaná obchodní značka společnosti Caldera Corporation `http://www.caldera.com/`.

S pozdravem Terry Dawson, VK2KTJ **<terry@perf.no.itg.telstra.com.au>.**

Kapitola

7

NFS

Nicolai Langfeldt **janl@math.uio.no** verze 0.7, 3. listopadu 1997

Jak nastavit NFS klienty a servery.

1 Úvod

1.1 Autorská práva

1.2 Další záležitosti

Tento dokument nebude nikdy dokončen. Pošlete mi prosím zprávy o vašich problémech a úspěších, dokument tak mohu vylepšovat. Peníze, komentáře nebo otázky posílejte na **janl@math.uio.no.** Jestliže budete odesílat elektronickou zprávu, *zajistěte prosím*, aby byla návratová adresa funkční. Dostávám *spousty* dopisů a zjišťování vaší adresy může být dost pracné.

Jestliže chcete tento dokument překládat, dejte mi vědět, abych věděl, ve kterých jazycích jsem byl publikován).

Stížnosti a poděkování předejte Olafu Kirchovi, který mě k napsání dokumentu přivedl a dal mi pár dobrých návrhů :-).

Tento dokument pokrývá NFS ve verzích jádra 2.0. Ve verzích jádra 2.1 jsou značné rozdíly a změny.

1.3 Věnování

Dokument bych rád věnoval Anne Line Norheim Langfeldtové. I když jej pravděpodobně nikdy nebude číst, protože není takovým typem dívky, která by jej četla.

2 README.first

NFS (Network File System – síťový systém souborů) má tři důležité charakteristiky:

- Umožňuje na síti sdílení souborů.
- Většinou funguje dostatečně dobře.
- Otevírá rizika v zabezpečení, velmi dobře známá crackerům, kteří snadno získají přístup (čtení, zápis a smazání) ke všem vašim souborům.

Ke zmíněným záležitostem se ještě v dokumentu dostanu. Jestliže přečtete část dokumentu, věnující se zabezpečení, budete méně zranitelní vůči některým rizikům zabezpečení. Pasáže, věnující se zabezpečení jsou v některých místech dosti odborné a vyžadují určité znalosti o IP a použitých termínech. Jestliže se v termínech nevyznáte, můžete se vrátit k dokumentu o sítích nebo si sežeňte knihu o správě TCP/IP-sítě. S její pomocí se s TCP/IP lépe sžijete. Mimochodem to je dobrý nápad, když provádíte správu unixových/linuxových serverů. Konkrétně bych doporučil *TCP/IP Network Administration* od Craiga Hunta, vydávanou O'Reilly & Associates, Inc. Když knihu přečtete a pochopíte, budete mít vyšší hodnotu na trhu práce, takže určitě neztratíte ;-).

Zde naleznete dvě části, určené k odstraňování obtíží s NFS – *Seznam problémů s programem mount* a *FAQ*. Jestliže vám něco nefunguje podle předpokladů, řešení hledejte nejprve zde.

3 Nastavení NFS-serveru

3.1 Předpoklady

Před dalším čtením dokumentu si musíte být jisti, že je možné provozovat telnet mezi stroji, které budete využívat jako server a klient. Jestliže to nefunguje, musíte korektně nastavit síť (dokument networking/NET-2).

3.2 První krok

Předtím, než můžeme provést cokoliv dalšího, musíme nastavit NFS server. Jestliže jste součástí nějaké úřední nebo univerzitní sítě, jistě zde již bude množství NFS serverů. Jestliže na ně máte přístup nebo jestli tento dokument čtete proto, abyste zjistili, jak přístup získat, můžete tuto část vynechat a přeskočit až na část 4.

Jestliže potřebujete server nastavit ne-linuxový, musíte si pročíst systémový manuál a zjistit, jak umožnit NFS-služby a export systémů souborů přes NFS. Jedna část tohoto dokumentu se této problematice věnuje pro mnoho různých systémů. Jakmile si vše urovnáte, můžete přikročit ke čtení další části. Nebo v této části ještě zůstaňte, protože některé věci, které zde odkrývám, jsou nezávislé na druhu stroje, použitého jako server.

Ti z vás, kteří tady čtou dál, budou muset nastavit množství programů.

3.3 Portmapper

Portmapperu se v Linuxu říká buď `portmap`, nebo `rpc.portmap`**.** Manuálová stránka na mém systému říká, že je to „DARPA port k mapovači čísel RPC-programů“. Je to první mezera v zabezpečení, se kterou se setkáte při čtení tohoto dokumentu. Popis zaplnění jedné z mezer naleznete v části 6. Tu vám opět vřele doporučuji k prostudování.

Spusťte `portmapper`. Jmenuje se buď `portmap`, nebo `rpc.portmap` a měl by bydlet v adresáři `/usr/sbin` (na některých strojích se nazývá `rpcbind`). Teď ho můžete spustit ručně, ale bude potřeba, aby se spustil při každém natažení systému, takže bude nutné vytvořit/editovat rc skripty. Vaše rc skripty jsou blíže popsány v manuálové stránce pro `init`. Většinou se nachází v `/etc/rc.d`, `/etc/init.d` nebo `/etc/rc.d/init.d`. Jestliže zde bude nějaký skript, který se nazývá nejspíše `inet`, je to ten, který budete editovat. Ale co napsat nebo provést je už záležitost, která leží mimo rozsah tohoto dokumentu. Spusťte `portmap` a spuštěním `ps aux` zkontrolujte, jestli žije. Žije? V pořádku.

3.4 Mountd a nfsd

Následující programy, které musíme spustit, jsou `mountd` a `nfsd`. Nejprve ale budeme editovat jiný soubor – `/etc/exports`. Řekněme, že chci, aby byl systém souborů `/mn/eris/local` (je na stroji **eris**) k dispozici pro stroj **apollon**. Potom do `/etc/exports` na stroji **eris** vložím následující:

```
/mn/eris/local apollon(rw)
```

Předchozí řádek dává stroji apollon na `/mn/eris/local` práva pro čtení a zápis. Místo **rw** by zde mohlo být **ro**, což znamená pouze čtení (jestliže zde nepoužijete nic, je to i implicitní hodnota, která se automaticky použije). Můžete zde použít ještě další volby, přičemž zabezpečovacími se budu ještě zabývat. Všechny volby jsou popsány na manuálových stránkách pro `exports`. Alespoň jednou v životě si je přečtěte. Existují i lepší způsoby, než je výpis všech serverů v souboru **exports**. Jestliže používáte NIS (nebo NYS, NIS byl znám jako YP), můžete využít síťové skupiny, určit zde domény pomocí zástupných znaků a IP-podsítě brát jako hostitele. Pak si ale uvědomte, kdo by tak mohl nechtěně získat přístup k serveru.

Poznámka: Tento soubor (`exports`) nemá stejnou syntaxi jako v ostatních Unixech. Souborům `exports` z ostatních Unixů zde v dokumentu věnujeme jednu samostatnou část.

Nyní můžeme spustit `mountd` (nebo se jedná o `rpc.mountd`) a `nfsd` (může být nazván `rpc.nfsd`). Oba budou číst soubor `exports`.

Jestliže budete editovat `/etc/exports`, musíte zajistit, aby `nfsd` a `mountd` věděly, že se soubory změnily. Většinou se to zajišťuje spuštěním `exportfs`. Mnoho distribucí Linuxu program `exportfs` neobsahuje. Jestliže je tomu tak i u vás, instalujte na vašem stroji následující skript:

```
#!/bin/sh
killall -HUP /usr/sbin/rpc.mountd
killall -HUP /usr/sbin/rpc.nfsd
echo re-exported file systems
```

Uložte jej například jako `/usr/sbin/exportfs` a nezapomeňte u něj provést `chmod a+rx`. Nyní po změně vašeho souboru `exports` jako root spustíte `exportfs`.

Nyní musíte zkontrolovat, jestli `mountd` a `nfs` skutečně fungují. Nejprve použijete `rpcinfo -p`. Mělo by se objevit zhruba následující:

```
program vers proto port
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
100005 1 udp 745 mountd
```

```
100005 1 tcp 747 mountd
100003 2 udp 2049 nfs
100003 2 tcp 2049 nfs
```

Jak je vidět, portmapper oznámil svoje služby, a totéž učinily i `mountd` a `nfsd`.

Jestliže obdržíte `rpcinfo: can't contact portmapper: RPC: Remote system error - Connection refused` nebo něco podobného, je jasné, že portmapper nefunguje. Napravte to. Jestliže dostanete `No remote programs registered`, pak buď s vámi `portmapper` nechce mluvit, nebo se něco pokazilo. Použijte na `nfsd`, `mountd` a `portmapper` příkaz `kill` a pokuste se o všechno znovu od začátku.

Po kontrole služeb `portmapperu` můžete také provést kontrolu pomocí `ps`. `Portmapper` bude hlásit svoje služby i v případě, že programy, které je využívaly, se zhroutí. Takže pokud se něco pokazí, kontrola pomocí `ps` bude na místě.

Samozřejmě potřebujete také upravit vaše systémové rc soubory, aby se `mountd` a `nfsd`, stejně jako `portmapper` spustily při startu systému. Většinou už se potřebné skripty na vašem stroji nachází, stačí v nich pouze zrušit komentář na patřičných místech nebo je aktivovat pro příslušné úrovně inicializace.

Nyní byste měli znát manuálové stránky pro: `portmap`, `mount`, `nfsd` a `exports`.

Jestliže jste všechno provedli, jak jsem navrhoval, můžete spustit NFS-klienta.

4 Nastavení NFS-klienta

Nejprve budete potřebovat jádro, kde bude NFS-systém zabudován nebo bude k dispozici jako modul. To se konfiguruje před kompilací jádra. Jestliže jste jádro ještě nekompilovali, pročtěte si dokument, týkající se jádra, a pokuste se o to. Jestliže používáte nějakou velmi dobrou distribuci (jako je Red Hat) a ještě jste se jádrem nebo moduly nezabývali (takže jste nic nezkazili :-), budete mít NFS k dispozici automaticky.

Nyní můžete jako root zadat příslušný příkaz `mount` a systém souborů se objeví. Když navážeme na příklad z předchozí části, budeme `mount` používat pro připojení `/mn/eris/local` ze stroje **eris**. To provedete následovně:

```
mount -o rsize=1024,wsize=1024 eris:/mn/eris/local /mnt
```

K volbám pro `rsize` a `wsize` se ještě dostaneme. Systém souborů je nyní k dispozici pod `/mnt`, takže tady můžete použít příkaz `cd` a `ls` a můžete se zde podívat na jednotlivé soubo-

ry. Zjistíte, že je to pomalejší než lokální systém souborů, ale naopak zase pohodlnější než FTP. Jestliže se po použití `mount` místo připojení systému objeví chybové hlášení, například: „`eris:/mn/eris/local failed, reason given by server`“**: Permission denied**, pak je špatný soubor exports nebo jste po jeho editaci zapomněli spustit `exportfs`. Jestliže se objeví „`mount clntudp_create: RPC: Program not registered`“, znamená to, že na serveru neběží `nfsd` nebo `mountd`.

Systému souborů se zbavíte takto:

```
umount /mnt
```

Aby se systém souborů nfs připojil při startu systému, musíte editovat `/etc/fstab`. Pro náš příklad stačí připojit takovýto řádek:

```
# device mountpoint fs-type options  dump fsckorder
...
eris:/mn/eris/local /mnt nfs rsize=1024,wsize=1024 0 0
...
```

K tomu vše, pokračujeme dál.

4.1 Volby pro mount

Ke zvážení zde máte několik možností. Jejich využitím je možné ovládat způsob, jakým se NFS-klient postaví k havárii serveru nebo poruše sítě. Jednou z dobrých vlastností NFS je to, že se takové situace zvládají s přehledem. Pokud je klient nastaven správně. Existují zde dva různé módy poruch:

`soft` NFS-klient nahlásí chybu procesu, který využívá soubor na přimontovaném systému souborů NFS. Některé programy se s takovým hlášením správně vyrovnají, některé ne. Takové nastavení nemohu doporučit.

`hard` Program, který přistupuje k souboru na přimontovaném systému souborů NFS, se při havárii serveru zastaví. Proces není možné přerušit nebo zrušit, pokud neurčíte také `intr`**.** Když je NFS-server opět funkční, program bude pokračovat tam, kde přestal. To je pravděpodobně i pro vás to pravé. Na všech připojených NFS-systémech souborů doporučuji použít `hard`**,** `intr`.

V návaznosti na předchozí příklad vypadají údaje v `fstab` takto:

```
# device mountpoint fs-type options  dump fsckorder
...
eris:/mn/eris/local /mnt nfs rsize=1024,wsize=1024,hard,intr 0 0
...
```

4.2 Optimalizace NFS

Normálně (když nejsou určeny volby `rsize` a `wsize`) bude NFS číst a zapisovat po 4 096 nebo 8 192 bajtech. Některé kombinace linuxových jader a síťových karet tak velké bloky nezvládají a navíc nemusí být optimální. Takže budeme muset experimentovat a nalézt nastavení `rsize` a `wsize`, která jsou funkční a co nejrychlejší. Rychlost vašeho nastavení můžete otestovat několika jednoduchými příkazy. Po použití předchozího příkazu `mount` a za předpokladu, že máte práva pro zápis na disk, můžete takto otestovat výkon při postupném zápisu:

```
time dd if=/dev/zero of=/mnt/testfile bs=16k count=4096
```

Tím vytvoříte 64 MB soubor nulových bajtů (je natolik velký, že by zde na výkon neměla mít vliv vyrovnávací paměť, jestliže máte velkou paměť použijte větší soubor). Operaci proveďte vícekrát (5-10) a spočítejte průměrný čas. To je doba trvání zápisu, která nás zajímá. Pak můžete zpětným čtením souboru otestovat výkony při čtení:

```
time dd if=/mnt/testfile of=/dev/null bs=16k
```

Opět proveďte vícekrát a spočítejte průměr. Proveďte odpojení a připojení (`mount` a `umount`), ale s většími hodnotami `rsize` a `wsize`. Mělo by se jednat o násobky 1 024 a neměly by být větší než 16 384 bajtů, což je maximální velikost u NFS verze 2. Ihned po připojení provádějte příkaz cd do připojeného systému a zde provádějte ls a trochu prozkoumejte fs, aby bylo zřejmé, že je vše v pořádku. Jestliže je nastavení `rsize`/`wsize` příliš vysoké, příznaky jsou *velice* podivné a někdy ne zcela zřejmé. Typický je nekompletní seznam souborů po zadání „`ls`“ a dále absence chybových hlášení. Nebo někdy bez chybových hlášení selže čtení souboru. Jakmile jste si jisti, že zadané hodnoty `rsize`/`wsize` jsou funkční, můžete opět přistoupit k testům rychlosti. Různé platformy serverů mají různé optimální hodnoty. SunOS a Solaris je například mnohem rychlejší při použití 4 096 bajtových bloků.

Nová jádra Linuxu (od 1.3) provádějí při nastavení `rsize` větším nebo rovném velikosti stránky procesoru čtení dopředu. Na procesorech Intel je velikost stránky 4 096 bajtů. Takové čtení zde *značně* zvýší čtecí výkony NFS. Takže na strojích Intel je dobré mít `rsize` alespoň 4 096.

Nezapomeňte nalezenou hodnotu rsize/wsize odrazit v `/etc/fstab`.

Výkony NFS při zápisu je možné zvýšit trikem – zrušením synchronních zápisů na serveru. Specifikace NFS určuje, že žádosti NFS o zápis by neměly být považovány za vyřízené, dokud jsou zapsaná data na stálém médiu (obyčejně disk). Tím se určitým způsobem sníží výkon při zápisu, asynchronní zápisy budou NFS-zápisy urychlovat. Linux nfsd nikdy neprovádí synchronní zápisy, protože implementace systému souborů v Linuxu se k tomu nepropůjčí, ale na ne-linuxových serverech můžete výkony zvýšit vložením následujícího řádku (nebo něčeho podobného) do vašeho souboru exports:

```
/dir -async,access=linuxbox
```

Podívejte se prosím u daného stroje na manuálovou stránku pro `exports`. Uvědomte si také, že se zvyšuje riziko ztráty dat.

5 NFS na pomalých linkách

Pomalé linky zahrnují modemy, ISDN a také další připojení na velké vzdálenosti.

Tato část je postavena na znalostech o použitých protokolech, ale zkušenosti z experimentování chybí. Můj domácí počítač byl 6 měsíců mimo provoz (nefunkční HDD), takže jsem nemohl testovat spojení přes modem. Jestli se o to pokusíte, dejte mi prosím vědět.

První věc, kterou musíme brát v úvahu, je to, že NFS je pomalý protokol. Má velké nároky. Použití NFS je skoro jako použití kermitu na přenos souborů. Je to *pomalé*. Skoro všechno je rychlejší než NFS. FTP je rychlejší. HTTP je rychlejší. Rcp je rychlejší. Ssh je rychlejší.

Pořád to chcete zkoušet? Dobrá.

Implicitní hodnoty NFS jsou určeny pro rychlé linky s malými prodlevami. Jestliže tyto implicitní hodnoty použijete na linkách s velkými prodlevami, NFS může hlásit chyby, ukončovat operace, předpokládat, že soubory jsou kratší než ve skutečnosti, nebo provádět jiné záhadné operace.

První věc, kterou musíte udělat, je *nepoužívat* pro příkaz `mount` volbu `soft`. Pak by prodlevy způsobily hlášení chyb aplikacím, které by většinou situaci správně nezvládly. Takto snadno vyvoláte různá záhadná selhání. Použijte tedy raději volbu `hard`. Když je aktivní **hard**, prodlevy místo ukončení způsobí další pokusy o pokračování, ať už se aplikace snaží o cokoliv. To je to, co potřebujete. Opravdu.

Nyní si musíte u příkazu `mount` pohrát s volbami `timeo` a `retrans`. Jsou popsány na manuálové stránce `nfs`(5), ale tady vám nabízím její kopii:

```
timeo=n The value in tenths of a second before
sending the first retransmission after an
RPC timeout. The default value is 7 tenths
of a second. After the first timeout, the
timeout is doubled after each successive
timeout until a maximum timeout of 60 sec-
onds is reached or the enough retransmis-
sions have occured to cause a major time-
out. Then, if the filesystem is hard
mounted, each new timeout cascade restarts
at twice the initial value of the previous
cascade, again doubling at each retransmis-
sion. The maximum timeout is always 60
seconds. Better overall performance may be
achieved by increasing the timeout when
mounting on a busy network, to a slow
server, or through several routers or gate-
ways.

retrans=n The number of minor timeouts and retrans-
missions that must occur before a major
timeout occurs. The default is 3 timeouts.
When a major timeout occurs, the file oper-
ation is either aborted or a "server not
responding" message is printed on the con-
sole.
```

Jinými slovy: Jestliže ve lhůtě 0,7 sekundy (700 ms) neobdržíte odpověď, NFS-klient bude žádost opakovat a lhůtu zdvojnásobí na 1,4 sekundy. Jestliže se odpověď opět neobjeví, žádost se opět opakuje a lhůta se opět zdvojnásobí (na 2,8 sekundy).

Rychlost linek je možné měřit pomocí `ping` se stejnou velikostí paketu, jakou mají nastaveny volby `rsize`/`wsize`.

```
$ ping -s 8192 lugulbanda
PING lugulbanda.uio.no (129.240.222.99): 8192 data bytes
8200 bytes from 129.240.222.99: icmp_seq=0 ttl=64 time=15.2 ms
8200 bytes from 129.240.222.99: icmp_seq=1 ttl=64 time=15.9 ms
8200 bytes from 129.240.222.99: icmp_seq=2 ttl=64 time=14.9 ms
```

```
8200 bytes from 129.240.222.99: icmp_seq=3 ttl=64 time=14.9 ms
8200 bytes from 129.240.222.99: icmp_seq=4 ttl=64 time=15.0 ms

--- lugulbanda.uio.no ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 14.9/15.1/15.9 ms
```

Čas (time) zde znamená, jak dlouho se paket ping dostává do lugulbandy a zpět. 15 ms je dost rychlých. Přes linku s kapacitou 28 000 bps můžete očekávat tak 4 000 – 5 000 ms a jestli je linka přetížená, může se čas prodloužit, snadno i zdvojnásobit. Když je čas dost vysoký, říkáme, že je velká prodleva (high latency). Obecně se při zvětšování paketů a vyšším zatížení sítě zvyšuje i prodleva. Timeo zvyšte podle vaší linky a jejího zatížení. A protože se prodleva zvyšuje, když linku využíváte pro další věci: Když budete chtít použít FTP a NFS současně, měli byste se pokusit změřit časy pro ping při využití FTP pro přenos souborů.

6 Zabezpečení a NFS

Nejsem žádný odborník na zabezpečení počítačů. Ale pro otázky zabezpečení mám pár *malých* rad. Ale nezapomeňte, že tohle není kompletní seznam problémů kolem NFS a jestli si myslíte, že po přečtení a provedení tohoto všeho jste v bezpečí, určitě se mýlíte.

Tahle část se vás nemusí týkat – a sice v případě, že je vaše síť *uzavřená*, vy věříte všem uživatelům a nikdo jiný (komu nevěříte) se na síť nemůže dostat. Znamená to, že se na síť nikdo jiný nemůže přihlásit ani přes modem a ani přes jinou síť, ve které nevěříte všem. Zní to paranoidně? Ne tak zcela. Je to jen *základní* rada pro zabezpečení. A uvědomte si, že to, co tady říkám, je jen začátek všeho. *Bezpečná* síť vyžaduje pilného a dobrého správce, který ví, kde najít informace o aktuálních a možných problémech.

NFS má základní problém v tom, že klient (nebylo-li řečeno jinak) bude věřit NFS-serveru a obráceně. To může být špatné. Znamená to, že pokud je narušen rootovský účet na serveru, není už takový problém narušit i rootovské účty na klientech. A opačně. K tomu existuje množství kopírovacích strategií, ke kterým se brzy dostaneme.

Něco, co byste si měli přečíst, jsou vývěsky CERT na NFS, většina z níže uvedeného textu se zabývá záležitostmi, o kterých CERT píše ve vývěskách. Zde následují některé vývěsky, vztahující se k NFS:

```
CA-91:21.SunOS.NFS.Jumbo.and.fsirand 12/06/91
Vulnerabilities concerning Sun Microsystems, Inc. (Sun) Network
File System (NFS) and the fsirand program. These vulnerabilities
```

```
affect SunOS versions 4.1.1, 4.1, and 4.0.3 on all architectures.
Patches are available for SunOS 4.1.1. An initial patch for SunOS
4.1 NFS is also available. Sun will be providing complete patches
for SunOS 4.1 and SunOS 4.0.3 at a later date.

CA-94:15.NFS.Vulnerabilities  12/19/94
This advisory describes security measures to guard against several
vulnerabilities in the Network File System (NFS). The advisory was
prompted by an increase in root compromises by intruders using tools
to exploit the vulnerabilities.

CA-96.08.pcnfsd  04/18/96
This advisory describes a vulnerability in the pcnfsd program (also
known as rpc.pcnfsd). A patch is included.
```

6.1 Zabezpečení klienta

U klienta se můžeme rozhodnout, že serveru nebudeme tolik věřit, několika způsoby a možnostmi pro mount. Můžeme například pomocí volby `nosuid` zakázat práci programů suid mimo NFS-systém. Tohle je dobrý nápad a stojí za to zvážit, zda jej nevyužít na všech připojených (mount) discích. Znamená to, že root na serveru nemůže v systému souborů vytvořit rootovský suid-program, přihlásit se na klienta jako normální uživatel a využít rootovský suid-program a stát se tak rootem i na klientu. Pomocí volby `noexec` je také možné na připojeném systému souborů zakázat spouštění souborů. Tohle je ale oproti `nosuid` méně praktické, protože systém souborů může obsahovat *nějaké* skripty nebo programy, které musí být spouštěny. Tyto volby zadáte ve sloupci pro volby s `rsize` a `wsize`, oddělené čárkami.

6.2 Zabezpečení serveru: nfsd

Na serveru se můžeme rozhodnout, že nebudeme věřit rootovskému účtu klienta. Dosáhneme toho využitím volby `root_squash` v `exports`:

```
/mn/eris/local apollon(rw,root_squash)
```

Nyní pokud se uživatel s UID 0 na klientovi pokusí přistoupit (číst, zapisovat, mazat) na systém souborů, server jeho UID nahradí UID účtu „nobody“ ze serveru. To znamená, že root z klienta nemůže pracovat se soubory, se kterými může pracovat jedině root na serveru. To je dobré, takže `root_squash` pravděpodobně použijete na všech exportovaných systémech souborů. Můžete ale namítnout, že root z klienta použije „su“, stane se tak jiným uživatelem

a stejně tyto uživatelské soubory použije! Zde je odpověď jasná: Ano, je to tak a v Unixu na NFS je to tak správně. Z toho plyne jedno ponaučení – všechny důležité soubory a binární programy by měl vlastnit **root**, ne tedy **bin** nebo jiný nerootovský účet, protože jediným účtem, na který se root z klienta nedostane, je rootovský účet na serveru. Na manuálových stránkách pro NFSd je vypsáno několik dalších voleb pro squash, takže se můžete rozhodnout, že někomu z klientů budete méně věřit. Pomocí squash je možné nastavit i libovolný rozsah UID a GID. Popis naleznete na manuálové stránce pro Linux NFSd.

V Linux NFSd je `root_squash` vlastně nastaveno implicitně. Pokud má mít root k systému souborů přístup, použijte

```
no_root_squash
```

Další důležitou věcí je zajistit, aby `nfsd` kontroloval, že všechny jeho žádosti přichází z privilegovaného portu. Jestliže přijme žádosti z jakéhokoliv starého portu na klientu, uživatel může bez zvláštních práv spustit program, který je možné snadno získat na Internetu. Hovoří protokolem nfs a určí, že uživatel je kdokoliv, kým chce být. No, trochu tady straším. Linux `nfsd` provádí kontrolu automaticky, v jiných operačních systémech ji musíte nastavit sami. Popis nastavení naleznete v manuálových stránkách pro nfsd v daném operačním systému.

Další věc. Nikdy neexportujte systém souborů na „localhost“ nebo na 127.0.0.1. Důvěřujte mi.

6.3 Zabezpečení serveru: portmapper

Základní `portmapper` má v kombinaci s `nfsd` problém, který umožňuje dostat se na soubory na NFS-serverech bez jakýchkoliv práv. Naštěstí je `portmapper`, využívaný v Linuxu, proti tomuto zneužití relativně bezpečný a může být ještě bezpečnější, pokud ve dvou souborech nakonfigurujete seznamy pro přístup.

Nejprve editujme `/etc/hosts.deny`. Měl by obsahovat řádek

```
portmap: ALL
```

Tento řádek zakáže přístup *všem*. Je to možná trochu drastické, takže přístup opět otevřeme editací `/etc/hosts.allow`. Nejprve si ale musíme rozmyslet, komu přístup otevřeme. Mělo by se jednat o všechny stroje, které by měly mít přístup k vašemu `portmapperu`. V běžícím linuxovém systému je jen málo strojů, které by z nějakého důvodu potřebovaly přístup. `Portmapper` se stará o `nfsd`, `mountd`, `ypbind/ypserv`, `pcnfsd` a „r“ služby, jako `ruptime` a `rusers`. Zde stojí za zvážení jedině `nfsd`, `mountd`, `ypbind/ypserv` a snad `pcnfsd`. Všechny stroje, které potřebují využívat služby na vašem stroji, by to měly mít také umožněno. Řekněme například, že adresa vašeho stroje je 129.240.223.254 a stroje, nalézající se v podsíti 129.240.223.0, by k němu měly mít přístup (použité termíny jsou vysvětleny v dokumentu k sítím). Potom v `hosts.allow` napíšete

```
portmap: 129.240.223.0/255.255.255.0
```

To je stejný případ jako síťová adresa, daná pro směrování, a podsíťová maska, použitá v `ifconfig`. Pro zařízení `eth0` by měl `ifconfig` na tomto počítači ukazovat

```
...
eth0 Link encap:10Mbps Ethernet HWaddr 00:60:8C:96:D5:56
inet addr:129.240.223.254 Bcast:129.240.223.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:360315 errors:0 dropped:0 overruns:0
TX packets:179274 errors:0 dropped:0 overruns:0
Interrupt:10 Base address:0x320
...
```

A `netstat -rn` by měl ukazovat

```
Kernel routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
...
129.240.223.0 0.0.0.0 255.255.255.0 U 0 0 174412 eth0
...
```

Síťová adresa je v prvním sloupci.

Soubory `hosts.deny` a `hosts.allow` jsou popsány na stejnojmenných manuálových stránkách.

DŮLEŽITÉ: V těchto souborech *nedávejte* na řádky `portmap` *nic jiného* než *IP-ČÍSLA*. Vyhledávání podle názvů lokací může nepřímo způsobit aktivitu `portmap`, která způsobí vyhledávání podle názvů lokací, které může nepřímo způsobit aktivitu `portmap`, která způsobí...

Předchozí věci posilují váš server. Jediným problémem, který zbývá (opravdu!), je něčí narušení roota (nebo natažení systému MS-DOS) na stroji, kterému věříme. Se získanými právy může dojít k odeslání žádostí ze zabezpečeného portu pod libovolným uživatelským jménem.

6.4 NFS a firewall

Dobrý nápad je použít na NFS firewall a portmapovat porty na vašem routeru a firewallu. `Nfsd` pracuje na portu 2 049 s oběma protokoly udp i tcp. `Portmapper` je na portu 111 (tcp i udp) a `mountd` je na portu 745 a 747 (tcp a udp). Obvykle. Porty raději zkontrolujte příkazem `rpcinfo -p`.

Jestliže ale na druhou stranu chcete, aby NFS procházel přes firewall, existují volby pro novější `NFSd` a `mountd`, aby používaly určitý (nestandardní) port, který může být ve firewallu otevřen.

6.5 Shrnutí

Jestliže u aplikací `portmapper/nfs` používáte `hosts.allow/deny`, `root_squash`, `nosuid` a funkce privilegovaných portů, vyhnete se mnoha známým nedostatkům v NFS a můžete se cítit téměř bezpečně.

Ale kdyby přece jenom: Když má na vaši síť přístup nějaký vetřelec a jsou-li `/home` a `/var/spool/mail` připojeny přes NFS, může vložit do vašich souborů `.forward` nebo do mailboxu nějaké podivné příkazy. Z tohoto důvodu nikdy nepřistupujte k vašemu soukromému klíči PGP přes NFS. Nebo alespoň počítejte s rizikem. A nyní už něco znáte.

NFS a `portmapper` tvoří dohromady složitý podsystém, takže se někdy může stát, že jsou objeveny nové chyby – ať už v základním návrhu nebo v použité implementaci. Vyskytují se i známé chyby, které někdo zneužívá. Ale takový je život. Tyto záležitosti můžete sledovat alespoň v newsgroups *comp.os.linux.announce* a *comp.security.announce*.

7 Seznam problémů s programem mount

Tato část je odvozena podle seznamu, vytvořeného v IBM Corporation, který se zabývá problémy při připojení NFS. Zde jim musím poděkovat za zpřístupnění pro účely tohoto dokumentu. Jestliže se při připojení NFS setkáte s problémem, proberte nejprve následující seznam. Teprve v případě neúspěchu svůj problém někam posílejte. Každá položka popisuje nějaký problém a nabízí jeho odstranění.

1. Systém souborů není exportován nebo není exportován danému klientovi.

   Náprava: exportujte jej.

2. Rozlišení názvu neodpovídá exportnímu seznamu.

   V exportním seznamu je například export na **johmad**, ale tento název je rozlišen jako **johmad.austin.ibm.com** a připojení není povoleno.

   Náprava: exportujte na obě formy názvu.

   Situace může nastat také v případě, že klient má 2 rozhraní s různými názvy pro oba adaptéry a v exportu je určen jen jeden.

   Náprava: exportujte obě rozhraní.

Situace může nastat také v případě, že server nemůže na klientu provést lookuphostbyname nebo lookuphostbyaddr (jsou to funkce knihoven). Zajistěte, aby klient mohl provést **host <name>; host <ip_addr>;** a aby oba příkazy ukazovaly stejný stroj.

Náprava: srovnejte rozlišení názvu.

3. Systém souborů byl připojen po spuštění NFS (na tom serveru). V takovém případě server exportuje vyznačené místo připojení, ne připojený systém souborů.

   Náprava: Ukončete `NFSd` a znovu jej spusťte.

   Poznámka: Klienti, na kterých je připojeno vyznačené místo připojení, budou mít po restartu problémy s přístupem.

4. Datum je na jednom nebo obou strojích úplně mimo (mohou vzniknout zmatky).

   Náprava: nastavte datum správně.

   Autor projektu HOWTO doporučuje k synchronizaci hodin použití NTP. Protože na NTP existují v USA exportní omezení, pro `Debian`, `RedHat` nebo `Slackware` jej získáte na adrese `ftp://ftp.hacktic.nl/pub/replay/pub/linux` nebo na mirrorech.

5. Server nedokáže přijmout připojení od uživatele, který je členem více než 8 skupin.

   Náprava: snižte počet skupin, ve kterých je uživatel členem, nebo proveďte připojení přes jiného uživatele.

8 FAQ

Tato část se věnuje FAQ, tedy často pokládaným otázkám. Většinu sepsal Alan Cox.

1. **Jestliže používám Linux jako NFS-server, často se mi objevují chyby „`stale nfs handle`“.**

   Objevuje se jako chyba v některých starších verzích `nfsd`. Od verze `nfs-server2.2beta16` je chyba odstraněna.

2. **Když se pokusím připojit systém souborů, dostávám**

   ```
   can't register with portmap: system error on send
   ```

   Pravděpodobně používáte systém Caldera. Je zde chyba v rc skriptech. Nápravu si zjednejte od Caldery.

3. **Proč není možné spustit soubor po jeho okopírování na NFS-server?**

   Důvod je v tom, že `nfsd` z důvodu zvýšení výkonů používá při práci se soubory vyrovnávací paměť (uvědomte si, že běží v uživatelském prostoru). `Nfsd` sice soubor otevře (jako v případě po zápisu do něj), ale jádro vám jej neumožní spustit. `Nfsd`, která jsou novější než z jara 95, otevírají soubory v několika sekundách, těm starším to může trvat i dny.

4. **Všechny moje NFS-soubory jsou pouze pro čtení.**

   Linuxový NFS-server je implicitně pouze pro čtení. Více podrobností viz manuálové stránky pro „`exports`“ a `nfsd`. Budete muset upravit `/etc/exports`.

5. **Připojuji se z linuxového NFS-serveru a `ls` mi funguje, ale číst a zapisovat soubory nemůžu.**

   Ve starších verzích Linuxu musíte NFS-server připojit s hodnotami `rsize=1024,wsize=1024`.

6. **Připojuji se z linuxového NFS-serveru s velikostí bloku mezi 3 500 – 4 000 a pravidelně se zhroutí linuxový box.**

   Tak to nedělejte.

7. **Dokáže Linux zvládnout NFS přes TCP?**

   Ne, zatím ne.

8. **Když se pokouším připojit stroj z linuxového boxu, dostanu spoustu podivných chyb.**

   Uživatelé musí být v 8 a méně skupinách. Starší servery to vyžadují.

9. **Když provedu restart svého stroje, někdy se při pokusu o odpojení NFS-serveru zasekne.**

   Při restartu **neprovádějte** odpojení NFS-serverů, ignorujte je, protože se stejně nic nestane. Příkaz pak zní `umount -avt nonfs`.

10. **NFS-klienty Linuxu jsou při zápisu do systémů Sun a BSD velice pomalé.**

    NFS-zápisy jsou normálně synchronní (můžete to vypnout, pokud nechcete riskovat ztrátu dat). Horší je, že od BSD odvozená jádra zatím nebývají schopna pracovat v malých blocích. Proto když zapisujete 4 KB dat z linuxového boxu v 1 KB paketech, které používá, BSD provede tohle

```
čtení 4KB stránky
změna 1KB
zápis 4KB zpět na disk
čtení 4KB stránky
```

```
změna 1KB
zápis 4KB zpět na disk
atd..
```

9 Exportování systémů souborů

Způsob exportování systémů souborů pomocí NFS není pochopitelně u všech platforem konzistentní. Linux a Solaris 2 jsou v tomto případě rozdílné. Tato část vypisuje, jak se to dělá na většině systémů. Jestliže zde není váš systém obsažen, musíte si k němu projít manuálové stránky. Klíčová slova jsou `nfsd`, system administration tool, rc scripts, boot scripts, boot sequence, `/etc/exports`, `exportfs`. V této části budu průběžně využívat jeden příklad: jak exportovat `/mn/eris/local` na apollon read/write.

9.1 IRIX, HP-UX, Digital-UNIX, Ultrix, SunOS 4 (Solaris 1), AIX

Tyto operační systémy využívají tradiční Sunovský exportní formát. Do `/etc/exports` zapište:

```
/mn/eris/local -rw=apollon
```

Kompletní dokumentace je na manuálové stránce `exports`. Po editaci souboru spusťte `exports -av` a systém souborů bude exportován.

Syntaxe příkazu `exportfs` se systém od systému může lišit. Na některých operačních systémech bude předchozí zadaný řádek znít:

```
/mn/eris/local apollon
```

nebo dokonce ještě takto:

```
/mn/eris/local rw=apollon
```

Doporučuji formálnost. Riskujete, že příští verze `exportfs` bude v syntaxi přísnější a nic vám nepoběží.

9.2 Solaris 2

Sun při tvorbě systému Solaris 2 znovu vynalezl kolo. Takže tento systém se od ostatních značně liší. Budete provádět editaci souboru `/etc/dfs/dfstab`. Zde uložíte podle manuálové stránky `share`(1M) příkazy pro sdílení. Asi takto:

```
share -o rw=apollon -d "Eris Local" /mn/eris/local
```

Po editaci spusťte program `shareall`, který systémy souborů exportuje.

10 PC-NFS

PC-NFS nespouštějte. Pusťte si sambu.

Pardon: O PC-NFS nevím nic. Jestli se na to někdo cítí, ať to udělá a já jeho práci zařadím na toto místo.

LINUX

České sdružení uživatelů
operačního systému Linux (CZLUG)

CZLUG – Czech Linux User Group

České sdružení uživatelů operačního systému Linux (CZLUG) je nezisková organizace, která spojuje lidi se společným zájmem - zájmem o operační systém Linux. Odkaz na základní informace o tomto sdružení lze nalézt na internetové stránce, která je „výchozím rozcestníkem“ informací o Linuxu v České republice: ***http://www.linux.cz***.

Linux je unixový operační systém, který se v posledních letech velmi dynamicky rozvíjí. Od prvních pokusů studenta informatiky Linuse Torvaldse v létě roku 1991, které se jen s velkou dávkou obrazotvornosti daly nazvat pokusy na novém operačním systému, uplynula relativně krátká doba. Svou prací chtěl Linus asi setřít propastný rozdíl mezi drahými unixovými pracovními stanicemi, se kterými se setkával ve škole, a levnými PC. Dnes můžeme s jistotou říci, že se mu povedlo splnit vytčený cíl. Linux je mladý operační systém, postavený na starých, léty ověřených unixových principech. Silně konkuruje komerčně šířeným Unixům a můžeme jej najít na velkém množství počítačových architektur. Procesory Intel (ona levná PC) jsou jen jednou z mnoha hardwarových platforem, kde se Linux provozuje. Geometrickou řadou roste počet aplikací pro Linux i počet jeho uživatelů. Na celém světě vznikají sdružení příznivců tohoto systému a CZLUG je jedním z takových sdružení.

CZLUG nestačil ještě oslavit ani své první narozeniny (ustavující shromáždění proběhlo v lednu 1998), nicméně uživatelé Linuxu už zaznamenali plno akcí, které jsou výsledkem nezištné práce aktivistů CZLUG. Každý měsíc vycházejí pod záštitou CZLUG Linuxové noviny, které si účastník Internetu může zkopírovat například z ***ftp://ftp.fi.muni.cz/pub/linux/local/noviny***. Jsou k dispozici formáty PostScript, PDF i HTML. V diskusní skupině ***linux@muni.cz*** je poměrně čilý provoz. Kromě otázek a odpovědí na technické problémy tam najdeme i lehce laděné příspěvky, týkající se například maskota Linuxu - tučňáka se žlutýma nohama. Koncem března zorganizovali lidé z CZLUG dvoudenní „Linuxový seminář“ ve školicím středisku Masarykovy university Cikháj. Byl o něj takový zájem, že kapacita střediska byla vyprodána velmi dlouho dopředu. Na tomto semináři si účastníci vyslechli plno zajímavých přednášek od odborníků z nejrůznějších počítačových specializací. Na místě byla též prodávána CD s aktuální distribucí RedHat Linux včetně úprav pro český jazyk. V polovině června proběhla na půdě Fakulty informatiky Masarykovy university akce s názvem „Linux - InstallFest“. Kdokoli si mohl vzít svůj počítač pod paží a přinést na fakultu. Tam pod odborným dohledem byl přístroj připojen k počítačové síti a nainstalován na něj operační systém Linux. V rámci této akce též proběhla přednáška o Linuxu. Počítačový měsíčník Softwarové noviny udělil Linuxu ocenění Produkt roku 1997 a u příležitosti slavnostního

udělování cen byli přizváni i zástupci CZLUG. Odborníci z řad CZLUG píší recenze k nově vznikajícím publikacím o Linuxu. Uvažuje se o sponzorování nějakého tučňáka v některé zoologické zahradě...

Je asi všeobecně známo, že Linux se šíří pod licencí GPL (General Public License), vydané Free Software Foundation (Nadací pro volně šířený software). Licence neklade mezi autory programů a uživatele jejich práce žádné překážky. Uživatel může získat programy například od zprostředkovatele jen za manipulační poplatek nebo cenu média, tedy neplatí licenční poplatky. Není přitom omezován ve způsobu použití díla - může jej dále modifikovat, dále šířit a není mu zabráněn přístup ke zdrojovým textům díla. Firmy zabývající se distribucí software mohou bez omezení poskytovat zákazníkům placené služby, které souvisejí s údržbou, školením a dalšími běžnými komerčními aktivitami spojenými s existencí softwarového díla šířeného podle GPL.

Domnívám se, že z výše uvedeného důvodu nemá dobrovolné sdružení uživatelů Linuxu ve svém poslání jen podporování určitého operačního systému, ale též šíření myšlenky ideálního vztahu mezi autorem softwarového díla a uživatelem, jako je tomu v případě programů s licencí GPL. Linux je jen jedním příkladem takového programu. Stovky, možná tisíce dalších aplikací (zvláště pro Linux) se šíří podobným způsobem. Proto CZLUG deklaroval už při svém vzniku úzkou spolupráci s Českou nadací pro podporu free software, která je českou alternativou americké Free Software Foundation.

Členem CZLUG se může stát kdokoli zaplacením členského příspěvku na kalendářní rok. Členský příspěvek pro rok 1998 je 220 Kč nebo 150 Kč pro studenty. Touto více méně symbolickou částkou dáváte najevo svou podporu s propagací Linuxu. Údaje o členech sdružení jsou zaneseny do databáze. Členové dostávají přednostně pozvánky na akce CZLUG, případně mohou dostat se slevou CD či jiný materiál vytvořený za podpory CZLUG. Protože CZLUG je registrovanou neziskovou organizací, má své stanovy, podle nichž členové na valných shromážděních volí výbor a mohou určit další směry činnosti CZLUG a propagace Linuxu. Sídlem CZLUG je adresa Fakulty informatiky MU, Botanická 68a, 602 00 Brno. Další informace včetně stanov naleznete na adrese ***http://www.linux.cz/czlug.***

Tuto knihu, Linux – dokumentační projekt, vydalo nakladatelství Computer Press ve spolupráci s Českým sdružením uživatelů operačního systému Linux, které ji doporučuje svým členům jako základní studijní a referenční materiál.

7. 9. 1998 Petr Olšák