LINUX

РЪКОВОДСТВО
НА
МРЕЖОВИЯ АДМИНИСТРАТОР

ОЛАФ КИРХ, ТЕРИ ДОУСЪН

O'REILLY®

СОФТПРЕС
ИЗДАТЕЛСТВО

Linux Network Administrator's Guide, Second Edition

by Olaf Kirch and Terry Dawson

Linux – ръководство на мрежовия администратор

от Олаф Кирх и Тери Доусън

Изданието на български език е публикувано от издателство СофтПрес ООД, 2001
ISBN 954-685-146-9

Издателски колектив:

***Редактори*:** Стефан Христов, Зоя Драговчева
***Предпечатна подготовка*:** Светослава Анева-Веселинова
***Художествен редактор*:** Владимир Владимиров

За контакти:
Адрес: София 1407, П.К. 114
тел.: (02) 958 25 80, 958 25 67; **факс:** (02) 58 62 04
e-mail: clients@soft-press.com; **Web site:** www.soft-press.com

За дистрибуция:

СофтПрес София – ул. "Искърско шосе" 19;
тел.: 02/ 973 15 06; **e-mail:** iliev@soft-press.com

СофтПрес Пловдив – бул. "Руски" 139, стая 104;
тел.: 032/ 62 75 62; **тел./факс:** 032/ 62 27 47; **e-mail:** plovdiv@soft-press.com

СофтПрес Бургас – пл. "Тройката" 4;
тел.: 056/ 80 02 31; **факс:** 056/ 80 31 39; **e-mail:** burgas@soft-press.com

СофтПрес Стара Загора – ул. "Цар Симеон Велики" 117;
тел.: 042/ 60 27 75; **e-mail:** stzagora@soft-press.com

СофтПрес Варна – бул. "Васил Левски" бл. 9, вх. А, ет. 2, ап. 11;
тел.: 052/ 30 42 69; **e-mail:** varna@soft-press.com

Съдържание

Глава 3

Глава 4

Глава 5

Глава 9

Глава 15

Глава 16

Глава 17

Глава 18

Приложение Б

Приложение В

Приложение Г

ПРЕДГОВОР

Днес Интернет е общоизвестен термин в много страни. След като дори сериозните хора започнаха да използват Информационната Супермагистрала за забавление, компютърните мрежи се приближиха по известност до телевизионните мрежи и микровълновите фурни. Интернет има необичайно голямо медийно покритие. Световноизвестни социолози се включват в дискусиите в групите по интереси на Usenet, в интерактивни среди за виртуална реалност и в Web, за да изследват новата "Интернет култура".

Разбира се, работата в мрежа е известна от много време. Свързването на компютри в локална мрежа или използвайки линиите за отдалечена връзка, осигурявани от телекомуникационните компании, е обичайна практика дори при малки инсталации. Бързо разрастващият се конгломерат от мрежи със световно покритие направи присъединяването към глобалното село една напълно разумна инвестиция дори за малки организации на частни компютърни потребители с идеална цел. Инсталирането на Интернет сървър, предоставящ електронна поща, връзка с Usenet и комутируем телефонен или ISDN достъп, стана достъпна възможност, а появата на технологии като DSL (Digital Subscriber Line – цифрова линия за абонати) и кабелния модем, без съмнение ще продължи тази тенденция.

Да говорим за компютърни мрежи, често означава да говорим за Unix. Разбира се, Unix не е единствената операционна система с мрежови възможности, нито пък ще остане на върха завинаги, но тя е отдавна в мрежовия бизнес и със сигурност ще продължи да бъде в него още дълго време.

Това, което прави Unix интересен за частните потребители, че се влагат много усилия в създаването на свободно достъпни Unix-подобни операционни системи за PC, например 386BSD, FreeBSD и Linux.

Linux е свободно разпространяван клон на Unix за персонални компютри. В момента той работи на разнообразни машини, използващи процесори на Intel, но също и на машини с процесори Motorola 680x0 като Comodore Amiga и Apple Macintosh; Sun SPARC и Ultra-SPARC станции; Alpha на Compaq; MIPS; чиповете PowerPC, например в новата генерация на Apple Macintosh и процесориге StrongArm в машините Netwinder на rebel.com и Palm на 3Com. Linux е адаптиран и за някои малко известни машини като Fujitsu AP-100 и System 390 на IBM. В момента в лабораториите на разработчиците се създават версии и за други интересни архитектури, а най-вероятно адаптирането на Linux за вградени контролери също ще завърши с успех.

Linux беше разработен от голяма група доброволци, комуникиращи чрез Интернет. Проектът беше започнат през 1990 г. от финландския студент Линус Торвалдс като курсов проект по операционни системи. Оттогава до днес Linux се разраства като лавина и се превърна в напълно функционален клон на Unix, способен да изпълнява разнообразни приложения като софтуер за симулиране и моделиране, текстообработващи пакети, системи за разпознаване на говор, web-браузъри и огромно количество друг софтуер, включително немалко превъзходни игри. Linux поддържа повечето съществуващ хардуер и предоставя пълни възможности за работа в TCP/IP мрежа, включително SLIP, PPP и защитни стени. Освен това, Linux предлага пълна реализация на IPX, както и много други възможности и протоколи, непознати в никоя друга операционна система. Linux е мощен, бърз и свободно достъпен и неговата популярност в света на Интернет нараства бързо.

Самата операционна система Linux е защитена от лиценза General Public License на GNU; същият лиценз, който се използва за софтуера, разработван от Free Software Foundation. Този лиценз позволява на всеки да разпространява или променя софтуера (без заплащане или срещу възнаграждение), ако всички изменения и дистрибуции също могат да се разпространяват свободно. Терминът “free software” означава, че софтуерът е свободен, а не просто безплатен.

Предназначение и аудитория на тази книга

Тази книга беше написана като цялостен справочник за мрежово администриране в Linux среда. Както начинаещите, така и напредналите потребители ще намерят информацията, от която се нуждаят за извършване на почти всички важни дейности по администрирането на базирана на Linux мрежова конфигурация. Разбира се, разнообразието на тези дейности е практически неограничено, поради което в една книга е невъзможно да се включи всичко, което може да се каже за мрежовото администриране. Опитали сме се да опишем само най-важните и основните дейности. Установихме, че начинаещите в мрежова работа под Linux, дори тези, които нямат предварителна представа за операционни системи от типа на Unix, смятат тази книга за достатъчно добра, за да им помогне успешно да поддържат тяхната базирана на Linux мрежа активна и работеща, като ги подготвя да научат и нещо повече.

Съществуват много книги и други източници на информация, от които можете да изучите всяка от темите, разгледани в тази книга, в по-голяма дълбочина (може би с изключение на някои от наистина специфичните за Linux особености, например новият интерфейс на защитната стена за Linux, който не е добре документиран другаде). Осигурили сме библиография, която можете да използвате, когато почувствате необходимост да научите повече.

Източници на информация

Ако сте нов в света на Linux, можете да използвате много източници за изучаване и усвояване на тази операционна система. Наличието на достъп до Интернет е полезно, но не е задължително.

Ръководства на LDP

Проектът за документиране на Linux (LDP - Linux Documentation Project) е група от доброволци, които създават книги, ръководства, HOWTO документи и наръчници по теми, започващи от инсталирането на Linux и стигащи до програмиране на ядрото. Книгите на LDP включват:

Linux Installation and Getting Started

от Мат Уелш и др. Тази книга описва как да намерим, инсталираме и използваме Linux. Тя включва встъпително ръководство за Unix и информация за системно администриране, системата X Window и работата в мрежа.

Linux System Administrators Guide

от Lars Wirzenius и Joanna Oja. Тази книга е общо ръководство за системно администриране на Linux и обхваща теми като създаване и конфигуриране на потребителски акаунти, създаване на резервни копия на системата, конфигуриране на основните софтуерни пакети и инсталиране и актуализиране на софтуер.

Linux System Administration Made Easy

от Steve Frampton. Тази книга описва дейностите по ежедневно администриране и поддръжка, които трябва да извършват потребителите на Linux.

Linux Programmers Guide

от B. Scott Burkett, Sven Goldt, John D. Harper, Sven van der Meer и Мат Уелш. В тази книга се разискват теми, които са от интерес за програмисти, желаещи да създават приложен софтуер за Linux.

The Linux Kernel

от David Rusling. Тази книга съдържа въведение към ядрото на Linux и описва как то е конструирано и работи. Хвърлете един поглед върху вашето ядро.

The Linux Kernel Module Programming Guide

от Ori Pomerantz. Това ръководство описва как се пишат модули за ядрото на Linux.

Разработват се и други справочници. Можете да намерите повече информация за LDP на WWW сървъра на проекта на адреса *http://www.linuxdoc.org* или на някой от многото негови огледални сървъри.

HOWTO документи

Документите *HOWTO* (букв. КАК ДА... – б.р.) за Linux представляват изчерпателни поредици от материали, детайлизиращи

различни аспекти на системата – например инсталиране и конфигуриране на софтуера, съставляващ системата X Window или как да пишем на асемблер програми за Linux. Тези документи обикновено се намират в поддиректория *HOWTO* на FTP сайтовете, които сме изброили по-долу, или са достъпни в Web на един от многото огледални сайтове на LDP. В библиографията в края на книгата и във файла *HOWTO-INDEX* можете да намерите списък на съществуващите *HOWTO* документи.

Препоръчваме ви да прочете *Installation HOWTO*, в който се описва как можете да инсталирате Linux на вашата система; *Hardware Compatibility HOWTO*, който съдържа списък с хардуера, за който е известно, че работи под Linux и *Distributions HOWTO*, в който са изброени софтуерните компании, които продават Linux на дискети или CD-ROM.

Библиографията на тази книга включва препратки към документите HOTWO, които са свързани с работата в мрежа под Linux.

Linux Frequently Asked Questions

Списъците FAQ (*Frequently Asked Questions with Answers* – често задавани въпроси и техните отговори) съдържат голям брой въпроси за системата и техните отговори. Тези документи са задължителни за всеки начинаещ потребител на Linux.

Документация, достъпна чрез FTP

Ако имате анонимен FTP достъп, можете да изтеглите цялата изброена по-горе документация за Linux от различни сайтове, в това число *metalab.unc.edu: /pub/Linux/docs* и *tsx-11.mit.edu:/pub/linux/docs*.

Тези сайтове имат огледални копия в целия свят.

Документация, достъпна чрез WWW

В Web съществуват много сайтове за Linux. Основният сайт на проекта LDP се намира на адрес *http://www.linuxdoc.org/*.

OSWG (Open Source Writes Guild – гилдия на авторите на отворен код) е проект с обхват, излизащ извън рамките на Linux. Целта на OSWG, подобно на тази книга, е да защитава и улеснява създаването на документация на софтуера с отворен код. Основният сайт на OSWG е на *http://www.oswg.org.8080/oswg*.

Всеки от тези сайтове съдържа хипертекстови и други версии на много свързани с Linux документи.

Комерсиална документация

Много от издателските компании и доставчици на софтуер публикуват книгите на LDP. Два такива доставчика са:

Specialized Systems Consultants, Inc. (SSC)
http://www.ssc.com/
P.O.Box 55549 Seattle, WA 98155-0549
1-206-782-7733
1-206-782-7191 (FAX)
sales@ssc.com

и:

Linux Systems Labs
http://www.lsl.com/
18300 Tara Drive
Clinton Township, MI 48036
1-810-987-8807
1-810-987-3562 (FAX)
sales@lsl.com

И двете компании продават съкратени варианти на документите HOWTO, както и друга отпечатана и подвързана документация за Linux.

O'Reilly & Associates публикува серия от книги за Linux. Тази книга е създадена от LDP, но повечето книги са написани специално за тази серия. В нейният състав влизат:

Running Linux[1]

Ръководство за инсталиране и работа със системата, описващо как да получим максимална производителност при персоналната работа с Linux.

[1] Книгата е издадена на български език под името *Ръководство за Linux* – б.р.

Learning Debian GNU/Linux

Learning Red Hat Linux

На по-елементарно ниво от *Running Linux*, тези книги са придружени с популярни дистрибуции на CD-ROM и съдържат ясни указания за инсталирането и използването им.

Linux in a Nutshell

Поредната успешна книга от серията "in a Nutshell" (накратко). Тази книга предоставя обширен справочен текст за Linux.

Linux Journal и Linux Magazine

Linux Journal и *Linux Magazine* са месечни списания за Linux обществото, които се пишат и издават от Linux активисти. Те съдържат статии, започващи от отговори на въпроси на новаци и стигащи до подробности за програмирането на ядрото. Дори да имате достъп до Usenet, тези списания са добър начин да бъдете в течение с дискусиите в Linux обществото.

Linux Journal е по-старото списание и се издава от фирмата S.S.C. Incorporated, за която по-горе бяха поместени подробности. Можете да намерите списанието и в Web на адрес *http://www.linuxjournal.com/*

Linux Magazine е по-ново, независимо издание. Web-сайта за списанието е *http://www.linuxmagazine.com/*

Usenet групи за Linux

Ако имате достъп до групите в Usenet, в тях ще намерите следните свързани с Linux дискусионни групи:

comp.os.linux.announce

Ненатоварена група, съдържаща анонси на нов софтуер, дистрибуции, сведения за грешки и тенденции в Linux обществото. Всички потребители на Linux би трябвало да четат съобщенията в тази група. Можете да изпращате съобщения за тази група на адрес *linux-announce@news.ornl.gov*.

comp.os.linux.help

Общи въпроси и отговори за инсталирането и използването на Linux.

comp.os.linux.admin

Дискусии, свързани със системното администриране под Linux.

comp.os.linux.networking

Дискусии, свързани с работата в мрежа под Linux.

comp.os.linux.development

Дискусии относно разработването на ядрото на Linux и самата система.

comp.os.linux.misc

Сборна група за различни дискусии, които не попадат в горните категории.

Съществуват и множество групи, отнасящи се за Linux, но на езици, различни от английски, например *fr.comp.os.linux* на френски език и *de.comp.os.linux* на немски.

Пощенски списъци за Linux

Можете да се абонирате за множество специализирани пощенски списъци за Linux, в които ще намерите много желаещи да ви помогнат за решаване на проблемите, които имате.

Най-популярните пощенски списъци са списъците, поддържани от Rutgers University. Можете да се абонирате за тези списъци като изпратите e-mail, оформен по следния начин:

```
To: majordomo@vger.rutgers.edu
Subject: anything at all
Body:

subscribe listname
```

Някои от списъците, свързани с Linux, са:

linux-net

Дискусии, свързани с работата в мрежа под Linux.

linux-ppp

Дискусии, свързани с реализацията на Linux PPP

linux-kernel

Дискусии, свързани с разработването на ядрото на Linux.

Интерактивна поддръжка на Linux

Съществуват много начини за получаване на интерактивна помощ. Доброволци от целия свят предлагат своята експертиза и услуги, за да помогнат на потребителите по техните въпроси и проблеми.

OpenProjects IRC Network е IRC мрежа, посветена изцяло на отворените проекти като Отворен Код и Отворен Хардуер. Някои от нейните канали са проектирани за предоставяне на интерактивна поддръжка на Linux. IRC е съкращение от Internet Relay Chat (разговор, предаван чрез Интернет) и е мрежова услуга, която дава възможност да разговаряте интерактивно през Интернет с други потребители. IRC мрежите поддържат множество канали, в които разговарят групи от хора. Текстът, което напишете в канала, се вижда от всички останали потребители на този канал.

Съществуват множество активни канали в IRC мрежата OpenProjects, където можете да намерите потребители 24 часа в денонощието и 7 дена в седмицата, които са готови и са в състояние да ви помогнат за разрешаването на всякакви свързани с Linux проблеми, които може да имате, или просто да си поговорите. Можете да използвате тази услуга като инсталирате IRC клиент, например *irc-II*, свържете се към сървъра **irc.openprojects.org:6667** и се присъедините към канала `#linpeople`.

Групи от потребители на Linux

Много групи от потребители на Linux в целия свят предлагат директна поддръжка на потребителите. Те се ангажират с дейности като дни за инсталации, разговори и семинари, демонстрационни вечери и други изцяло социални събития. Групите от потребители на Linux са чудесен начин да се срещнете с други потребители на Linux във вашия град или област. Публикувани са много бюлетини на потребителските групи. Някои от най-известните са:

Groups of Linux Users Everywhere

http://www.ssc.com/glue/groups/

LUG list project

http://www.nllgg.nl/lugww/

LUG registry

http://wwwlinux.org/users/

Как да се снабдим с Linux

Не съществува само една дистрибуция на софтуера за Linux; всъщност, има много дистрибуции като Debian, RedHat, Caldera, Corel, SuSE и Slackware. Всяка дистрибуция съдържа всичко, от което се нуждаете, за да използвате пълна Linux система: ядрото, основни инструменти, библиотеки, поддържащи файлове и приложен софтуер.

Можете да се сдобиете с дистрибуциите на Linux от множество електронни източници, например от Интернет. Всяка от основните дистрибуции има собствен FTP и web-сайт. Някои от тези сайтове са:

Caldera

http://www.caldera.com/ftp://ftp.caldera.com/

Corel

http://www.corel.com/ftp://ftp.corel.com/

Debian

http://www.debian.org/ftp://ftp.debian.org/

RedHat

http://www.redhat.com/ftp://ftp.redhat.com/

Slackware

http://www.slackware.com/ftp://ftp.slackware.com/

SuSE

http://www.suse.com/ftp://ftp.suse.com/

Много от популярните FTP сайтове с архиви съдържат огледални копия на дистрибуции на Linux. Най-известните такива сайтове са:

metalab.unc.edu:/pub/Linux/distributions/
ftp.funet.fi:/pub/Linux/mirrors/
tsx-11.mit.edu:/pub/linux/distributions/
mirror.aarnet.edu.au:/pub/linux/distributions/

Много от модерните дистрибуции могат да бъдат инсталирани директно от Интернет. За типична инсталация трябва да изтеглите много софтуер, затова вероятно ще използвате такъв тип инсталиране само, ако имате високоскоростна непрекъсната връзка с Мрежата или ако просто искате да актуализирате съществуваща инсталация[2].

Можете да закупите Linux на CD-ROM от постоянно нарастващ брой доставчици. Ако вашият местен компютърен магазин го няма, вероятно ще можете да го поръчате. Повечето от популярните дистрибуции могат да бъдат получени на CD-ROM. Някои доставчици произвеждат продукти, съдържащи се на няколко CD-ROM-а, всеки от които съдържа различна дистрибуция на Linux. Това е много добър начин за изпробване на различни дистрибуции, докато се спрете на най-удобната за вас.

Стандарти за файловата система

В миналого един от проблемите, който пречеше на дистрибуциите на Linux, както и на софтуерните пакети, работещи под Linux, беше липсата на единна приета от всички организация на файловата система. Това водеше до несъвместимост между различните пакети и поставяше администраторите и потребителите пред задачата да откриват разположението на различни файлове и програми.

За подобряване на тази ситуация през август 1993 г. група експерти формираха така наречената Група за стандартизиране на файловата система на Linux (FSSTND – File System Standard Group). След шест месеца дискусии групата създаде проект, който представя схема на структура на файлова система и определи разположението на най-важните програми и конфигурационни файлове.

Този стандарт бе предназначен за внедряване от повечето основни дистрибуции и пакети на Linux. За съжаление, макар че в повечето

[2] ... или сте крайно нетърпеливи и знаете, че 24-те часа, необходими за изтегляне на софтуера от Интернет, са по-малко от 72-те часа, през които трябва да чакате доставката на компакт-диска с дистрибуцията!

дистрибуции бяха направени опити да се постигне съвместимост с FSSTND, много малко са тези, които го въведоха изцяло. В тази книга ще приемем, че всички файлове, за които ще става дума, са разположени на определените в стандарта места. Алтернативни места ще бъдат споменавани само, когато има силни традиции, които са в конфликт с тези препоръки.

Стандартът FSSTND на Linux продължава да се развива, но през 1997 г. беше заменен със стандарта FHS (File Hierarchy Standard – стандарт за йерархията на файловете). Стандартът FHS решава проблемите, които възникват при дистрибуциите за много архитектури, които не са решени от FSSTND. Текстът на стандарта FHS може да се изтегли от директорията за документация на всички основни FTP сайтове за Linux и техните огледални копия или от основният му сайт на адрес *http:// www. pathname.com/fhs/*. Daniel Quinlan, координаторът на групата за разработване на стандарта FHS, може да бъде намерен на адрес *quinlan@transmeta.com*.

Стандартна база на Linux

Големият брой различни дистрибуции на Linux, въпреки че осигури много алтернативи за потребителите, създаде проблем за разработчиците на софтуер – особено за разработчиците на не-свободен софтуер.

Всяка дистрибуция пакетира и доставя определени базови библиотеки, конфигурационни инструменти, системни приложения и конфигурационни файлове. За нещастие, различията в техните версии, имена и местоположения, правят много труден анализа на инсталирания софтуер в дадена дистрибуция. Това прави трудно създаването на двоични (изпълними) приложения, които да работят надеждно на всички разпространени дистрибуции на Linux.

За да помогне за преодоляването на този проблем се появи нов проект, наречен "Стандартна база на Linux" (LSB – Linux Standard Base). Неговата цел е да опише стандартната база на дистрибуцията, която съвместимите дистрибуции могат да използват. Ако разработчикът проектира едно приложение да работи със стандартната базова платформа, това приложение ще работи и ще бъде преносимо на всяка съвместима дистрибуция на Linux.

Можете да намерите информация за състоянието на проекта LSB на неговия web-сайт на адрес *http://www.linuxbase.org/*.

Ако се грижите за съвместимостта, особено на платен софтуер, трябва да се убедите, че създателите на вашата дистрибуция на Linux са положили необходимите усилия за включване в стандартизационния проект.

За книгата

Когато Олаф се присъедини към Проекта за документиране на Linux (LDP) през 1992 г., той написа две малки глави за UUCP и *smail*, с които смяташе да допринесе към усилията за създаване на System Administrator's Guide. Разработката на поддръжката за TCP/IP мрежа току-що започваше и когато тези "малки глави" започнаха да нарастват, той се чудеше на глас дали не би било хубаво да се направи ръководство за работа в мрежа. "Чудесно! Направи го!" казваше всеки. Така че, Олаф направи необходимото и написа първата версия на Ръководството на мрежовия администратор на Linux, което беше издадено през 1993 г.

Олаф продължи работата си върху Ръководството и в резултат създаде много по-добра негова версия. Винс Скаан добави първоначалната глава за *sendmail*, която беше напълно заменена в това издание поради новия интерфейс за конфигуриране на *sendmail*.

Вариантът на ръководството, което сега четете, е редактиран и осъвременен от O'Reilly & Associates под ръководството на Тери Доусън*. Тери е радио-оператор любител от над 20 години, като над 15 от тях е работил в телекомуникационната индустрия. Той е съавтор на първоначалния списък NET-FAQ, а сега създава и поддържа множество свързани с работата в мрежа HOWTO-документи. Тери винаги е бил ентусиазиран поддръжник на проекта за Ръководство на мрежовия администратор и добави няколко нови глави към тази версия, описвайки възможности на Linux за работа в мрежа, които бяха разработени след първото издание, като освен това направи множество изменения за осъвременяване на останалата част от книгата.

Главата за *exim* беше добавена от Philip Hazel♦, който е водещ разработчик на пакета.

---

* Тери Доусън може да бъде намерен на адрес *terry@linux.org.au*.

♦ Philip Hazel може да бъде намерен на адрес *ph10@cus.cam.ac.uk*.

Книгата е организирана приблизително според последователността на стъпките, които трябва да извършите, за да конфигурирате вашата система за работа в мрежа. Тя започва с обсъждане на основни концепции на мрежите и в частност на TCP/IP мрежите. След това текста плавно преминава от конфигурирането на TCP/IP на ниво устройство към защитна стена, отчитане на трафика и конфигуриране на маскирането и стига до инсталиране на общи приложения като *rlogin* и приятели, мрежовата файлова система (NFS) и мрежовата информационна система (NIS). Следва главата за начина, по който да настроите вашата машина като UUCP възел. Повечето от оставащите части са посветени на две основни приложения, които работят върху TCP/IP и UUCP: електронната поща и новините. На протокола IPX и файловата система NCP е посветена специална глава, защото те се използват в много корпоративни среди, където Linux намира приложение.

Частта за електронната поща съдържа въведение към детайлите за прехвърлянето и маршрутизирането на пощата и безкрайните схеми на адресиране, с които можете да се срещнете. В нея се описва конфигурирането и управлението на *exim*, агент за прехвърляне на поща, който е идеален за използване в повечето ситуации, които не изискват UUCP, и *sendmail*, който е за хора, които трябва да конфигурират по-сложно маршрутизиране, включващо UUCP.

Частта за новини съдържат преглед на начина, по който работи системата Usenet news. Тя включва INN и C News, двата най-широко използвани в момента софтуерни пакета за прехвърляне на новини и работата с NNTP за осигуряване на достъп за четене на новини в локалната мрежа. Книгата завършва с глава за поддръжката на най-популярните четци на новини за Linux.

Разбира се, една книга никога не може изчерпателно да отговори на всички въпроси, които можете да зададете. Затова, ако следвате инструкциите в тази книга и въпреки това, нещо не работи, моля бъдете търпеливи. Възможно е някои от вашите проблеми да се дължат на наши грешки (вижте раздела “Изпращане на промени”, по-долу в този Предговор), но те могат да бъдат причинени и от изменения в мрежовия софтуер. Ето защо, първо трябва да прегледате изброените източници на информация. Съществува голяма вероятност да не сте единствения, който се е сблъскал с проблема, така че решение или поне начин за заобикаляне на проблема вероятно вече е открит. Ако имате възможност, трябва също така да опитате да се сдобиете с последните версии на ядрото и мрежовата част на софтуера от някой от

FTP сайтовете за Linux или BBS близо до вас. Много проблеми се причиняват от софтуерни компоненти в различни етапи на разработка, които не могат да работят правилно съвместно. В края на краищата, Linux е развиваща се система.

Официалната печатна версия

През есента на 1993 г. Енди Орам, който участва в пощенския списък на LDP от самото му създаване, попита Олаф дали е съгласен тази книга да се публикува от O'Reilly & Associates. Той беше възхитен от идеята, но не можеше да си представи, че тя ще има такъв успех. Олаф и Енди накрая се съгласиха O'Reilly да издаде подобрена Официална печатна версия на Ръководството на мрежовия администратор, като Олаф запази авторските права над оригинала, така че текстът на книгата да може свободно да се разпространява. Това означава, че можете сами да избирате: да изтеглите различни свободни форми на документа от най-близкия ви огледален сайт на LDP и да го отпечатате или да закупите официалната печатна версия от O'Reilly.

Защо трябва да давате пари за нещо, което можете да получите безплатно? Дали Tim O'Reilly е загубил разсъдъка си и затова публикува нещо, което всеки може сам да отпечата и дори, ако иска, да го продава? Има ли някаква разлика между тези версии?

Отговорите са "зависи", "не, определено не" и "да и не". O'Reilly & Associates пое риск с издаването на Ръководството на мрежовия администратор и изглежда, че рискът си струваше (хората искат още такива книги). Вярваме, че този проект служи като хубав пример за това как свободният софтуер и компаниите могат да обединят усилията си, за да създадат нещо, от което и двете страни да имат полза. Според нас, голямата услуга, която O'Reilly прави на Linux обществото (освен, че книгата вече се намира във вашата квартална книжарница) е това, че помогна Linux да бъде идентифициран като нещо, което трябва да бъде взето на сериозно: жизнеспособна и полезна алтернатива на другите комерсиални операционни системи. Няма сериозна книжарница за техническа литература без поне една лавица, пълна с книги за Linux на O'Reilly.

Защо O'Reilly издава тези книги? Защото гледа на тях като на новия тип литература. Това са книгите, които компанията се надява да издава, ако се договори с автор, който пише за Linux. Темпът, нивото на детайлност и стилът съвпадат с останалите предложения на O'Reilly.

Основната идея в лиценза на LDP е да се направи така, че никой да не може да го пренебрегне. Други хора могат да отпечатат копие на тази книга и никой няма да ви упрекне, ако си вземете едно от тези копия. Но ако не сте имали шанса да видите книгата на O'Reilly, отидете в книжарницата или разгледайте книгата на ваш приятел. Смятаме, че ще харесате онова, което ще видите, и ще поискате да си го купите.

И така, каква е разликата между подвързаната и отпечатаната в къщи версия? Енди Орам извърши много работа, трансформирайки нашите безпорядъчни познания в нещо, което действително си струва да се отпечата. (Освен това той е направил рецензии на някои други книги, създадени от проекта LDP, допринасяйки с професионалните си умения на Linux обществото).

Откакто Енди започна преглеждането на Ръководството на мрежовия администратор и редактирането на изпратените му копия, книгата претърпя големи подобрения в сравнение с първоначалната си форма и с всеки цикъл представяне – препоръки се подобряваше отново. Възможността да се извлече полза от опита на професионалния редактор е нещо, което не бива да се пропуска. В много отношения приносът на Енди е толкова значим, колкото и този на авторите. Това важи и за редакторите на копията, които доведоха книгата до вида, в който я виждате в момента. Всички тези редакции са отразени и в електронната версия, така че няма разлика между тях в съдържанието.

Все пак, версията на O'Reilly *е* различна. Тя е професионално подвързана и макар че вие може би ще успеете да преодолеете проблемите при отпечатването на своята безплатна версия, едва ли ще получите същото качество на крайния продукт, а още по-малко вероятно е да го получите на същата цена. На второ място, вашите любителски опити за илюстриране ще бъдат заменени от красиво изработените фигури на професионалните художници от O'Reilly. Хората, работили по индекса, са създали подобрен продукт, което прави намирането на информация в книгата много по-лесно. Ако смятате да прочетете тази книга от начало до край, бихте предпочели да четете официалното печатно издание.

Кратък преглед на книгата

В Глава 1, *Въведение в работата в мрежа*, се разглежда историята на Linux и се дава основна мрежова информация за UUCP, TCP/IP, различните протоколи, хардуера и сигурността. Следващите няколко глави са посветени на конфигурирането на Linux за работа в TCP/IP

мрежа и стартиране на някои основни приложения. Малко по-подробно изучаваме IP в Глава 2, *Въпроси на работата в TCP/IP мрежа*, преди да започнем редактирането на файлове и други подобни дейности. Ако вече знаете как работи IP маршрутизацията и как се извършва преобразуването на адресите, можете да прескочите тази част.

Глава 3, *Конфигуриране на мрежовия хардуер*, е посветена на много основни конфигурационни въпроси като компилиране на ядро и настройване на Ethernet контролера. Конфигурирането на серийните портове е разгледано отделно в Глава 4, *Конфигуриране на серийния хардуер*, защото дискусията се отнася не само за TCP/IP мрежите, но и за UUCP.

Глава 5, *Конфигуриране на TCP/IP мрежа*, помага да настроите вашата машина за мрежова работа с TCP/IP. Тя съдържа съвети за инсталирането на хостове, свързани към Ethernet или самостоятелни хостове, имащи само loopback интерфейс. Освен това, в тази глава ще се запознаете и с някои полезни инструменти, с които можете да тествате и отстранявате грешки във вашата инсталация. Глава 6, *Конфигуриране на услугата за имена и резолвера*, разглежда как се конфигурира преобразуването на хост-имена в адреси и обяснява как се инсталира сървър за имена.

Глава 7, *IP през серийна линия*, обяснява как се установяват SLIP връзки и дава подробна информация за *dip*, инструмент, който дава възможност за автоматизиране на повечето от необходимите действия. Глава 8, *Протоколът* PPP, описва протоколът за връзка от точка до точка PPP и демона *pppd*.

Глава 9, *Защитна стена за TCP/IP*, разширява дискусията върху мрежовата сигурността и описва TCP/IP защитната стена за Linux и нейните конфигурационни инструменти: *ipfwadm*, *ipchains* и *iptables*. Защитната стена, базирана на IP, осигурява средства за много прецизно контролиране на достъпа до вашата мрежа и вашите компютри.

Глава 10, *IP счетоводство*, обяснява как се конфигурира IP счетоводството в Linux, което позволява да се следи какво количество трафик преминава през вашия хост, накъде, и кой го генерира.

Глава 11. *IP маскиране и транслиране на мрежови адреси*, описва функцията на мрежовия софтуер на Linux, наречена IP маскиране, която позволява цяла IP мрежа да се свърже и използва Интернет чрез един-единствен адрес, скривайки вътрешните системи от външните по време на този процес.

Глава 12, *Важни мрежови възможности*, дава кратко описание на настройката на най-важните мрежови приложения като *rlogin*, *ssh* и др. Тази глава показва и как супер-потребителя *inetd* управлява мрежовите сървъри и как можете да ограничите връзките към определени свързани със сигурността услуги, така че те да са достъпни само за конкретни хостове.

Глава 13, *Мрежова информационна система* и Глава 14, *Мрежова файлова система*, разглеждат NIS и NFS. NIS е инструмент, използван за разпространяване в локалната мрежа на административна информация (например потребителски пароли). NFS позволява да споделяте файловите системи между множество хостове във вашата мрежа.

В Глава 15, *IPX и файловата система NCP*, разглеждаме протокола IPX и файловата система NCP. Те позволяват на Linux да се интегрира в мрежова среда, базирана на Novell NetWare, като споделя файлове и принтери с не-Linux машини.

Глава 16, *Управление на Taylor UUCP* съдържа подробно встъпление в администрирането на Taylor UUCP, свободна реализация на сюитата UUCP.

Останалата част на книгата се заема от подробен преглед на електронната поща и новините в Usenet. Глава 17, *Електронна поща*, ви запознава с основните концепции в електронната поща, как изглежда пощенският адрес и как работи системата за обработка на пощата, когато доставя вашето съобщение до получателя.

Глава 18, *Sendmail*, и Глава 19, *Конфигуриране и използване на exim* описват конфигурирането на *sendmail* и *exim*, два агента за прехвърляне на поща, които можете да използвате в Linux. В тази книга са описани и двата агента, защото *exim* е по-лесен за инсталиране от начинаещи, а *sendmail* предоставя поддръжка на UUCP.

Главите от 20, *Netnews* до 23, *Интернет новини*, обясняват начина, по който се управляват новините в Usenet и как да инсталирате и използвате C News, *nntpd* и INN – три популярни софтуерни пакета за управление на новините в Usenet. След краткото въведение в Глава 20, можете да прочетете Глава 21, *C News*, ако искате да обменяте новини с помощта на CNews – традиционната услуга, която обикновено се използва с UUCP. Следващите глави разглеждат по-модерни алтернативи на CNews, които използват базирания на Интернет протокол NNTP (Network News Transfer Protocol – мрежов протокол за прехвърляне на новини). Глава 22, *NNTP и демона nntpd* показва как

се инсталира обикновен NNTP демон – демона *nntpd*, осигуряващ достъп за четене на новини в локална мрежа. Глава 23 описва помощен сървър за глобално прехвърляне на новини през Интернет – демона INN (InterNet News). Накрая Глава 24, *Конфигуриране на четец за новини*, показва как се конфигурират и поддържат различни потребителски програми за четене на новини.

Конвенции, използвани в книгата

Всички примери, представени в тази книга, предполагат, че използвате съвместима с *sh* обвивка (shell, или както още е известен, команден интерпретатор). Обвивката *bash* е съвместима с *sh* и е стандартна за всички дистрибуции на Linux. Ако сте потребител на *csh*, ще трябва да направите съответните промени в примерите.

Следва списък на полиграфските конвенции, използвани в книгата:

Наклонен текст

Използва се за имена на файлове, директории, имена на команди и програми, опции на командни редове, email адреси и пътища, URLs и за повдигане на нови термини.

Удебелен текст

Използва се за имена на машини, хостове, сайтове, потребители и идентификатори, както и за наблягане върху важността на някои места в текста.

`Текст с постоянна ширина`

Използва се в примери за показване съдържанието на файлове с код, отпечатания текст от различни команди, обозначаване на променливи от обкръжението и ключови думи, появяващи се в кода.

`Наклонен текст с постоянна ширина`

Използва се за означаване на опции на променливи, ключови думи или текст, който потребителят трябва да замени с действителна стойност.

`Удебелен текст с постоянна ширина`

Използва се в примери за показване на команди или друг текст, който трябва да бъде въведен буквално от потребителя.

Текст, който е обозначен по този начин, съдържа предупреждение. Тук можете да направите грешка, която да повреди вашата система или да е трудно да се отмени.

Изпращане на промени

Информацията в тази книга е тествана и проверена по възможно най-добрия начин, но е не е изключено да установите, че някои неща са се променили (възможно е дори да откриете, че сме направили грешки!). Моля уведомете ни за всяка грешка, която сте открили, както и за вашите предложения за бъдещите издания, като ни пишете на адрес:

O'Reilly & Assosites, Inc.
101 Morris Street
Sebastopol, Ca 95472
1-800-998-9938 (в U.S. или Canada)
1-707-829-0515 (международно или локално)
1-707-829-0104 (FAX)

Можете да ни изпращате съобщения по електронен път. За да се включите в нашия пощенски списък или за да заявите каталог, изпратете e-mail на адрес:

info@oreilly.com

За да поставите технически въпроси или за коментари върху книгата, изпратете e-mail на адрес:

bookquestions@oreilly.com

O'Reilly поддържа web-сайт за книгата, където можете да намерите примери, списък на допуснати грешки и планове за бъдещи издания. Адресът на тази страница е:

http://www.oreilly.com/catalog/linag2

Повече информация за тази и други книги на O'Reilly вижте можете да намерите на адрес:

http://www.oreilly.com

Благодарности

Това издание на Ръководството на мрежовия администратор дължи почти всичко на забележителната работа на Олаф и Винс. Трудно е да се оцени усилието, което е необходимо при проектирането и написването на книга от такъв характер, докато не получите шанса сами да извършите тази работа. Актуализацията на книгата също беше предизвикателна работа, но при наличието на такава прекрасна база, беше приятно занимание.

Тази книга дължи много на различни хора, които намериха време да я прегледат и помогнат за изглаждането на много грешки – както технически, така и граматически (ние дори не знаехме, че съществува такова нещо като свободно причастие). Phil Hughes, John MacDonalds и Eric Ratcliffe предложиха много полезни (и като цяло, твърде значителни) корекции на съдържанието на книгата.

Освен това, дължим много благодарности на хората от O'Reilly, с които имахме удоволствието да работим: Sarah Jane Shangrow, която доведе книгата до вида, в който я виждате сега; Maureen Dempsey, която редактира текста за печат; Rob Romano, Rhon Porter и Chris Railly, които създадоха фигурите; Hana Dayer, която проектира корицата; Alisa Cech, David Futato и Jennifer Niedherst за вътрешния дизайн; Лар Кауфман за предложените стари дърворезби като визуални теми; Judy Hoer за индекса; и накрая, Tim O'Reilly за смелостта да се заеме с такъв проект.

Много сме задължени на Andre Sepulveda, Wolfgang Michaelis, Michael K. Johnson и всички разработчици, които отделиха време, за да проверят информацията в Ръководството на мрежовия администратор. Phil Hughes, John MacDonalds и Eric Ratcliffe предоставиха неоценими коментари за второто издание. Също така, искаме да благодарим на тези, които прочетоха първата версия на Ръководството и ни изпратиха корекции и предложения. Можете да намерите пълен списък на сътрудниците във файла *Thanks* в електронната дистрибуция. Накрая, тази книга не би била възможна без помощта на Holger Grothe, който осигури на Олаф необходимата връзка с Интернет, за да направи първоначалната версия възможна.

Олаф би искал също така да благодари на следните групи и компании, които отпечатаха първото издание на Ръководството на мрежовия администратор и предоставиха част от печалбата лично на него или на Проекта за документиране на Linux като цяло: Linux Support Team, Erlangen, Германия; S.u.S.E. GmbH, Fuerth, Германия и Linux

System Labs, Inc., Clinton Twp., САЩ и RedHat Software, Северна Каролина, САЩ.

Тери благодари на своята съпруга, Маги, която търпеливо го подкрепяше през цялото време на участието му в проекта, независимо от предизвикателствата от раждането на първото им дете, Джак. Освен това, Тери благодари на *много* хора от Linux обществото, които го обучаваха независимо от трудностите до момента, в който и той можеше да вземе реално участие и активно да допринася в работата. "Ще ти помогна, ако обещаеш и ти да помогнеш на някой друг в замяна на това".

Залата на славата

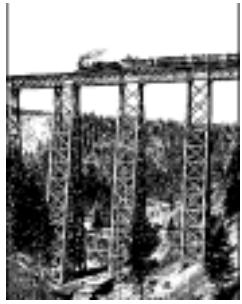
Освен тези, които вече споменахме, много други хора допринесоха при създаването на Ръководството на мрежовия администратор като го прегледаха и ни изпратиха поправки и предложения. Ние сме им много благодарни.

Ето списък на тези, чиито приноси оставиха следи в пощенските ни папки.

Al Longyear, Alan Cox, Andres Sepúlveda, Ben Cooper, Cameron Spitzer, Colin McCormack, D.J. Roberts, Emilio Lopes, Fred N. van Kempen, Gert Doering, Greg Hankins, Heiko Eissfeldt, J.P. Szikora, Johannes Stille, Karl Eichwalder, Les Johnson, Ludger Kunz, Marc van Diest, Michael K. Johnson, Michael Nebel, Michael Wing, Mitch D'Souza, Paul Gortmaker, Peter Brouwer, Peter Eriksson, Phil Hughes, Raul Deluth Miller, Rich Braun, Rick Sladkey, Ronald Aarts, Swen Thüemmler, Terry Dawson, Thomas Quinot и Yury Shevchuk.

ГЛАВА 1

ВЪВЕДЕНИЕ В РАБОТАТА В МРЕЖА



История

Идеята за работа в мрежа вероятно е толкова стара, колкото и самата телекомуникация. Представете си хората, живеещи в Каменната епоха, когато за предаване на съобщения между двама души са се използвали барабани. Да предположим, че пещерният човек А иска да покани пещерния човек В, за да си поиграят, като се замерят с камъни. Двамата, обаче, живеят много далеч един от друг и В не може да чуе, когато А удря по барабана си. Какви са възможностите на А? Той би могъл: 1) да отиде до пещерата на В, 2) да вземе по-голям барабан или 3) да поиска от С, който живее на половината път между А и В да предаде съобщението. Последната възможност се нарича *работа в мрежа*.

Разбира се, ние сме изминали дълъг път от примитивните действия и средства на нашите прародители. Сега имаме компютри, разговаряме помежду си през сложни плетеници от проводници, оптични влакна, микровълни и други подобни, за да направим, например, уговорка за съботния мач*. В следващите редове ще разгледаме средствата и начините, чрез които се осъществява това, но ще оставим настрана въпроса за проводниците и мачовете.

* Първоначалният дух на играта (виж по-горе) все още понякога се забелязва в Европа.

В това ръководство ще опишем три вида мрежи. Ще обърнем най-голямо внимание на TCP/IP, защото това е най-популярната група протоколи, използвана както в локалните мрежи, така и в големите мрежи като Интернет. Освен това ще разгледаме протоколите UUCP и IPX. Преди време UUCP беше масово използван за пренос на новини и писма през комутируеми телефонни линии. Днес той е по-малко популярен, но все още е полезен в различни ситуации. Протоколът IPX се използва най-често в корпоративните среди, базирани на Novell NetWare, затова ще обясним как да го използвате за да свържете вашата Linux машина в мрежа с Netware. Всеки от тези протоколи е мрежов протокол и се използва за пренасяне на данни между компютри-хостове. Ще разгледаме как можете да ги използвате и ще ви запознаем с принципите, на които те са базирани.

Ще дефинираме мрежата като множество от *хостове*, всеки два от които са в състояние да комуникират помежду си. Често за комуникация между хостовете се разчита на услугите на специално предназначени за целта други хостове, които препредават данни между участниците. Хостовете обикновено са компютри, но това не е задължително; X-терминалите и интелигентните принтери също могат да се смятат за хостове. Малки групи от хостове се наричат още *сайтове*.

Комуникацията е невъзможна без някакъв вид език или код. В компютърните мрежи тези езици се наричат *протоколи*. Не трябва обаче да мислите за тях като за писмени протоколи, а по-скоро като за високо формализиран начин на поведение, например като този, който се наблюдава, когато се посрещат държавни глави. По твърде подобен начин, протоколите използвани в компютърните мрежи, не са нищо друго, освен много строги правила за обмяна на съобщения между два или повече хоста.

TCP/IP мрежи

Модерните мрежови приложения изискват сложен начин за пренасяне на данни от една машина към друга. Ако управлявате Linux машина, която има много потребители, всеки от които иска едновременно да се свърже с няколко отдалечени хоста в мрежата, трябва да разполагате с начин, който им осигурява възможност съвместно да използват мрежата, без да пречат един на друг. Подходът, който се използва от голям брой от модерните мрежови протоколи, се нарича *комутиране на пакети* (*packet switching*). *Пакетът* е малък блок от данни, който се пренася от една машина до друга през мрежата. Комутирането се извършва, като дейтаграмата се предава през нова връзка

от мрежата. Мрежата с комутиране на пакети разпределя една мрежова линия между много потребители, като алтернативно изпраща пакети от един потребител до друг през тази линия.

Решението, което Unix-системите, а след тях и много други системи възприеха, е известно като TCP/IP. Когато говорим за TCP/IP мрежи, ще срещнете термина *дейтаграма*, който технически има специално значение, но често се използва взаимозаменяемо с термина *пакет*. В тази част ще разгледаме концепциите, които лежат в основата на TCP/IP протоколите.

Въведение в TCP/IP мрежите

TCP/IP води своя произход от един изследователски проект на американската Агенция за перспективни изследователски проекти за отбраната (Defense Advanced Research Project Agency – DARPA) от 1969 г. ARPANET беше експериментална мрежа, която, след като се доказа, че е успешна, през 1975 г. се превърна в работеща.

През 1983 г. новата група протоколи TCP/IP беше приета като стандарт и всички хостове в мрежата бяха задължени да го използват. Когато ARPANET накрая прекрати съществуването си (самата ARPANET излезе от употреба през 1990 г.), използването на TCP/IP се разпространи и в мрежи извън самия Интернет. Много компании сега изграждат корпоративни мрежи, базирани на TCP/IP, а Интернет се разрасна до такава степен, че почти може да се счита за основна потребителска технология. Трудно е да срещнете вестник или списание без препратка към Интернет и почти всеки може да използва мрежата.

Като конкретен пример, по време на описанието на TCP/IP в следващите части ще използваме Университета Groucho Marx (GMU), разположен някъде във Фредландия. Повечето факултети използват свои собствени локални мрежи, но някои използват една мрежа съвместно, а други имат по няколко. Всичките мрежи са взаимно свързани и имат достъп до Интернет през една високоскоростна линия.

Да предположим, че вашата Linux машина е свързана към локална мрежа от Unix хостове в Математическия факултет и нейното име е **erdos**. За да достигнете до хост във Физическия факултет, да кажем **quark**, вие въвеждате следните команди:

```
$ rlogin quark.physics
Welcome to the Physics Department at GMU
(ttyq2) login:
```

В поканата за влизане трябва да въведете вашето име, например **andres**, както и паролата си. След това получавате shell-достъп* до **quark**, с който можете да въвеждате команди точно както, ако седяхте пред системната конзола. След като прекратите достъпа до отдалечената машина, вие се връщате на поканата за въвеждане на команди на вашата собствена машина. Току-що използвахте едно от директните интерактивни приложения, които предлага TCP/IP: отдалечено влизане (remote login).

Докато сте свързани към **quark**, можете да поискате да стартирате и приложение с графичен потребителски интерфейс, например текстообработваща програма, програма за рисуване на графики или дори web-браузър. Системата X Window е изцяло мрежово-ориентирана графична потребителска среда и е достъпна за много различни компютърни системи. За да укажете на избраното приложение, че искате неговия прозорец да се показва на екрана на вашия хост, трябва да настроите променливата `DISPLAY`:

```
$ DISPLAY=erdos.maths:0.0
$ export DISPLAY
```

Ако сега стартирате приложението, то ще се свърже с вашия X-сървър, вместо с този на **quark**, и ще показва всички свои прозорци на вашия монитор. Разбира се, това изисква да имате работещ X11 на **erdos**. Важното в случая е, че TCP/IP дава възможност на **quark** и **erdos** да изпращат X11-пакети помежду си, като създават илюзията, че работите на една-единствена система. Мрежата в тази ситуация е почти прозрачна.

Друго много важно приложение в TCP/IP-мрежите е NFS, което е съкращение от *Networks File System* (Мрежова Файлова Система). Това е друг начин да направите мрежата прозрачна, защото позволява да третирате йерархията на директориите от други хостове все едно, че са локални файлови системи и изглеждат като всички други директории на вашия хост. Например, всички лични потребителски директории могат да бъдат пазени на централен сървър, от където всички други хостове в локалната мрежа могат да ги използват. Ефектът е, че потребителите могат да влязат на който и да е машина и да работят в своята директория. Аналогично е възможно много хостове съвместно да използват големи обеми от данни (например бази данни,

* Обвивката (shell) е интерфейса с команден ред на операционната система Unix. Тя е подобна на т.нар. DOS prompt в обкръжението Microsoft Windows, но е много по-мощна.

документации или приложни програми), като се поддържа едно копие на данните на един сървър и се позволява достъп на другите хостове до тях. Ще се върнем пак към NFS в Глава 14, *Мрежова файлова система*.

Разбира се, това са само примери за това, какво можете да правите с TCP/IP мрежите. Възможностите са почти неограничени и ще ви запознаем с повече от тях по-нататък в книгата.

Сега ще разгледаме по-подробно как работи TCP/IP. Тази информация ще ви помогне да разберете как и защо трябва да конфигурирате вашата машина. Ще започнем с изучаване на хардуера и бавно ще се придвижваме напред.

Ethernet мрежи

Най-масовият тип хардуер за локални мрежи е известен като *Ethernet*. В неговата най-проста форма той се състои от един кабел и хостове, свързани към него чрез съединители (конектори), разклонители или предаватели. Простите Ethernet мрежи са сравнително евтини за инсталиране, което заедно със скоростите за предаване 10, 100 или дори 1000 мегабита/сек (Mbps) обяснява тяхната популярност.

Съществуват три варианта Ethernet мрежи: дебела (*thick*), тънка (*thin*) и усукана двойка (*twisted pair*). Дебелата и тънката Ethernet мрежи използват коаксиален кабел, различаващ се по диаметъра и начина на присъединяване на хоста към този кабел. Тънкият Ethernet използва Т-образен съединител тип "BNC", който се поставя към кабела и се завива в гнездо отзад на компютъра. Дебелият Ethernet изисква да пробиете малка дупка в кабела и да прикрепите предавателя с помощта на т.нар. "вампирски разклонител". Към предавателя могат да се свържат един или повече хостове. Тънката и дебелата Ethernet мрежи могат да се използват на разстояния съответно до 200 и 500 метра и се наричат също 10base-2 и 10base-5. В тези обозначения "base" (база) идва от термина "baseband modulation" и означава просто, че данните се подават директно в кабела без модем. Числото в началото означава скоростта в Mbps, а числото накрая е максималната дължина на кабела в стотици метри. Усуканата двойка използва кабел, направен от два чифта медни проводници и обикновено изисква допълнителен хардуер известен като *активен концентратор* (хъб). Усуканата двойка е известна като 10base-T, като "T" е съкращение от Twisted (усукан). Версията, работеща със скорост 100 Mbps, се нарича 100base-T.

За да добавите хост към инсталация с тънка Ethernet мрежа се налага да спрете работата на мрежата поне за няколко минути, защото трябва да отгрижете кабела и да поставите съединителя. Въпреки, че добавянето на хост към система с дебел Ethernet е малко по-сложно, то обикновено не води до спиране на мрежата. При усуканата двойка е още по-просто. Тя използва устройство, наречено "концентратор", което служи като място за свързване. Можете да добавяте и да премахвате хостове, без да се налага да прекъсвате работата на другите потребители в мрежата.

Много потребители предпочитат тънкият Ethernet за малки мрежи, защото е много евтин – PC картите струват под $30 (много компании буквално бълват такива карти), а кабелът струва по няколко цента на метър. Все пак, за по-големи инсталации са по-подходящи дебелият Ethernet и усуканата двойка. Например, мрежата в Математическия факултет на GMU първоначално беше изградена с дебел Ethernet, защото разстоянията са доста големи и не се налага трафикът да бъде прекъсван всеки път, когато се добавя нов хост. Инсталациите с усукана двойка сега са много разпространени и се използват в разнообразни конфигурации. Цената на концентраторите спада и се предлагат малки устройства на цени, които са достъпни дори за малки домашни мрежи. Използването на кабели с усукана двойка може да бъде значително по-евтино при големи инсталации, а самият кабел е много по-гъвкав в сравнение с коаксиалните кабели, използвани в другите Ethernet системи. Мрежовите администратори в Математическия факултет на GMU планират да заменят през следващата финансова година съществуващата коаксиална кабелна мрежа с усукана двойка, защото това ще ги доближи до съвременните технологии и ще им спести много време при инсталиране на нови и разместване на съществуващите компютри.

Един от недостатъците на Ethernet технологията е ограничението в дължината на кабелите, което прави невъзможно тяхното използване за друго, освен локални мрежи. Все пак, отделни Ethernet сегменти могат да бъдат свързани един към друг с помощта на повторители, мостове или маршрутизатори. Повторителите просто копират сигналите между два или повече сегмента така, че всички сегменти да работят, както ако са една Ethernet мрежа. Поради времеви ограничения, между кои да е два хоста в мрежата не може да има повече от четири повторителя. Мостовете и маршрутизаторите са по-сложни. Те анализират постъпващите данни и ги препредават само, ако получателят не е в локалния Ethernet.

Ethernet работи като шинна система, в която един хост може да изпраща пакети (или *кадри*) с големина до 1500 байта до друг хост в същата мрежа. Хостът се адресира с 6-байтов адрес, кодиран в неговата мрежова интерфейсна карта (NIC – Network Interface Card) за Ethernet. Тези адреси обикновено се записват като последователност от двуцифрови шестнадесетични числа, разделени с двоеточия, например **aa:bb:cc:dd:ee:ff**.

Кадърът, изпращан от една станция, се вижда от всички свързани станции, но само хостът-получател в действителност го получава и обработва. Ако две станции се опитат да предават едновременно, възниква *колизия*. Колизиите в Ethernet се откриват много бързо от електрониката в интерфейсните карти и се разрешават като двете станции прекъсват предаването, всяка изчаква случаен интервал от време и отново се опитва да предаде същите данни. Ще чуете много истории, че колизиите в Ethernet мрежи са проблем и поради това използваемостта им е само около 30 процента от тяхната пропускателна способност. Колизиите в Ethernet са *нормално* явление и в много натоварена мрежа не бива да се изненадвате от нива на колизии до около 30 процента. Едно по-реалистично ограничение за използваемостта на Ethernet мрежите е до около 60 процента, преди да започнете да се безпокоите за това.*

Други типове хардуер

В по-големи инсталации, като тази на GMU, Ethernet обикновено не е единствения тип оборудване. Съществуват много други протоколи за предаване на данни, които също се използват. Всички изброени подолу протоколи се поддържат от Linux, но поради липса на място ще ги опишем накратко. За много от протоколите са написани HOWTO-документи, където те са описани детайлно, така че трябва да се обърнете към тях, ако се интересувате от изучаването на протоколите, които не са описани в тази книга.

В GMU локалната мрежа на всеки факултет е свързана към университетската високоскоростна "опорна" мрежа, която е изградена от оптични кабели по технологията, наречена FDDI (*Fiber Distribution*

* Този проблем се обсъжда подробно в Ethernet FAQ (виж *http://www.faqs.org/faqs/LANs/ethernet-faq/*), а изобилие от историческа и техническа информация можете да намерите в web-сайта за Ethernet на Charles Spurgeon на адрес *http:// www.host.ots.utexas.edu/ethernet*.

Data Interface – Интерфейс за разпространение на данни по стъкловлакно). FDDI използва изцяло различен подход за предаване на данни, който се базира на изпращане на *маркери* (*tokens*) между машините. В този протокол всяка станция може да изпраща кадър само, ако маркерът е в нея. Основното предимство на протоколите, изпращащи маркери е намаляването на колизиите. Ето защо, този протокол може по-лесно да достигне пълната скорост на преносната среда, до 100 Mbps в случая на FDDI. FDDI, базиран на оптични влакна, предлага значителни предимства, защото максимално допустимата дължина на оптичния кабел при него е много по-голяма от жичната технология – приблизително 200 км, което го прави идеален за свързване на много сгради в един град или както е в случая с GMU, много сгради в района на университета.

Аналогично, ако наблизо има компютърно оборудване от IBM, най-вероятно в него се използва мрежата Token Ring на IBM. Token Ring се използва като алтернатива на Ethernet в някои локални мрежи и предлага същите предимства като FDDI по отношение на постигане на максималната пропускателна способност на преносната среда, но при по-ниска скорост (4 Mbs или 16 Mbps) и по-ниска цена, защото е базирана на обикновени проводници, а не оптични кабели. В Linux мрежата Token Ring се конфигурира точно по същия начин, както Ethernet, затова няма да я разглеждаме специално.

Макар че сега е по-малко вероятно отколкото в миналото, могат да бъдат инсталирани и други LAN технологии като ArcNet и DECNet. Те също се поддържат от Linux, но тук няма да ги разглеждаме.

Много национални мрежи на телекомуникационните компании поддържат протоколи за комутиране на пакети. Може би най-популярен от тях е стандартът, наречен X.25. Много обществени мрежи за предаване на данни като Tymnet в САЩ, Austpac в Австралия и Datex-P в Германия, предлагат тази услуга. X.25 дефинира набор от мрежови протоколи, които описват как апаратурата на терминала за данни, какъвто е един хост, комуникира с апаратурата за предаване на данни (X.25 комутатор). X.25 изисква линия за синхронно предаване на данни и поради това, специален хардуер за синхронен сериен порт. Възможно е да използвате X.25 с обикновени серийни портове, ако разполагате със специално устройство, наречено PAD (Packet Assembler Disassembler). PAD е самостоятелно устройство, което предоставя няколко асинхронни серийни порта и един синхронен сериен порт. То работи по протокола X.25, така че обикновени терминални устройства могат чрез него да осъществят и приемат X.25 връзки.

X.25 често се използва за пренасяне на други мрежови протоколи, например TCP/IP. Тъй като IP дейтаграмите не могат просто да се преобразуват в X.25 дейтаграми (и обратно), те се капсулират в X.25 пакети и се предават през мрежата. Съществува експериментална реализация за Linux на протокола X.25.

Най-новият протокол, предлаган от телекомуникационните компании, се нарича *Frame Relay* (букв. препредаване на кадри). Протоколът Frame Relay има много общи технически характеристики с X.25, но по своето поведение наподобява повече протокола IP. Както X.25, Frame Relay изисква специален синхронен сериен хардуер. Поради тяхното подобие, много карти поддържат и двата протокола. Предлага се алтернатива, която не изисква специален вътрешен хардуер, а отново разчита на независимо външно устройство, наречено FRAD (Frame Relay Access Device – устройство за достъп до Frame Relay), което капсулира на Ethernet пакети във Frame Relay пакети за предаването им през мрежата. Frame Relay е идеален за пренасяне на TCP/IP между сайтове. Linux предоставя драйвери, които поддържат някои видове вградени Frame Relay устройства.

Ако се нуждаете от по-висока скорост на работа в мрежа, която наред с вашите обикновени данни може да пренася много други типове данни като дигитализирани глас и видео, вероятно ще ви заинтересува технологията ATM (Asynchronous Transfer Mode – асинхронен режим на предаване). ATM е нова мрежова технология, която беше специално проектирана, за да се осигурят управляеми, високоскоростни средства за пренасяне на данни с малко забавяне, като се предоставя контрол върху качеството на услугата (Q.S. – Quality of Service). Много телекомуникационни компании разгръщат ATM инфраструктури, защото това позволява съсредоточаването на много различни мрежови услуги върху една платформа, с надеждата да се намалят разходите за управление и поддръжка. ATM често се използва за пренасяне на TCP/IP. В документа Networking-HOWTO можете да намерите информация за поддръжката за ATM под Linux.

Често радио-любителите използват своето радио-оборудване, за да свързват своите компютри в мрежа; обикновено това се нарича *пакетно радио* (*packet radio*). Един от протоколите, използвани от радио-операторите – любители, се нарича AX.25 и е производен (с известни разлики) на X.25. Радио-любителите използват AX.25, за да пренасят TCP/IP и други протоколи. AX.25, както X.25, изисква сериен хардуер с възможност за синхронна работа или външно устройство, наречено "Terminal Node Controller" (контролер за краен въ-

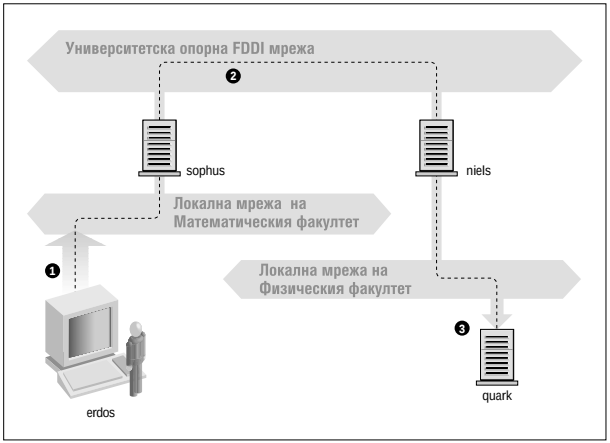
зел), което да превръща пакетите, предавани по асинхронна серийна свръзка в синхронно предавани пакети. Съществува голямо разнообразие от различни интерфейсни карти за осигуряване работата на пакетното радио. Тези карти обикновено се наричат "Z8530 SCC-базирани" и са наречени така по името на най-популярния тип комуникационен контролер, използван в проектите за тази цел. Два от другите протоколи, които обикновено се пренасят от AX.25, са NetRom и Rose, които са протоколи на мрежово ниво. Тъй като тези протоколи работят върху AX.25, те имат същите хардуерни изисквания. Linux поддържа пълноценни версии на протоколите AX.25, NetRom и Rose. Документът AX.25-HOWTO е добър източник на информация за реализацията на тези протоколи в Linux.

Друг начин за достъп до Интернет е избирането на централна система по бавна, но евтина серийна линия (телефон, ISDN и т.н.). Това изисква още един протокол за предаване на пакети, например SLIP или PPP, които ще разгледаме по-късно.

Протоколът IP

Разбира се, едва ли бихте искали вашата мрежова работа да бъде ограничена само до Ethernet или една връзка от тип точка-до-точка. Идеалният вариант е, ако можете да комуникирате с всеки хост, независимо към какъв тип физическа мрежа е свързан той. В по-големи инсталации, например в GMU, обикновено има няколко отделни мрежи, които трябва да бъдат свързани по някакъв начин. В Математическият факултет на GMU работят две Ethernet мрежи – една с бързи машини за преподавателите и аспирантите и друга с бавни машини за студентите. Двете мрежи са свързани към университетската опорна FDDI мрежа.

Тази връзка се управлява от предназначен за целта хост, наречен *шлюз* (*gateway*), който обработва пристигащите и изпращаните пакети, като ги копира между двете Ethernet мрежи и оптичния FDDI кабел. Например, ако сте в Математическия факултет и искате от вашата Linux машина да достигнете **quark** от локалната мрежа на Физическия факултет, мрежовият софтуер не може да изпрати пакетите директно до **quark**, защото тази машина не е в същата локална мрежа. Следователно, софтуерът трябва да използва шлюза като ретранслатор. Шлюзът (наречен **sophus**) препраща тези пакети на съответния шлюз **niels** във Физическия факултет като използва опорната мрежа, а **niels** ги доставя на машината-получател. Потокът данни между **erdos** и **quark** е показан на Фигура 1-1.



Фигура 1-1. Трите стъпки при изпращането на дейтаграма от erdos до quark.

Тази схема за изпращане на данни към отдалечения хост се нарича маршрутизация, а пакетите в този контекст често се наричат *дайтаграми*. За улесняване на нещата, обменът на дейтаграми се управлява от един-единствен протокол, който е независим от използвания хардуер: това е IP или *Интернет протокола*. В Глава 2, *Въпроси на работата в TCP/IP мрежа*, ще разгледаме IP и въпросите на маршрутизирането по-големи подробности.

Главната полза от IP е, че превръща физически различни мрежи в една от гледна точка на потребителя хомогенна мрежа. Това се нарича междумрежова работа, а получената "мета-мрежа" се нарича *интернет*. Забележете тънката разлика между една интернет-мрежа и Интернет. Последното е официалното име на една конкретна глобална интернет-мрежа.

Разбира се, IP изисква и хардуерно независима схема за адресиране. Това се постига чрез задаване на всеки хост на уникален 32-битов номер, наричан *IP адрес*. Един IP адрес обикновено се записва като четири десетични числа, разделени от точки, като всяко число съответства на една 8-битова порция. Например, **quark** може да има IP адрес **0x953C0C04**, който се записва като **149.76.12.4**. Този формат се нарича още десетично-точково означаване, а понякога точкувано

четиризначно означаване. Все по-често този формат се обозначава като IPv4 (от Интернет протокол, Версия 4), защото новият стандарт IPv6 предлага много по-гъвкаво адресиране, както и други модерни възможности. Ще измине може би поне година след излизането на това издание преди IPv6 да влезе във употреба.

Забележете, че сега имаме три различни вида адреси: първият е името на хоста, например **quark**, следват са IP адресите и накрая – хардуерните адреси като 6-байтовия Ethernet адрес. Всички тези адреси трябва да си съответстват по някакъв начин, така че когато въведете *rlogin quark*, мрежовият софтуер да може да получи IP адреса на **quark**, а когато IP доставя данни за локалната мрежа на Физическия факултет, той по някакъв начин трябва да намери какъв Ethernet адрес съответства на IP адреса.

Ще разгледаме тези въпроси в Глава 2. Засега е достатъчно да запомните, че тези стъпки за намиране на адресите се наричат *разпознаване на име на хост* (*hostname resolution*), когато името на хоста се съпоставя с неговия IP адрес и *разпознаване на адрес* (*address resolution*) при намиране на съответствие между IP адрес и хардуерен адрес.

IP през серийни линии

SLIP или *Serial Line IP* (IP през серийна линия) е на практика стандартен протокол за серийни линии. Модификацията на SLIP, наречена CSLIP или *Компресиран SLIP*, извършва компресиране на заглавната част на IP, за да се използва по-добре сравнително ниската пропускателна способност, предоставяна от повечето серийните линии. Друг сериен протокол е PPP или *Point-to-Point Protocol* (протокол точка-до-точка). PPP е по-модерен от SLIP и предоставя редица възможности, които го правят по-привлекателен. Неговото основно преимущество пред SLIP е, че не е ограничен до прехвърлянето на IP дейтаграми, а е проектиран така, че да позволява да се пренася практически всеки протокол.

Протоколът TCP

Изпращането на дейтаграми от един хост до друг не е цялата история. Ако сте влезли в **quark**, бихте искали да имате надеждна връзка между вашия процес *rlogin* на **erdos** и shell-процеса на **quark**. Затова информацията, изпращана между двете машини, трябва да бъде разделена от изпращача на пакети и събрана отново на поток от символи от получателя. На пръв поглед това изглежда тривиално, но изисква решаването на множество сложни задачи.

Много важно нещо, което трябва да се знае за IP е, че той съзнателно не е създаден като надежден протокол. Представете си, че десет души от вашата локална мрежа са започнали да изтеглят последната версия на изходния код на web-браузъра на Netscape от FTP-сървъра на GMU. Количеството на генерирания трафик може да бъде много голямо за станцията-шлюз и тя да не може да се справи с него, защото е много бавна или няма достатъчно памет. Ако в този момент изпратите пакет на **quark**, **sophus** може да няма свободни буфери и затова да не може да препрати пакета. IP разрешава този проблем като просто игнорира пакета и той се губи необратимо. Ето защо, комуникационните хостове са длъжни да проверяват интегритета и пълнотата на данните и да ги предават повторно в случай на грешка.

Тази обработка се извършва от още един протокол, наречен TCP (*Transmission Control Protocol* – протокол за управление на предаването), който изгражда надеждна услуга над IP. Основно свойство на TCP е, че той използва IP, за да създаде илюзията за проста връзка между двата процеса на вашия хост и отдалечената машина, така че вие не трябва да се грижите за това как и по какъв път вашите данни пътуват в действителност. Всъщност, TCP връзката работи като двупосочен канал, от който двата процеса могат да четат и пишат. Можете нагледно да си представите това като телефонен разговор.

TCP идентифицира крайните точки на връзката по IP адресите на двата участващи хоста и номера на *порт* на всеки хост. Портовете могат да се разглеждат като точки на свързване за мрежовите връзки. Ако разширим примера за телефонния разговор още малко и си представим, че градовете са като хостове, можем да сравним IP адресите с регионалните телефонни кодове (където числата съответстват на градовете), а номерата на портовете са локалните кодове (където номерата съответстват на индивидуалните телефонни номера на абонатите). Един отделен хост може да поддържа много различни услуги, всяка от които се определя от номера на нейния порт.

В примера с *rlogin*, клиентското приложение (*rlogin*) отваря порт на **erdos** и се свързва с порт 513 на **quark**, на който знае, че слуша сървърът *rlogind*. Това действие създава TCP връзка. Използвайки тази връзка, *rlogind* извършва идентифицираща процедура и след това стартира нов процес за обвивката. Стандартните вход и изход на обвивката се пренасочват към TCP връзката, така че всичко, което въвеждате в *rlogin* на вашата машина, ще бъде пренесено през TCP потока и подадено на обвивката като стандартен вход.

Протоколът UDP

Разбира се, TCP не е единственият потребителски протокол в TCP/IP мрежите. Макар че е подходящ за приложения като *rlogin*, количеството на използваните служебни данни е недопустимо за приложения като NFS, които вместо това използват сродния на TCP протокол, наречен UDP (*User Datagram Protocol* – потребителски протокол за дейтаграми). Точно като TCP, UDP позволява на приложенията да се свържат с услуга през определен порт на отдалечената машина, но не създава връзка за целта. Вместо това, вие използвате UDP, за да изпратите отделни пакети до получаващата услуга – от там идва и неговото име.

Да предположим, че искате да заявите малко количество данни от сървър за база данни. За създаване на TCP връзка са необходими поне три дейтаграми, други три при всяко изпращане и потвърждаване на малкото количество данни и още три за затваряне на сесията. UDP ни дава възможност да постигнем същото само с две дейтаграми. UDP работи без установяване на постоянна връзка и не изисква от нас да създаваме и затваряме сесия. Ние просто поставяме нашите данни в дейтаграма и я изпращаме на сървъра; сървъра подготвя нейният отговор, поставя данните в дейтаграма, адресирана обратно до нас и я изпраща. Макар че това е както по-бързо, така и по-ефективно от TCP за прости транзакции, UDP не е проектиран да се справя със загуба на дейтаграми. Приложението, например сървъра за имена, трябва да се справя с такива ситуации.

Още за портовете

Портовете могат да се разглеждат като точки на присъединяване за мрежовите връзки. Ако дадено приложение иска да предложи определена услуга, то се прикрепя към порт и очаква клиенти (това се нарича още *слушане* на порт). Всеки клиент, който иска да използва тази услуга, заделя порт на своя локален хост и се свързва към порта на сървъра в отдалечения хост. Същият порт може да бъде отворен на много различни машини, но на всяка машина само един процес може да отвори конкретен порт в даден момент.

Важно свойство на портовете е, че след като бъде установена връзка между клиента и сървъра, друго копие на сървъра може да се прикрепи към порта на сървъра и да слуша за други клиенти. Това свойство позволява, например, множество едновременни отдалечени свързвания към същия хост, всяко от които използва същия порт 513.

TCP е в състояние да различава тези връзки една от друга, защото те идват от различни портове или хостове. Например, ако влезете два пъти в **quark** от **erdos**, първият *rlogin* клиент ще използва локален порт 1023, а вторият ще използва порт 1022. И двата обаче ще се свържат с порт 513 на **quark**. Двете връзки ще се различават с помощта на номерата на портовете на **erdos**.

Този пример показва използването на портовете като места за среща, където клиентът се свързва с определен порт, за да получи конкретна услуга. За да може клиентът да знае правилния номер на порт, трябва да бъде постигнато споразумение между администраторите на двете системи относно назначаването на тези номера. За услуги с широко приложение като *rlogin* тези номера трябва да бъдат администрирани централно. За тази задача отговаря организацията IETF (Internet Engineering Task Force), която периодично издава RFC-документ, наречен *Assigned Numbers* (RFC-1700). Освен други неща, той описва номерата на портове, резервирани за известните услуги. Linux използва файла */etc/services*, в който са описани съответствията между видовете услуги и техните номера.

Макар че и TCP, и UDP връзките разчитат на портове, техните номера са независими. Това означава, че TCP порт 513, например, е различен от UDP порт 513. Всъщност, точно тези портове служат като входни точки за две различни услуги – *rlogin* (TCP) и *rwho* (UDP).

Библиотека за гнезда

В операционната система Unix софтуерът, осъществяващ всички описани по-горе задачи и протоколи, обикновено е част от ядрото. Същото е и при Linux. Най-известният програмен интерфейс в света на Unix е *Berkeley Socket Library* (библиотека за гнезда от Бъркли). Неговото име произлиза от популярната аналогия, която разглежда портовете като гнезда, а свързването към порт – като включване в гнездо. Този интерфейс предоставя системната функция *bind*, с която се задава отдалечен хост, транспортен протокол и услуга, към която програмата може да се свърже или да слуша (като използва *connect*, *listen* и *accept*). Библиотеката за гнезда е нещо по-общо в смисъл, че осигурява не само клас за TCP/IP-базираните гнезда (гнездата *AF_INET*), но и клас, който обработва връзките, които са локалните за машината (класът *AF_UNIX*). Някои реализации поддържат и други класове като протокола XNS (*Xerox Networking System*) или X.25.

В Linux, библиотеката за гнезда е част от стандартната библиотека на C *libc*. Тя поддържа *AF_INET* и *AF_INET6* гнезда за TCP/IP и

AF_UNIX гнезда за Unix домейн. Освен това, библиотеката поддържа *AF_IPX* за мрежовите протоколи на Novell, *AF_X.25* за мрежовия протокол X.25, *AF_ATMPVC* и *AF_ATMSVC* за мрежовия протокол ATM и *AF_X25*, *AF_NETROM* и *AF_NTROSE* гнезда за поддръжка на протокола за любителското (аматьорско) радио. В процес на разработване са други групи протоколи, които ще бъдат включени към библиотеката, след като бъдат завършени.

UUCP Мрежи

UUCP (Unix-to-Unix Copy – копиране от Unix към Unix) започна като пакет от програми, които пренасяха файлове през серийни линии, планираха тези трансфери и инициираха изпълнението на програми на отдалечени сайтове. По пакета бяха извършени основни изменения от момента на първата му реализация в края на седемдесетте години, но все още е твърде спартански от гледна точка на услугите, които предлага. Неговото основно приложение е още в глобалните мрежи, базирани на периодично създавани комутируеми телефонни връзки.

UUCP беше разработен най-напред от Bell Laboratories през 1977 г. за комуникация между техните Unix центрове за разработки. В средата на 1978 г. тази мрежа вече свързваше над 80 центъра. Тя поддържаше като приложение електронната поща, както и отдалечен печат. Все пак, основното предназначение на системата беше разпространението на нов софтуер и поправки на програмни грешки. Пакетът UUCP не е огрничен само за Unix обкръжение. Съществуват безплатни и комерсиални адаптации за разнообразие от платформи, включително AmigaOS, DOS и TOS на Atari.

Един от основните недостатъци на UUCP мрежите е, че те работят на пакетни задания. Вместо да предоставят постоянна връзка между хостовете, те използват временни връзки. Възможно е един UUCP хост да се свърже с друг UUCP хост само веднъж на ден и то само за кратък период от време. По време на тази връзка се предават всичките новини, електронна поща и файлове, които са поставени в опашката на хоста и след това връзката се прекъсва. Именно поставянето на заявките в опашка ограничава типовете приложения, за които може да се използва UUCP. В случая с електронната поща, потребителят може да подготви едно съобщение и да го изпрати. Съобщението ще стои в опашката на UUCP хоста, докато той позвъни на друг UUCP хост, за да предаде съобщението. Това е достатъчно за мрежо-

ви услуги като електронната поща, но е неприемливо за услуги като *rlogin*.

Независимо от тези ограничения, все още съществуват много UUCP мрежи, работещи по целия свят и поддържани предимно от ентусиасти, които предлагат на частните потребители мрежов достъп на разумни цени. Основната причина за дълговременната популярност на UUCP беше, че тя е много по-евтина в сравнение с директния достъп до Интернет. За да направите вашия компютър UUCP възел, всичко от което се нуждаете, е един модем, работеща реализация на UUCP и друг UUCP възел, съгласен да ви подава новини и поща. Много хора бяха готови да осигурят UUCP връзка на частните потребители, защото тези връзки не поставят високи изисквания към тяхната съществуваща мрежа.

В тази книга разглеждаме конфигурирането на UUCP в отделна глава, но няма да се съсредоточаваме много върху този пакет, защото сега, когато евтиният достъп до Интернет стана нещо обикновено навсякъде по света, UUCP бързо се измества от TCP/IP.

Работа в мрежа под Linux

Тъй като е резултат от съгласуваните усилия на програмисти от целия свят, Linux не би бил възможен без наличието на глобалната мрежа. Ето защо не е изненадващо, че още в ранните етапи от неговото развитие, много хора започнаха да работят по осигуряването му с мрежови възможности. Почти от самото начало съществуваше адаптация на UUCP за Linux, а работата по TCP/IP-базирана мрежа започна около есента на 1992 г., когато Ross Biro и други създадоха това, което сега е известно като Net-1.

След като Ross спря да участва в активното разработване през май 1993 г., Fred van Kempen започна да работи върху нова реализация, пренаписвайки основни части от кода. Този проект беше известен като Net-2. Първото публично издание – Net-2d – беше направено през лятото на 1993 г. (като част от ядрото 0.99.10) и от тогава беше поддържано и разширявано от много хора, като специално трябва да отбележим Alan Cox*. Първоначалната работа на Alan беше известна като Net-2Debugged. След множество корекции на грешки и подобрения върху кода, той промени своето име на Net-3, след издаването

* Alan може да бъде намерен на адрес *alan@lxorguk.ukuu.org.uk*

на Linux 1.0. Кодът на Net-3 след това беше доразвит за Linux 1.2 и Linux 2.0. Ядрата 2.2 и по-новите версии използват мрежовата поддръжка на Net-4, която остана официалния стандарт, предлаган днес.

Мрежовият код на Net-4 за Linux предлага драйвери за голямо разнообразие от устройства и други разширени възможности. Стандартните протоколи на Net-4 включват SLIP и PPP (за предаване на мрежов трафик през серийни линии), FLIP (за паралелни линии), IPX (за съвместимис Novell мрежи, които ще разгледаме в Глава 15, *IPX и файловата система NCP*), Appletalk (за мрежи на Apple) и AX.25, NetRom и Rose (за мрежи на радио-любители). Други стандартни възможности на Net-4 включват IP защитна стена, IP счетоводство (разгледани по-късно в Глава 9, *Защитна стена за TCP/IP* и Глава 10, *IP счетоводство*), както и IP маскиране (разглежда се в Глава 11, *IP маскиране и транслиране на мрежови адреси*). Поддържа се IP тунелиране по няколко различни начина и усъвършенствана система за маршрутизиране. Поддържат се много голям брой Ethernet устройства, а освен тях някои FDDI, Token Ring, Frame Relay, ISDN и ATM карти.

Освен това, съществуват много други възможности, които силно подобряват гъвкавостта на Linux. Тези възможности включват реализация на файловата система SMB, наречена Samba, която взаимодейства с приложения като *lanmanager* и Microsoft Windows, написана от Andrew Tridgell, както и версия на Novell NCP (NetWare Core Protocol)*.

Различни линии на развитие

В различни периоди са полагани разнообразни усилия за разработване на мрежовия софтуер под Linux.

След като Net-2Debugged беше обявена за официалната реализация на мрежа под Linux, Fred продължи работата по нея. Тази разработка доведе до Net-2e, която имаше доста променен дизайн на мрежовия слой. Fred работеше по създаването на стандартизиран интерфейс за драйвери (DDI – Device Driver Interface), но работата по Net-2 вече приключи.

Още една реализация на TCP/IP мрежа беше направена от Matthias Urlichs, който написа ISDN драйвер за Linux и FreeBSD. За този

* NCP е протоколът, на който са базирани файловите и услугите за печат на Novell

драйвер той интегрира част от мрежовия код на BSD в ядрото на Linux. Този проект също повече не се разработва.

Бяха извършени множество промени в мрежовата реализация в ядрото на Linux, но промяната все още е ключова дума, защото разработката продължава. Понякога това означава, че трябва да се извършват изменения и в друг софтуер, например в средствата за конфигуриране на мрежата. Въпреки че това вече не е толкова голям проблем, както някога, все още е възможно да откриете, че да извършите upgrade на ядрото си до най-новата версия означава, че трябва да подновите и инструментите за конфигуриране на мрежата. За щастие при повечето разпространени днес дистрибуции на Linux, това е проста задача.

Мрежовата реализация Net-4 е зрял продукт и се използва в голям брой сайтове по света. Беше извършена много работа за подобряване на възможностите на реализацията Net-4 и днес тя може да се конкурира с най-добрите реализации за съответните хардуерни платформи. Linux се разпространява бързо сред доставчиците на Интернет услуги и често се използва за изграждане на евтини и надеждни web-сървъри, пощенски сървъри и сървъри за новини за този тип организации. Съществува достатъчен интерес по разработката на Linux, благодарение на който Linux е в крак с най-съвременните тенденции в мрежовите технологии, а текущите версии на ядрото предлагат следващата генерация на протокола IP – IPv6 – като стандартна възможност.

Откъде да вземем кода

Днес изглежда странно като си спомним, че в ранните дни на разработката на мрежовия код за Linux, стандартното ядро изискваше огромен допълващ пакет, за да се добави мрежова поддръжка. Днес, разработването на мрежовите възможности се осъществява като част от основния процес по разработването на ядрото на Linux. Последните стабилни ядра на Linux могат да бъдат намерени на сървъра **ftp.kernel.org** в директорията */pub/linux/kernel/v2x/*, където *x* е четно число. Последните експериментални ядра на Linux могат да бъдат намерени на **ftp.kernel.org** в */pub/linux/kernel/v2y/*, където *y* е нечетно число. Има огледални образи на кода на ядрото на Linux по целия свят. Сега е трудно да си представим Linux без стандартна поддръжка на работа в мрежа.

Поддържане на системата

В тази книга главно ще се занимаваме предимно с въпросите на инсталирането и конфигурирането. Администрирането обаче е нещо повече от това – след като сте настроили една услуга, трябва и да я поддържате работеща. В повечето случаи са достатъчни съвсем малко грижи, но някои услуги като пощата и новините изискват постоянно да изпълнявате рутинни задачи, за да поддържате системата си актуална. Ще разгледаме тези задачи в следващите глави.

Абсолютният минимум при поддръжката е редовно да се проверяват системните дневници (log-файлове) и дневниците на приложенията за грешки и необикновени събития. Често за целта се създават няколко административни shell-скрипта, които се стартират периодично от *cron*. Дистрибуциите с изходен код на някои основни приложения като *inn* и C News съдържат такива скриптове. Вие трябва само да ги настроите според вашите нужди и предпочитания.

Отпечатаният текст от всяка от вашите *cron*-задачи трябва да бъде изпратен с e-mail на административен акаунт. По подразбиране много приложения изпращат доклади за грешки, статистика на използването си или обобщения на log-файла до акаунта **root**. Това е достатъчно само, ако често влизате като **root**; много по-добра идея е пощата на **root** да се препраща към вашия личен акаунт. За целта трябва да се конфигурират пощенски псевдоними, както е описано в Глава 19, *Конфигуриране и използване на exim* и Глава 18, *Sendmail*.

Колкото и внимателно да сте конфигурирали вашия сайт, законът на Мърфи гарантира, че рано или късно *ще* се появи някакъв проблем. Ето защо, да поддържаш система, означава също да бъдеш абониран за оплаквания. Обикновено, хората очакват системният администратор да може да бъде достигнат поне с e-mail до *root*, но има и други адреси, които обикновено се използват за комуникация с лицето, отговарящо за специфични аспекти от поддръжката на системата. Например, оплаквания за неработеща пощенска конфигурация обикновено се адресират до *postmaster*, а проблемите със системата за новини могат да бъдат докладвани на *newsmaster* или *usenet*. Писмата до *hostmaster* трябва да бъдат препращани към човека, отговарящ за базовите мрежови услуги на хоста и сървъра за DNS имена, ако поддържате такъв сървър.

Сигурност на системата

Друга много важна страна на системното администриране в мрежова среда е защитата на системата и потребителите ѝ от нарушители. Безгрижно управляваните системи предлагат на злонамерените хора много цели. Атаките варират от отгатване на паролите до подслушване на мрежата, а нанесените поражения започват от подправяне на писма, минават през загуба на данни и стигат до нарушаване на личната неприкосновеност на потребителите. Ще се спрем на някои конкретни проблеми, когато разглеждаме контекста, в който те могат да възникнат и ще дадем някои стандартни мерки за защита против тях.

В този раздел ще разгледаме няколко примера и основни техники за осигуряване на сигурност на системата. Разбира се, с разгледаните тук теми не обхващаме подробно всички проблеми на сигурността, с които можете да се сблъскате. Те служат само като илюстрация на проблемите, които могат да възникнат. Ето защо, прочитането на хубава книга, специално посветена на сигурността, е абсолютно наложително, особено когато става въпрос за мрежови системи.

Сигурността на системата започва с доброто администриране. Това включва проверка на собствеността и правата за достъп до всички важни файлове и директории и наблюдаване на използването на привилегированите акаунти. Програмата COPS, например, ще провери вашата файлова система и общите конфигурационни файлове за нетипични права за достъп или други аномалии. Освен това е разумно да се използва софтуер за управление на паролите, налагащ определени изисквания към паролите на потребителите, които ги правят потрудни за отгатване. Например пакета shadow password изисква паролата да се състои от поне пет знака и да съдържа едновременно главни и малки букви или цифри, както и други знаци.

Когато правите дадена услуга достъпна през мрежата, погрижете се да ѝ дадете "най-ниски привилегии". Не ѝ давайте права да върши неща, които не се изискват от нея по проект. Например, трябва да давате право на програмите да работят с привилегиите на **root** (setuid root) или някой друг привилегирован акаунт само, когато това е необходимо. Освен това, ако искате да използвате дадена услуга само за строго определено приложение, не се колебайте да я конфигурирате толкова ограничаващо, колкото позволява вашето конкретно приложение. Например, ако искате да разрешите на бездискови хостове да зареждат операционна система от вашата машина, трябва да предоставите протокола *TFTP* (*Trivial File Transfer Protocol* – тривиален

протокол за прехвърляне на файлове), за да могат те да изтеглят основни конфигурационни файлове от директорията */boot*. Ако се използва без ограничение обаче, TFTP позволява на потребители от цял свят да изтеглят всеки достъпен за четене файл от вашата система. Ако не искате това, ограничете услугата TFTP само до директорията */boot*.*

Можете също да ограничите достъпа до конкретни услуги само за потребители от определени хостове, например от вашата локална мрежа. В Глава 12 ви запознаваме с демона *tcpd*, който прави това за различни мрежови приложения. По-сложни методи за ограничаване на достъпа до определени хостове или услуги ще бъдат разгледани в Глава 9.

Друг важен момент е избягването на "опасен" софтуер. Разбира се, всеки софтуер, който използвате, може да бъде опасен, защото в софтуера може да има грешки, които умели хора могат да използват, за да получат достъп до вашата система. Такива неща се случват и срещу тях няма сигурна защита. Този проблем засяга както свободния софтуер, така и комерсиалните продукти.+ Все пак, програми, които изискват специални привилегии са значително по-опасни, отколкото останалите, защото всеки пропуск може да има драстични последствия.# Ако инсталирате setuid-програма за мрежови цели, бъдете двойно по-внимателни и проверете документацията, за да не създадете неволно пробив в сигурността.

Друг източник на проблеми могат да бъдат програмите, които позволяват влизане в системата или изпълнение на команди, като извършват ограничена проверка на самоличността. Командите *rlogin*, *rsh* и *rexec* са много полезни, но предлагат много малка възможност за идентифициране на викащата страна. Удостоверяването на самоличността се основава на доверяване на името на викащия хост, получено от сървъра за имена (ще разгледаме тези сървъри по-късно), което може да бъде фалшифицирано. Днес е стандартна практика *r-*

---

* Ще се върнем отново на този въпрос в Глава 12, *Важни мрежови възможност.*

\+ Съществуват комерсиални версии на Unix (за които трябва да платите доста пари), които се доставят със setuid-**root** shell-скрипт, който позволява на потребителите да получат **root** привилегии с помощта на прост стандартен трик

\# През 1988 г., червеят RTM спря работата на голяма част от Интернет, отчасти като използва пропуски в сигурността на някои програми, включително *sendmail*. Тези пропуски са отстранени отдавна.

командите да се забраняват изцяло и да се заместват с комплекта инструменти *ssh*, който използват много по-надежден метод за идентификация и освен това осигурява допълнителни услуги като шифриране и компресиране.

Никога не трябва да изключвате вероятността вашите предпазни мерки да бъдат преодолени, независимо от това колко внимателни сте били. Затова, първо трябва да направите необходимото, за да откривате нарушителите навреме. Проверката на системните дневници е добра изходна точка, но нарушителят вероятно ще бъде достатъчно умен, за да предвиди това действие и да изгрие всички очевидни следи, които е оставил. Все пак, съществуват инструменти като *tripwire* от Gene Kim и Gene Spafford, които ви дават възможност да проверявате важните системни файлове и да установите дали тяхното съдържание или права за достъп са променяни. *tripwire* изчислява различни мощни контролни суми за тези файлове и ги записва в база данни. При следващите изпълнения на програмата, контролните суми се преизчисляват и сравняват със записаните суми, за да се открие дали са извършвани промени.

ГЛАВА 2

ВЪПРОСИ НА РАБОТАТА В TCP/IP МРЕЖА

В тази глава ще разгледаме конфигурационните решения, които трябва да вземете, когато свързвате вашата Linux машина към TCP/IP мрежа, в това число избирането на IP адрес, име на хост и въпросите на маршрутизирането. Тази глава ви дава основата, от която се нуждаете, за да разберете какво изисква конфигурирането, а следващите глави описват средствата, които ще използвате за целта.

Можете да научите повече за комплекта протоколи TCP/IP и принципите в него от тригомника *Internetworking with TCP/IP* от Douglas R. Comer (издание на Prentice Hall). За по-детайлно ръководство по управлението на TCP/IP мрежа, вижте *TCP/IP Network Administration* от Craig Hunt (издание на O'Reilly).

Мрежови интерфейси

За да скрие разнообразието в оборудването, което може да се използва в мрежови среди, TCP/IP дефинира абстрактен *интерфейс*, който се използва за достъп до хардуера. Този интерфейс предлага комплект от операции, които са едни и същи за всички видове хардуер и се използват основно за изпращане и приемане на пакети.

За всяко периферно мрежово устройство в ядрото трябва да съществува съответния интерфейс. Например, Ethernet интерфейсите в Linux се наричат с имена като *eth0* и *eth1*; PPP-интерфейсите (разглеждат се в Глава 8, *Протоколът PPP*) са *ppp0* и *ppp1*, а на FDDI-интерфейсите се дават имена като *fddi0* и *fddi1*. Тези имена на интерфейси се използват само за нуждите на конфигурирането, когато искате да

посочите конкретно физическо устройство в дадена конфигурационна команда, но нямат значение извън тази употреба.

Преди да бъде използван за работа в TCP/IP мрежа, на интерфейса трябва да бъде зададен IP адрес, който служи за идентифицирането му при комуникиране с останалия свят. Този адрес е различен от името на интерфейса, което споменахме по-горе. Ако сравните интерфейса с врата, адресът е като табелката с името, поставена на нея.

Могат да бъдат зададени и други параметри на устройството, например максималната големина на дейтаграма, която може да бъде обработена от конкретен хардуер. Тази големина се обозначава като *MTU* (съкращение от *Maximum Transfer Unit* – максимална единица за предаване). По-късно ще ви запознаем и с други атрибути. За щастие, повечето атрибути имат разумни подразбиращи се стойности.

IP Адреси

Както споменахме в Глава 1, *Въведение в работата в мрежа*, мрежовият протокол IP възприема адресите като 32-битови числа. На всяка машина трябва да бъде зададен уникален в мрежовата среда номер.* Ако поддържате локална мрежа, която няма TCP/IP трафик с други мрежи, можете да зададете тези номера според личните си предпочитания. Съществуват няколко интервали от IP адреси, които са резервирани за такива частни мрежи. Тези интервали са посочени в Таблица 2-1. За сайтове в Интернет, обаче, номерата се определят от централен пълномощен орган, наречен *NIC* (*Network Information Center* – мрежов информационен център)+.

IP адресите се разделят за четивност на четири осембитови числа, наречени *октети*. Например, **quarkphisycs.grouchо.edu** има IP адрес

---

* Най-често използваната в Интернет версията на IP Протокола е Версия 4. Бяха положени много усилия за разработването на заместител, наречен IP Версия 6. IPv6 използва различна схема за адресация и по-дълги адреси. Съществува реализация на IPv6 за Linux, но тя още не беше готова за документиране, когато подготвяхме това издание. Поддръжката на IPv6 в ядрото на Linux е добра, но освен това е необходимо да бъдат модифицирани и голям брой мрежови приложения.

\+ Най-често IP адресите се задават от доставчика, от когото купувате IP достъп. Въпреки това, можете да се обърнете и директно към NIC, за да получите IP адрес за вашата мрежа, като изпратите e-mail до *hostmaster@internic.net* или чрез формата на адрес *http://www.internic.net/*

0x954C0C04, който се записва като **149.76.12.4**. Този формат често се нарича *точкувано четиризначно означаване* (*dotted quad notation*).

Друга причина за това означаване е, че IP адресите се разделят на *мрежов* номер, който се съдържа в левите октети, и номер на *хост*, който е останалата част от адреса. Когато се обърнете към NIC за IP адреси, не получавате адрес за всеки отделен хост, който планирате да използвате. Вместо това, получавате номер на мрежа и правото да задавате адреси на хостовете от вашата мрежа според предпочитанията си, като използвате всички валидни IP адреси в обхвата на мрежата.

Размерът на частта, предназначена за хостовете, зависи от размера на мрежата. За задоволяване на различните нужди бяха създадени няколко класа мрежи, които дефинират местата, на които се разделя IP адреса. Класовете мрежи са описани по-долу:

Клас A

Клас A включва мрежите от **1.0.0.0** до **127.0.0.0**. Номерът на мрежата се съдържа в първия октет. Този клас осигурява 24-битова част за номер на хост, позволяваща всяка мрежа да съдържа около 1,6 милиона хоста.

Клас B

Клас B обхваща мрежите от **128.0.0.0** до **191.255.0.0**. Номерът на мрежата е в първите два октета. Този клас дефинира 16,320 мрежи с по 65,024 хоста във всяка.

Клас C

Мрежите от клас C са от **192.0.0.0** до **223.255.255.0**, като номерът на мрежата се съдържа в първите три октета. Този клас съдържа около 2 милиона мрежи с по 254 хоста.

Класове D, E и F

Адресите, попадащи в обхвата от **224.0.0.0** то **254.0.0.0** са или експериментални, или са запазени за използване за специални цели и не задават номер на мрежа. Например услугата IP Multicast, която позволява да се предават данни едновременно до много точки от една интернет, използва адреси от този обхват.

Ако се върнем към примера в Глава 1, ще видим, че **149.76.12.4**, адресът на **quark**, указва хост **12.4** от мрежата от клас B **149.76.0.0**.

Може би сте забелязали, че не всички възможни стойности в горния списък бяха разрешени за всеки октет в частта за хоста. Това е така, защото октетите **0** и **255** са запазени за специални цели. Адрес, в който всички битове в хост-частта са 0, указва мрежата, а адрес, в който всички битове в частта за хоста са 1, се нарича *broadcast адрес*. С този адрес се посочват едновременно всички хостове в дадена мрежа. Например, **149.76.255.255** не е валиден адрес на хост, а се използва за посочване на всички хостове в мрежата **149.76.0.0**.

Известен брой мрежови адреси са запазени за специални цели. Два такива адреса са **0.0.0.0** и **127.0.0.0**. Първият се нарича *маршрут по подразбиране*, а вторият е т.нар. *loopback адрес (адрес-примка)*. Подразбиращият се маршрут се използва при определяне на пътя, по който IP препраща дейтаграми.

Мрежата **127.0.0.0** е запазена за IP трафик, който е локален за вашия хост. Обикновено, адресът **127.0.0.1** се задава на специален интерфейс, наречен примка (*loopback*), който работи като затворена верига. Всеки IP пакет, който се изпрати през този интерфейс от TCP или UDP, ще бъде върнат към тях, все едно, че току-що пристига от някоя мрежа. Това ви позволява да разработвате и тествате мрежов софтуер, без въобще да използвате "истинска" мрежа. Освен това, мрежата-примка видава възможност да използвате мрежов софтуер на самостоятелен хост. Това не е толкова необичайно, колкото изглежда на пръв поглед; например, много UUCP-сайтове изобщо нямат IP връзка, но въпреки това на тях работи системата за новини INN. За правилна работа под Linux, INN изисква loopback интерфейс.

Някои адресни интервали от всеки от мрежовите класове са отделени и обозначени като "запазени" или "частни". Тези адреси са резервирани за използване от частни мрежи и не се маршрутизират в Интернет. Обикновено те се използват от организации, изграждащи своя собствена вътрешна мрежа, но дори и малки мрежи администраторите ги смятат за полезни. Запазените мрежови адреси са показани на Таблица 2-1.

Таблица 2-1. Интервали на IP адреси, запазени за частна употреба.

Клас	Мрежи
A	10.0.0.0 до 10.255.255.255
B	172.16.0.0 до 172.31.0.0
C	192.168.0.0 до 192.168.255.0

Разпознаване на адреси

След като видяхте как се образуват IP адресите, вероятно се питате как те се използват в една Ethernet или Token Ring мрежа за адресиране на отделните хостове. В крайна сметка, тези протоколи имат свои собствени адреси за обозначаване на хостовете, които нямат абсолютно нищо общо с един IP адрес, нали така? Правилно.

За съпоставяне на IP адресите с адресите на физическата мрежа е необходим механизъм. Използваният механизъм се нарича *ARP* (съкращение от *Address Resolution Protocol* – протокол за разпознаване на адреси). Всъщност, ARP не е ограничен до Ethernet и Token Ring и се използва и с други типове мрежи, например протокола AX.25 за аматьорско радио. Идеята, лежаща в основата на ARP е това, което правят повечето хора, когато трябва да намерят г-н Х в тълпа от 150 души: човекът, който го търси, вика достатъчно високо, така че всеки да го чуе и очаква г-н Х да отговори, ако е там. Когато отговори, научаваме кой е той.

Когато ARP иска да определи Ethernet адреса, съответстващ на даден IP адрес, той използва една възможност на Ethernet, наречена *broadcasting* (предаване до всички), при която дейтаграмата се адресира едновременно до всички станции в мрежата. Broadcast-дейтаграмата, изпратена от ARP, съдържа запитване за IP адреса. Всеки хост, който получи това запитване, сравнява търсения адрес със своя собствен IP адрес и ако двата съвпадат, връща ARP отговор на питащия хост. След получаване на отговора, питащия хост може да извлече Ethernet-адреса на изпращача от неговия отговор.

Може би се сега се питате, как един хост може да достигне до Интернет адрес, който се намира в мрежа на другия край на света. Отговорът на този въпрос се съдържа в маршрутизацията, с която именно се намира физическото място на един хост в мрежата. Ще разгледаме този въпрос в следващия раздел.

Сега нека поговорим още малко за ARP. След като хоста открие един Ethernet адрес, той го запазва в своя ARP кеш, така че да не се налага да го търси отново следващия път, когато иска да изпрати дейтаграма на въпросния хост. Все пак, не е много разумно тази информация да се пази завинаги; Ethernet картата на отдалечения хост може да бъде заменена поради технически проблеми и информацията в ARP записа ще стане невалидна. Ето защо, записите в ARP кеша се анулират след определено време, което инициира нова процедура за определяне на IP адреса.

Понякога е необходимо да се намери IP адреса, съответстващ на даден Ethernet адрес. Това се случва например, когато бездискова машина иска да зареди операционна система от сървър в мрежата, което е често срещана ситуация в локалните мрежи. Бездисковият клиент, обаче, практически няма информация за самия себе си, освен своя Ethernet адрес! Затова, той изпраща broadcast съобщение, съдържащо заявка към сървъра за зареждане да му предостави IP адрес. В тази ситуация се използва друг протокол, наречен *RARP (Reverse Address Resolution Protocol* – протокол за обратно разпознаване на адреси). Заедно с протокола BOOTP, той служи за дефиниране на процедурата за зареждане на операционна система от бездисковите клиенти в мрежата.

IP маршрутизация

Сега ще се заемем с въпроса за намирането на хоста, към който отиват дейтаграмите според техния IP адрес. Различните части от адреса се обработват по различни начини; вашата работа е да настроите файловете, които указват как да се обработва всяка част.

IP мрежи

Когато пишете писмо до някого, обикновено поставяте пълния адрес върху плика, определяйки страната, щата и пощенския код. След като пуснете писмото в пощенската кутия, пощенската служба ще го достави на неговия получател: то ще бъде изпратено в посочената страна, където националната пощенска служба ще го препрати в съответния щат и район. Предимството на тази йерархична схема е очевидно: където и да пуснете писмото, местната пощенска служба знае приблизително в каква посока да го изпрати, но не се интересува по какъв път ще пътува то, след като достигне страната, за която е предназначено.

IP мрежите са структурирани по подобен начин. Целият Интернет се състои от множество независими мрежи, наричани *автономни системи*. Всяка система извършва вътрешна маршрутизация между своите хостове, така че задачата за доставяне на една дейтаграма се свежда до намиране на път до мрежата на получаващия хост. След като дейтаграмата пристигне в *произволен* хост от тази конкретна мрежа, по-нататъшната обработка се извършва изключително от самата мрежа.

Подмрежи

Тази структура се отразява чрез разделяне на IP адреса на мрежова част и хост-част, както споменахме по-горе. По подразбиране, получаващата мрежа се извлича от мрежовата част на IP адреса. Затова, хостове с еднакви IP *мрежови* номера трябва да се намират в една и съща мрежа.*

Има смисъл да се приложи подобна схема и *вътре* в мрежата, защото тя може да се състои от стотици по-малки мрежи, като най-малките единици са физически мрежи като Ethernet. Затова IP ви позволява да разделите една IP мрежа на множество *подмрежи*.

Една подмрежа се грижи за доставянето на дейтаграми до определен диапазон от IP адреси. Това е разширение на концепцията за разделяне на битовите полета, както е при класовете A, B и C. Мрежовата част обаче сега е разширена като включва няколко бита от частта на хоста. Броят на битовете, които се интерпретират като номер на подмрежата, се дава от така наречената *маска на подмрежата* (*subnet mask*) или *мрежова* маска (*netmask*). Това също е 32-битово число, което определя битовата маска за мрежовата част на IP адреса.

Университетската мрежа на GMU е пример за такава мрежа. Тя има номер на мрежа от клас B - **149.76.0.0**, следователно нейната мрежова маска е **255.255.0.0**.

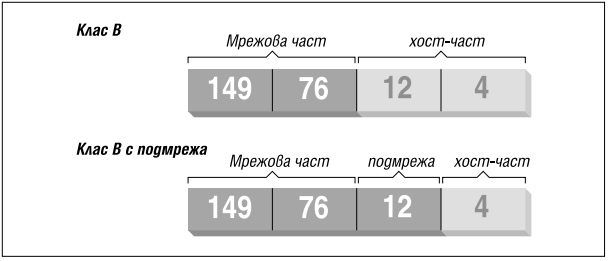
Вътрешно, университетската мрежа на GMU се състои от множество по-малки мрежи като различните факултетски локални мрежи. Затова, обхвата на IP адресите е разделен на 254 подмрежи – от **149.76.1.0** до **149.76.254.0**. Например, Факултетът по Теоретична физика използва мрежа с номер **149.76.12.0**. Университетската опорна мрежа е самостоятелна мрежа с номер **149.76.1.0**. Тези подмрежи използват съвместно един и същи номер на IP мрежа, но третият октет се използва, за да се разграничават помежду си. Следователно, тяхната маска на подмрежа е **255.255.255.0**.

На Фигура 2-1 е показано как **149.76.12.4**, адресът на **quark**, се интерпретира различно, когато се възприема като обикновена мрежа от клас B, и когато се използват подмрежи.

Няма никакво значение, че техниката за генериране на подмрежи е само едно *вътрешно деление* на мрежата. Подмрежите се създават от

* Автономните системи са малко по-общи. Те могат да се състоят от повече от една IP мрежа.

собственика на мрежата (или администраторите). Често подмрежите се създават, за да отразят съществуващи граници, били те физически (между две Ethernet мрежи), административни (между два факултета) или географски (между два района), а правата над всяка подмрежа се делегират на някой представител. Тази структура обаче се отразява само на вътрешното поведение на мрежата и е напълно невидима за външния свят.



Фигура 2-1. Разделяне на мрежа от клас B на подмрежи.

Шлюзове

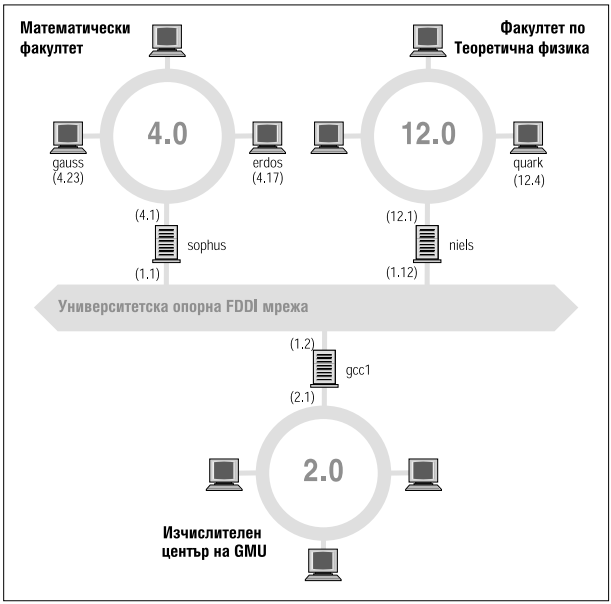
Разделянето на подмрежи е не просто от полза на организациите; често то е естествен резултат от хардуерни граници. Видимата област на един хост от дадена физическа мрежа, например Ethernet, е много ограничен: той може да разговаря само с хостове от неговата мрежа. Всички други хостове могат да бъдат достигнати само чрез специално предназначени за целта машини, наречени *шлюзове* (gateway). Шлюзът е хост, който е свързан едновременно към две или повече физически мрежи и е конфигуриран да препраща пакети между тях.

На фигура 2-2 е показана част от мрежовата топология на GMU. Хостове, които са едновременно в две подмрежи, са показани с двата си адреса.

Физически различните мрежи трябва да принадлежат на различни IP мрежи, за да може IP да разпознае дали хостът е в локалната мрежа. Например, номерът на мрежа **149.76.4.0** е запазен за хостове от локалната мрежа на Математическия факултет. Когато изпраща дейтаграма на **quark**, мрежовият софтуер на **erdos** незабавно вижда от IP

адреса **149.76.12.4**, че получателят е във физически различна мрежа и поради това може да бъде достигнат само чрез шлюз (по подразбиране **sophus**).

Самият **sophus** е свързан към две различни подмрежи: тази на Математическия факултет и университетската опорна мрежа. Достъпът му до всяка от тях е през различен интерфейс – съответно *eth0* и *fddi0*. И сега, какъв IP адрес трябва да му зададем? В коя подмрежа трябва да е хоста – в **149.76.1.0** или в **149.76.4.0**?



Фигура 2-2. Част от мрежовата топология на GMU

Отговорът е "и в двете". На **sophus** са зададени адресите **149.76.1.1** за работа в мрежата **149.76.1.0** и **149.76.4.1** за работа в мрежата **146.76.4.0**. На шлюза трябва да се зададе по един IP адрес за всяка мрежа, към която принадлежи. Тези адреси – заедно със съответната

мрежова маска – са свързани с интерфейса, през който се извършва достъпа до подмрежата. Затова, съответствията между интерфейса и адресите за **sophus** биха изглеждали например така:

Интерфейс	Адрес	Мрежова маска
eth0	149.76.4.1	255.255.255.0
fddi0	149.76.1.1	255.255.255.0
lo	127.0.0.1	255.0.0.0

Последният ред описва loopback интерфейса *lo*, за който говорихме по рано.

В общия случай можете да игнорирате малката разлика между свързването на адрес към хост или към негов интерфейс. За хостове, които са само в една мрежа (като **erdos**), можете да обозначавате хоста с еди-кой-си IP адрес, въпреки че, ако говорим стриктно, Ethernet интерфейса е този, който има въпросния IP адрес. Разграничаването е действително важно само, когато става въпрос за шлюз.

Таблица за маршрутизиране

Сега ще насочим вниманието си върху това, как IP избира шлюз, който да използва за доставяне на дейтаграми до отдалечена мрежа.

Видяхме, че когато **erdos** изпраща дейтаграма за **quark**, проверява адреса на получателя и определя, че той не е в локалната мрежа. Ето защо **erdos** изпраща дейтаграмата на подразбиращия се шлюз **sophus**, който сега е поставен пред същата задача. **sophus** определя, че **quark** не е свързан към нито една от мрежите, с които този шлюз е свързан директно, така че трябва да намери друг шлюз, чрез който да препрати дейтаграмата. Правилният избор би бил **niels**, шлюзът към локалната мрежа на Физическия факултет. Затова **sophus** се нуждае от информация, за да асоциира мрежата на получателя с подходящ шлюз.

За тази цел IP използва таблица, която съдържа съответствия между мрежите и шлюзовете, чрез които те могат да бъдат достигнати. Тази таблица трябва да съдържа и един универсален запис (*маршрут по подразбиране*); това е шлюз, който е асоцииран с мрежа **0.0.0.0**. Всички адреси съответстват на този маршрут, защото не е необходимо да съвпада нито един от 32-та бита. Затова, пакетите към една неиз-

вестни мрежи се изпращат по подразбиращия се маршрут. Таблицата за маршрутизиране на **sophus** може да изглежда например по този начин:

Мрежа	Мрежова маска	Шлюз	интерфейс
149.76.1.0	255.255.255.0	-	*fddi0*
149.76.2.0	255.255.255.0	149.76.1.2	*fddi0*
149.76.3.0	255.255.255.0	149.76.1.3	*fddi0*
149.76.4.0	255.255.255.0	-	*eth0*
149.76.5.0	255.255.255.0	149.76.1.5	*fddi0*
...	...	...	...
0.0.0.0	0.0.0.0	149.76.1.2	*fddi0*

Ако вие се налага да използвате маршрут до мрежа, към която **sophus** е свързан директно, не се нуждаете от шлюз; затова колоната на шлюза в такъв случай съдържа тире.

Процесът за определяне дали даден адрес съответства на конкретен маршрут е математическа операция. Този процес е много прост, но изисква познаването на двоичната аритметика и логика. Един маршрут съответства на получателя, ако след извършване на Логическо И (AND) между адреса на мрежа и мрежовата маска, резултатът е равен точно на резултата от Логическо И между адреса на получателя и мрежовата маска.

Превод: маршрутът може да се използва, ако битовете от адреса на мрежа, задавани от мрежовата маска (започвайки от най-левия бит от първия байт на адреса) съвпадат с аналогичните битове в адреса на получателя.

Когато реализацията на IP търси най-добрия маршрут до получателя, тя може да намери записи за няколко маршрута, които съответстват на крайната цел. Например, знаем, че подразбиращия се маршрут съответства на всеки получател, но за дейтаграми, предназначени за локално свързани мрежи може да се използва и техния локален маршрут. Как IP решава кой маршрут да използва? Именно тук мрежовата маска играе важна роля. Макар че и двата маршрута водят до получателя, единият маршрут има по-дълга мрежова маска от другия. По-горе споменахме, че мрежовата маска се използва за разделяне на нашето адресно пространство на малки мрежи. Колкото по-

дълга е мрежовата маска, толкова по-добре съвпада с адреса на получателя; когато препращаме дейтаграми, трябва винаги да избираме маршрута, който има най-дълга мрежова маска. Маршрутът по подразбиране има мрежова маска с дължина нула бита, а в показаната по-горе конфигурация, локално свързаните мрежи имат 24-битова мрежова маска. Ако дейтаграма съответства на локално свързана мрежа, тя ще бъде насочена към съответното устройство, вместо да се следва маршрута по подразбиране, защото маршрутът към локалната мрежа съответства с по-голям брой битове. Единствените дейтаграми, които ще бъдат насочени по подразбиращия се маршрут, са тези, които не съответстват на никой друг маршрут.

Можете да създадете таблици за маршрутизация по разнообразни начини. За малки локални мрежи, обикновено най-ефективно е те да се конструират ръчно и да се зададат на IP по време на началното зареждане с помощта на командата *route* (виж Глава 5, *Конфигуриране на TCP/IP мрежа*). За по-големи мрежи, таблиците се създават и настройват по време на работа от *демони за маршрутизация*; тези демони се работят на централни хостове в мрежата и обменят информация за маршрутите, за да определят "оптимални" маршрути между свързаните мрежи.

В зависимост от размера на мрежата трябва да използвате различни протоколи за маршрутизация. За маршрутизиране вътре в автономни системи (като тази на университета GMU) се използват *вътрешни протоколи за маршрутизиране*. Най-известен от тях е протоколът *RIP* (*Routing Information Protocol*), който се реализира от BSD-демона *routed*. За маршрутизиране между автономни мрежи трябва да се използват протоколи за външно маршрутизиране като *EGP* (*External Gateway Protocol*) или *BGP* (*Border Gateway Protocol*). Тези протоколи и RIP са реализирани в демона *gated* на Университета в Cornell.

Метрични стойности

Изборът на най-добър маршрут до получателя-хост или мрежа се извършва с динамична маршрутизация според броя *ретранслации* (*hops*). Ретранслациите съответстват на шлюзовете, през които трябва да премине дейтаграмата, преди да достигне до хоста или мрежата, за които е адресирана. Колкото по-къс е маршрутът, толкова по-добре го оценява RIP. Много дългите маршрути с 16 и повече ретранслации се считат за неизползваеми и се отхвърлят.

RIP управлява вътрешната за локалната мрежа информация за маршрутизиране, но на всички хостове трябва да работи *gated*. По време на зареждането си, *gated* търси всички активни мрежови интерфейси. Ако открие повече от един активен интерфейс (без да се брои loopback интерфейса), демона предполага, че хостът комутира пакети между различни мрежи и активно ще обменя и разпространява информация за маршрутите. В противен случай, той само пасивно ще приема актуализациите на RIP и ще обновява локалната таблица за маршрутизация.

Когато разпространява информация от локалната таблица за маршрутизация, *gated* изчислява дължината на маршрута като използва така наречената *метрична стойност*, свързана с реда от таблица за маршрутизация. Тази метрична стойност се задава от системния администратор при конфигурирането на маршрута и би следвало да отразява реалното тегло на маршрута.* Ето защо, метриката на един маршрут до подмрежа, към която хостът е свързан директно, трябва винаги да бъде нула, докато маршрут, минаващ през два шлюза трябва да има метрика две. Не е нужно да се занимавате с метриката, ако не използвате *RIP* или *gated*.

Протоколът ICMP

Съществува един помощен протокол на IP, за който още не сме говорили. Това е протоколът ICMP (*Internet Control Message Protocol* – протокол за управляващи съобщения в Интернет), използван от мрежовия код в ядрото, за да се обменят съобщения за грешки с други хостове. Като пример ще допуснем, че отново сте на **erdos** и искате да се свържете с *telnet* с порт 12345 на **quark**, но за този порт няма слушащ процес. Когато първият TCP пакет за този порт пристигне в **quark**, мрежовото ниво ще разпознае това и незабавно ще върне ICMP съобщение на **erdos**, с което съобщава, че "портът е недостъпен".

Протоколът ICMP предоставя множество различни съобщения, много от които се отнасят за различни грешки. Има обаче едно много интересно съобщение, наречено съобщение за пренасочване (redirect

* Можете да мислите за теглото на маршрута, в най-простия случай като за броя на необходимите ретранслации, за да се достигне до получателя. Точното определяне на теглото на маршрута може да бъде като изящно изкуство в сложните мрежови топологии.

message). То се генерира от маршрутизиращия модул, когато открие, че друг хост го използва като шлюз, въпреки че съществува и по-кратък маршрут. Например, след стартирането на **erdos**, неговата таблица за маршрутизация може да бъде непълна. Тя може да съдържа маршрути до мрежата на Математическия факултет, до опорната FDDI мрежа и подразбиращия се маршрут, сочещ към шлюза на Изчислителния център на GMU (gcc1). С тази конфигурация, пакетите за **quark** ще бъдат изпратени до **gcc1** вместо до **niels** – шлюза на Физическия факултет. Когато получи такава дейтаграма, **gcc1** ще забележи, че това е лош избор на маршрут и ще препрати пакета към **niels**, като междувременно върне ICMP съобщение за пренасочване на **sophus**, с което му съобщава за по-добрия маршрут.

Това изглежда като много разумен начин за избягване на ръчното настройване на маршрутите, с изключение на най-важните от тях. Все пак, имайте предвид, че разчитането на схемите за динамично маршрутизиране, независимо дали са RIP или съобщенията за пренасочване на ICMP, не винаги е добра идея. Пренасочването на ICMP и RIP предлага малък или никакъв избор за проверка дали дадена информация за маршрутизиране наистина е автентична. Тази ситуация позволява на злонамерени негодници да разрушат целия вътрешен трафик на вашата мрежа или дори по-лоши неща. Ето защо, мрежовият код на Linux разглежда съобщенията за пренасочване за адрес на мрежа като съобщения за пренасочване за адрес на хост. Това намалява пораженията от една евентуална атака, като ги ограничава само до един хост, вместо до цялата мрежа. Недостатък на този подход е, че води до генериране на малко повече трафик в случай на легитимни заявки, тъй като за всеки хост се генерира ICMP съобщение за пренасочване. В днешни дни по принцип се счита за лоша практика да се разчита на ICMP пренасочването.

Разпознаване на имена на хостове

Както описахме по-горе, адресирането в TCP/IP мрежа, поне в IPv4, е свързано с 32-битовите номера. Все пак, едва ли ще можете да запомните повече от няколко такива числа. Ето защо, хостовете обикновено са известни с "обикновени" имена като **gauss** или **strange**. Намирането на IP адреса, съответстващ на тези имена, е грижа на приложението. Този процес се нарича *разпознаване името на хоста* (*hostname resolution*).

Когато едно приложение трябва да намери IP адреса на даден хост, то се обръща към библиотечните функции *gethostname(3)* и *gethos-*

tbyaddr(3). Традиционно, тези и редица свързани с тях процедури са групирани в отделна библиотека, наречена *resolverlibrary*; в Linux тези функции са част от стандартната библиотека *libc*. Затова, разговорно тази колекция от функции се нарича "резолвера" (the resolver). Конфигурирането на резолвера на имена е описано подробно в Глава 6, *Конфигуриране на услугата за имена и резолвера*.

В малки мрежи като една Ethernet мрежа или дори клъстер от такива мрежи, не е много трудно да се поддържат таблици със съответствията между имената на хостовете и техните адресите. Тази информация обикновено се пази във файл, наречен */etc/hosts*. Когато добавяте/премахвате хостове или променяте адресите им, всичко, което трябва да направите, е да актуализирате файла *hosts* на всички хостове. Очевидно, това става много трудно в мрежи, които се състоят от повече от няколко машини.

Едно решение на този проблем е Мрежовата информационна система (NIS), разработена от Sun Microsystems, която разговорно се нарича YP или Жълти Страници. NIS съхранява файла *hosts* (и друга информация) в база данни в главен хост, от където клиентите могат да я извличат при нужда. Все пак, този подход е подходящ само за мрежи със среден размер като локалните, защото изисква централно поддържане на цялата база данни *hosts* и разпространяването ѝ до всички сървъри. Инсталирането и конфигурирането на NIS е разгледано подробно в Глава 13, *Мрежова информационна система*.

В Интернет, адресната информация отначало също беше пазена в една база данни – файлът *HOSTS.TXT*. Този файл се поддържаше от организацията *NIC* (*Network Infomation Center* – мрежов информационен център) и трябваше да се изтегля и инсталира върху всички участващи сайтове. Когато мрежата се разрасна, възникнаха множество проблеми с тази схема. Освен административните дейности при периодичното инсталиране на *HOST.TXT*, натоварването на сървърите, които го разпространяваха, също стана много голямо. Имаше дори още по-сериозен проблем – всички имена трябваше да бъдат регистрирани от NIC, които трябваше да гарантира, че те да не се повтарят.

Ето защо, през 1994 г. беше внедрена нова схема за разпознаване на имената: *DNS* (*Domain Name System* – система за имена на домейни). DNS беше разработена от Paul Mockapetris и беше адресирана едновременно и към двата проблема. Ще разгледаме подробно DNS в Глава 6.

ГЛАВА 3

КОНФИГУРИРАНЕ НА МРЕЖОВИЯ ХАРДУЕР

Вече говорихме доста за мрежовите интерфейси и основните въпроси на TCP/IP, но все още не сме разгледали истински какво се случва, когато "мрежовият код" в ядрото работи с дадено устройство. За да опишем това по подходящия начин, трябва да поговорим малко за концепцията на интерфейсите и драйверите.

Първо, разбира се, е самият хардуер, например Ethernet, FDDI или Token Ring картата – това е една платка, покрита с множество малки чипове със странни означения върху тях, която е поставена в един от слотовете на вашето PC. Това е, което обикновено наричаме физическо устройство.

За да използвате мрежовата карта, във вашето ядро на Linux трябва да има специални функции, които разбират конкретния начин за достъп до това устройство. Софтуерът, който реализира тези функции, се нарича *драйвер на устройство*. Linux има драйвери за много видове мрежови интерфейсни карти: ISA, PCI, MCA, EISA, за паралелен порт, PCMCIA, и отскоро, за USB.

Но какво имаме предвид, когато казваме, че един драйвер "управлява устройство"? Да вземем като пример една Ethernet карта. Драйверът трябва да бъде в състояние по някакъв начин да комуникира с логиката, вградена в периферната карта: той трябва да изпраща команди и данни към картата, а тя трябва да предава всички получени данни на драйвера.

В съвместимите с IBM персонални компютри, тази комуникация се осъществява през блок от входно/изходни адреси, които съответстват на регистри на картата и/или чрез обща памет или директен достъп до паметта. Всички команди и данни, които ядрото изпраща на картата, трябва да преминат през тези адреси. Входно/изходните адреси и адресите на паметта обикновено се описват като се задава стартов или *базов адрес*. Типични базови адреси за Ethernet карти за ISA шина са `0x280` и `0x300`. Мрежовите карти за PCI шина обикновено автоматично получават своите входно/изходни адреси.

Обикновено не трябва да се грижите за каквито и да е хардуерни въпроси като базовият адрес, защото по време на зареждането си, ядрото се опитва да открие разположението на картата. Това се нарича *автоматична проверка* (*auto probing*), което означава, че ядрото прочита множество в/и адреси и адреси в паметта и сравнява получените данни с това, което се очаква да има там, ако на съответния адрес е инсталирана конкретна мрежова карта. Все пак, възможно е да има мрежови карти, които ядрото да не може да идентифицира автоматично; това се случва понякога с евтини мрежови контролери, които не са добри копия на стандартни карти от други производители. Освен това, по време на стартирането си, обикновено ядрото се опитва да открие само едно мрежово устройство. Ако използвате повече от една карта, трябва изрично да зададете останалите карти на ядрото.

Друг параметър, който понякога трябва да се укаже на ядрото, е линията на заявката за прекъсване. Хардуерните компоненти обикновено прекъсват ядрото, когато се нуждаят от обслужване – например, когато са пристигнали данни или е възникнало специално събитие. В PC компютър с ISA шина, прекъсванията могат да се осъществят по един от 15-те канала за прекъсване, номерирани 0, 1, 3 и т.н. до 15. Номерът на прекъсването, зададен за даден хардуерен компонент, се нарича *IRQ* (*Internupt Request Number* – номер на заявка за прекъсване).*

Както беше описано в Глава 2, *Въпроси на работата в TCP/IP мрежа*, ядрото работи с мрежовия хардуер с помощта на софтуерна конструкция, наречена *интерфейс*. Интерфейсите предоставят абстрактен набор от функции като изпращане и получаване на дейтаграма, които са еднакви за всички видове хардуер.

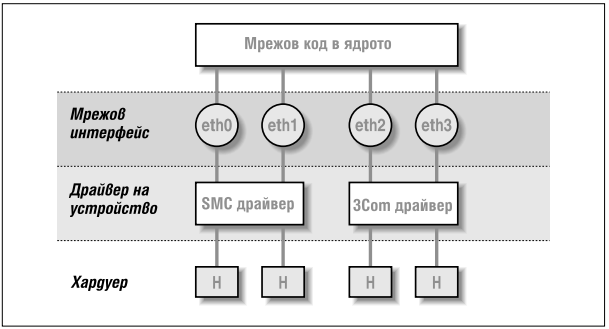
* Заявките за прекъсване 2 и 9 са едни и същи, защото в хардуерния дизайн на IBM PC има два каскадни процесора за обработка на прекъсванията, всеки от които има по осем входа; вторият процесор се свързва към IRQ 2 на основния.

Интерфейсите се идентифицират със своите имена. В много други подобни на Unix операционни системи, мрежовият интерфейс се реализира като специален файл за устройство в директорията */dev/*. Ако въведете командата `ls – las /dev/`, ще видите как изглеждат тези файлове на устройства. Във втората колона, съдържаща разрешенията (permissions) за файла ще видите, че информацията за файловете за устройства започва с буква, вместо с тире, както е при нормалните файлове. Тази буква показва типа на устройството. Най-често срещаните типове са `b`, който означава, че това устройство е *блоково* и на всеки запис/четене работи с цели блокове данни, и `c`, който означава, че това устройство е *символно* и обработва данните знак по знак. На местото в изхода от командата *ls*, където обикновено виждате дължината на файла, сега ще видите две числа, наричани основен (major) и вторичен (minor) номер на устройството. Тези числа указват конкретното устройство, с което е асоциран този файл.

Всеки драйвер на устройство регистрира уникален основен номер в ядрото. Всеки отделен *представител* на това устройство регистрира уникален вторичен номер за този основен номер на устройство. Например, интерфейсите `tty` – */dev/tty*, са символни устройства, обозначавани със "`c`" като всяко от тях има основен номер `4`, но устройството */dev/tty1* има вторичен номер `1`, а устройство */dev/tty2* има вторичен номер `2`. Файловете на устройства са много полезни за голям брой устройства, но могат да бъдат неудобни, когато се опитвате да намерите едно неизползвано устройство, за да го отворите.

Имената на интерфейсите в Linux се дефинират вътрешно в ядрото и не са файлове на устройства в директорията */dev*. Някои типични имена на устройства са изброени по-долу в раздела "Преглед на мрежовите устройства на Linux". Свързването на интерфейсите към устройства обикновено зависи от реда, по който устройствата се конфигурират. Например, първата инсталирана карта Ethernet ще бъде *eth0*, а следващата – *eth1*. SLIP-интерфейсите се обслужват различно от другите, защото се създават динамично. Когато се установява SLIP-връзка, със серийния порт се свързва интерфейс.

На Фигура 3-1 е показана връзката между хардуера, драйверите на устройства и интерфейсите.



Фигура 3-1. Връзката между драйверите, интерфейсите и хардуера

По време на зареждане, ядрото показва устройствата, които открива и интерфейсите, които инсталира. Следващите редове съдържат част от типичните съобщения при зареждане на ядрото:

```
.
.  This processor honors the WP bit even when in supervisor mode./
   Good.
Swansea University Computer Society NET3.035 for Linux 2.0
NET3: Unix domain sockets 0.13 for Linux NET3.035.
Swansea University Computer Society TCP/IP for NET3.034
IP Protocols: IGMP,ICMP, UDP, TCP
Swansea University Computer Society IPX 0.34 for NET3.035
IPX Portions Copyright (c) 1995 Caldera, Inc.
Serial driver version 4.13 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16550A
tty01 at 0x02f8 (irq = 3) is a 16550A
CSLIP: code copyright 1989 Regents of the University of California
PPP: Version 2.2.0 (dynamic channel allocation)
PPP Dynamic channel allocation code copyright 1995 Caldera, Inc.
PPP line discipline registered.
eth0: 3c509 at 0x300 tag 1, 10baseT port, address 00 a0 24 0e e4 e0,/
    IRQ 10.
3c509.c:1.12 6/4/97 becker@cesdis.gsfc.nasa.gov
Linux Version 2.0.32 (root@perf) (gcc Version 2.7.2.1)
#1 Tue Oct 21 15:30:44 EST 1997
.
.
```

Този пример показва, че ядрото е компилирано с поддръжка на TCP/IP и съдържа драйвери за SLIP, CSLIP и PPP. Третият ред отдо-

лу нагоре показва, че е открита Ethernet карта 3C509 и е инсталирана като интерфейс *eth0*. Ако имате друг тип мрежова карта – например джобен адаптер D-Link – ядрото обикновено отпечатва ред, започващ с името на устройството – *dl0* в примера с D-Link – следван от типа на откритата карта. Ако имате инсталирана мрежова карта, но не видите подобно съобщение, значи ядрото не е успяло да разпознае коректно вашата карта. Тази ситуация ще бъде разгледана по-долу в раздела "Автоматична проверка за Ethernet контролер".

Конфигуриране на ядрото

Повечето дистрибуции на Linux се доставят с дискове за начално зареждане, които работят с всички масови хардуерни устройства за PC. Обикновено, ядрото в тях е конфигурирано като множество модули и включва почти всички възможни драйвери. Идеята за такива дискети за начално зареждане е много добра, но може би не е точно това, което бихте искали за продължителна употреба. Няма голям смисъл да си препълвате диска с драйвери, които никога няма да използвате. Ето защо, обикновено си струва да създадете свое собствено ядро, в което ще включите само тези драйвери, от които действително имате нужда или искате; по този начин спестявате малко дисково пространство и намалявате времето, необходимо за компилиране на новото ядро.

При всички случаи, когато използвате Linux система, би трябвало да сте на "ти" с компилирането на ядрото. Мислете за това като за ваше право, едно потвърждение на факта, който прави свободния софтуер толкова мощен, колкото е – вие разполагате с изходния код. Не трябва да смятате, че "трябва да компилирате ядрото", а че "*можете* да компилирате ядрото". Основите на компилирането на ядрото на Linux са описани в книгата на Мат Уелш "*Running Linux*" (издание на O'Reilly)*. Затова в този раздел ще разгледаме само конфигурационните опции, които се отнасят за работата в мрежа.

Един важен момент, който си струва да повторим тук, е начинът, по който се номерира версията на ядрото. Ядрата на Linux се номерират, като се използва следния формат: `2.2.14`. Първата цифра показва *основния* номер на версията. Тази цифра се променя, когато се извършват големи и значими изменения в архитектурата на ядрото.

* Издадена на български език под името "Ръководство за Linux" – б.р

Например, основния номер на ядрото се промени от 1 на 2, когато ядрото предостави поддръжка за машини, различни от базираните на процесор Intel. Второто число е *вторичен* номер на версията. В много отношения този номер е най-важното число.

Обществото от разработчици на Linux е възприело стандарт, според който с *четен* вторичен номер на версия се обозначават *готовите за употреба* или *стабилните* версии на ядрото, а с нечетен вторичен номер на версии се обозначават *разработвани* или *нестабилни* ядра. Стабилните ядра са това, което бихте трябвало да използвате на важните за вас машини, защото те са по-цялостно тествани. Ако се интересувате от експериментиране с най-новите възможности на Linux, можете да използвате разработваните ядра, но те могат да имат проблеми, които все още не са открити и отстранени. Третото число просто се увеличава с всяко ново издание на вторичната версия.*

Когато изпълнявате *make menuconfig*, получавате текстово меню, което предлага списък от конфигурационни въпроси – например, дали искате емулация на математически копроцесор в ядрото. Един от тези въпроси е искате ли поддръжка на работа в TCP/IP мрежа. За да получите ядро, което има възможност за работа в мрежа, трябва да отговорите с `y`.

Опции на ядрото в Linux версия 2.0 или по-висока

След като завършите частта с общите опции, конфигурирането ще продължи с въпроси дали искате да добавите поддръжка на разнообразни възможности като SCSI драйвери и звукови карти. Поканата за въвеждане показва възможните отговори. Можете да натиснете `?`, за да получите описание на това, какво всъщност предлага опцията. Винаги можете да отговорите с да (`y`), за да включите компонентата статично в ядрото, или с не (`n`), за да изключите компонентата изцяло. За компонентите, които могат да бъдат компилирани като зареждаеми модули, ще видите като възможен отговор компилирай като модул (`m`). Модулите трябва да бъдат заредени, за да могат да се из-

* Хората могат да използват разработваните ядра и да изпращат съобщения за грешки, ако намерят такива; това е много полезна дейност, ако имате машина, която можете да отделите като тестова. Инструкциите за начина, по който да съобщите за открита грешка, са описани подробно във файла */usr/src/linux/REPORTING-BUGS* в изходния код на ядрото на Linux.

ползват и са полезни за драйвери на компоненти, които използвате рядко.

Следващият списък с въпроси се отнася за поддръжката на работа в мрежа. Поради продължаващите разработки, точният набор на конфигурационните възможности непрекъснато се променя. Типичният списък от опции, предлаган от повечето ядра с версии 2.0 и 2.1 изглежда по следния начин:

```
*
* Network device support (Поддръжка на мрежови устройства)
*
Network device support (CONFIG_NETDEVICES) [Y/n/?]
```

Ако искате да използвате *някакъв* тип мрежови устройства, независимо дали те са Ethernet, SLIP, PPP или нещо друго, трябва да отговорите на този въпрос с да (y). След като отговорите с y, поддръжката на Ethernet устройства се разрешава автоматично. Ако искате да добавите поддръжка за други типове мрежови устройства, трябва да отговорите на допълнителни въпроси:

```
PLIP (parallel port) support (CONFIG_PLIP) [N/y/m/?] y
PPP (point-to-point) support (CONFIG_PPP) [N/y/m/?] y
*
* CCP compressors for PPP are only built as modules
*
SLIP (serial line) support (CONFIG_SLIP) [N/y/m/?] m
 CSLIP compressed headers (CONFIG_SLIP_COMPRESSED) [N/y/?] (NEW) y
 Keepalive and linefill (CONFIG_SLIP_SMART) [N/y/?] (NEW) y
 Six bit SLIP encapsulation (CONFIG_SLIP_MODE_SLIP6) [N/y/?] (NEW) y
```

Тези въпроси се отнасят за различните протоколи от каналния слой, които поддържа Linux. PPP и SLIP ви дават възможност да пренасяте IP дейтаграми през серийни линии. PPP всъщност е комплект от протоколи, използвани за предаване на мрежов трафик по серийни линии. Някои от протоколите, образуващи PPP, управляват начина, по който удостоверявате самоличността си пред избирания сървър, а други управляват начина, по който определени протоколи се пренасят през връзката – PPP не е ограничен до пренасянето само на TCP/IP дейтаграми; той може да пренася и други протоколи, например IPX.

Ако отговорите с y или m на въпроса за поддръжка на SLIP (SLIP support), ще трябва да отговорите на още три въпроса, които се появяват след това. Опцията за компресирано заглавие (compressed headers) осигурява поддръжката на CSLIP – техника, която компре-

сира TCP/IP заглавията само до три байта. Обърнете внимание, че тази опция на ядрото не включва автоматично CSLIP; тя просто осигурява необходимите функции на ядрото за поддръжката на протокола. Опцията `Keepalive and linefill` указва на функциите, поддържащи SLIP, да генерират периодично активност по линията, за да се избегне нейното прекъсване от таймер за неактивност. Опцията `Six bit SLIP encapsulation` ви дава възможност да използвате SLIP по линии и вериги, които не са в състояние да предават коректно целия 8-битов пакет данни. Тази техника е подобна на unencoding и binhex, които се използват за изпращане на двоични файлове по електронната поща.

PLIP осигурява възможност за изпращане на IP дейтаграми през връзка към паралелен порт. Той се използва предимно за комуникация с PC-та, работещи под DOS. При типичен PC хардуер, PLIP може да бъде по-бърз от PPP или SLIP, но генерира по-голямо натоварване на процесора, затова макар че скоростта на обмен може да е добра, другите задачи на машината могат да се забавят.

Следващите въпроси се отнасят за мрежовите карти от различни производители. Тъй като постоянно се разработват нови драйвери, вероятно ще видите нови въпроси, освен изброените по-долу. Ако искате да създадете ядро, което да можете да използвате върху множество различни машини, или ако във вашата машина има инсталирани няколко мрежови карти, можете да разрешите повече от един драйвер:

```
.
.
Ethernet (10 or 100Mbit) (CONFIG_NET_ETHERNET) [Y/n/?]
3COM cards (CONFIG_NET_VENDOR_3COM) [Y/n/?]
3c501 support (CONFIG_EL1) [N/y/m/?]
3c503 support (CONFIG_EL2) [N/y/m/?]
3c509/3c579 support (CONFIG_EL3) [Y/m/n/?]
3c590/3c900 series (592/595/597/900/905) "Vortex/Boomerang" support/
    (CONFIG_VORTEX) [N/y/m/?]
AMD LANCE and PCnet (AT1500 and NE2100) support (CONFIG_LANCE) [N/y/?]
AMD PCInet32 (VLB and PCI) support (CONFIG_LANCE32) [N/y/?] (NEW)
Western Digital/SMC cards (CONFIG_NET_VENDOR_SMC) [N/y/?]
WD80*3 support (CONFIG_WD80x3) [N/y/m/?] (NEW)
SMC Ultra support (CONFIG_ULTRA) [N/y/m/?] (NEW)
SMC Ultra32 support (CONFIG_ULTRA32) [N/y/m/?] (NEW)
SMC 9194 support (CONFIG_SMC9194) [N/y/m/?] (NEW)
Other ISA cards (CONFIG_NET_ISA) [N/y/?]
Cabletron E21xx support (CONFIG_E2100) [N/y/m/?] (NEW)
DEPCA, DE10x, DE200, DE201, DE202, DE422 support (CONFIG_DEPCA)
[N/y/m/?]/
    (NEW)
```

```
EtherWORKS 3 (DE203, DE204, DE205) support (CONFIG_EWRK3)
[N/y/m/?] (NEW)
EtherExpress 16 support (CONFIG_EEXPRESS) [N/y/m/?] (NEW)
HP PCLAN+ (27247B and 27252A) support (CONFIG_HPLAN_PLUS)
[N/y/m/?] (NEW)
HP PCLAN (27245 and other 27xxx series) support (CONFIG_HPLAN)
[N/y/m/?]/
    (NEW)
HP 10/100VG PCLAN (ISA, EISA, PCI) support (CONFIG_HP100)
[N/y/m/?] (NEW)
NE2000/NE1000 support (CONFIG_NE2000) [N/y/m/?] (NEW)
SK_G16 support (CONFIG_SK_G16) [N/y/?] (NEW)
EISA, VLB, PCI and on card controllers (CONFIG_NET_EISA) [N/y/?]
Apricot Xen-II on card ethernet (CONFIG_APRICOT) [N/y/m/?] (NEW)
Intel EtherExpress/Pro 100B support (CONFIG_EEXPRESS_PRO100B)
[N/y/m/?]/
    (NEW)
DE425, DE434, DE435, DE450, DE500 support (CONFIG_DE4X5)
[N/y/m/?] (NEW)
DECchip Tulip (dc21x4x) PCI support (CONFIG_DEC_ELCP) [N/y/m/?] (NEW)
Digi Intl. RightSwitch SE-X support (CONFIG_DGRS) [N/y/m/?] (NEW)
Pocket and portable adaptors (CONFIG_NET_POCKET) [N/y/?]
AT-LAN-TEC/RealTek pocket adaptor support (CONFIG_ATP) [N/y/?] (NEW)
D-Link DE600 pocket adaptor support (CONFIG_DE600) [N/y/m/?] (NEW)
D-Link DE620 pocket adaptor support (CONFIG_DE620) [N/y/m/?] (NEW)
Token Ring driver support (CONFIG_TR) [N/y/?]
IBM Tropic chipset based adaptor support (CONFIG_IBMTR) [N/y/m/?] (NEW)
FDDI driver support (CONFIG_FDDI) [N/y/?]
Digital DEFEA and DEFPA adapter support (CONFIG_DEFXX) [N/y/?] (NEW)
ARCnet support (CONFIG_ARCNET) [N/y/m/?]
  Enable arc0e (ARCnet "Ether-Encap" packet format) (CONFIG_ARCNET_ETH)/
      [N/y/?] (NEW)
  Enable arc0s (ARCnet RFC1051 packet format) (CONFIG_ARCNET_1051)/
      [N/y/?] (NEW)
.
.
```

Накрая, в раздела за файловата система, конфигурационният скрипт ще ви попита дали искате поддръжка на мрежовата файлова система (NFS). NFS ви дава възможност да предоставите файлови системи за достъп през мрежата от множество хостове, благодарение на което файловете изглеждат така, като че са на обикновен твърд диск, свързан към хоста:

```
NFS file system support (CONFIG_NFS_FS) [Y]
```

Ще опишем NFS подробно в Глава 14, *Мрежова файлова система*.

Опции на ядрото за работа в мрежа за Linux версия 2.0.0 или по-висока

Linux 2.0.0 бележи значителни изменения в поддръжката за работа в мрежа. Много възможности (например поддръжката на IPX) бяха направени стандартна част от Ядрото. Бяха добавени много нови опции и възможности за тяхното конфигуриране. Голяма част от тези опции се използват само в много специфични условия и затова няма да ги разглеждаме подробно. Най-вероятно ще намерите това, което го няма тук, в документа Networking HOWTO. В този раздел ще опишем някои полезни опции и ще обясним кога бихте могли да използвате всяка от тях.

Основи

За да използвате TCP/IP мрежа, трябва да отговорите на следващия въпрос с `y`. Дори да отговорите с `n`, обаче, ще можете да компилирате ядрото с поддръжка на IPX:

```
Networking options---->
[*] TCP/IP networking
```

Шлюзове

Ако вашата система работи като шлюз между две мрежи или между локална мрежа и външна връзка (например SLIP), трябва да разрешите следващата опция. Обикновено не е проблем, ако я разрешите по подразбиране, освен когато конфигурирате хоста като *защитна стена*. Защитни стени са хостовете, които са свързани към две или повече мрежи, но не маршрутизират трафик между тях. Обикновено те се използват, за да предоставят на потребителите достъп до Интернет при минимален риск за вътрешната мрежа. На потребителите се разрешава да влизат в хоста-защитна стена и да използват Интернет услуги, но машините на компанията са защитени от външни атаки, защото входящите връзки не могат да преминат защитната стена (защитните стени са разгледани подробно в Глава 9, *Защитна стена за TCP/IP)*:

```
[*] IP: forwarding/gatewaying
```

Виртуален хостинг

Тези опции дават възможност да конфигурирате няколко IP адреса за един интерфейс. Понякога това е полезно, ако искате да предоставите "виртуален хостинг", чрез който една машина може да бъде конфигурирана да изглежда и работи все едно, че е

няколко отделни машини, всяка със своя собствена мрежова индивидуалност. Ще поговорим повече за IP псевдонимите (IP aliases) след малко:

```
[*] Network aliasing
  <*> IP: aliasing support
```

Счетоводство

Следващата опция ви позволява да събирате данни за обема на изходящия и входящия IP трафик във вашата машина (Ще разгледаме тази възможност подробно в Глава 10, *IP счетоводство*):

```
[*] IP: accounting
```

PC hug

Тази опция заобикаля несъвместимост с някои версии на PC/TCP – комерсиална реализация на TCP/IP за базирани на DOS PC-та. Ако я разрешите, ще можете да комуникирате с нормални Unix машини, но производителността може да намалее при бавни линии:

```
--- (it is safe to leave these untouched)
[*] IP: PC/TCP compatibility mode
```

Бездисково зареждане

Тази функция разрешава протокола RARP (*Reverse Address Resolution Protocol* – протокол за обратно разпознаване на адреси). RARP се използва от бездискови клиенти и X-терминали за получаване на техния IP адрес по време на зареждане. Ако смятате да обслужвате такъв вид клиенти, трябва да разрешите RARP. За добавяне на записи в таблицата на RARP в ядрото се използва малка програма, наречена *rarp*, която е част от стандартните мрежови инструменти:

```
<*> IP: Reverse ARP
```

MTU

Когато изпраща данни чрез TCP, ядрото трябва да раздели потока на блокове данни, които да ги подаде на IP. Размерът на блока се нарича MTU (*Maximum Transfer Unit* – максимална единица за предаване). За хостове, които могат да бъдат достигнати по локална мрежа като Ethernet, обикновено се използва толкова голям MTU, колкото е максималната дължина на пакета в Ethernet

- 1500 байта. Когато се маршрутизира IP по глобална мрежа като Интернет, за предпочитане е да се използват дейтаграми с по-малък размер, за да се гарантира, че те няма да бъдат разделени по-късно по маршрута на още по-малки части от процеса, наречен *IP фрагментация*.* Ядрото е в състояние автоматично да определи най-малкия MTU за даден IP маршрут и да конфигурира TCP връзката да го използва. Това поведение по подразбиране е включено. Ако отговорите с y на следващата опция, тази възможност ще бъде дезактивирана.

Ако искате да използвате по-малки размери на пакети за данните, изпращани към определени хостове (например, защото данните преминават през SLIP връзка), можете да направите това с опцията *mss* на командата *route*, която е разгледана накратко в края на тази глава:

```
[ ] IP: Disable Path MTU Discovery (normally enabled)
```

Особеност на сигурността

Протоколът IP поддържа една възможност, наречена *Source Routing* (определянето на маршрута от източника). Тя позволява да се посочи маршрута, през който трябва да премине дейтаграмата, чрез кодирането му в самата дейтаграма. Преди маршрутизиращите протоколи като RIP и OSPF да станат масова практика, това може би беше полезно. Но днес тази възможност се счита за опасност за сигурността, защото може да предостави на умелите нападатели начин за заобикаляне на определени типове защити чрез прескачане на таблицата с пътища на маршрутизатора. Нормалната практика е да се игнорират дейтаграмите със зададен маршрут, поради което следващата опция обикновено е разрешена:

```
[*] IP: Drop source routed frames
```

Поддръжка на Novell

Тази възможност позволява поддръжката на IPX – транспортния протокол, който използват мрежите на Novell. Linux може да ще работи чудесно като IPX маршрутизатор; тази поддръжка е по-

* Спомнете си, че IP протокола може да бъде пренасян през много различни видове мрежи и не всяка от тях поддържа размер на пакета толкова голям, колкото е при Ethernet

лезна в мрежови среди, в които се използват файлови сървъри на Novell. Файловата система NCP също изисква поддръжката на IPX в ядрото да бъде разрешена. Ако искате да се свържете и монтирате вашите файлови системи на Novell сървър, трябва да разрешите тази опция (ще разгледаме IPX и файловата система NCP в Глава 15, *IPX и файловата система NCP*):

```
<*> The IPX protocol
```

Любителско радио

Тези три опции задават поддръжката на трите протокола за Любителско радио в Linux: AX.25, NetRom и Rose (тук няма да описваме тези протоколи, но те са разгледани подробно в AX25 HOWTO):

```
<*> Amateur Radio AX.25 Level 2
<*> Amateur Radio NET/ROM
<*> Amateur Radio X.25 (Rose)
```

Linux поддържа един друг тип драйвер: dummy (фиктивен) драйвер. При започването на раздела за драйверите на устройства се появява следния въпрос:

```
<*> Dummy net driver support
```

Фиктивния драйвер в действителност не прави много, но е доста полезен при самостоятелни или PPP/SLIP хостове. Той всъщност е маскиран интерфейс-примка (loopback). На хостове, които предлагат PPP/SLIP, но нямат друг мрежов интерфейс, трябва да имате интерфейс, който постоянно дава вашия IP адрес. Това е разгледано малко по-подробно в раздела "Фиктивен интерфейс" в Глава 5, *Конфигуриране на TCP/IP мрежа*. Днес можете да постигнете същия резултат с помощта на IP псевдоними, като конфигурирате вашия IP адрес като псевдоним на интерфейса-примка.

Преглед на мрежовите устройства на Linux

Ядрото на Linux поддържа множество хардуерни драйвери за разнообразни типове апаратури. Този раздел съдържа кратък преглед на достъпните фамилии драйвери и на имената на интерфейсите, които те използват.

В Linux се използват множество стандартни имена на интерфейси. Повечето драйвери поддържат повече от един интерфейс, при което се номерират последователно, например *eth0* и *eth1*:

lo Това е локален интерфейс-примка. Той се използва за тестване и за комуникация между двойка мрежови приложения, намиращи се на една и съща машина. Този интерфейс работи като затворена верига в смисъл, че всяка изпратена към него дейтаграма незабавно ще бъде върната на мрежовото ниво на хоста. В ядрото винаги има едно loopback устройство и няма смисъл да има повече от едно.

eth0, eth1, ...

Това са интерфейсите на мрежовите карти за Ethernet. Те се използват за повечето Ethernet карти, включително и за Ethernet-устройствата за паралелен порт.

tr0, tr1, ...

Това са интерфейсите на картите Token Ring. Те се използват за повечето Token Ring карти, включително за контролери, които не произведени от IBM.

sl0, sl1, ...

Това са SLIP интерфейсите. Тези интерфейси се асоциират със серийните линии по реда, по който те се отделят за SLIP.

ppp0, ppp1, ...

Това са PPP интерфейсите. Точно както SLIP интерфейсите, PPP интерфейсът се асоциира със серийните линии, след като те се преобразуват в PPP режим.

plip0, plip1, ...

Това са PLIP интерфейсите. PLIP прехвърля IP дейтаграми през паралелни линии. Интерфейсите се създават от PLIP драйвера по време на зареждането на системата и се асоциират с паралелните портове. В ядрата *2.0.x* има директна връзка между името на устройството и входно/изходния адрес на паралелния порт, но в следващите ядра имената на устройствата се задават последователно, точно както при SLIP и PPP устройствата.

ax0, ax1, ...

Това са AX.25 интерфейсите. AX.25 е основния протокол, използван от радио-операторите любители. Интерфейсите AX.25 се създават и асоциират подобно на SLIP устройствата.

Съществуват много други типове интерфейси за други мрежови драйвери. Тук изброихме само най-често срещаните.

В следващите два раздела ще разгледаме подробностите за използването на описаните по-горе драйвери. Документът Networking HOWTO съдържа подробности относно начина, по който можете да конфигурирате повечето от останалите, а AX25 HOWTO обяснява как се конфигурират мрежовите устройствата на радио-любителите.

Инсталиране за Ethernet

Сегашният мрежов код на Linux поддържа голям брой Ethernet карти. Повечето драйвери са написани от Donald Becker, който създаде фамилия драйвери за картите, базирани на чипа 8390 на National Semiconductors; тези драйвери станаха известни като Серията драйвери на Becker. Много други разработчици добавиха свои драйвери и днес има много малко Ethernet карти, които не се поддържат от Linux. Списъкът на поддържаните Ethernet карти нараства постоянно, така че ако за вашата карта все още няма поддръжка, най-вероятно скоро и тя ще бъде добавена към списъка с поддържани контролери.

В по-ранната история на Linux бихме се опитвали изброим всички поддържани Ethernet карти, но днес това би отнело твърде много време и място. За щастие, Paul Gortmaker поддържа документа Ethernet HOWTO, който съдържа пълен списък на поддържаните карти и полезна информация за конфигурирането на всяка една от тях в Linux.[+] Този документ се публикува ежемесечно в групата за новини *comp.os.linux.answers* и може да бъде намерен и на всички огледални сайтове на LDP.

Дори да сте сигурни в това, как се инсталира определен тип Ethernet карта на вашата машина, често си струва да хвърлите поглед на Ethernet HOWTO, за да видите какво е казано по този въпрос в офи-

[+] Paul може да бъде намерен на адрес *gpg109@rsphy1.anu.edu.au*

циалния документ. Там ще намерите информация, която излиза извън рамките на обикновените въпроси по конфигурирането. Например, ще си спестите много главоболия, ако знаете поведението на една DMA Ethernet карта, която по подразбиране използва същия DMA канал, както и SCSI контролерът Adaptec 1542. Докато не преместите единия от тях на друг DMA канал, рискувате Ethernet картата на запише получените данни на случайно място на вашия твърд диск.

За да използвате коя да е от поддържаните Ethernet карти в Linux, можете да използвате предварително компилирано ядро от някоя от основните дистрибуции на Linux. Те обикновено имат модули за всички поддържани драйвери, а инсталационния процес ви дава възможност да изберете кои драйвери искате да бъдат заредени. В дългосрочен план, обаче, е по-добре да компилирате свое собствено ядро, в което да включите само онези драйвери, от които действително се нуждаете; това спестява дисково пространство и памет.

Автоматична проверка за Ethernet контролер

Много от Ethernet драйверите за Linux са достатъчно интелигентни, за да знаят как да търсят разположението на вашата Ethernet карта. Това ви спестява неудобството ръчно да информирате ядрото къде се намира тя. От Ethernet HOWTO можете да разберете дали даден драйвер поддържа автоматична проверка и в какъв ред търси входно/изходните адреси на картата.

Съществуват три ограничения на кода за автоматична проверка. Първо, той може да не разпознае правилно всички карти. Това става най-често с някои от евтините клонинги на основните карти. Второ, ако не бъде специално инструктирано, ядрото няма да търси автоматично повече от една карта. Това беше съзнателно проектно решение, защото се предполага, че искате изрично да укажете коя карта към кой интерфейс се свързва. Най-добрият начин гарантирано да постигнете това, е да конфигурирате ръчно Ethernet картите във вашата машина. Трето, драйверът може да не провери адреса, на който е конфигурирана вашата карта. Обикновено, драйверите проверяват автоматично адресите, на които е възможно да бъде конфигурирано съответното устройство, но понякога определени адреси се игнорират, за да се избегнат хардуерни конфликти с другите типове карти, които използват обикновено тези адреси.

Мрежовите карти с PCI интерфейс би трябвало да бъдат откривани надеждно. Но ако използвате повече от една карта или ако автома-

тичния тест не успее да разпознае вашата карта, имате възможност изрично да укажете на ядрото базовия адрес и името на картата.

По време на зареждане можете да зададете аргументи и информация на ядрото, които могат да бъдат прочетени от всеки от неговите компоненти. Този механизъм ви дава възможност да подадете информация на ядрото, която Ethernet драйверите да използват, за да определят мястото на вашия Ethernet хардуер, без да се опитват автоматично да го открият.

Ако използвате lilo, за да заредите системата си, можете да зададете параметри на ядрото, като използвате опцията `append` във файла *lilo.conf*. За да информирате ядрото за Ethernet устройство, можете да използвате следните параметри:

```
ether=irq,базов_адрес,[параметър1,][ параметър2,]име
```

Първите четири параметъра са числови, а последният е името на устройството. Параметрите *irq*, *базов_адрес* и *име* са задължителни, а двете стойности параметър не са. Всяка от числовите стойности може да бъде зададена като нула, което означава, че ядрото трябва да я определи чрез тестване.

Първият параметър задава номера на IRQ, използван от устройството. По подразбиране ядрото ще се опита автоматично да определи IRQ канала на устройството. Например, драйверът 3c503 има специалната възможност да избира свободен IRQ канал от списъка 5, 9, 3, 4 и конфигурира картата да го използва. Параметърът *базов_адрес* дава базовия входно/изходен адрес на картата; стойност нула указва на ядрото да провери адресите, изредени по-горе.

Различните драйвери използват следващите два параметъра различно. За карти със споделена памет WD80x3, тези адреси задават началните и крайните адреси на областта със споделена памет. Други карти обикновено използват *параметър1* за задаване на нивото на отпечатваната debug-информация. Стойности от 1 до 7 означават увеличаване нивото на детайлност, а 8 ги изключва цялата информация; с 0 се означава подразбиращото се ниво. Драйверът 3c503 използва *параметър2* за избор между вътрешен предавател (по подразбиране) или външен предавател (стойност 1). Вътрешния предавател използва BNC съединителя на картата, а външния – нейния AUI порт. Ако нямате нещо специално за конфигуриране, не е необходимо да задавате стойности *параметър*.

Първият не-числов аргумент се интерпретира от ядрото като име на устройството. Трябва да зададете име на устройството за всяка Ethernet карта, която описвате.

Ако имате две Ethernet карти, можете да оставите Linux автоматично да открие едната и да зададете параметриге на втората карта посредством *lilo*, но най-вероятно ще искате ръчно да конфигурирате и двете карти. Ако сте решили да използвате ядрото за автоматично откриване на първата карта и ръчно да конфигурирате втората, трябва да се уверите, че ядрото няма да намери случайно първо втората карта или с други думи, първата да не бъде регистрирана изобщо. За целта трябва да използвате опцията `reserve` на *lilo*, която изрично указва на ядрото да не проверява входно/изходната област, заета от втората карта. Например, за да накарате Linux да инсталира втората Ethernet карта на адрес `0x300` като *eth1*, трябва да предадете на ядрото следните параметри:

```
reserve=0x300,32 ether=0,0x300,eth1
```

Опцията *reserve* гарантира, че никой драйвер при автоматична проверка за някое устройство няма да тества входно/изходното пространство на втората карта. Освен това, можете да използвате параметрите на ядрото за да отмените автоматичното търсене за *eth0*:

```
reserve=0x340,32 ether=0,0x340,eth0
```

Можете да изключите автоматичната проверка изобщо. Това се прави, например, за да прекратите търсенето на Ethernet карта, която временно сте я демонтирали. Забраняването на автоматичната проверка се извършва просто със задаване на –1 като стойност за *базов_адрес*:

```
ether=0,-1,eth0
```

За да се подадат тези параметри на ядрото по време на зареждането му, трябва да ги въведете при получаване на поканата "`boot:`" на *lilo*. За да получите тази покана, трябва да натиснете един от клавишите Control, Alt или Shift по време на зареждането на *lilo*. Ако натиснете клавиша Tab в нея, ще получите списък на ядрата, които можете да заредите. За да заредите дадено ядро с параметри, въведете името на ядрото, което искате да заредите, следвано от интервал и параметрите, които подавате. Когато натиснете клавиша Enter, *lilo* ще зареди това ядро и ще му подаде параметрите, които сте въвели.

За да направите така, че тези изменения да се извършват автоматично при всяко рестартиране, въведете параметрите във файла */etc/*

lilo.conf като използвате ключовата дума `append=`. Ето един пример как би изглеждало това:

```
boot=/dev/hda
root=/dev/hda2
install=/boot/boot.b
map=/boot/map
vga=normal
delay=20
append="ether=10,300,eth0"

image=/boot/vmlinuz-2.2.14
label=2.2.14
read-only
```

След всяко редактиране на *lilo.conf*, трябва отново да изпълните командата *lilo*, за да активирате измененията.

PLIP драйвер

PLIP (*Parallel Line IP* – IP през паралелна линия) е евтин начин за работа в мрежа, когато искате да свържете само две машини. Използват се паралелните портове и специален кабел, а достигаите скорости са от 10 до 20 килобайта за секунда.

PLIP първоначално беше разработен от Crynwr, Inc. Тяхната разработка за времето си беше много изобретателна (или, ако предпочитате, хак), защото оригиналните паралелни портове на IBM PC бяха проектирани да работят като еднопосочни портове за печатащи устройства; осемте линии за данни можеха да се използват само за изпращане на данни от компютъра към периферното устройство, но не и в обратната посока.[#] Разработката PLIP на Crynwr заобикаляше това ограничение, като използваше петте линии за състояние на порта за вход, което даваше възможност за прехвърляне на данните като полубайтове, но позволяваше двупосочна комуникация. Този режим на работа беше наречен PLIP "режим 0". Днес, паралелните портове монтирани на персоналните компютри, позволяват пълна двупосочна

[#] Борете се за изчистване името на хакерите! Винаги използвайте "кракер", когато става дума за хора, които съзнателно се опитват да нарушат сигурността на системата, а "хакер", когато става въпрос за хора, които са намерили умен начин за решаване на даден проблем. Хакерите могат да бъдат кракери, но двете понятия не трябва никога да се объркват. За пълно изясняване на термините се консултирайте с Новия хакерски речник (New Hackers Dictionary – обикновено се намира като жаргонен файл).

обмяна на 8-битови данни и PLIP беше доразвит да използва тази възможност със създаването на PLIP "режим 1".

Ядрата на Linux до Версия 2.0 включително поддържат само PLIP режим 0. За осигуряване на работа в режим 1 съществува подобрен драйвер за паралелен порт като patch за версия 2.0 и като стандартен елеменг за версия 2.2 на ядрото.[%] За разлика от първите версии на кода на PLIP, сега драйверът се опитва да бъде съвместим с реализацията на PLIP на Crynwr, както и с PLIP драйвера в NCSA *telnet*.[!] За да свържете две машини посредством PLIP, се нуждаете от специален кабел, продаван в някои магазини като Null Printer или Turbo Laplink cable. Можете обаче доста лесно сами да направите такъв кабел; ще научите как става това от Приложение Б, *Полезни кабелни конфигурации*.

PLIP драйверът за Linux е резултат от работата на почти безкрайно много хора. В момента той се поддържа от Niibe Yutaka.[*] Ако бъде компилиран в ядрото, той създава мрежов интерфейс за всеки от възможните принтерски портове като *plip0* съответства на паралелния порт *lp0*, *plip1* съответства на *lp1* и т.н. Съответствието на интерфейсите с портовете се различава в ядрата с версия 2.0 и версия 2.2. В кода на ядрата 2.0 съответствието е твърдо зададено във файла *drivers/net/Spacd.c* от изходния код на ядрото. Зададените съответствия в този файл са:

Интерфейс	В/И Порт	IRQ
plip0	0x3BC	7
plip1	0x378	7
plip2	0x278	5

Ако сте конфигурирали вашия принтерски порт по различен начин, трябва да промените тези стойности в *drivers/net/Spacd.c* и да компилирате ново ядро.

[%] Patch-кода, съдържащ подобрения драйвер за паралелен порт за версия 2.0 на ядрото може да бъде намерен на адрес *http://www.cyberelc.demon.co.uk/parport.html*.

[!] NCSA *telnet* е популярна програма за DOS, която предоставя TCP/IP през Ethernet или PLIP и поддържа *telnet* и FTP.

[*] Можете да намерите Niibe на адрес *gniibe@mri.co.jp*.

В ядрата 2.2, PLIP драйверът използва драйвер за споделяне на паралелния порт "parport", разработен от Philip Blundell.* Новият драйвер задава последователно имената на паралелните мрежови устройства, аналогично на Ethernet и PPP драйверите, така че първото създадено PLIP устройство е *plip0*, второто е *plip1* и т.н. Хардуерът на физическия паралелен порт също се резервира последователно. По подразбиране, драйверът за паралелния порт се опитва да открие хардуера на вашия паралелен порт с помощта на процедура за автоматична проверка като записва информацията за физическото устройство по реда на намирането му. По-добра практика е физическите входно/изходни параметри да се зададат изрично на ядрото. Това може да се направи или чрез подаване на аргументи на модула *parport_pc.o* по време на зареждането му, или, ако сте компилирали драйвера в ядрото, чрез подаване на аргументи по време на зареждането на ядрото с помощта на lilo. Настройката на IRQ линия за кое да е устройство може да бъде променена по-късно чрез записване на новата стойност на IRQ в съответния файл *proc/parport/*/irq*.

Конфигурирането на физическите входно/изходни параметри при ядрата от серия 2.2 по време на зареждане на модула е съвсем просто. Например, за да укажете на драйвера, че имате два паралелни порта (от PC-тип) на входно/изходни адреси `0x278` и `0x378`, използващи съответно IRQ 5 и 7, трябва да заредите модула със следните аргументи:

```
modprobe parport_pc io=0x278,0x378 irq=5,7
```

Съответните аргументи, които трябва да се подадат на ядрото при компилиран в него драйвер са:

```
parport=0x278,5 parport=0x278,7
```

За да подадете тези аргументи на ядрото автоматично по време на неговото зареждане, трябва да използвате ключовата дума `append`.

Когато PLIP драйверът се инициализира, независимо дали по време на зареждането, ако е вграден, или когато се зарежда модула *plip.o*, с всеки паралелен портове се асоциира мрежово устройство *plip*. С първото паралелно устройство ще бъде свързан *plip0*, с второто – *plip1* и т.н. Можете ръчно да промените тези асоциации като използвате друг набор от аргументи за ядрото. Например, за да свържете

* Можете да намерите Philip на адрес *Philip.Blundell@pobox.com*

`parport1` с мрежовото устройство `plip0` и `parport0` на мрежово устройство `plip1`, трябва да подадете следните аргументи на ядрото:

```
plip=parport1 plip=parport0
```

Това задаване обаче не означава, че не можете да използвате тези паралелни портове за печат или за други цели. Физическите паралелни портове се използват от PLIP драйвера само когато съответният интерфейс е конфигуриран.

PPP и SLIP драйвери

Протоколите PPP и SLIP се използват широко за пренасяне на IP пакети през серийна линия. Голям брой организации предлагат комутируем PPP и SLIP достъп до свързани с Интернет машини, като по този начин осигуряват IP връзка на частни лица (нещо, което по друг начин е трудно осъществимо).

За да се използва PPP или SLIP не са нужни хардуерни изменения; можете да използвате всеки сериен порт. Тъй като конфигурирането на серийния порт не е специфична за работа в TCP/IP мрежа, ние сме отделили специална глава за това. За повече информация разгледайте Глава 4, *Конфигуриране на серийния хардуер*. PPP се разглежда подробно в Глава 8, *Протоколът PPP*, а SLIP – в Глава 7, *IP през серийна линия*.

Други типове мрежи

Повечето други типове мрежи се конфигурират подобно на Ethernet. Аргументите, подавани на зареждаемите модули ще бъдат различни, а някои драйвери могат да не поддържат повече от една карта, но всичко останало е същото. Документацията за тези карти обикновено се намира в директорията */usr/src/linux/Documentation/networking* в дървото с изходен код на ядрото на Linux.

ГЛАВА 4

КОНФИГУРИРАНЕ НА СЕРИЙНИЯ ХАРДУЕР

Интернет се разраства с невероятните темпове. До голяма степен това развитие се дължи на Интернет потребителите, които не могат да си позволят високоскоростна постоянна връзка към мрежата и използват протоколи като SLIP, PPP или UUCP, за да се свържат с някой мрежов доставчик и да получат дневната си порция електронна поща и новини.

Предназначението на тази глава е да помогне на всички хора, които разчитат на модем, за да поддържат своята връзка с останалия свят. Няма да се спираме на механичното конфигуриране на вашия модем (упътването, предоставяно с него, вероятно повече информация по темата, отколкото ние можем да ви дадем), а ще разгледаме специфичните за Linux аспекти на управлението на устройства, които използват серийни портове. Темите в главата включват софтуер за серийни комуникации, създаване на файловете за серийни устройства, сериен хардуер и конфигуриране на серийни устройства чрез командите *setserial* и *stty*. Много други въпроси, свързани с тези теми, са разгледани в документа Serial HOWTO от David Lawyer.

Комуникационен софтуер за връзки през модем

Съществуват множество комуникационни пакети за Linux. Много от тях са *терминални програми*, които позволяват на потребителя да влезе в друг компютър, както ако стои пред прост терминал. Традиционната терминална програма за Unix средие *kermit*. Днес, обаче, тя е доста остаряла и използването ѝ създава трудности. Съществуват по-удобни програми, поддържащи възможности като телефонни указатели, скрипт-езици, автоматизиращи свързването и влизането в отдалечени компютърни системи и разнообразие от протоколи за обмен на файлове. Една от тези програми е *minicom*, която е моделирана по подобие на някои от най-известните терминални програми за DOS. Потребителите на X11 също не са забравени. *seyon* е X11-базирана програма за комуникация с всички необходими възможности.

Терминалните програми не са единствения достъпен тип програми за серийно комуникиране. Други програми ви позволяват да се свържете с хост и да изтеглите като един пакет новини и електронна поща и да ги четете или отговаряте по-късно по ваше желание. Това може да ви спести доста време и е доста полезно, ако за ваше нещастие живеете в област, в която местните разговори се таксуват на време. Можете да прекарате времето за четене и написване на отговор без да имате връзка. Когато сте готови, просто отново установявате връзка и изпращате отговорите си отново като един-единствен пакет. За целта трябва да използвате малко повече дисково пространство, защото съобщенията трябва да се запишат на вашия твърд диск, за да ги прочетете после, но това вероятно е по-добрата алтернатива при днешните цени на устройства за съхранение на данни.

UUCP е символ на този стил комуникационен софтуер. Това е комплект програми, които копират файлове от един хост на друг и изпълняват програми на отдалечен хост. Използва се често за прехвърляне на поща или новини в частни мрежи. UUCP пакетът на Ian Taylor, който работи и под Linux, е описан подробно в глава 16: *Управление на Taylor UUCP*. Друг неинтерактивен комуникационен софтуер се използва при мрежи от типа на Fidonet. Достъпни са и приложни адаптации на Fidonet като *ifmail*, но очакваме, че хората, които все още ги използват, не са много.

PPP и SLIP се намират по средата на току-що описаните тенденции и позволяват както интерактивна, така и неинтерактивна употреба. Много хора използват PPP или SLIP, за да се свържат с техния дос-

тавчик на Интернет услуги и да използват FTP или да разглеждат web-страници. PPP и SLIP се използват често и при постоянни или полупостоянни връзки за свързване на локални мрежи, но това представлява интерес само при ISDN или друг вид високоскоростна мрежова връзка.

Запознаване със серийните устройства

Ядрото на Unix предоставя устройства за достъп до серийния хардуер, които обикновено се наричат *tty* устройства. *tty* е съкращение от *Teletype device* (устройство на Teletype), който беше един от основните производители на терминални устройства в ранните години на Unix. Този термин днес се използва за всеки символно-базиран терминал за данни. В тази глава използваме термина изключително за обозначаване на файловете за устройства на Linux, а не за самия физически терминал.

Linux предоставя три класа tty устройства: серийни устройства, виртуални терминали (всеки от които е достъпен с натискане на съответния клавиш от Alt-F1 до Alt-*Fnn* в локалната конзола) и псевдо терминали (подобни на двупосочните канали, използвани от приложения като X11). Първите се наричаха tty устройства, защото първоначалните символно-базирани терминали се свързваха към Unix машина със сериен кабел или през телефонна линия и модем. Последните две бяха наречени tty устройства, защото бяха проектирани да работят по подобен начин от програмна гледна точка.

SLIP и PPP най-често се реализират в ядрото. Ядрото в действителност не работи с *tty* устройството като с мрежово устройство, което можете да управлявате като Ethernet устройство, използвайки команди като *ifconfig*. Все пак, ядрото третира tty устройствата като места, където могат да се свържат мрежови устройства. За да направи това, ядрото променя така наречената "дисциплина на линията" на tty устройството. И SLIP, и PPP са дисциплини на линията, които могат да се активират на tty устройства. Общата идея е, че серийният драйвер обработва подадените му данни различно, в зависимост от дисциплината на линията, за която е конфигуриран. Според подразбиращата се дисциплина, той просто предава последователно всеки получен символ. Когато се зададе дисциплина на линията SLIP или PPP, драйверът чете блок от данни, прикрепя към него заглавие, което позволява на отсрещната страна да идентифицира този блок от данни

от потока, и предава новия блок от данни. На този етап все още не е от особена важност да разберете този механизъм; ще разгледаме SLIP и PPP в следващите глави, а всичко това така или иначе става автоматично.

Достъп до серийните устройства

Като всички устройства в една Unix система, серийните портове са достъпни чрез специални файлове, разположени в директорията */dev*. Съществуват две разновидности на файловете на устройства, свързани със сериен хардуер, а за всеки тип на всеки порт има по един такъв файл. Устройството ще работи по малко по-различен начин в зависимост от това, кой от неговите файлове сме отворили. Ще разгледаме тези разлики, защото това ще ви помогне да разберете някои от конфигурациите и съветите, свързани със серийните устройства, които може би ще срещнете, но на практика ще трябва да използвате само един файл на устройство. В бъдеще, някой от тези файлове може дори напълно да изчезне.

Най-важният от двата класа серийни устройства има основен номер 4, а неговите специалните файлове на устройство са с имена *ttyS0, ttyS1* и т.н. Втората разновидност има основен номер 5 и е създадена за случаите, в които набирате телефонен номер през порт; нейните специални файлове са *cua0, cua1* и т.н. В света на Unix, броенето започва от 0, докато обикновено хората започват от 1. Това създава малко объркване, защото `COM1`: съответства на */dev/ttyS0,* `COM2`: – на */dev/ttyS1* и т.н. Всеки, който е запознат с хардуера в IBM PC знае, че `COM3`: и следващите го портове никога не са били стандартизирани.

Устройствата *cua* (или "callout") са създадени, за да решават проблемите за избягване на конфликти при серийни устройства за модеми, които трябва да поддържат и входящи, и изходящи връзки. За съжаление, те самите създават нови проблеми и е много вероятно използването им да се преустанови. Да разгледаме накратко тези проблеми.

Linux, аналогично на Unix, позволява едно устройство, както и всеки друг файл, да бъде отворено едновременно от няколко процеса. За съжаление, това много рядко е полезно при tty устройствата, защото двата процеса почти сигурно ще си пречат един на друг. Все пак бе създаден механизъм, който дава възможност един процес да провери дали едно *tty* устройство е вече отворено от друг процес, преди самият той да го отвори. Този механизъм използва така наречените заключващи файлове (*lock files*). Идеята е преди един процес да отвори

едно tty устройство, той да провери дали съществува определен файл на специално място, като името на файла е подобно на устройството, което процеса иска да отвори. Ако не съществува такъв файл, процесът го създава и отваря tty устройството. Ако файлът съществува, процесът приема, че друг процес вече е отворил това устройство и предприема съответните действия. Още един хитър трик, за да може системата за управление на заключващите файловете да работи добре, е идентификатора на процеса, който е създал заключващия файл, да се запише в самия файл; ще поговорим повече за това след малко.

Механизмът на заключващите файлове работи перфектно в случаите, в които имате дефинирано място за тези файлове и всички програми знаят къде да ги намерят. Уви, в Linux това не винаги е така. Не беше до момента, в който документа Linux Filesystem Standard определи стандартно място за заключващите файлове, след което заключващите файлове за *tty* започнаха да работят правилно. Имаше момент, в който съществуваха поне четири, а може би дори повече места, в които софтуерните разработчици съхраняваха заключващите файлове: */usr/spool/locks/*, */var/spool/locks/*, */var/lock/* и */usr/lock/*. Объркването причиняваше хаос. Програмите отваряха на различни места заключващи файлове, които са предназначени за управление на едно и също tty устройство; поради това ситуацията беше като че ли заключващите файлове изобщо не се използват.

Устройствата *cua* бяха създадени, за да предоставят решение на този проблем. Вместо да разчитат на заключващите файлове за предотвратяване на противоречията между програмите, желаещи да използват серийните устройства, беше решено, че ядрото може да осигури просто средство, което да определя на кого да бъде даден достъп. Ако устройството *ttyS* вече е било отворено, опитът да се отвори *cua* ще доведе до грешка, която клиента може да интерпретира като индикатор, че устройството вече се използва. Ако устройството *cua* вече е отворено и се направи опит да се отвори *ttyS*, заявката се блокира, т.е. тя се оставя да чака, докато устройството *cua* се затвори от другия процес. Това работи много добре, ако имате единствен модем, който сте конфигурирали за входящ достъп и от време на време искате да използвате същото устройство за изходящ достъп. Но същото не работи толкова добре в среди, в които имате няколко програми, които искат да наберат номер през същото устройство. Единственият начин да се реши този проблем е да се използват заключващи файлове! Връщаме се там, откъдето започнахме.

Достатъчно е да се каже, че Linux Filesystem Standard дойде на помощ и сега е определено, че заключващите файлове трябва да се съхраняват в директорията */var/lock/* и по конвенция, името на заключващия файл например за устройство *ttyS1* е *LCK.ttyS1*. Заключващите файлове за устройствата *cua* също трябва да се съхраняват в тази директория, но днес използването на тези устройствата не се препоръчва.

Устройствата *cua* вероятно ще се запазят още известно време, за да се осигури период на обратна съвместимост с по-старите версии, но след време ще бъдат премахнати. Ако не знаете какво да използвате, придържайте се към устройствата *ttyS* и се уверете, че системата ви съответства на стандарта Linux FSSTND, или поне, че всички програми, използващи серийни устройства, използват едно и също място за съхранение на заключващите файлове. Повечето софтуер, използващ tty устройства, дава възможност по време на компилиране да се определи разположението на заключващите файлове. Често това се прави чрез променлива с име като `LOCKDIR` в *Makefile*-а или в заглавен конфигурационен файл. Ако сами компилирате софтуера, най-добре е да го промените така, че да използва определеното от FSSTND място. Ако използвате предварително компилирани програми и не сте сигурни къде програмата ще запише нейните заключващи файлове, може да използвате следната команда, за да проверите това:

```
strings binaryfile | grep lock
```

Ако намереното място не съвпада с използваното във вашата система, можете да създадете символна връзка от директорията за заключващия файл, която иска да използва външната програма, към директорията */var/lock/*. Това е грозно, но ще работи.

Специални файлове за серийните устройства

Вторичните номера за двата типа серийни устройства са идентични. Ако вашият модем е на някой от портовете от COM1: до COM4:, неговия вторичен номер ще бъде номера на COM-порта плюс 63. Ако използвате специален сериен хардуер като високопроизводителен сериен контролер с много портове, най-вероятно ще ви се наложи да създадете специални файлове за устройства за него; вероятно той няма да използва стандартния драйвер. Документът Serial-HOWTO съдържа полезна информация, която може да ви помогне в намирането на съответните подробности за решаването на проблема.

Да предположим, че модемът вие на COM2:. Неговият вторичен номер ще бъде 65, а основният му номер ще бъде 4 при нормална употреба. Трябва да разполагате с устройство, наречено *ttyS1*, което има тези номера. Разгледайте имената на серийните tty устройства в директорията */dev/*. Петата и шестата колони показват основните и вторичните номера, съответно:

```
$ ls -l /dev/ttyS*
0 crw-rw---- 1 uucp  dialout  4,  64 Oct 13 1997  /dev/ttyS0
0 crw-rw---- 1 uucp  dialout  4,  65 Jan 26 21:55 /dev/ttyS1
0 crw-rw---- 1 uucp  dialout  4,  66 Oct 13 1997  /dev/ttyS2
0 crw-rw---- 1 uucp  dialout  4,  67 Oct 13 1997  /dev/ttyS3
```

Ако няма устройство с основен номер 4 и вторичен номер 65, ще трябва да създадете такова. Влезте като супер потребител и въведете командите:

```
# mknod -m 666 /dev/ttyS1 c 4 65
# chown uucp.dialout /dev/ttyS1
```

Различните дистрибуции на Linux използват леко различаващи се стратегии за това, кой трябва да бъде собственик на серийните устройства. Понякога те са собственост на *root*, друг път – на различен потребител; в нашия пример – **uucp**. Модерните дистрибуции имат група специално за устройства, позволяващи набиране и всеки потребител, на който е позволено да ги използва, се добавя към тази група.

Някои хора предлагат да се направи символна връзка с име */dev/modem* към устройството-модем, за да не се налага потребителите да помнят донякъде неинтуитивното *ttyS1*. Все пак, не можете да използвате в една програма *modem*, а в друга – реалното име на файла на устройството. Техните заключващи файлове ще имат различни имена и заключващия механизъм няма да работи.

Сериен хардуер

В момента RS-232 е най-популярния стандарт за серийни комуникации в света на персоналните компютри. В него се използват няколко вериги за предаване на единични битове и за синхронизация. Могат да се използват допълнителни линии за сигнализиране на наличието на носеща честота (използва се от модемите) и за договаряне. Linux поддържа голямо разнообразие от серийни карти, използващи стандарта RS-232.

Хардуерното договаряне не е задължително, но е доста полезно. То позволява на всяка от двете станции да сигнализира дали е готова да приеме още данни или дали другата станция трябва да направи пауза, докато приемащият свърши с обработката на получените данни. Линиите, използвани за това, се наричат съответно CTS (Clear to Send) и RTS (Ready to Send), което обяснява неофициалното име на хардуерното договаряне: RTS/CTS. Другият тип такова договаряне, за който може би сте запознати, се нарича XON/XOFF. В XON /XOFF се използват два определени символа, по конвенция Ctrl-S и Ctrl-R, за да се сигнализира на другата страна съответно да спре и да започне да предава данни. Макар че този метод е лесен за реализация и добър за употреба от "глупави" терминали, той причинява големи проблеми, когато се работи с двоични данни, защото е възможно тези символи да са просто част от вашия поток данни, но да бъдат интерпретирани като символи, управляващи самия поток. Освен това, този начин е малко по-бавен от хардуерното договаряне. Самото хардуерно договаряне е ясно, бързо и се предпочита пред XON/XOFF, когато има възможност за избор.

В оригиналния компютър IBM PC, интерфейсът RS-232 се управлява от UART чип, наречен 8250. PC-компютрите по времето на 486 използваха нова версия на UART, наречена 16450. Тя беше малко по-бърза от 8250. Почти всички машини, базирани на Pentium, бяха снабдени с още по-нова версия на UART, наречена 16550. Някои производители (най-често на вътрешни модеми, снабдени с чипове Rockwell) използват напълно различни чипове, които симулират поведението на 16550 и могат да се управляват по подобен начин. Linux поддържа всеки от тези чипове със своя стандартен драйвер за сериен порт.

16550 беше значително подобрение на 8250 и 16450, защото предлага 16-байтов FIFO буфер (опашка). 16550 всъщност е семейство от UART устройства, състоящо се от 16550, 16550A и 16550AFN (по-късно преименуван на 16550DN). Разликата между тях е свързана с това дали FIFO действително работи; 16550AFN е единственият, при които буфера работи със сигурност. Преди време се произвеждаше и NS16550, но при него FIFO буфера в действителност никога не е работил.

8250 и 16450 UART имат прост 1-байтов буфер. Това означава, че 16450 генерира прекъсване за всеки изпратен или получен символ. Всяко прекъсване се нуждае от кратко време за обслужване и това

малко забавяне ограничава 16450 до максимална надеждна скорост от около 9,600 bps на типична машина с ISA шина.

В конфигурацията по подразбиране, ядрото проверява четирите стандартни серийни порта от COM1: до COM4:. Освен това, ядрото може автоматично да открие какъв UART се използван на всеки от стандартните серийни портове и при възможност ще използва увеличения FIFO буфер на 16550, ако той е налице.

Използване на конфигурационните инструменти

Сега да отделим малко време, за да разгледаме два от най-полезните конфигурационни инструмента за серийни устройства: *setserial* и *stty*.

Командата setserial

Ядрото ще направи всичко възможно, за да определи правилно как е конфигуриран вашия сериен хардуер. За съжаление, вариациите на конфигурациите на серийните устройства прави много трудно на практика това определяне да бъде със 100% надеждност. Един добър пример за това, къде се среща такъв проблем, са вътрешните модеми, за които говорихме по-рано. UART-чипа, който те използват, има 16-байтов FIFO буфер, но той изглежда като 16450 UART за драйвера на устройството в ядрото: докато изрично не зададем на драйвера, че този порт е 16550 устройство, ядрото няма използва разширения буфер. Друг пример са простите 4-портови карти, които позволяват използването на един IRQ канал от няколко серийни устройства. Може да се наложи специално да укажете на ядрото, кой IRQ порт би трябвало да използва и че IRQ линиите могат да бъдат споделени.

Програмата *setserial* беше създадена, за да конфигурира серийния драйвер по време на работата му. Командата *setserial* се изпълнява най-често по време на зареждане от скрипт с име *Osetserial* при някои дистрибуции или *rc.serial* при други. Този скрипт се грижи за инициализирането на серийния драйвер, за да се адаптира за всякакъв нестандартен или необикновен сериен хардуер в машината.

Общият синтаксис на командата *setserial* е:

```
setserial устройство [параметри]
```

където *устройство* е едно от серийните устройства, например *ttyS0*.

Командата *setserial* има голям брой параметри. Най-често срещаните от тях са описани в Таблица 4-1. За информация относно останалите параметри, разгледайте справочната страница на *setserial*.

Таблица 4-1. Параметри от командния ред на setserial

Параметър	Описание
`port` *`номер_на_порт`*	Задава адреса на I/O порта на серийното устройство. Стойностите трябва да бъдат задавани като шестнадесетично число, например `0x2f8`.
`irq` *`номер`*	Задава линията на заявката за прекъсване, която използва серийното устройство.
`uart` *`uart_type`*	Задава типа UART на серийното устройство. Най-често използваните стойности са `16450` и `16550`. Задаването на стойност `none` ще забрани това устройство.
`fourport`	Задаването на този параметър инструктира серийния драйвер в ядрото, че този порт е от картата ASP Fourport.
`spd_hi`	Програмира UART да използва скорост от 57.6 kbps, когато някой процес изисква 38.4 kbps.
`spd_vhi`	Програмира UART да използва скорост от 115 kbps, когато някой процес изисква 38.4 kbps.
`spd_normal`	Програмира UART при заявка да използва подразбиращата се скорост 38.4 kbps. Този параметър се използва, за да се премахне ефекта от `spd_hi` и `spd_vhi` върху определено устройство.
`auto_irq`	Този параметър ще укаже на ядрото да опита автоматично да определи IRQ линията на съответното устройство. Този опит не дава пълна гаранция за успех, затова можете да мислите за него като заявка към ядрото да предположи кой е номера на IRQ. Ако знаете IRQ линията на устройството, можете да го зададете с параметъра `irq`.

Параметър	Описание
`autoconfig`	Този параметър трябва да се използва заедно с параметъра port. Когато зададете този параметър, *setserial* инструктира ядрото да се опита автоматично да определи типа UART, разположен на дадения адрес. Ако е зададен и параметърът `auto_irq`, ядрото ще се опита автоматично да определи и номера на IRQ.
`skip_test`	Този параметър инструктира ядрото да не се опитва да открие типа UART по време на автоматичната конфигурация. Това е необходимо в случаите, когато ядрото не успява да определи правилно версията на UART.

Един типичен прост *rc*-файл, който конфигурира вашия сериен порт по време на зареждане на операционната система, може да изглежда като представения в Пример 4-1. Повечето дистрибуции на Linux съдържат малко по-усъвършенстван скрипт от представения по-долу.

Пример 4-1: Примерни команди setserial в скрипта rc.serial

```
# /etc/rc.serial - конфигурационен скрипт за серийната линия.
#
# Конфигуриране на серийни устройства
/sbin/setserial /dev/ttyS0 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS1 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS2 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS3 auto_irq skip_test autoconfig
#
# Отпечатване на конфигурацията на серийното устройство
/sbin/setserial -bg /dev/ttyS*
```

Аргументът `-bg /dev/ttyS*` в последната команда ще изведе на екрана добре оформено обобщение на хардуерната конфигурация на всички активни серийни устройства. Резултатът ще изглежда подобно на показания в Пример 4-2.

Пример 4-2: Резултат от командата setserial -bg /dev/ttyS

```
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

Командата stty

Името *stty* вероятно означава "set tty" (настрой tty), но може да се използва и за отпечатване на конфигурацията на терминала. Може би дори повече от *setserial*, командата *stty* осигурява огромен брой възможности за конфигуриране. Ние ще обърнем внимание само на най-важните от тях. Информация относно другите параметри можете да намерите в справочната страница на *stty*.

Командата *stty* се използва най-често за конфигуриране на параметрите на терминали, например дали въведените символи да бъдат отпечатвани или кой ключ трябва да генерира сигнал за прекъсване. По-горе обяснихме, че серийните устройства са tty устройства и следователно командата *stty* е приложима и за тях.

Едно от най-важните приложения на *stty* за серийни устройства е да разрешава хардуерното договаряне за устройството. Вече споменахме накратко за хардуерното договаряне по-горе. Конфигурацията по подразбиране за серийните устройства е то да е забранено. Тази настройка позволява работата по "трижични" серийни кабели; те не поддържат необходимите сигнали за хардуерно договаряне и ако по подразбиране то беше активирано, те нямаше да могат да изпращат символи, които да го променят.

Учудващо някои програми за серийни комуникации не разрешават хардуерно договаряне, така че ако вашият модем го поддържа трябва да го конфигурирате да го използва (вижте в упътването към модема каква команда трябва да използвате), а също да конфигурирате и серийното си устройство. Командата *stty* има флаг `crtscts`, който разрешава хардуерното договаряне на устройството; този флаг ще ви бъде необходим. Вероятно е най-добре да изпълните тази команда от файла *rc.serial* (или негов еквивалент) по време на зареждане на операционната система, използвайки команди като показаните в Пример 4-3.

Пример 4-3: Примерни команди stty в rc.serial

```
#
stty crtscts < /dev/ttyS0
stty crtscts < /dev/ttyS1
stty crtscts < /dev/ttyS2
stty crtscts < /dev/ttyS3
#
```

По подразбиране, командата *stty* работи на текущия терминал, но като използваме пренасочване на входния поток ("<") на обвивката,

можем да използваме *stty* за управление на всяко tty устройство. Често срещана грешка е да се забрави кой знак трябва да се използва – "<" или ">", затова новите версии на *stty* имат доста по-изчистен синтаксис за правене на това. Използвайки този нов синтаксис ще преначертаем нашия пример, както е показано в Пример 4-4.

Пример 4-4: Примерни команди stty в rc.serial, използвайки модерния синтаксис

```
#
stty crtscts -F /dev/ttyS0
stty crtscts -F /dev/ttyS1
stty crtscts -F /dev/ttyS2
stty crtscts -F /dev/ttyS3
#
```

Както споменахме, командата *stty* може да се използва за отпечатване на конфигурационните параметри на терминала за tty устройство. За да изведете всички активни настройки на едно такова устройство, използвайте:

```
$ stty -a -F /dev/ttyS1
```

Резултатът от тази команда, показан в Пример 4-5, показва състоянието на всички флагове за това устройство; ако флага се предхожда от минус, както при *-crtscts*, това означава, че флагът е бил изключен.

Пример 4-5: Резултат от изпълнението на командата stty -a

```
speed 19200 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>
   eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R;
   werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr -icrnl -
ixon
   -ixoff -iuclc -ixany -imaxbel
-opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0
   bs0 vt0 ff0
-isig -icanon -iexten echo echoe echok -echonl -noflsh -xcase -tostop
   -echoprt echoctl echoke
```

Описание на най-важните от тези флагове можете да намерите в Таблица 4-2. Всеки от тях се разрешава като се добави към *stty* и се забранява като го зададете на *stty* със знак минус пред него. Следователно, за да забраните хардуерното договаряне на устройството `ttyS0`, трябва да използвате:

```
$ stty -crtscts -F /dev/ttyS0
```

Таблица 4-2: stty флагове, свързани с конфигурирането на серийни устройства

Флагове	Описание
`N`	Задава скорост на линията N бита в секунда.
`crtsdts`	Разрешава/забранява хардуерното договаряне.
`ixon`	Разрешава/забранява XON/XOFF контрол на потока.
`clocal`	Разрешава/забранява управляващи сигнали на модема като DTR/DTS и DCD. Това е необходимо, ако използвате "прижичен" сериен кабел, тъй като той не поддържа тези сигнали.
`cs5 cs6 cs7 cs8`	Задава броя на битовете с данни съответно на 5, 6, 7 или 8.
`parodd`	Разрешава контрол по нечетност. Забраняването на този флаг активира контрол по четност.
`parenb`	Разрешава контрол по четност. Забраняването на този флаг дезактивира контрола по четност.
`cstopb`	Разрешава използването на два стоп-бита на символ. Когато този флаг се забрани, се използва един стоп-бит на символ.
`echo`	Разрешава/забранява ехото на получени символи обратно до изпращача.

В следващият пример са комбинирани някои от тези флагове за настройка на устройството *ttyS0* на 19,200 bps, 8 битови данни, без контрол по четност, забранено ехо и хардуерното договаряне:

```
$ stty 19200 cs8 -parenb crtscts -echo -F /dev/ttyS0
```

Серийните устройства и поканата за влизане в системата

Преди време беше нещо обикновено една инсталация на Unix да се състои от машина-сървър и множество обикновени (dumb) символни терминали или модеми за телефонна линия. Днес този вид инсталация е по-малко разпространена, което е добра новина за хората, кои-

то искат да използват този стил, защото е доста евтино да се добиеш с необходимите обикновени терминали. Конфигурациите с модеми си остават разпространени, но днес те се използват повече, за да поддържат SLIP и PPP влизане (виж Глава 7, *IP през серийна линия* и Глава 8, *Протоколът PPP*) отколкото да се използват за обикновено влизане. Независимо от това обаче, за всяка една от тези конфигурации може да използвате една проста програма, наречена *getty*.

Терминът *getty* вероятно е съкращение от "get tty". Програмата *getty* отваря серийно устройство, конфигурира го по подходящия начин, а със съответната опция конфигурира модем и изчаква да се осъществи връзка. Активната връзка на серийно устройство обикновено се разпознава по вдигането на сигнала Data Carrier Detect (DCD) от устройството. Когато се установи връзка, програмата *getty* извежда поканата за влизане и стартира програмата *login*, за да се извърши реално влизане в системата. За всеки от виртуалните терминали в Linux (например */dev/tty1*) има работеща команда *getty*.

Съществуват множество различни реализации на *getty*, всяка от които е подходяща за определена конфигурация повече от другите. Варианта на *getty*, който ние ще опишем, се нарича *mgetty*. Той е доста популярен, защото има всички възможности, които го правят непретенциозен спрямо модемите, включително и поддръжка на автоматични факс-програми и гласови модеми. Ще се концентрираме върху конфигурирането на *mgetty* за отговаряне на конвенционалните телефонни обаждания за прехвърляне на данни, а останалите възможности ще оставим да разучите сами.

Конфигуриране на демона mgetty

Демонът *mgetty* може да бъде намерен на адрес *ftp://alpha.greenie.net/pub/mgetty/source* и е достъпен в предварително пакетирана форма за почти всички дистрибуции на Linux. Той се различава от повечето други реализации на *getty* по това, че е проектиран специално за Hayes-съвместими модеми. Той продължава да поддържа директни терминални връзки, но е най-подходящ за приложения за телефонна линия. Вместо да използва линията DCD за разпознаване на входящо позвъняване, той следи за съобщението `RING`, което се генерира от по-модерните модеми при откриване на входящо позвъняване, когато те не са конфигурирани за автоматично отговаряне.

Основната изпълнима програма се нарича */usr/sbin/mgetty*, а нейния главен конфигурационен файл е */etc/mgetty/mgetty.config*. Съществуват и други двоични програми и конфигурационни файлове, които реализират другите възможности на *mgetty*.

За повечето инсталации, конфигурирането представлява просто редактиране на файла */etc/mgetty/mgetty.config* и добавяне на съответните записи за автоматично стартиране на *mgetty* във файла */etc/inittab*.

На Пример 4-6 е показан един прост конфигурационен файл за *mgetty*. В този пример се конфигурират две серийни устройства. Първото, */dev/ttyS0*, поддържа Hayes-съвместим модем със скорост 38,400 bps. Второто, */dev/ttyS1*, поддържа директно свързан VT100 терминал със скорост 19,200 bps.

Пример 4-6: Примерен конфигурационен файл /etc/mgetty/mgetty.config

```
#
# конфигурационен файл за mgetty
#
# това е примерен конфигурационен файл, за подробности виж mgetty.info
#
# редовете с коментари започват с "#", празните редове се игнорират
#
# ---- общ раздел -----
#
# В този раздел се задават глобалните подразбиращи се стойности,
# нещата за отделните портове са по-надолу
#
# достъп до модем(ите) със скорост 38400 bps
speed 38400
#
# определя глобалното ниво за debug на "4" (идва от policy.h)
debug 4

# ----- раздел за конкретни портове ------
#
# Тук се поставят нещата, валидни само за една линия, но не за всички
#
# Hayes модем, свързан към ttyS0: без факс, по-малко статистика
#
port ttyS0
  debug 3
  data-only y
#
# директна връзка с VT100 терминал, който не обича свалянето на DTR
#
```

```
port ttyS1
  direct y
  speed 19200
  toggle-dtr n
#
```

Конфигурационният файл поддържа глобални и специфични за всеки порт опции. В нашия пример използвахме една глобална опция, за да зададем скорост 38,400 bps. Тази стойност се наследява от порта *ttyS0*. Портовете, за които използваме *mgetty*, работят на тази скорост, освен ако тя не се промени от специфична за порта настройка на скоростта, както направихме в нашия пример с конфигурирането на *ttyS1*.

Запазената дума `debug` контролира подробността на записваната в дневниците информация от *mgetty*. Ключовата дума `data-only` в конфигурацията на *ttyS0* указва на *mgetty* да игнорира всякакви възможности на факс-модеми и да работи просто като модем за данни. Опцията `direct` в конфигурацията на *ttyS1* инструктира *mgetty* да не се опитва да инициализира модем на този порт. И накрая, `toggle-dtr` задава на *mgetty* да не затваря връзката чрез сваляне на сигнала DTR (Data Terminal Ready – терминала за данни е готов) от серийния интерфейс; някои терминали не обичат да се случва това.

Възможно е да оставите файла *mgetty.config* празен и да използвате аргументи от командния ред, за да зададете повечето параметри. Документацията, която съпътства приложението, включва пълно описание на параметриге на конфигурационния файл и аргументиге от командния ред на *mgetty*. Вижте следващия пример.

Трябва да добавим два елемента във файла */etc/inittab*, за да активираме тази конфигурация. Файлът *inittab* е конфигурационния файл на командата *init* под Unix System V. Тази команда се грижи за инициализиране на системата и предоставя възможност за автоматично изпълнение на програми по време на зареждане и повторното им изпълнение, когато те приключат. Това е идеално за работата с една *програма getty*.

```
T0:23:respawn:/sbin/mgetty ttyS0
T1:23:respawn:/sbin/mgetty ttyS1
```

Всеки ред от файла */etc/inittab* се състои от четири полета, разделени от двуеточие. Първото поле е идентификатор, който уникално идентифицира един запис от файла; традиционно се състои от два символа, но по-новите версии позволяват и четири. Второто поле е списък на нивата на работа, на които трябва да се изпълни командата от този запис. Нивото на работа е средство за осигуряване на алтернативна конфигурация на машината и се реализира посредством дървета от инициализационни скриптове, съхранявани в директории с имена */etc/rc1.d, /etc/rc2.d* и т.н. Тази възможност обикновено се реализира много просто; можете да построите своите записи просто като използвате другите записи във файла; в документацията на своята система ще намерите повече информация. Третото поле описва кога да се извърши действието. Ако искаме да конфигурираме работата на една *getty* програма, в това поле трябва да се постави `respawn`, което означава, че командата ще бъде изпълнена автоматично повторно, след като приключи работата си. Съществуват и няколко други възможности, но те не са необходими за нашите цели. Четвъртото поле е самата команда, която трябва да се изпълни; това е мястото, където задаваме командата *mgetty* и всички аргументи, които искаме да ѝ подадем. В нашия прост пример ние стартираме и рестартираме *mgetty* когато системата работи на ниво на работа две или три и подаваме като аргумент само името на устройството, което искаме да използваме. Командата *mgetty* по подразбиране поставя */dev/* пред името на устройството, затова не е необходимо ние да го пишем.

Тази глава съдържаше кратко запознаване с *mgetty* и как да се предоставят покани за влизане чрез серийни устройства. По-подробна информация можете да намерите в документа Serial-HOWTO.

След като вече редактирахте конфигурационните файлове, трябва да презаредите *init*, за да могат направените промени да влязат в действие. Просто изпратете сигнал за прекъсване към процеса *init*; той винаги има идентификатор на процес (PID) 1, така че спокойно можете да използвате следната команда:

```
# kill -HUP 1
```

ГЛАВА 5

КОНФИГУРИРАНЕ НА TCP/IP МРЕЖА

В тази глава ще разгледаме всички необходими стъпки за конфигуриране на TCP/IP мрежа на вашата машина. Ще започнем със задаването на IP адреси, постепенно ще се придвижим до конфигурирането на мрежовите TCP/IP интерфейси и ще представим няколко инструмента, които са доста полезни, когато се отстраняват проблемите при инсталирането на мрежа.

Повечето от задачите, разгледани в тази глава, обикновено трябва да се изпълнят само веднъж. След това ще ви се наложи да променяте тези конфигурационни файлове само, когато добавяте нова система към вашата мрежа или ако изцяло преконфигурирате системата си. Някои от командите, използвани за конфигурирането на TCP/IP, обаче, ще трябва да се изпълняват при всяко зареждане на системата. Това обикновено се прави чрез стартирането им от системните скриптове */etc/rc**.

Обикновено, мрежово-специфичната част от тази процедура се съдържа в един скрипт. Името му е различно при различните дистрибуции на Linux. В много от старите дистрибуции той е известен като *rc.net* или *rc.inet*. Понякога можете да видите два скрипта с имена *rc.inet1* и *rc.inet2*; първият от тях инициализира частта от поддръжката на мрежа, намираща се в ядрото, а вторият стартира основните мрежови услуги и приложения. В модерните дистрибуции, *rc*-файловете са структурирани по доста по-добър начин; вече в директорията */etc/init.d/* (или */etc/rc.d/init.d/*) можете да намерите скриптовете, които създават мрежови устройства или стартират мрежови приложни

програми. Примерите в тази книга се базират на по-новото разположение.

В тази глава разглеждаме тези части от скриптовете, които конфигурират вашите мрежови интерфейси; приложенията се описват в следващите глави. След завършване на тази глава, би трябвало да сте определили набора от команди, които могат правилно да конфигурират TCP/IP мрежа на вашия компютър. Тогава ще трябва да замените всички примерни команди в конфигурационните ви скриптове с ваши команди, да се уверите, че скрипта се изпълнява от базовия *rc*-скрипт по време на зареждане и да рестартирате машината си. Мрежовите *rc*-скриптове от любимата ви дистрибуция на Linux трябва да ви осигурят солиден пример, от който да започнете.

Монтиране на файловата система /proc

Някои от конфигурационните инструменти на версиите Linux NET-2 и NET-3 разчитат на файловата система за комуникиране с ядрото. Този интерфейс позволява достъп до информация за работата на ядрото чрез механизъм, наподобяващ файлова система. Когато тя е монтирана, можете да получите списък на файловете в нея като при всяка друга файлова система, или пък да разгледате тяхното съдържание. Типични файлове са *loadavg*, който съдържа информация за средното натоварване на системата и *meminfo*, който показва текущото използване на физическата памет и виртуалната памет.

Към това мрежовият код прибавя и директорията *net*. Тя съдържа множество файлове, които показват данни като ARP таблиците на ядрото, състоянието на TCP връзките, и таблици за маршрутизация. Повечето инструменти за администриране на мрежата получават необходимата им информация от тези файлове.

Файловата система *proc* (или както още е известна, *procfs*) обикновено се монтира в поддиректорията */proc* по време на зареждане на системата. Най-добрия метод за това е да се добави следния ред към файла */etc/fstab*:

```
# място за монтиране на procfs:
none        /proc       proc     defaults
```

След това изпълнете *mount /proc* от вашия */etc/rc* скрипт.

Днес *procfs* по подразбиране се конфигурира в повечето ядра. Ако във вашето ядро няма *procfs*, ще получите съобщение от вида: `mount: fs type procfs not supported by kernel` (`mount`: файловата система *procfs* не се поддържа от ядрото). В такъв случай ще ви се наложи отново да компилирате ядрото и да отговорите с "yes" на въпроса, дали искате поддръжка на *procfs*.

Инсталиране на двоичните файлове

Ако използвате предварително пакетирана дистрибуция на Linux, тя ще съдържа основните мрежови приложения и инструменти, заедно със съответния набор от примерни файлове. Единствения случай, при който може да ви се наложи да си набавите и инсталирате нови инструменти е, когато инсталирате нова версия на ядрото. Тъй като понякога това води до промяна в мрежовия слой на ядрото, ще трябва да подновите основните конфигурационни инструменти. Това подновяване включва поне прекомпилиране, но понякога се налага да си набавите и най-актуалния набор от двоични файлове. Те могат да бъдат намерени на техния официален сайт на адрес *ftp.inka.de/pub/comp/Linux/networking/ NetTools/*, пакетирани в архив, наречен *net-tools-XXX.tar.gz*, където *XXX* е номера на версията. Пакетът, отговарящ на Linux 2.0, е *net-tools-1.45*.

Ако искате сами да компилирате и инсталирате стандартните мрежови приложения за TCP/IP, можете да ги изтеглите от повечето Linux FTP сървъри. Всички модерни дистрибуции на Linux включват пълен набор от TCP/IP мрежови приложения като World Wide Web браузъри, *telnet* и *ftp* програми и други мрежови приложения, например *talk*. Ако разполагате със софтуер, който трябва да компилирате самостоятелно, има голяма вероятност това да стане под Linux доста просто от изходния код, ако спазвате включените в пакета инструкции.

Задаване името на хоста

Повечето, ако не и всички, мрежови приложения разчитат на вас да зададете смислена стойност като име на локалния хост. Тази настройка обикновено се прави по време на процедурата за начално зареждане посредством изпълнение на командата *hostname*. За да зададете стойност *name* като име на хоста, въведете:

```
# hostname name
```

Обичайна практика е да се използва непълно име на хоста, без да се задава името на домейна. Например, хостовете във Виртуалната пивоварна (описани в Приложение А, *Примерна мрежа: Виртуалната пивоварна*) могат да бъдат наречени **vale.vbrew.com** или **vlager.vbrew.com.** Това са техните официални пълни домейн имена (*fully qualified domain name* – FQDN). Техните локални имена на хост ще са първите части на името като vale. Тъй като, обаче, локалното име на хоста често се ползва за откриване на IP адреса на хоста, трябва да се уверите, че библиотеката resolver може да разпознае IP адреса на хоста. Това обикновено означава, че трябва да въведете името на хоста в */etc/hosts*.

Някои хора предлагат да се използва командата *domainname*, за да се зададе на ядрото името на домейна от останалата част от FQDN. По този начин може да се комбинират резултатите от *hostname* и *domainname*, за да се получи FQDN. Това обаче е вярно само наполовина. Командата *domainname* обикновено се използва, за да се зададе NIS домейна на хоста, който може да е напълно различен от DNS домейна, към който принадлежи хоста ви. Вместо това, за да сте сигурни, че кратката форма на името на вашия хост може да се разпознае от всички съвременни версии на командата *hostname*, или го добавете като запис във вашия локален сървър за имена, или поставете пълното домейн име във файла */etc/hosts*. След това можете да използвате аргумента `-fqdn` на командата hostname и тя ще изведе пълното домейн име на вашия хоста.

Задаване на IP адреси

Ако конфигурирате мрежовия софтуер на вашия хост за самостоятелна работа (например, за да може на него да работи софтуера за мрежови новини INN), може спокойно да пропуснете тази част, защото в такъв случай единствения IP адрес, който ще ви е необходим, е за loopback-интерфейса, който винаги е **127.0.0.1**.

Нещата са малко по-сложни с истинските мрежи като Ethernet. Ако желаете да свържете вашия хост към вече съществуваща мрежа, ще трябва да помолите администраторите ѝ да ви дадат IP адрес за тази мрежа. Когато създавате мрежа изцяло самостоятелно, вие сами задавате IP адресите.

Хостовете в рамките на една локална мрежа трябва обикновено да използват адреси от една и съща логическа IP мрежа. Оттук следва, че трябва да зададете IP адрес на мрежата. Ако имате няколко физически мрежи, трябва или да им зададете различни мрежови номера,

или да използвате подмрежи, за да разделите вашия диапазон от IP адреси на няколко подмрежи. Подмрежите се разглежда в следващата част, "Създаване на подмрежи".

Получаването на IP номер на мрежата зависи до голяма степен от намерението ви да се свържете към Интернет в близко бъдеще. Ако възнамерявате да направите това, трябва *веднага* да получите официален IP адрес. Можете да се обърнете към вашия доставчик на мрежови услуги за помощ. Ако желаете да получите мрежов номер само за в случай, че някой ден решите да се включите в Интернет, заявете формата *Network Address Application Form* от *hostmaster@internic.net* или от Мрежовия информационен център във вашата страна, ако съществува такъв.

Ако вашата мрежата не е свързана към Интернет и няма да бъде в близко бъдеще, вие сте свободни да изберете произволен легален мрежов адрес. Само се уверете, че никакви пакети от вашата вътрешна мрежа не излизат навън в Интернет. За да сте сигурни, че няма да има вреди дори и ако стане такова изтичане, използвайте някой от мрежовите номера, запазени за частно използване. Организацията IANA (*Internet Assigned Number Authority*) е запазила няколко номера на мрежи от класовете A, B и C, които могат да бъдат използвани без регистрация. Тези адреси са валидни само в рамките на частната мрежа и не се маршрутизират между истинските Интернет сайтове. Тези номера са дефинирани във RFC 1597 и са изброени в Таблица 2-1 в Глава 2, *Въпроси на работата в TCP/IP мрежа*. Обърнете внимание, че втория и третия блок съдържат съответно 16 и 256 мрежи.

Избирането на вашите адреси между някой от тези мрежови номера е удобно не само за мрежи без всякаква връзка с Интернет; можете по този начин да реализирате малко по-ограничен достъп, използвайки един-единствен хост като шлюз. За вашата локална мрежа шлюзът е достъпен чрез своя вътрешен IP адрес, докато останалия свят знае за него с неговия официално регистриран адрес (даден ви от вашия доставчик). Ще се върнем отново на тази концепция във връзка с възможността за IP маскиране в Глава 11, *IP маскиране и транслиране на мрежови адреси*.

Оттук до края на книгата ще предполагаме, че мрежовият администратор на пивоварната използва мрежов адрес от клас B, например **172.16.0.0**. Разбира се, един мрежов адрес от клас C определено ще удовлетвори условието да е приложим и за мрежата на Пивоварната, и за тази на Винарната. Ние ще използваме мрежа от клас B с цел простота; това ще направи примерите за подмрежи в следващата част на тази глава малко по-лесни за възприемане.

Създаване на подмрежи

За да работите с няколко Ethernet мрежи (или други мрежи, стига да разполагате с необходимия драйвер), трябва да разделите вашата мрежа на подмрежи. Забележете, че разделянето на подмрежи се налага само ако имате повече от една *broadcast* мрежа – връзките от тип точка-до-точка не се броят. Например, ако имате една Ethernet и една или повече SLIP връзки с външния свят, не е нужно да разделяте вашата мрежа. Това е обяснено по-подробно в Глава 7, *IP през серийна линия*.

За да обедини двете Ethernet мрежи, мрежовия управител на Пивоварната решава да използва 8 бита от частта за хоста като допълнителни битове за подмрежа. Това остава други 8 бита за хост частта, позволявайки 254 хоста във всяка от подмрежите. След това той дава номер на подмрежа 1 на пивоварната и номер 2 на винарната. Респективно, мрежовите им адреси в този случай са **172.16.1.0** и **172.16.2.0**. Маската на подмрежа е **255.255.255.0.**

Шлюзът между двете подмрежи е **vlager**, който получава номер на хост 1 за всяка от тях, което прави IP адресите му съответно **172.16.1.1** и **172.16.2.1.**

Забележете, че в този пример използваме мрежа от клас B, за да запазим нещата прости, но мрежа от клас C би била по-реалистична. С новия код за поддръжка на мрежа, разделянето на подмрежи не е ограничено байтови граници, така че дори една мрежа от клас C може да бъде разделена на няколко подмрежи. Например, можете да използвате два бита от хост-частта за мрежова маска, което ви дава четири възможни подмрежи с по 64 хоста във всяка.[3]

Създаване на файловете hosts и networks

След като разделите вашата мрежа на подмрежи, трябва да подготвите просто разпознаване имената на хостовете посредством файла */etc/hosts*. Ако не смятате да използвате DNS или NIS за разпознаване на адреса, трябва да поставите информация за всички хостове във файла *hosts*.

[3] Първият номер във всяка подмрежа е нейният адрес, а последният номер е запазен като broadcast адрес, така че всъщност остават 62 хоста на подмрежа.

Дори ако искате да използвате DNS или NIS по време на нормалната работа, трябва да имате някакво подмножество от всички имена на хостове в */etc/hosts*. Трябва да имате и някакъв вид разпознаване на името, дори когато няма активни мрежови интерфейси, например по време на зареждане. Това е не необходимо не просто за удобство, но и позволява да се използват символни имена на хостове във вашите мрежови *rc*-скриптове. По този начин, когато промените IP адреси, трябва само да копирате обновения файл *hosts* на всички машини и да рестартирате, вместо да редактирате поотделно множество *rc*-файлове. Обикновено трябва да поставите всички локални имена и адреси на хостове в *hosts*, като добавяйки и информация за използваните шлюзове и NIS сървъри.[4]

Трябва да се уверите, че вашият резолвер използва информация от файла *hosts* само по време на първоначалното тестване. Примерните файлове, които придружават вашия DNS или NIS софтуер, могат да предизвикат странни резултати. За да накарате всички приложения да използват единствено */etc/hosts*, когато търсят IP адреса на хост, трябва да редактирате файла */etc/host.conf*. Коментирайте всеки ред, който започва с ключовата дума `order` като поставите пред него знака диез (#) и вмъкнете реда:

```
order hosts
```

Конфигурацията на библиотеката-резолвер е разгледана подробно в Глава 6, *Конфигуриране на услугата за имена и резолвера*.

Файлът *hosts* се състои от един запис на ред, включващ IP адрес, име на хост и незадължителен списък от псевдоними на името на хоста. Полетата са разделени от интервали или табулации, а полето с адреса трябва да започва в първата колона. Всичко след знака диез (#) се счита за коментар и се игнорира.

Имената на хостовете могат да бъдат както пълни, така и относителни спрямо локалния домейн. За **vale** във файла *hosts* обикновено трябва да зададете пълното име – **vale.vbrew.com** – и отделно само **vale**, така че хоста да е известен както с официалното си име, така и с по-краткото си локално име.

[4] Адресът на даден NIS сървър ви е необходим само, ако използвате NYS на Peter Eriksson. Другите реализации на NIS откриват своите сървъри само по време на работа като използват *ypbind*.

Ето един пример как може да изглежда файла *hosts* на Виртуалната пивоварна. Включени са две специални имена – **vlager-if1** и **vlager-if2**, които дават адресите за двата интерфейса, използвани от **vlager**:

```
#
# Файлът hosts за Виртуалната пивоварна / Виртуалната винарна
#
# IP            FQDN                    псевдоними
#
localhost
#
172.16.1.1      vlager.vbrew.com        vlager vlager-if1
172.16.1.2      vstout.vbrew.com        vstout
172.16.1.3      vale.vbrew.com          vale
#
172.16.2.1      vlager-if2
172.16.2.2      vbeaujolais.vbrew.com   vbeaujolais
172.16.2.3      vbardolino.vbrew.com    vbardolino
172.16.2.4      vchianti.vbrew.com      vchianti
```

Точно както при IP адреса на хоста, понякога се налага да използвате символно име за адрес на мрежа. Затова за файла *hosts* съществува съответен файл */etc/networks*, който съдържа връзките между имената на мрежите и номерата им и обратно. При Виртуалната пивоварна, можем да използваме файл *network*, изглеждащ по следния начин[5]:

```
# Файлът /etc/networks за Виртуалната пивоварна
brew-net        172.16.1.0
wine-net        172.16.2.0
```

Конфигуриране на интерфейс за IP

След като настроите вашия хардуер, както е описано в Глава 4, *Конфигуриране на серийния хардуер*, трябва да направите тези устройства известни на мрежовия софтуер в ядрото. За конфигурирането на мрежовите интерфейси и инициализирането на таблицата за маршрутизация се използват няколко команди. Тази операция обикновено се изпълнява от инициализиращия мрежата скрипт при всяко зареждане на системата. Основните инструменти за този процес се наричат *ifconfig* (if е съкращение от интерфейс) и *route*.

[5] Забележка: имената в *network* не трябва да са в противоречие с имената на хостовете от файла *hosts*, в противен случай някои програми могат да работят по странен начин.

ifconfig се използва за осигуряване на достъп до даден интерфейс за мрежовия слой в ядрото. Това включва задаването на IP адрес и други параметри, и активирането на интерфейса, известно още като "вдигане" на интерфейса. Да бъде активен тук означава, че ядрото ще изпраща и получава IP дейтаграми през интерфейса. Най-простият начин да използвате тази команда е чрез:

```
ifconfig интерфейс ip-адрес
```

Тази команда задава *ip-адреса* на *интерфейса* и го активира. Всички други параметри се настройват според подразбиращите им се стойности. Например, подразбиращата се мрежова маска извлича от мрежовия клас на IP адреса, например **255.255.0.0** за адрес от клас B. Инструментът *ifconfig* е описан по-подробно в раздела "*Всичко за ifconfig*".

Инструментът *route* ви позволява да добавяте или премахвате маршрути от таблицата за маршрутизация. Той може да бъде извикан по следния начин:

```
route [add|del] [-net|-host] цел [if]
```

Аргументите *add* и *del* определят дали добавяте или премахвате маршрут към *целта*. Аргументите *–net* и *–host* указват на командата `route` дали целта е мрежа или хост (ако не е посочено нищо се предполага, че е хост). Аргументът *if* отново не е задължителен и ви позволява да посочите към кой мрежов интерфейс трябва да се насочи маршрута –ядрото на Linux прави разумно предположение, в случай че не дадете такава информация. Тази тема ще бъде разгледана по-подробно в следващите раздели.

Интерфейсът Loopback

Първият интерфейс, който се активира, е интерфейсът loopback (примка):

```
# ifconfig lo 127.0.0.1
```

Понякога можете да видите фиктивното име **localhost** да се използва вместо IP адреса. *ifconfig* ще потърси името във файла *hosts*, където някой запис трябва да го декларира като име на хост за **127.0.0.1:**

```
# Примерен запис за localhost в /etc/hosts
localhost       127.0.0.1
```

За да видите конфигурацията на даден интерфейс, стартирайте *ifconfig* като зададете като аргумент само името на интерфейса:

```
$ ifconfig lo
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:3924 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          Collisions:0
```

Както виждате, на интерфейса loopback е зададена мрежова маска **255.0.0.0**, тъй като **127.0.0.1** е адрес от клас A.

Сега вече сте почти готови да започнете да експериментирате с вашата мини мрежа. Това, което все още липсва, е запис в таблицата за маршрутизация, който указва на IP, че може да използва този интерфейс като маршрут за направление **127.0.0.1.** Това се постига с командата:

```
# route add 127.0.0.1
```

Отново можете да използвате **localhost** вместо IP адреса, при условие че сте го въвели във вашия файл */etc/hosts*.

Сега трябва да проверите дали всичко работи добре, например като използвате *ping*. *ping* е мрежовия еквивалент на сонарно устройство[6]. Тази команда се използва за проверка дали даден адрес в действителност е достъпен и за измерване на закъснението при изпращане на дейтаграма до него и обратно. Необходимото време за този процес често се нарича и "време за пътуване в двете посоки":

```
# ping localhost
PING localhost (127.0.0.1) : 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=0.4 ms
64 bytes from 127.0.0.2: icmp_seq=0 ttl=255 time=0.4 ms
64 bytes from 127.0.0.3: icmp_seq=0 ttl=255 time=0.4 ms
^C
---- localhost ping statistics ----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.4 / 0.4 / 0.4 ms
#
```

Когато извикате *ping* както е показано тук, той ще продължи да генерира пакети, докато не се прекъсне от потребителя. С ^C се обозначава мястото, където сме натиснали Ctrl-C.

[6] Някой да си спомня "Echoes" на Pink Floyd?

Предишният пример показва, че пакетите за **127.0.0.1** се доставят правилно и отговор се връща до *ping* почти мигновено. Това показва, че успешно сте настроили вашия пръв мрежов интерфейс.

Ако резултатът, който получите от *ping*, не прилича на показания в горния пример, значи имате проблем. Проверете всички грешки, ако те показват, че някой файл не е инсталиран правилно. Проверете дали командите *ifconfig* и *route*, които използвате, са съвместими с версията на работещото ядро и най-вече, че ядрото е компилирано с разрешена поддръжка на мрежа (можете да установите това от наличието на директорията */proc/net*). Ако получите съобщение за грешка гласящо "Network unreachable" (мрежата е недостижима), най-вероятно сте задали погрешна командата *route*. Уверете се, че използвате същия адрес, който сте дали на *ifconfig*.

Описаните дотук стъпки са напълно достатъчни за използване на мрежови приложения на самостоятелен хост. След като добавите горните редове към вашия инициализиращ мрежата скрипт и се уверите, че той ще се изпълни по време на зареждане, можете да рестартирате машината си и да тествате различни приложения. Например, командата *telnet localhost* трябва да установи *telnet* връзка към вашия хост като изведе поканата за влизане `login:`.

Интерфейсът loopback е полезен не само като пример в книгите за мрежи или като средство за тестове по време на разработка, но се използва и при нормална работа от някои приложения[7]. Затова трябва винаги да го конфигурирате, независимо дали машината ви е свързана към мрежа или не е.

Ethernet интерфейси

Конфигурирането на един Ethernet интерфейс е почти същото като на интерфейса loopback; то изисква само няколко нови параметъра, когато използвате подмрежи.

Във Виртуалната пивоварна ние разделихме IP мрежата, която в началото беше от клас B, на подмрежи от клас C. За да конфигурираме интерфейса да разпознае това, параметрите на *ifconfig* трябва да бъдат следните:

[7] Например, всички приложения, базирани на RPC, използват интерфейса *loopback*, за да се регистрират в демона *portmapper* при стартирането си. Това включва NIS и NFS.

```
# ifconfig eth0 vstout netmask 255.255.255.0
```

Тази команда задава IP адреса на **vstout (172.16.1.2)** на интерфейса *eth0*. Ако бяхме пропуснали мрежовата маска, *ifconfig* щеше да я извлече от класа на IP мрежата, което щеше да доведе до неправилната мрежова маска **255.255.0.0.** Сега една бърза проверка показва:

```
# ifconfig eth0
eth0  Link encap 10Mps Ethernet HWaddr 00:00:C0:90:B3:42
      inet addr 172.16.1.2 Bcast 172.16.1.255 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
```

Можете да видите, че *ifconfig* автоматично конфигурира `broadcast` адреса (полето *Bcast*) с обичайната стойност, която е номера на мрежата на хоста с единици във всички битове в полето за адрес на хоста. Освен това, максималната единица за предаване (максималният размер на IP дейтаграми, които ядрото ще генерира са този интерфейс) е зададена като максималния размер на Ethernet пакетите: 1500 байта. Обикновено се използват стойностите по подразбиране, но ако се наложи, всички тези стойности могат да бъдат променени със специални опции, които ще бъдат описани в раздела "*Всичко за ifconfig*".

Също както за интерфейса loopback ще трябва да инсталирате един запис за маршрутизация, който информира ядрото за мрежата, която може да бъде достигната през *eth0*. За Виртуалната пивоварна можете да изпълните *route* със следните параметри:

```
# route add -net 172.16.1.0
```

На пръв поглед това изглежда малко като магия, защото не е съвсем ясно как *route* открива през кой интерфейс да маршрутизира. Обаче този трик е доста прост: ядрото проверява всички интерфейси, които са конфигурирани досега и сравнява адреса на целта (в този случай **172.16.1.0**) с мрежовата част от адреса на интерфейса (т.е. извършва побитов AND между адреса на интерфейса и мрежовата маска). Единственият интерфейс, който съответства, е *eth0*.

Добре, а за какво е опцията *–net*? Тя се използва, защото *route* може да обслужва едновременно маршрути към мрежи и към отделни хостове (както видяхме преди малко с **localhost**). Когато е зададен адрес в десетично-точков формат, *route* се опитва да разпознае дали това е име на мрежа или на хост като преглежда битовете от хост-частта. Ако хост-частта на адреса е нула, *route* предполага, че този адрес е на

мрежа; в противен случай *route* го счита за адрес на хост. Следователно, *route* ще сметне, че **172.16.1.0** е адрес на хост, а не на мрежа, защото не може да знае, че използваме подмрежи. Затова трябва изрично да уведомим *route*, че това е мрежа и затова му подаваме флага *–net*.

Разбира се, малко досадно е да се пише командата *route*, а и това е предпоставка за допускане на правописни грешки. По-удобен подход е да се използват имената на мрежите, които дефинирахме в */etc/networks*. Този подход прави командата по-лесна за четене; дори флагът *–net* може да бъде пропуснат, защото *route* знае, че **172.16.1.0** обозначава мрежа:

```
# route add brew-net
```

Сега, след като завършихте базовите стъпки на конфигурацията, трябва да се уверите, че вашият Ethernet интерфейс наистина работи добре. Изберете един хост от вашата мрежа, например **vlager** и въведете:

```
# ping vlager
PING vlager: 64 byte packets
64 bytes from 172.16.1.1: icmp_seq=0. time=11. ms
64 bytes from 172.16.1.1: icmp_seq=1. time=7. ms
64 bytes from 172.16.1.1: icmp_seq=2. time=12. ms
64 bytes from 172.16.1.1: icmp_seq=3. time=3. ms
^C
----vstout.vbrew.com PING Statistics----
4 packets transmitted, 4 packets received, 0
round-trip (ms)  min/avg/max = 3/8/12
```

Ако не видите подобен резултат, значи нещо не е наред. Ако срещнете необичайно голям брой загубени пакети, това вероятно е хардуерен проблем като повредени или липсващи терминатори. Ако въобще не получите никакъв отговор, трябва да проверите конфигурацията на интерфейса с инструмента *netstat*, който е описан по-долу в раздела "*Командата netstat*". Статистиката на пакетите, която се извежда от *ifconfig*, трябва да ви покаже дали въобще са били изпратени някакви пакети през интерфейса. Ако имате достъп до отдалечения хост, трябва да влезете в тази машина и да проверите статистиката на интерфейса. По този начин може да установите дали пакетите са били отхвърлени. Освен това, трябва да отпечатате информацията за маршрутизиране с *route*, за да проверите дали и двата хоста имат правилни записи за маршрутите. *route* отпечатва пълната таблица в ядрото с информацията за маршрутизиране, когато се извиква

без аргументи (*-n* просто указва да се извеждат адресите с десетично-точков формат, вместо да се използва името на хоста):

```
# 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
172.16.1.0   *        255.255.255.0   U     1      0       10 eth0
```

По-подробно описание на значението на тези полета е дадено в следващия раздел "*Командата netstat*". Колоната `Flags` съдържа списък с флаговете, зададени на всеки интерфейс. Флагът U винаги е вдигнат за активните интерфейси, а H указва, че целевият адрес е на хост. Ако флагът `H` е вдигнат за маршрут, който трябва да е маршрут към мрежа, ще трябва отново да стартирате командата *route* с опция *–net*. За да проверите дали въведено от вас маршрутизиране въобще се използва, проверете дали полето `Use` от втората до последната колона се увеличава между две извиквания на *ping*.

Маршрутизиране през шлюз

В предишния раздел описахме само конфигурирането на хост, имащ връзка само с една Ethernet мрежа. Много често, обаче, можете да срещнете мрежи, свързани една с друга посредством шлюзове (gateways). Тези шлюзове могат просто да свързват две или повече Ethernet мрежи, но могат и да осигуряват връзка с външния свят, например връзка с Интернет. За да използвате шлюз трябва да дадете допълнителна информация за маршрутизиране на мрежовия слой.

Ethernet мрежите на Виртуалната пивоварна и Виртуалната винарна са свързани чрез подобен шлюз, и по-точно хостът **vlager**. Ако предположим, че **vlager** вече е конфигуриран, трябва да добавим само един нов ред към таблицата с информацията за маршрутите на **vstout**, които да указва на ядрото, че то може да достигне всички хостове от мрежата на Винарната чрез **vlager**. Необходимото заклинание с *route* е показано по-долу; запазената дума `gw` означава, че следващия я аргумент обозначава шлюз:

```
# route add wine-net gw vlager
```

Разбира се, всеки хост от мрежата на Винарната, с който искате да кореспондирате, трябва да има съответния запис за маршрутизация към мрежата на Пивоварната. В противен случай, ще можете само да изпращате данни от мрежата на Пивоварната към тази на Винарната, но хостовете от последната няма да могат да отговарят.

Този пример описва само един шлюз, който прехвърля пакети между две изолирани Ethernet мрежи. Сега си представете, че **vlager** е свързан и към Интернет (например, чрез допълнителна SLIP връзка). В такъв случай бихме искали дейтаграмите към *която и да е* друга мрежа, освен тази на Пивоварната, да бъдат подавани на **vlager**. Това може да бъде постигнато като той се направи подразбиращ се шлюз за **vstout**:

```
# route add default gw vlager
```

Името на мрежа **default** е синоним на **0.0.0.0** и обозначава подразбиращия се маршрут. Подразбиращия се маршрут съответства на всяко направление и ще се използва, ако няма по-точен маршрут за това направление. Не е необходимо да добавяте това име в */etc/networks*, защото то е вградено в *route*.

Ако откриете, че имате големи загуби на пакети при ping към хост, намиращ се зад един или няколко шлюза, това може да означава, че мрежата е пренатоварена. Загубата на пакети е резултат не толкова на технически недостатъци, колкото на временно прекомерно натоварване на препредаващите хостове, което ги кара да се забавят или дори да игнорират входящите дейтаграми.

Конфигуриране на шлюз

Конфигурирането на машина, която комутира пакети между две Ethernet мрежи е доста просто. Да предположим, че отново се намираме при **vlager**, който има две Ethernet карти, всяка от които е свързана с някоя от двете мрежи. Всичко, което трябва да направите, е да конфигурирате поотделно двата интерфейса, като им дадете съответните IP адреси и маршрути.

Много полезно е да добавите информация за двата интерфейса към файла *hosts*, както е показано в следващия пример, така че да имаме удобни имена и за тях:

```
172.16.1.1      vlager.vbrew.com    vlager vlager-if1
172.16.2.1      vlager-if2
```

Тогава последователността от команди за конфигуриране на двата интерфейса е:

```
# ifconfig eth0 vlager-if1
# route add brew-net
# ifconfig eth1 vlager-if2
# route add wine-net
```

Ако тази последователност не работи, уверете се, че вашето ядро е компилирано с разрешена поддръжка на IP препращане (IP forwarding). Един добър начин да направите това е да проверите дали първото число на втория ред в `/proc/net/snmp` е `1`.

Интерфейсът PLIP

PLIP връзката, използвана за свързване на две машини, е малко по-различна от Ethernet. PLIP връзките са пример за така наречените връзки *точка-до-точка* (*point-to-point*), което всъщност означава, че има единствен хост на всеки от краищата на връзката. Мрежите като Ethernet се наричат *излъчващи* (*broadcast*) мрежи. Конфигурирането на връзките точка-до-точка е по-различно, защото те за разлика от излъчващите мрежи, не поддържат своя собствена мрежа.

PLIP предоставя много евтини и преносими връзки между компютрите. Като пример ще вземем преносимия компютър на работник във Виртуалната пивоварна, който се свързва с **vlager** чрез PLIP. Самият преносим компютър се казва **vlite** и има само един паралелен порт. По време на зареждане този порт ще бъде регистриран като *plip1*. За да активирате връзката, трябва да конфигурирате интерфейса *plip1*, използвайки следните команди[8]:

```
# ifconfig plip1 vlite pointopoint vlager
# route add default gw vlager
```

Първата команда конфигурира интерфейса, като указва на ядрото, че това е връзка от тип точка-до-точка, като отдалечената страна има адреса на **vlager**. Втората команда инсталира подразбиращ се маршрут, използвайки **vlager** като шлюз. На **vlager** е необходима подобна команда *ifconfig*, за да активира връзката (не е необходимо извикване на *route*):

```
# ifconfig plip1 vlager pointopoint vlite
```

Забележете, че интерфейсът *plip1* на **vlager** не се нуждае от отделен IP адрес, но също така може да се зададе и **172.16.1.1**. Мрежите точка до точка не поддържат директно мрежа, затова и интерфейсите не изискват адрес на поддържаната мрежа. Ядрото използва информа-

[8] Забележка: **pointopoint** не е правописна грешка. То наистина се записва така.

цията за интерфейса в таблицата с маршрутите, за да избегне всякакви възможни объркания[9].

По този начин конфигурирахме маршрутизирането от преносимия компютър към мрежата на Пивоварната; това, което все още липсва, е начин на маршрутизиране от хостовете на Пивоварната към **vlite**. Един определено трудоемък начин е да се добави специфичен маршрут към таблиците с маршрути на всеки хост, който да определя **vlager** като шлюз за **vlite**:

```
# route add vlite gw vlager
```

Динамичното маршрутизиране предлага много по добра възможност за временни маршрути. Може да използвате *gated* – демона за маршрутизиране, който трябва да инсталирате на всеки хост в мрежата за да разпространите динамично информацията за маршрутизиране. Най-лесната възможност, обаче, е да използвате *proxy ARP* (Address Resolution Protocol – протокол за разпознаване на адреси). С proxy ARP, **vlager** ще отговаря на всяка ARP заявка за **vlite** като изпраща своя собствен Ethernet адрес. Всички пакети за **vlite** ще пристигнат при **vlager**, който след това ще ги препрати към преносимия компютър. Ще се върнем отново на proxy ARP в раздела "Проверяване на ARP таблиците".

Настоящите версии на *net-tools* съдържат инструмент, наречен *plipconfig*, които ви позволява да задавате определени PLIP таймингови параметри. Номера на IRQ, което се използва от порта за принтер, може да се зададе с командата *ifconfig*.

Интерфейсите SLIP и PPP

Въпреки че SLIP и PPP връзките са само прости връзки от типа точка-до-точка като PLIP връзките, има доста повече какво да се каже за тях. Обикновено, установяването на SLIP връзка включва набиране на отдалечената страна през вашия модем и задаване на SLIP режим в серийната линия. PPP се използва по подобен начин. Ще разгледаме SLIP и PPP по-подробно в Глава 7 и Глава 8, *Протоколът PPP*.

[9] Като предохранителна мярка, трябва да конфигурирате вашите PLIP или SLIP връзки само, след като напълно сте конфигурирали записите за Ethernet в таблицата с маршрути. В противен случай, при някои по-стари ядра, мрежовия маршрут може да сочи към връзката точка-до-точка.

Фиктивният интерфейс

Фиктивният интерфейс е малко екзотичен, но въпреки това е много полезен. Основното му предимство е при самостоятелни хостове и машини, чиито единствени IP мрежови връзки са чрез набиране през телефонна линия. Всъщност, тези машини в повечето време са самостоятелни хостове.

Дилемата при самостоятелните хостове е, че имат само едно активно мрежово устройство – устройството loopback, на което обикновено се задава адрес **127.0.0.1**. В някои случаи, обаче, трябва да изпратите данни към "официалния" IP адрес на локалния хост. Като пример си представете преносимия компютър **vlite**, който не е свързан с мрежа в този пример. Някое приложение на **vlite** може да иска да изпрати данни към друго приложение на същия хост. Анализа *на /etc/hosts* на **vlite** дава IP адрес **172.16.1.65**, така че приложението се опитва да изпрати данни на този адрес. Тъй като в момента единствения активен интерфейсът на машината е само loopback, ядрото не знае, че **172.16.1.65** всъщност е същата машина! Като резултат, ядрото игнорира дейтаграмата и връща на приложението съобщение за грешка.

Тук е мястото, където се намесва фиктивното устройство. То решава дилемата като просто изпълнява противоположното на интерфейса loopback. В случая с **vlite**, просто му задавате адрес **172.16.1.65** и добавяте маршрут към хост, който да сочи към него. Тогава всяка дейтаграма за **172.16.1.65** ще се доставя локално. Правилното извикване е следното[10]:

```
# ifconfig dummy vlite
# route add vlite
```

IP псевдоними

Новите ядра имат възможност, която може напълно да замести фиктивния интерфейс и да изпълнява и други полезни функции. *IP псевдонимите* ви позволяват да конфигурирате няколко IP адреса на едно физическо устройство. В най-простия случай, можете да репликирате функцията на фиктивния интерфейс посредством конфигуриране на адреса на хоста като псевдоним на loopback интерфейса и напълно да

[10] Фиктивният интерфейс се нарича *dummy0*, ако сте го заредили като модул, вместо да сте го избрали като вградена опция на ядрото. Това е така, защото можете да зареждате няколко модула и да имате повече от едно фиктивно устройство.

избегнете използването на фиктивен интерфейс. В по-сложни ситуации, можете да конфигурирате вашия хост да изглежда като няколко различни хоста, всеки със свой собствен IP адрес. Тази конфигурация понякога се нарича "Виртуален хостинг", въпреки че технически този термин се използва и за множество други техники[11].

За да конфигурирате псевдоним на интерфейс, първо трябва да се уверите, че ядрото ви е компилирано с поддръжка на IP псевдоними (проверете дали имате файл */proc/net/ip_alias*; ако го нямате ще трябва да прекомпилирате ядрото си).

Конфигурирането на IP псевдоним теоретично е идентично на конфигурирането на реално мрежово устройство, но използвате специално име, за да покажете, че това е псевдоним. Например:

```
# ifconfig lo:0 172.16.1.1
```

Тази команда ще създаде псевдоним на loopback интерфейса с адрес `172.16.1.1`. IP псевдонимите се посочват като се добави `:n` към действителното мрежово устройство, където "n" е цяло число. В нашия пример, мрежовото устройство, на което правим псевдоним, е `lo` и за него правим псевдоним с номер 0. По този начин едно физическо устройство може да поддържа множество псевдоними.

Всеки псевдоним може да се третира като отделно устройство и от гледна точка на IP софтуера в ядрото, той ще бъде такова; този интерфейс обаче, ще споделя своя хардуер с друг интерфейс.

Всичко за ifconfig

Съществуват още много повече параметри на *ifconfig*, отколкото сме описали досега. Нормалното извикване на този инструмент е следното:

```
ifconfig интерфейс [адрес [параметри] ]
```

интерфейс е името на интерфейса, а *адрес* е IP адреса, който ще се зададе на интерфейса. Това може да бъде или IP адрес, записан в де-

[11] По-конкретно, използването на IP псевдонимите е известно като виртуален хостинг в мрежовия слой. В световете на WWW и SMTP е по-обичайно да се използва виртуален хостинг в приложния слой, в който един и същ IP адрес се използва за всеки виртуален хост, но при всяка заявка в приложния слой се подава различно име на хост. Услуги като FTP не могат да работят по този начин и изискват виртуален хостинг в мрежовия слой.

сетично-точков формат, или име, което *ifconfig* ще потърси в */etc/hosts*.

Ако *ifconfig* се извика само с името на интерфейса, ще бъде изведена конфигурация за този интерфейс. Когато се извиква без някакви параметри, се извеждат всички интерфейси, които сте конфигурирали досега; опцията *–a* показва неактивните от тях. Примерно извикване за Ethernet интерфейса *eth0* ще изглежда така:

```
# ifconfig eth0
eth0      Link encap 10Mbps Ethernet  HWaddr 00:00:C0:90:B3:42
          inet addr 172.16.1.2 Bcast 172.16.1.255 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
```

Полетата MTU и `Metric` показват текущите MTU и метрични стойности за този интерфейс. Метричната стойност традиционно се използва от някои операционни системи за изчисляване на стойността на маршрута. Linux все още не използва тази стойност, но я дефинира с цел съвместимост.

Редовете, започващи с `RX` и `TX` показват колко пакети са получени и изпратени без грешки, колко грешки са възникнали, колко пакета са игнорирани (най-вероятно поради недостиг на памет) и колко са изгубени поради препълване. Препълване при получаване обикновено възниква, когато пакетите пристигат по-бързо, отколкото ядрото може да ги обслужи. Флаговите стойности, изведени от *ifconfig*, съответстват приблизително на имената на опциите от командния ред; те са описани по-долу.

Следва списък от параметрите, разпознавани от *ifconfig*, със съответните имена на флаговете. Опции, които просто включват определена функция, позволяват тя отново да бъде спряна, като се постави тире (–) пред опцията.

`up` Тази опция прави даден интерфейс достъпен за IP слоя. Тя се подразбира, когато с командния ред е даден *адрес*. Може да се използва също и отново да разреши интерфейс, който временно е бил спрян с опцията `down`.

Тази опция съответства на флаговете *UP* и *RUNNING*.

`down`
Тази опция маркира интерфейс като недостъпен за IP слоя. Резултатът е, че какъвто и да е IP трафик през този интерфейс се

забранява. Забележка: тази опция автоматично ще изтрие всички маршрути, които използват този интерфейс.

`netmask` *маска*

Тази опция задава маска на подмрежата, която да се използва от интерфейса. Тя може да се даде като 32-битово шестнадесетично число, предхождано от 0x или в десетично-точков формат. Макар че вторият формат е по-често използван, с шестнадесетичното представяне често се работи по-лесно. Мрежовите маски всъщност са двоични числа и е по-лесно да се извършва двоично-шестнадесетично, отколкото двоично-десетично преобразуване.

`pointopoint` *адрес*

Тази опция се използва за IP връзки от тип точка-до-точка, която включва само два хоста. Тя е необходима например за конфигуриране на SLIP и PLIP интерфейси. Ако е бил зададен адрес точка-до-точка, *ifconfig* извежда флага *POINTOPOINT*.

`broadcast` *адрес*

Адресът за предаване до всички станции обикновено се образува от номера на мрежата, като във всички битове от хост-частта се записват единици. Някои реализации на IP (например системи, произлезли от BSD 4.2) използват различна схема, при която всички битове от хост-частта се изчистват. Опцията *broadcast* се използва за адаптиране към тези странни среди. Ако е зададен *broadcast* адрес, *ifconfig* извежда флага *BROADCAST*.

`irq`

Тази опция ви позволява да зададете IRQ линията, използвана от определени устройства. Това е особено полезно за PLIP, но може също да се използва и за някои Ethernet карти.

`metric` *число*

Тази опция може да се използва, за да се зададе метрична стойност на запис в таблицата с маршрути за интерфейса. Тази метрика се използва от протокола RIP (Routing Information Protocol)

за създаване на таблици за маршрутизиране за мрежата[12]. Подразбиращата се метрика, използвана от *ifconfig*, е нула. Ако на машината ви не работи RIP демон, тази опция не ви е необходима; а ако имате такъв, много рядко ще ви се налага да променяте метричната стойност.

`mtu` *байтове*

Задава максималната дължина на единица за предаване (Maximum Transmission Unit), която е максималния брой октети, които интерфейсът може да обработи за една транзакция. За Ethernet мрежите, MTU по подразбиране е 1500 (най-големият позволен размер за Ethernet пакет); за SLIP интерфейсите MTU е 296. (Няма ограничение за MTU при SLIP връзки; тази стойност просто е добър компромис.)

`arp`

Това е опция, специфична за broadcast мрежи като Ethernet или пакетно радио. Тя позволява използването на протокола ARP (Address Resolution Protocol – протокол за разпознаване на адреси) за откриване на физическите адреси на свързаните към мрежата хостове. За broadcast мрежите този протокол е активиран по подразбиране. Ако ARP е забранен, *ifconfig* извежда флага *NOARP*.

`-arp`

Тази опция забранява използването на ARP на този интерфейс.

`promisc`

Тази команда поставя интерфейса в т.нар promiscuous режим. При една broadcast мрежа това конфигурира интерфейса да приема всички пакети, независимо дали са предназначени за този хост или не. Това позволява анализиране на мрежовия трафик посредством пакетни филтри, наречено още *Ethernet snooping* (Ethernet следене). Обикновено това е добра техника за откриване на мрежови проблеми, които иначе са трудно откриваеми. Инструменти като *tcpdump* разчитат на това.

---

[12] RIP избира оптимален маршрут към даден хост, базирайки се на "дължината" на пътя. Тази дължина се изчислява като се съберат отделните метрични стойности на всяка връзка хост-до-хост. По подразбиране, едно препращане има дължина 1, но може да бъде всяко положително число, по-малко от 16. (Маршрут с дължина 16 е равен на безкрайност. Такива маршрути се считат за неизползваеми.) Параметърът *метрика* определя цената на това препращане, която след това се разпространява от маршрутизиращия демон.

От друга страна, тази опция позволява на недобронамерени хора да вършат опасни неща, например да следят трафика на вашата мрежа за пароли. Можете да се защитите от този тип атаки като забраните на който и да е да включва своя компютър към вашата мрежа. Може също така да използвате сигурни протоколи за удостоверяване на самоличността като Kerberos или комплекта за влизане secure shell[13]. Тази опция съответства на флага *PROMISC*.

`-promisc`
Тази опция изключва режима promiscuous.

`allmulti`
Multicast адресите са като broadcast Ethernet адресите, само че вместо автоматично да изпращат до всички станции, единствените, които получават пакети, изпратени на един multicast адрес, са тези, които са програмирани да следят пакетите за него. Това е полезно за приложения като Ethernet-базирани видео конференции или мрежово радио, които са само за тези, които искат да ги приемат. Адресирането до ограничен брой потребители се поддържа от повечето, но не от всички Ethernet драйвери. Когато тази опция е разрешена, интерфейсът получава и предава multicast пакети за обработка. Тя съответства на флага *ALLMULTI*.

`-allmulti`
Тази опция изключва multicast адресите.

Командата netstat

netstat е полезен инструмент за проверка на конфигурацията и работата на мрежата. Всъщност това са няколко инструмента, събрани заедно. В следващите раздели ще дискутираме всяка една от функциите им.

Отпечатване на таблицата с маршрути

Когато стартирате *netstat* с флага *–r*, ще бъде отпечатана таблицата с маршрути в ядрото по начина, по който правехме това с *route*. За **vstout** ще получим следния резултат:

[13] ssh може да се вземе от сървъра **ftp.cs.hut.fi**, директория */pub/ssh*.

```
# netstat -nr
Kernel IP routing table
Destination Gateway    Genmask         Flags  MSS Window  irtt Iface
127.0.0.1   *          255.255.255.255 UH       0 0          0 lo
172.16.1.0  *          255.255.255.0   U        0 0          0 eth0
172.16.2.0  172.16.1.1 255.255.255.0   UG       0 0          0 eth0
```

Опцията *–n* указва на *netstat* да отпечатва адресите посредством като IP номера в десетично-точков формат, вместо да използва символните имена на хостове и мрежи. Тази опция е особено полезна, когато искате да избегнете разпознаването на адресите в мрежата (например, чрез DNS или NIS сървър).

Втората колона от резултата от *netstat* показва шлюза, към който сочи маршрута. Ако не се използва шлюз се извежда звезда. Третата колона показва "общността" на маршрута, т.е. мрежовата маска за този маршрут. Когато е даден IP адрес, за който се търси подходящ маршрут, ядрото преминава през всеки от записите в таблицата с маршрути, като извършва побитово AND на адреса и маската, преди да го сравни с целта на маршрута.

Четвъртата колона отпечатва следните флагове, които описват маршрута:

- `G` Маршрута използва шлюз.
- `U` Интерфейсът, който ще се използва, е активен (up).
- `H` През този маршрут може да се достигне само един хост. Например, това е случая с loopback записа **127.0.0.1.**
- `D` Този маршрут е създаден динамично. Той се конфигурира, ако записът в таблицата е бил генериран от маршрутизиращ демон като *gated* или от ICMP съобщение за пренасочване (виж раздела "Протоколът ICMP" в Глава 2).
- `M` Този маршрут се задава ако записът в таблицата е модифициран от ICMP съобщение за пренасочване.
- `!` Маршрутът е отхвърлящ маршрут и дейтаграмите ще бъдат игнорирани.

Следващите три колони показват MSS, Window и `irtt`, които се прилагат към TCP връзките, установени през този маршрут. MSS е съкращение от Maximum Segment Size (максимален размер на сегмент) и представлява размера на най-голямата дейтаграма, която ядрото ще конструира за предаване през този маршрут. Window (прозорец) е максималното количество данни, които системата ще прие-

ме от един отдалечен хост за една поредица (burst). Съкращението `irtt` идва от "initial round trip time" (начално време за пътуване в двете посоки). Протоколът TCP гарантира надеждното доставяне на данни между хостовете посредством повторно изпращане на дейтаграми, в случай че те се изгубят. Този протокол пази текуща информация за това колко време е необходимо на една дейтаграма, за да бъде доставена до отдалечения хост и да се получи потвърждение за доставката. По този начин TCP знае колко трябва да изчака, преди да приеме, че дейтаграмата трябва да се изпрати отново; тази информация се нарича време за пътуване в двете посоки. Първоначалното време за пътуване е стойността, която TCP ще използва, когато една връзка се създава за пръв път. За повечето типове мрежи стойността по подразбиране е подходяща, но за някои по-бавни мрежи, например за определени любителски радио-мрежи, това време е твърде кратко и причинява излишно препредаване. Стойността на `irtt` може да се зададе чрез командата *route*. Стойност нула в това поле означава, че се използват подразбиращите се стойности.

И накрая, последното поле показва мрежовия интерфейс, който се използва от този маршрут.

Показване на статистика за интерфейса

Когато се извиква с флага *–i*, *netstat* показва статистика за конфигурираните в момента мрежови интерфейси. Ако е зададена и опцията *–a*, ще бъдат изведени *всички* интерфейси, които се намират в ядрото, а не само конфигурираните в момента. За **vstout** изходът от *netstat* ще бъде следния:

```
# netstat -i
Kernel Interface table
Iface MTU Met  RX-OK RX-ERR RX-DRP RX-OVR  TX-OK TX-ERR TX-DRP TX-OVR Flags
lo      0   0   3185      0      0      0   3185      0      0      0 BLRU
eth0 1500   0 972633     17     20    120 628711    217      0      0 BRU
```

Полетата `MTU` и `Met` показват текущите MTU и метрични стойности за този интерфейс. Колоните `RX` и `TX` показват колко пакети са изпратени или получени без грешки (`RX-OK/ TX-OK`) или повредени (`RX-ERR/ TX- ERR`); колко са били игнорирани (`RX-DRP/ TX-DRP`); и колко са изгубени поради препълване (`RX-OVR/ TX-OVR`).

Последната колона показва зададените за този интерфейс флагове. Тези символи са едносимволната версия на пълните имена на флаговете, които се извеждат, когато се проверява конфигурацията на интерфейса с *ifconfig*:

- B Зададене broadcast адрес.
- L Този интерфейс е loopback устройство.
- M Приемат се всички пакети (promiscuous режим).
- O ARP е забранен за този интерфейс.
- P Това е връзка от тип точка-до-точка.
- R Интерфейсът е работещ.
- U Интерфейсът е активен.

Извеждане на връзките

netstat поддържа комплект от опции за показване на активни и пасивни гнезда (sockets). Опциите *-t*, *-u*, *-w* и *-x* показват активни TCP, UDP, RAW или Unix socket-връзки. Ако в подадете и флага *–a*, ще бъдат показани също и гнездата, които очакват връзка (т.е. слушащите). Това ще ви даде списък на всички сървъри, които в момента работят на вашата система.

Стартирането на *netstat -ta* на **vlager** дава следния резултат:

```
$ 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
```

Този резултат показва, че повечето сървъри просто очакват входяща връзка. Все пак обаче, четвъртият ред показва входяща SMTP връзка от **vstout**, а шестият ред ни показва, че имаме изходяща *telnet* връзка към **vbardolino**.[14]

[14] Може да прецените дали една връзка е изходяща от номерата на портовете. Номерът на порта за *викащия* хост винаги е обикновено цяло число. На извикания хост за активните стандартни портове на услуги, *netstat* ще използва символни имена като `smtp`, които се намират в */etc/services*.

Използването на флага *–a* самостоятелно ще изведе всички гнезда от всички семейства.

Проверяване на ARP таблиците

В някои случаи е полезно да прегледате и промените съдържанието на ARP таблиците на ядрото, например ако подозирате, че дублиращ се Интернет адрес е причината за някои появяващ се от време на време мрежов проблем. Инструментът *arp* е създаден за такива ситуации. Опциите от командния му ред са:

```
arp [-v] [-t hwtype] -a [hostname]
arp [-v] [-t hwtype] -s hostname hwaddr
arp [-v] -d hostname [hostname...]
```

Всички аргументи `hostname` (име на хост) могат да бъдат или символни имена, или IP адреси в десетично-точков формат.

Първото извикване показва ARP записа за определения IP адрес или хост, или всички известни хостове, ако не е зададено *име на хост*. Например, стартирането на *arp* на **vlager** може да даде:

```
# arp -a
IP address       HW type                 HW address
172.16.1.3       10Mbps Ethernet         00:00:C0:5A:42:C1
172.16.1.2       10Mbps Ethernet         00:00:C0:90:B3:42
172.16.2.4       10Mbps Ethernet         00:00:C0:04:69:AA
```

което показва Ethernet адресите на **vlager**, **vstout** и **vale**.

Може да ограничите извеждането само за определен хардуерен тип чрез опцията *–t*. Можете да използвате аргументи `ether`, `ax25` или `pronet`, означаващи съответно 10 Mbps Ethernet, AMPR AX.25, и IEEE 802.5 оборудване.

Опцията *-s* се използва за постоянно добавяне към ARP таблиците на Ethernet адреса на хоста, зададен с `hostname`. Аргументът `hwaddr` задава хардуерния адрес, които по подразбиране се очаква да е Ethernet адрес, зададен като шест шестнадесетични числа (байтове), разделени с двоеточия. Освен това, можете да зададете хардуерния адрес за други типове хардуер, използвайки опцията *–t*.

Понякога ARP запитванията за отдалечен хост се провалят, например когато отдалечения ARP драйвер съдържа програмни грешки или съществува друг хост в мрежата, който по погрешка се идентифицира с IP адреса на този хост; този проблем изисква вие ръчно да доба-

вите IP адрес към ARP таблицата. Писането на IP адреси в ARP таблицата освен това е (много драстична) мярка, с която се защитавате от хостове във вашата мрежа, които се представят като някой друг.

Стартирането на *arp* с ключа *-d* изгрива всички ARP записи, свързани с дадения хост. Този ключ може да бъде използван, за да укаже на интерфейса отново да опита да получи Ethernet адреса за посочен IP адрес. Това е полезно, когато една неправилно конфигурирана система е разпространила погрешна ARP информация (разбира се, първо ще трябва да преконфигурирате неизправния хост).

Опцията *-s* може също да се използва, за да реализирате *proxy* ARP. Това е специална техника, чрез която един хост, например **gate**, работи като шлюз към друг хост, да кажем **fnord**, като симулира, че и двата адреса се отнасят за един и същи хост, а именно **gate**. Това се постига, като се направи ARP запис за **fnord**, който да сочи към собствения Ethernet интерфейс на **gate**. Сега, когато един хост изпрати ARP запитване за **fnord**, **gate** ще върне отговор, съдържащ неговия собствен Ethernet адрес. След това запитващия хост ще изпраща съответните дейтаграми към **gate**, който ще ги препраща към **fnord**.

Тези усложнения могат да ви бъдат необходими, когато желаете да получите достъп до **fnord** от DOS машина с лоша реализация на TCP, която не разбира добре маршрутизирането. Когато използвате proxy ARP, за DOS машината ще изглежда сякаш **fnord** е в локалната подмрежа, затова не ѝ се налага да знае нещо за това как се маршрутизира през шлюз.

Друго полезно приложение на proxy ARP е когато един от вашите хостове работи само временно като шлюз към някой хост, например, през телефонна връзка. В един от предишните примери разгледахме преносимия компютър **vlite**, който от време на време беше свързан към **vlager** чрез PLIP връзка. Разбира се, това приложение ще работи само ако адресът на хоста, за който искате да осигурите proxy ARP, е в същата IP подмрежа като вашия шлюз. **vstout** може да работи като proxy ARP за всеки хост от подмрежата на Пивоварната (**172.16.1.0**), но никога за хост от подмрежата на Винарната (**172.16.2.0**).

Правилното извикване, за да предоставите proxy ARP за **fnord** е дадено на следващите редове; разбира се, даденият Ethernet адрес трябва да е този на **gate**:

```
# arp -s fnord 00:00:c0:a1:42:e0 pub
```

Записът за proxy ARP може да бъде премахнат по следния начин:

```
# arp -d fnord
```

ГЛАВА 6

КОНФИГУРИРАНЕ НА УСЛУГАТА ЗА ИМЕНА И РЕЗОЛВЕРА

Както казахме в Глава 2, *Въпроси на работата в TCP/IP мрежа*, TCP/IP мрежите могат да използват различни схеми за преобразуване на имената в адреси. Най-простият начин е да се използва таблица на хостовете, съхранявана във файла */etc/hosts*. Това обаче е приложимо само за малки мрежи, които се поддържат от един-единствен администратор и нямат IP трафик с външния свят. Форматът на файла *hosts* беше вече описан в Глава 5, *Конфигуриране на TCP/IP мрежа*.

Като алтернатива, за преобразуване на името на хоста в IP адрес можете да използвате услугата *BIND* (*Berkeley Internet Name Domain* – област от имена в Интернет; софтуерът е създаден в Бъркли). Конфигурирането на BIND може да бъде много отегчителна задача, но след като веднъж го извършите, можете лесно да правите промени в топологията на мрежата. Под Linux, както под много други Unix системи, услугата за имена се предоставя от програма, наречена *named*. При стартирането си тя зарежда набор от управляващи файлове в своя вътрешен кеш и чака запитвания от отдалечени или локални потребителски процеси. Съществуват различни начини за конфигуриране на BIND, като не всеки от тях изисква на всеки хост да работи сървър за имена.

Тази глава ще ви предостави малко повече от груба скица на начина, по който работи DNS и как да използвате сървър за имена. Това ще ви бъде достатъчно, ако имате малка мрежа и една връзка с Интернет. Можете да намерите най-нова информация в документацията, която се съдържа в пакета BIND, която съдържа справочни страници,

бележки по версията и ръководство на оператора на BIND (BIND *Operator's Guide*, или съкратено BOG). Не се плашете от това име; в действителност това е един доста полезен документ. Като по-пълно описание на DNS и свързаните с него проблеми ви препоръчваме *DNS and BIND* от Paul Albitz и Cricket Liu (O'Reilly). Вероятно ще намерите отговор на въпросите си относно DNS в групата по интереси в Usenet, наречена *comp. protocols.tcp-ip.domains*. Ако са ви необходими техническите детайли, системата DNS е описана в RFC документите с номера 1033, 1034 и 1035.

Библиотеката резолвер

Терминът *resolver*, се отнася не за специално приложение, а за библиотеката резолвер. Тя представлява колекция от функции, които могат да се намерят в една стандартна библиотека на C. Централно място от тях заемат *gethostbyname(2)* и *gethostbyaddr(2)*, които намират всички IP адреси, асоциирани с дадено име на хост и обратно. Те могат да бъдат конфигурирани просто да преглеждат информацията в *hosts*, да изпращат запитвания към няколко DNS сървъра или да използват базата данни *hosts* от Мрежовата информационна система (NIS).

Функциите на резолвера четат конфигурационните файлове, когато бъдат извикани. От тези конфигурационни файлове те определят към какви бази от данни да отправят запитвания, в какъв ред и други детайли, свързани с начина, по който сте конфигурирали средата си. Старата стандартна библиотека на Linux (libc) използваше */etc/host.conf* като свой главен конфигурационен файл, но версия 2 на стандартната библиотека на GNU (glibc) използва файла */etc/nsswitch.conf*. Ние ще опишем и двете библиотеки, защото и двете се използват масово.

Файлът host.conf

Файлът */etc/host.conf* указва на функциите за преобразуване (функциите на резолвера) от старите стандартни библиотеки на Linux кои услуги да използват и в какъв ред.

Опциите в *host.conf* трябва да се намират на различни редове. Полетата могат да бъдат разделени от празно пространство (интервали или табулации). Символът диез (#) обозначава начало на коментар, който продължава до края на текущия ред. Предоставят се следните опции:

`order`

Тази опция определя реда, по който трябва да се опитват услуги за преобразуване. Валидните стойности са *bind* за запитвания към сървъра за имена, *hosts* за търсене в */etc/hosts* и `nis` за търсене в NIS. Може да бъде зададена една или всички от тях. Редът, в който те се указани в тази опция, определя реда, в който се опитват съответните услуги.

`multi`

Като стойност на `multi` можете да зададете `on` или `off`. Това определя дали на един хост в */etc/hosts* може да има няколко IP адреса, което обикновено се нарича multi-homed (букв. с няколко дома). Подразбиращата се стойност е off. Този флаг няма влияние върху DNS или NIS запитванията.

`nospoof`

Както ще обясним в раздела "Обратно търсене", DNS ви позволява да откриете името на хоста, свързано с даден IP адрес, като използвате домейна **in-addr.arpa**. Опитите на сървъри за имена да предоставят фалшиво име на хост се наричат *spoofing* (мамене). Като предпазна мярка срещу това, резолвера може да бъде конфигуриран така, че да проверява дали оригиналният IP адрес в действителност е асоцииран с полученото име на хост. Ако не е, името се отхвърля и се връща съобщение за грешка. Това поведение се активира като се зададе опцията `nospoof`.

`alert`

Тази опция приема като аргумент on или `off`. Ако се зададе on, след всеки опит за мамене (*spoofing*), резолвера ще запише съобщение в дневника чрез инструмента *syslog*.

`trim`

Тази опция приема аргумент, задаващ име на домейн, което ще бъде премахнато от имената на хостовете преди търсене. Това е полезно при записите в *hosts*, за които може би искате да задавате само имената на хостовете без локален домейн. Ако зададете вашия локален домейн тук, той ще бъде премахнат при търсенето на хост, към чието име е добавено името на локалния домейн, като по този начин позволява на търсенето в */etc/hosts* да бъде успешно. Името на домейна, което задавате, трябва да завършва със символа точка (.) (например, `linux.org.au`.) ако искате `trim` да работи правилно.

Опциите `trim` се натрупват; можете да считате вашия хост като локален за няколко домейна.

Примерен файл за **vlager** е показан на Пример 6-1.

Пример 6-1: Примерен файл host.conf

```
# /etc/host.conf
# Имаме работещ named, но (все още) не и NIS
order    bind,hosts
# Позволяваме няколко адреса
multi    on
# Защита от опити за измамване
nospoof  on
# Изрежи локалния домейн (не действително необходимо)
trim     vbrew.com
```

Променливи от обкръжението за резолвера

Настройките от *host.conf* може да бъдат предефинирани посредством няколко променливи от обкръжението:

`RESOLV_HOST_CONF`
: Тази променлива задава файл, който да се чете вместо */etc/host.conf*.

`RESOLV_SERV_ORDER`
: Тази променлива замяня опцията `order`, зададена в *host.conf*. Услугите се задават с имената `hosts`, `bind` и `nis`, разделени от интервал, запетая, двоеточие или точка и запетая.

`RESOLV_SPOOF_CHECK`
: Тази променлива задава мерките, които се вземат срещу измамене. Те се забраняват напълно с `off`. Стойностите `warn` и `warn off` разрешават проверката за измамене като съответно включва и изключват отбелязването в дневника. Стойност * активира проверката за измамене, но оставя конфигурацията за дневника както е зададена в *host.conf*.

`RESOLV_MULTI`
: Тази променлива използва стойност `on` или `off` за предефиниране на опцията `multi` от *host.conf*.

`RESOLV_OVERRIDE_TRIM_DOMAINS`
: Тази променлива задава списък с имена на домейни за отрязване, които заместват зададените в *host.conf*. Имената за отрязване бяха описани по-горе при дискусията за ключовата дума `trim`.

`RESOLV_ADD_TRIM_DOMAINS`
: Тази променлива задава списък с имена на домейни за отрязване, които се добавят към зададените в *host.conf*.

Файлът nsswitch.conf

Версия 2 на стандартната библиотека на GNU включва много по-мощен и гъвкав заместител на по-стария механизъм *host.conf*. Концепцията на услугата за имена е разширена така, че да включва разнообразие от различни типове информация. Конфигурационните опции за всички различни функции, които извършват запитвания към тези бази от данни, са събрани отново в един-единствен конфигурационен файл; файлът *nsswitch.conf*.

Файлът *nsswitch.conf* позволява на системния администратор да конфигурира голямо разнообразие от различни бази данни. Ще ограничим нашата дискусия до опциите, свързани с разпознаването на IP адресите на хостове и мрежи. Можете лесно да намерите информация за другите възможности като прочетете документацията на стандартната библиотека на GNU.

Опциите в *nsswitch.conf* трябва да се задават на отделни редове. Полетата могат да бъдат разделени от празно пространство (интервали или табулации). Символът диез (#) обозначава начало на коментар, който продължава до края на текущия ред. Всеки ред описва конкретна услуга; разпознаването на имена на хостове е една от тях. Първото поле на всеки ред е името на базата данни, завършващо с двоеточие. Името на базата данни, асоциирана с разпознаването на адреса на хоста, е `hosts`. Подобна база данни е файлът `networks`, който се използва за преобразуване на имената на мрежите до адреси на мрежи. Останалата част от реда съдържа опции, които определят начина, по който се извършва търсенето в тази база данни.

Съществуват следните опции:

`dns`
: Използва услугата DNS за разпознаване на адреса. Приложима е само за преобразуването на адреси на хостове, а не на мрежи. Този механизъм използва файла */etc/resolv.conf*, който ще опишем по-долу в тази глава.

`files`
: Търси локален файл за името на хост или мрежа и съответстващия му адрес. Тази опция използва традиционните файлове */etc/hosts* и */etc/networks*.

`nis` *или* `nisplus`
: Използва Мрежовата информационна система (NIS) за разпознаване на адресите на хоста или мрежата. NIS и NIS+ се разглеждат подробно в Глава 13, *Мрежова информационна система*.

Редът, в който са изброени услугите, определя реда, по който ще се извършват запитвания към тях, когато се разпознава име. Списъкът с реда на отправяне на запитванията се намира в описанието на услугата във файла */etc/nsswitch.conf*. Запитванията към услугите по подразбиране се извършват отляво надясно и спират, когато разпознаването е успешно.

В Пример 6-2 е показан прост пример за задаване на бази данни за хост и мрежа, който е аналогичен на нашата конфигурация посредством старата стандартна библиотека libc.

Пример 6-2: Примерен файл nsswitch.conf

```
# /etc/nsswitch.conf
#
# Примерна конфигурация на функционалността на GNU за смяна на услугата
# за имена.
# Информация за този файл може да се намери в пакета 'libc6-doc'.

hosts:      dns files
networks:   files
```

Този пример указва на системата да търси хостове първо в DNS и ако не ги намери там, във файла */etc/hosts*. Търсенето на имена на мрежи ще се извършва само във файла */etc/networks*.

Имате възможност да контролирате по-прецизно поведението при търсене посредством "елементите за действие", които описват какво действие трябва да се извърши, като се има предвид резултата от предишното търсене. Елементите за действие се задават между задаването на услугите и са оградени в квадратни скоби *[ ]*. Обичайният синтаксис на декларирането на действието е:

```
[ [ ! ] състояние = действие ... ]
```

Има две възможни действия:

`return`
: Управлява връщането към програмата, която е поискала разпознаване на името. Ако търсенето е било успешно, резолвера ще върне детайлите, в противен случай ще върне нулев резултат.

`continue`
: Резолвера ще продължи със следващата услуга в списъка и ще се опита да разпознае името, използвайки нея.

Незадължителния символ (!) указва, че стойността на състоянието трябва да бъде инвертирана преди тестването, т.е. означава представка "не".

Възможните стойности за състояние, с които можем да работим, са следните:

`success`
: Заявеният елемент е намерен без грешка. Действието по подразбиране за това състояние е `return`.

`notfound`
: При търсенето не е възникнала грешка, но търсения хост или мрежа не могат да бъдат намерени. По подразбиране, действието за това състояние е `continue`.

`unavail`
: Търсената услуга не е достъпна. Това може да означава, че файлът *hosts* или *networks* не може да бъде прочетен от услугата `files` или че DNS или NIS сървър не отговарят на dns или nis запитвания. По подразбиране, действието за това състояние е `continue`.

`tryagain`
: Това състояние означава, че услугата временно не е достъпна. За файловата услуга `files` това обикновено означава, че съответния файл е заключен от някой процес. За други услуги може да означава, че сървърът временно не може да приеме връзки. Подразбиращото се действие за това състояние е `continue`.

В пример 6-3 е показан прост пример за това, как можете да използвате този механизъм.

Пример 6-3: Примерен файл nsswitch.conf, използвайки действия

```
# /etc/nsswitch.conf
#
# Примерна конфигурация на функционалността на GNU за смяна на услугата
# за имена.

# Информация за този файл може да се намери в пакета 'libc6-doc'.
Hosts:      dns [!UNAVAIL=return] files
Networks:   files
```

Този пример се опитва да извърши разпознаване на хоста, използвайки системата DNS. Ако върнатото състояние е различно от "unavail" (не е достъпен), резолвера връща каквото е намерил. Ако, и само ако опита за DNS търсене върне състояние "unavail", резолвера

се опитва да използва локалния */etc/hosts*. Това означава, че ще използваме файла *hosts* само, ако нашият сървър за имена по някаква причина не е достъпен.

Конфигуриране с resolv.conf на търсенето чрез сървър за имена

Когато се конфигурира библиотеката резолвер да използва BIND услуга за имена, трябва да укажете и кои сървъри за имена да се използват. За целта се използва отделен файл, който се нарича *resolv.conf*. Ако този файл не съществува или е празен, резолвера приема, че сървъра за имена се намира на вашия локален хост.

Ако искате на вашия локален хост да работи сървър за имена, трябва да го конфигурирате отделно, както ще опишем в следващата част. Ако се намирате на локална мрежа с работещ и достъпен сървър за имена, винаги е за предпочитане да използвате тази възможност. Ако използвате IP връзка към Интернет през телефонна линия, обикновено във файла *resolv.conf* трябва да зададете сървъра за имена на вашия доставчик.

Най-важната опция в *resolv.conf*е *name* `server`, с която се задава IP адреса на сървъра за имена, който ще използвате. Ако зададете няколко такива сървъра като използвате няколко пъти опцията `name server`, те се опитват в посочения ред. Затова трябва да поставите най-надеждния сървър на първо място. Текущата реализация ви дават възможност да имате до три конструкции `name server` в *resolv.conf*. Ако не е зададена опцията `name server`, резолвера се опитва да се свърже към сървъра за имена, работещ на локалния хост.

Две други опции – `domain` и `search`, ви позволяват да използвате съкратени имена за хостовете от локалния домейн. Обикновено, когато просто се свързвате чрез telnet с друг хост от вашия локален домейн, не искате да пишете пълното име на хоста, а в командния ред използвате име като **gauss**, като очаквате резолвера да добави липсващата част **mathematics.groucho.edu.**

Точно това е предназначението на конструкцията `domain`. Тя ви позволява да зададете подразбиращо се име на домейн, което да се добави, когато DNS не успее да открие името на хоста. Например, когато се подаде името **gauss**, резолвера не успява да намери **gauss**. в DNS, защото няма такъв домейн от най-високо ниво. Когато като домейн

по подразбиране е зададен **mathematics.grouchо.edu**, резолвера повтаря запитването за **gauss** с добавен домейн по подразбиране, което този път ще завърши с успех.

Може би мислите, че това е прекрасно, но веднага щом излезете от домейна на Математическия факултет, отново се сблъсквате с пълните имена на домейните. Разбира се, бихте искали и да използвате съкращения като **quarkphysics** за хостове в домейна на Физическия факултет.

Точно затова се използва опцията *search list* (списък за търсене). Един списък за търсене може да се зададе с опцията `search`, която е обобщение на конструкцията *domain*. Докато с `domain` се задава един-единствен подразбиращ се домейн, със *search list* се задава цял списък от домейни, като последователно се опитва всеки от тях, докато търсенето приключи успешно. Елементите в този списък трябва да бъдат разделени от интервали или табулации.

Конструкциите `search` и `domain` са взаимно изключващи се и не могат да се задават повече от веднъж. Ако не е зададена нито една от тези опции, резолвера ще се опита да отгатне подразбиращия се домейн от името на локалния хост, като използва системната функция *getdomainname(2)*. Ако името на локалния хост няма домейн-частта, за домейн по подразбиране ще се счита основния домейн.

Ако решите да поставите конструкция `search` в *resolv.conf*, трябва да внимавате какви домейни добавяте към този списък. Библиотеките резолвер преди BIND 4.9 създаваха подразбиращ се списък за търсене, когато не е зададен такъв. Този списък по подразбиране се състоеше от самия подразбиращ се домейн плюс всички негови родителски домейни до корена. Това причиняваше някои проблеми, защото DNS запитванията достигаха до сървъри за имена, които никога не би трябвало да ги виждат.

Да предположим, че сте във Виртуалната пивоварна и искате да влезете във **foot.grouchо.edu**. По погрешка обаче вместо **foot**, вие въвеждате foo, което не съществува. В такъв случай сървърът за имена на GMU ще ви каже, че не знае за такъв хост. Със стария списък за търсене, резолвера би продължил като опитва това име с добавени **vbrew.com** и **com**. Последното е проблем, защото **grouchо.edu**.com може да е валидно име на домейн. Техният сървър за имена, може дори да намери foo в техния домейн и да ви посочи някой от техните хостове – което очевидно не е това, което искахме да постигнем.

За някои приложения тези фалшиви търсения на хостове могат да се окажат дори проблем от гледна точка на сигурността. Следователно, обикновено трябва да ограничите домейните във вашия списък за търсене до рамките на вашата организация или нещо съизмеримо. В Математическия факултет на Университета Groucho Marx, списъка за търсене обикновено трябва да се състои от **maths.groucho.edu** и **groucho.edu**.

Ако домейните по подразбиране ви изглеждат объркващо, разгледайте следния примерен файл *resolv.conf* за Виртуалната пивоварна:

```
# /etc/resolv.conf
# Нашият домейн
domain      vbrew.com
#
# Използваме vlager като централен сървър за имена:
name server 172.16.1.1
```

Когато се разпознава името **vale**, резолвера търси **vale** и, ако не може да го намери, **vale.vbrew.com**.

Устойчивост на резолвера

Когато работите в локална мрежа в рамките на по-голяма мрежа, определено трябва да използвате централни сървъри за имена, ако има такива. Тези сървъри създават богат кеш, който ускорява отговорите на повтарящите се запитвания, защото всички запитвания се изпращат на тях. Все пак, тази схема има и недостатък: когато пожар унищожи главния кабел в Университета на Олаф, работата в локалната мрежа на неговия отдел беше невъзможна, защото резолвера вече не беше в състояние да достигне до който и да е от сървърите за имена. Тази ситуация причини трудности с повечето мрежови услуги, например с влизането от X терминали и печата.

Въпреки че не се случва често главният кабел на университета да е в пламъци, могат да се вземат мерки за защита от такива ситуации.

Една от възможностите е да се използва локален сървър за имена, които да разпознава имената във вашия локален домейн и да препраща всички запитвания за други имена на хостове към главните сървъри. Разбира се, това е приложимо само ако имате свой собствен домейн.

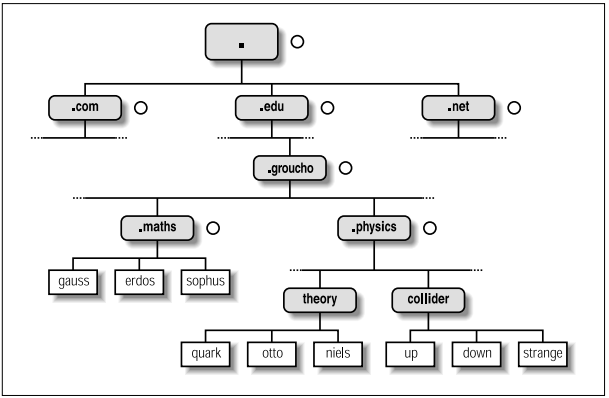
Алтернатива на тази възможност е да се поддържа резервна таблица с хостовете във вашия домейн или локална мрежа в */etc/hosts*. Това се прави много просто. Трябва просто да се уверите, че библиотеката

резолвер запитва първо DNS, а след това файла *hosts*. Във файла */etc/host.conf* трябва да използвате "`order bind, hosts`", а във файла */etc/nsswitch.conf* – "`hosts: dns files`", за да укажете на резолвера да използва файла `hosts`, ако централния сървър за имена не е достъпен.

Как работи DNS

DNS организира имената на хостовете в йерархия от домейни. *Домейнът* (областта – б.р.) е множество от сайтове, които в някакъв смисъл са свързани един с друг – защото формират истинска мрежа (например, всички машини в един университет или всички хостове на BITNET), защото всички принадлежат на определена организация (например, Правителството на САЩ) или защото те просто са географски близо. Например, университетите обикновено се обединяват в домейна **edu**, като всеки университет или колеж използва отделен поддомейн, в който се разполагат неговите хостове. Домейнът на Университета Grouch Marx е **grouchо.edu**, а на локалната мрежа на Факултета по математика е зададен домейнът **maths.grouchо.edu**. Към имената на хостовете от този факултет ще бъде добавено името на техния домейн, т.е. **erdos** ще бъде известен като **erdos. maths. grouchо.edu**. Това се нарича *пълно домейн име* (FQDN – *fully qualified domain name*), което идентифицира уникално този хост в целия свят.

На Фигура 6-1 е показана част от пространството на имената. Елементът в корена на дървото, който е отбелязан с една точка, доста подходящо се нарича основен домейн и съдържа всички други домейни. За да се покаже, че едно име на хост е пълно домейн име, а не име, което е относително спрямо някой (подразбиращ се) локален домейн, то понякога се пише завършващо с точка. Тя указва, че последният компонент от името е основния домейн.



Фигура 6-1. Част от домейн пространството на имената

В зависимост от своето разположение в йерархията на имената, един домейн може да бъде от горно (първо) ниво, от второ ниво или от трето ниво. Съществуват и повече нива на подразделяне, но те се срещат по-рядко. Следващият списък съдържа някои често срещани домейни от горно ниво:

Домейн	Описание
edu	(Предимно в САЩ) образователни институции като университети.
com	Комерсиални организации и компании.
org	Некомерсиални организации. Частните UUCP мрежи често са този домейн.
net	Шлюзове и други административни хостове в една мрежа.
mil	Военни институции на САЩ.
gov	Правителствени институции на САЩ.
uucp	Официално, всички сайтове, които преди са били използвани като UUCP имена без домейни, са преместени на този домейн.

Исторически, първите четири от тези домейни са били дадени на САЩ, но неотдавнашни промени в политиката продиктуваха днес тези домейни, наречени глобални домейни от горно ниво (gTLD), да се считат за глобални по природа. В момента се водят преговори за разширяването на кръга на gTLD, което може да доведе като резултат до по-голям избор в бъдеще.

Извън САЩ, всяка страна обикновено използва свой домейн от горно ниво, именован по двубуквения код на страната, определен от стандарта ISO-3166. Финландия, например, използва домейн **fi**; **fr** се използва от Франция, **de** от Германия, а **au** от Австралия (домейнът за България е **bg – б.р.**). Под този домейн от горно ниво, NIC на всяка страна (организацията, грижеща се за регистриране на домейни от съответното ниво – б.р.) може свободно да организира имената на хостовете по избрания от нея начин. Австралия има домейни от второ ниво подобни на международните домейни от горно ниво, наречени **com.au** и **edu.au**. Други държави, например Германия, не използват това допълнително ниво, но имат малко по-дълги имена, които сочат директно организациите, поддържащи определен домейн. Не е обичайно да срещате имена на хостове като **ftp.informatik.uni-erlangen.de.** Това несъмнено се дължи на немската ефикасност.

Разбира се, тези национални домейни не означават, че един хост, намиращ се в такъв домейн, в действителност е разположен в тази страна; това просто означава, че този хост е регистриран при NIC на съответната страна. Един шведски производител може да има клон в Австралия и въпреки това, всичките му хостове да са регистрирани в домейна se от горно ниво.

Организирането на пространството на имената в йерархия от имена на домейни напълно решава проблема с уникалността на имената; с DNS, едно име на хост трябва да бъде уникално само в рамките на своя домейн, за да бъде различно от всички останали в световен мащаб. Освен това, пълните имена са лесни за запомняне. Дори сами по себе си, те са достатъчно добра причина да разделиш един голям домейн на няколко поддомейна.

DNS ви дава дори нещо повече от това. Тя още ви позволява да делегирате права за даден поддомейн на неговите администратори. Например, хората от поддръжката в компютърния център на Университета Groucho Marx могат да създадат по един поддомейн за всеки факултет; вече се срещнахме с поддомейните **math** и **physics** по-горе. Когато решат, че мрежата на Физическия факултет е твърде голяма и хаотична за управление отвън (все пак, физиците са известни като

доста безпорядъчни хора), те могат просто да дадат контрола върху домейна **physics.groucho.edu** на администраторите на тази мрежа. Тези администратори са свободни да използват каквито искат имена и да им дават произволни IP адреси от своята мрежа без външна намеса.

С тази цел, пространството на имена е разделено на *зони*, всяка от които има като корен един домейн. Забележете тънката разлика между *зона* и *домейн*: домейнът **groucho.edu** съдържа всички хостове на Университета Groucho Marx, докато зоната **groucho.edu** включва само хостовете, които се управляват директно от компютърния център: например, тези от Математическия факултет. Хостовете от Физическия факултет принадлежат на различна зона, наречена **physics.groucho.edu**. На Фигура 6-1 началото на зона е отбелязано с малко кръгче в дясно от името на домейна.

Търсене на имена с DNS

На пръв поглед, цялата тази суета около домейните и зоните изглежда прави разпознаването на имената ужасно сложен процес. В крайна сметка, ако никаква централна власт не контролира какви имена се задават на различните хостове, как може да се очаква едно просто приложение да знае?

Точно тук се проявява гениалното в DNS. Ако искате да намерите IP адреса на **erdos**, DNS ви казва, "Попитайте хората, които гоу управляват и те ще ви кажат".

Всъщност, DNS е една гигантска разпределена база данни. Тя се реализирана от така наречените сървъри за имена, които предоставят информация за даден домейн или набор от домейни. За всяка зона има поне два, или най-много няколко такива сървъра, които съхраняват цялата достоверна информация за хостовете в тази зона. За да получите IP адреса на **erdos**, всичко което трябва да направите е да се свържете със сървъра за имена на зоната **groucho.edu**, който ще ви даде необходимите данни.

По-лесно е да се каже, отколкото да се направи, ще си помислите. А как да достигна до сървъра за имена на Университета Groucho Marx? Ако в компютъра ви няма оракъл за разпознаване на адреси, DNS ще се погрижи и за това. Когато вашето приложение иска да получи информация за **erdos**, то се свързва с локален сървър за имена, който извършва така наречената итеративно запитване за нея. То започва с изпращането на запитване към сървър за имена от основния домейн,

в което се пита за адреса на **erdos.maths.groucho.edu**. Основният сървър за имена разпознава, че това име е не в неговата юрисдикция, а в тази на домейна **edu**. Затова той ви казва, че за повече информация трябва да се свържете със сървър за имена от зоната **edu** и прилага списък с всички сървъри за имена от **edu** заедно с техните адреси. Вашия локален сървър продължава и отправя запитване към някой от тях, например към **a.isi.edu.** По подобен начин на основния сървър за имена, **a.isi.edu** знае, че хората от **groucho.edu** използват собствена зона и ви насочва към техните сървъри. Локалният сървър изпраща своето запитване за **erdos** към един от тях, който в крайна сметка ще разпознае името като принадлежащо на неговата зона и ще върне съответстващия IP адрес.

На пръв поглед изглежда, че за намирането на един IP адрес се генерира много трафик, но той всъщност е само малка частица в сравнение с количеството данни, които би трябвало да се прехвърлят, ако все още използвахме *HOSTS.TXT*. Все пак, в тази схема има място за оптимизация.

За намаляване на времето за отговор на бъдещи запитвания, сървърът за имена съхранява получените данни в своя вътрешен *кеш*. Така че следващия път, когато някой от вашата локална мрежа иска да научи адреса на хост от домейна **groucho.edu**, вашият сървър за имена ще отиде директно на сървъра за имена на **groucho.edu**.[15]

Разбира се, сървърът за имена няма да пази тази информацията вечно; той ще я игнорира след известно време. Интервала, за който се пази информацията, се нарича *време на живот* или TTL (от *time to live*). На всички данни в DNS базата данни се задава TTL от администраторите на съответната зона.

Типове сървъри за имена

Сървърите за имена, които пазят цялата информация за хостовете в рамките на една зона се наричат *упълномощени* (*authoritative*) за тази зона, а понякога и главни (*master*) сървъри за имена. Всяко запитване за хост в тази зона ще достигне до един от тези главни сървъри за имена.

[15] Ако информацията не беше кеширана, тогава DNS щеше да бъде неефикасен като всеки друг метод, защото за всяко запитване ще се използват основните сървъри за имена

Главните сървъри трябва да са доста добре синхронизирани. Затова, мрежовият администратор на тази зона трябва да направи един първичен (*primary*) сървър, който зарежда информация за своята зона от файлове с данни и да направи останалите сървъри вторични (*secondary*), които прехвърлят данните за зоната от първичния сървър на равни интервали.

Наличието на няколко сървъра за имена разпределя натоварването, а освен това осигурява резерв. Ако един от сървърите за имена стане недостъпен, например след срив или загуба на връзката си с мрежата, всички запитвания ще се поемат от останалите сървъри. Разбира се, тази схема не ви предпазва от повреда в сървъра, в следствие на която се генерират грешни отговори на всички DNS запитвания, като например софтуерни грешки в самата програма на сървъра.

Можете освен това да използвате сървър за имена, който да не е упълномощен за никой домейн[16]. Това е полезно, защото този сървър също е способен да приема DNS запитвания от приложения, работещи в локалната мрежа и да кешира информацията. Оттук идва и името му – *сървър само за кеширане* (*caching-only*).

Базата данни на DNS

Вече видяхме, че DNS се грижи не само за IP адресите на хостовете, но и обменя информация между сървърите за имена. Всъщност, в базите данни на DNS може да има много различни типове записи.

Една частица информация от базата данни на DNS се нарича *запис за ресурс* (*resource record* или RR). Всеки запис има асоцииран с него тип, описващ типа данни, който представлява, и клас, определящ типа мрежа, за който се отнася. Типът мрежа се използва за нуждите на различни адресни схеми като IP адреси (класът IN), Hesiod адреси (използват се от системата Kerberos на MIT) и още няколко. Прототипният запис за ресурс е записът A, който асоциира пълното домейн име с IP адрес.

Един хост може да бъде известен с повече от едно име. Например, възможно е да имате машина, на която работят два типа сървъри – FTP и World Wide Web, на която сте задали две имена: **ftp.machine.org** и **www.machine.org.** Все пак, едно от тези имена трябва да се оп-

[16] Е, почти. Един сървър за имена трябва да осигурява поне услуга за името **localhost** и обратното търсене за адрес **172.0.0.1.**

редели като официално или *канонично* име на хоста, докато другите ще са просто псевдоними, сочещи към официалното име на хоста. Разликата е в това, че каноничното име на хоста е това, което има асоциран A запис, докато другите имат само запис от тип CNAME, който сочи към каноничното име.

Тук няма да разглеждаме всички типове записи, но ще ви дадем кратък пример. В Пример 6-4 е показана част от базата данни на домейн, която се зарежда от сървърите за имена за зоната **physics.groucho.edu**.

Пример 6-4: Извадка от файла named.hosts за Физическия факултет

```
; Достоверна информация за physics.groucho.edu.
@  IN SOA niels.physics.groucho.edu. janet.niels.physics.groucho.edu. (
                  1999090200      ; сериен номер
                  36000           ; опресняване
                  3600            ; повторен опит
                  3600000         ; изтичане
                  3600            ; подразбиращо се ttl
                )
;
; Сървъри за имена
              IN    NS       niels
              IN    NS       gauss.maths.groucho.edu.
gauss.maths.groucho.edu. IN A 149.76.4.23
;
; Теоретична физика (подмрежа 12)
niels         IN    A        149.76.12.1
              IN    A        149.76.1.12
name server   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
...
; Изследователска лаборатория (подмрежа 14)
boson         IN    A        149.76.14.1
muon          IN    A        149.76.14.7
bogon         IN    A        149.76.14.12
...
```

Освен записите A и CNAME, можете да забележите специален запис в началото на файла, състоящ се от няколко реда. Това е записа за ресурс SOA (*Start of Authority* – начало на пълномощия), в който се съхранява обща информация за зоната, за която сървъра има пълномощия. Записът SOA включва, например, подразбиращото се време за живот на всички записи.

Обърнете внимание, че всички имена, които не завършват с точка, трябва да се интерпретират като относителни спрямо домейна **physics.groucho.edu.** Специалното име (@), използвано в записа SOA, се отнася до самото име на домейна.

По-горе видяхме, че сървърите за имена в домейна **groucho.edu** трябва по някакъв начин да знаят за зоната **physics**, така че да могат да насочват запитванията към нейните сървъри за имена. Обикновено това се постига чрез двойката записи: записът NS, който дава пълното домейн име на сървъра и записът A, който асоциира адрес към това име. Тъй като тези записи са това, което съединява пространството на имена като едно цяло, те често се наричат *свързващи записи*. Те са единствените примери на записи, в които родителските зони в действителност държат информация за хостове в подзоната. Свързващите записи, сочещи към сървърите за имена във **physics.groucho.edu**, са показани в Пример 6-5.

Пример 6-5: Извадка от файла named.host за GMU

```
; Данни за зоната groucho.edu.
@  IN  SOA vax12.gcc.groucho.edu. joe.vax12.gcc.groucho.edu. {
                        1999070100       ; сериен номер
                        360000           ; опресняване
                        3600             ; повторен опит
                        3600000          ; изтичане
                        3600             ; подразбиращо се ttl
                }
....
; Свързващи записи за зоната 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
...
```

Обратно търсене

Намирането на IP адрес, принадлежащ на даден хост, определено е най-често използваният механизъм от DNS, но понякога ви се налага да намерите каноничното име на хоста, съответстващ на определен адрес. Намирането на това име на хост се нарича *намиране на обратно съответствие* и се използва от различни мрежови услуги за проверка на идентичността на клиента. Когато се използва единединствен файл *hosts*, обратното търсене включва просто претърсване на файла за хост, който притежава търсения IP адрес. Когато се използва DNS, едно пълно претърсване на пространството на имена-

та е немислимо. Вместо това е създаден специален домейн, наречен **in-addr.arpa**, който съдържа IP адресите на всички хостове в обратен десетично-точков запис. Например, на IP адреса **149.76.12.4** съответства името **4.12.76.149.in-addr.arpa**. Записът за ресурс, който свързва тези имена с каноничните имена на хостове, е PTR.

Създаването на зона с пълномощия обикновено означава, че нейните администратори имат пълен контрол над начина, по който на имената се задават адреси. Тъй като обикновено те имат в ръцете си една или повече IP мрежи, съществува съответствие от тип едно-към-много между DNS зоните и IP мрежите. Физическият факултет, например, се състои от подмрежите **149.76.8.0**, **149.76.12.0** и **149.76.14.0.**

В резултат на това, трябва да бъдат създадени нови зони в домейна **in-addr.arpa** заедно със зоната **physics** и да се предоставят на администраторите на мрежата в отдела: **8.76.149.in-addr.arpa**, **12.76.149.in-addr. arpa** и **14.76.149.in-addr. arpa**. В противен случай, инсталирането на нов хост в Изследователската лаборатория ще се изисква от тях да се свържат със своя родителски домейн, за да може новия адрес да се въведе в неговия файл със зоната **in-addr.arpa**.

Базата данни за зоната на подмрежа 12 е показана в Пример 6-6. Съответния свързващ запис в базата данни на нейната родителска зона е показан в Пример 6-7.

Пример 6-6: Извадка от файла named.rev за подмрежа 12

```
; Домейнът 12.76.149.in-addr.arpa.
@  IN  SOA niels.physics.groucho.edu. janet.niels.physics.groucho.edu.{
                        1999090200  36000 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.
```

Пример 6-7: Извадка от файла named.rev за мрежа 149.76

```
; Домейнът 76.149.in-addr.arpa.
@  IN  SOA vax12.gcc.groucho.edu. joe.vax12.gcc.groucho.edu. {
                      1999070100  36000 3600 3600000  3600
                    }
...
; подмрежа 4: Математически факултет.
1.4       IN      PTR       sophus.maths.groucho.edu.
17.4      IN      PTR       erdos.maths.groucho.edu.
```

```
23.4        IN      PTR       gauss.maths.groucho.edu.
...
; подмрежа 12: Физически факултет, отделна зона.
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
...
```

Системните зони на **in-addr.arpa** могат да се създават само като обединение от IP мрежи. Още по-сериозно ограничение е това, че техните мрежови маски трябва да граничат на байт. Всички подмрежи на Университета Groucho Marx имат мрежови маски **255.255.255.0**, следователно за всяка подмрежа може да се създаде in-addr.arpa зона. Ако маската обаче беше **255.255.255.128**, създаването на зони за подмрежата **149.76.12.128** щеше да бъде невъзможно, защото няма начин да се укаже на DNS, че домейна **12.76.149. in-addr.arpa** е бил разделен на две зони на пълномощия с имена на хостове от **1** до **127** и от **128** до **255** съответно.

Използване на named

Демонът *named* предоставя DNS на повечето Unix машини. Той е сървърна програма, чиято оригинална версия е разработена за BSD, за да предоставя услуга за имена на клиентите и евентуално на други сървъри за имена. BIND версия 4 се използваше известно време и се срещаше в повечето дистрибуции на Linux. Новото издание, Версия 8, се разпространява с повечето съвременни дистрибуции на Linux и представлява сериозна промяна в сравнение с предишните версии.[17] Тя има много нови възможности като поддръжка на динамично актуализиране на DNS, DNS съобщения за промени, значително по-голяма производителност и нов синтаксис за конфигурационните файлове. Можете да намерите повече подробности в документацията, която се съдържа в дистрибуцията с изходен код.

Тази част изисква известни познания за начина, по който работи DNS. Ако следващата дискусия ви изглежда като написана на гръцки, прочетете отново раздела "Как работи DNS".

[17] BIND 4.9 е разработен от Paul Vixie (*paul@vix.com*), но днес BIND се поддържа от Internet Software Consortium (*bind-bugs@isc.org*)

named обикновено се стартира по време на зареждането на системата и работи докато машината не се спре отново. Реализациите на BIND преди Версия 8 получаваха необходимата им информация от конфигурационен файл, наречен */etc/named.boot* и различни файлове със съответствия между домейн имена и адреси. Тези файлове се наричат *файлове със зони*. Версиите на BIND с версия 8 или по-голяма използват */etc/named.conf* вместо */etc/named.boot*.

За да стартирате *named* от командния ред, въведете:

```
# /usr/sbin/named
```

named ще се стартира и ще прочете файла */etc/named.boot* и всеки файл със зона, посочен там. Демонът записва своя идентификатор на процес във файла */var/run/named.pid* в ASCII код, ако е необходимо, изтегля посочените файлове със зони от първичните сървъри и започва да слуша на порт 53 за DNS запитвания.

Файлът named.boot

Конфигурационният файл на BIND преди Версия 8 имаше много проста структура. Версия 8 на BIND използва доста различен синтаксис на конфигурационния файл, за да могат да се настроят големия брой нови възможности. Името на конфигурационния файл се промени от */etc/named.boot* в по-старите версии на BIND, на */etc/named.conf* във BIND Версия 8. Ние ще се съсредоточим върху конфигурирането на по-старата версия, защото вероятно тя все още се ползва от повечето дистрибуции, но ще представим и еквивалентния *named.conf* за илюстриране на разликите, като ще посочим и как да преобразувате стария формат в новия.

Файлът *named.boot* обикновено е малък и съдържа почти само указатели към главните файлове със информация за зоната, както и указатели към други сървъри за имена. Коментариите във файла .boot започват със символите (#) или (;) и продължават до началото на нов ред. Преди да обсъдим подробно формата на *named.boot*, да разгледаме примерния файл за **vlager**, даден в Пример 6-8.

Пример 6-8: Файлът named.boot за vlager

```
;
; Файлът /etc/named.boot за vlager.vbrew.com
;
directory       /var/named
;
;               домейн                  файл
;-----------------
cache           .                       named.ca
primary         vbrew.com               named.hosts
primary         0.0.127.in-addr.arpa    named.local
primary         16.172.in-addr.arpa     named.rev
```

Да разгледаме поотделно всяка от конструкциите. Ключовата дума `directory` указва на *named*, че всички имена на файлове, които се използват по-долу в този файл, например файлове със зони, са разположени в директорията */var/named*. Това спестява малко писане.

Ключовата дума `primary`, показана в този пример, зарежда информация в *named*. Тази информация се взима от главните файлове, зададени като последен аргумент. Тези файлове представляват DNS записите за ресурси, които ще разгледаме по-долу.

В този пример конфигурираме *named* като първичен сървър за имена за три домейна, както се задава от трите конструкции `primary`. Първата от трите конструкции указва на *named* да работи като първичен сървър за имена за **vbrew.com**, като вземе данните за зоната от файла *named.hosts*.

Ключовата дума `cache` е много специална и трябва да се задава практически на всички машини, на които работи сървър за имена. Тя инструктира *named* да разреши кеша си и да зареди *указанията за основните сървъри за имена* от указания кеш-файл (в нашия пример, *named.ca*). Ще се върнем на указанията за сървърите за имена в следващия списък.

Следва списък на най-важните опции, които можете да използвате в *named.boot*:

`directory`
: С тази опция се задава директорията, в която се намират файловете със зоните. Имената на файлове в други опции могат да се дадат като относителни спрямо тази директория. Могат да се зададат няколко директории, като се използва повторно `directory`. Стандартът за файловата системата на Linux предполага, че тя е */var/named*.

`primary`

Тази опция приема като аргументи име на домейн и име на файл, като декларира, че локалния сървър е упълномощен за указания домейн. Като първичен сървър, *named* зарежда информация за зоната от дадения главен файл.

Във всеки .boot файл трябва да има поне една конструкция `primary`, която се използва за обратно търсене на мрежа **127.0.0.0**, която е локалната loopback мрежа.

`secondary`

В тази конструкция като аргументи се задават име на домейн, списък с адреси и име на файл. Тя декларира локалния сървър като вторичен главен сървър за зададения домейн.

Един вторичен сървър пази също и достоверните данни за домейна, но не ги извлича от файлове; вместо това той се опитва да ги изтегли от първичния сървър. Чрез списъка с адреси на *named* трябва да се даде IP адреса на най-малко един първичен сървър. Локалният сървър се свързва последователно с всеки от тях, докато успее да прехвърли базата данни на зоната, която след това се записва в резервния файл, подаден като трети аргумент. Ако никой от първичните сървъри не отговори, информацията за зоната се възстановява от резервния файл.

След това *named* се опитва да опреснява данните за зоните на равни интервали. Този процес е описан по-долу във връзка с типа записа за ресурс SOA.

`cache`

Тази опция приема като аргументи име на домейн и име на файл. Този файл съдържа указания за основните сървъри, които представляват списък от записи, сочещи към основните сървъри за имена. Разпознават се само NS и A записи. В полето *domain* трябва да е зададено името на основния домейн – просто една точка (.).

Тази информация е от изключителна важност за *named*; ако конструкцията `cache` не се среща в .boot файла, *named* няма въобще да създаде локален кеш. Тази ситуация сериозно ще намали производителността и ще увеличи натоварването на мрежата, ако следващият сървър, към който се отправя запитване, не е на локалната мрежа. Нещо повече, *named* няма да е в състояние да достигне до който и да е основен сървър за имена и като резултат няма да може да разпознава други имена освен тези, за които

е упълномощен. Изключение от това правило са препращащите сървъри (виж следващата опция `forwarders`).

`forwarders`
: Тази конструкция приема като аргументи списък с адреси, разделени от празно пространство. IP адресите в този списък указват списък от сървъри за имена, които *named* може да запита, ако не успее да отговори на запитването от своя локален кеш. Те се опитват подред, докато някой не удовлетвори запитването. Обикновено като препращащ сървър трябва да използвате сървъра за имена на вашия мрежов доставчик или друг добре известен сървър.

`slave`
: Тази конструкция указва на сървъра за имена, че е второстепенен (*slave*) сървър. Той никога не изпълнява самостоятелно рекурсивни запитвания, а само ги препраща на сървърите, определени с конструкцията `forwarders`.

Съществуват две опции, които няма да описваме тук: `sortlist` и `domain`. Освен това, в тези файлове с бази данни могат да се използват още две директиви: `$INCLUDE` и `$ORIGIN`. Тъй като те са необходими рядко, тях също няма да описваме.

Файлът host.conf на BIND 8

BIND Версия 8 въведе няколко нови възможности, поради които се наложи да се използва нов синтаксис на конфигурационните файлове. Файлът *named.boot* със своите прости едноредови конструкции бе заменен от файла *named.conf*, със синтаксис, близък до този на *gated* и наподобяващ синтаксиса на изходния код на C.

Новият синтаксис е по-сложен, но за щастие съществува инструмент, който автоматизира преобразуването от стария синтаксис към новия. Пакетът с изходния код на BIND 8 съдържа една програма на *perl*, наречена *named-bootconf.pl*, която чете съществуващия файл *named.boot* от стандартния вход и го преобразува в еквивалентния му формат *named.conf* на стандартния изход. За да можете да я използвате, трябва да имате инсталиран интерпретатор на *perl*.

Можете да използвате скрипта например по следния начин:

```
# cd /etc
# named-bootconf.pl <named.boot >named.conf
```

Скриптът ще създаде файл *named.conf*, който изглежда като представения в Пример 6-9. Ние сме премахнали някои от полезните коментари, които се добавят от скрипта, за да ви покажем почти директната връзка между стария и новия синтаксис.

Пример 6-9: Еквивалентният BIND 8 named.conf за vlager

```
//
// Файлът /etc/named.conf за vlager.vbrew.com
options {
        directory "/var/named"
};

zone "." {
        type hint;
        file "named.ca"
};

zone "vbrew.com" {
        type master;
        file "named.hosts"
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local"
};

zone "16.172.in-addr.arpa" {
        type master;
        file "named.rev"
};
```

Ако се вгледате внимателно, ще забележите, че във файла *named.conf* всяка от едноредовите конструкции в *named.boot* е преобразувана в C-подобна конструкция, заградена със символите { }.

Коментариге, които във файла *named.boot* започваха с точка и запетая (;), сега започват с две наклонени черти (//).

Конструкцията `directory` е преведена до параграф `options` с клауза `directory`.

Конструкциите `cache` и `primary` са преобразувани до параграфи `zone` с клаузи `type` съответно `hint` и `master`.

Файлове със зоните не трябва да се променят по никакъв начин; техният синтаксис остава непроменен.

Новият синтаксис за конфигуриране дава възможност на много опции, които не сме описали. Ако ви е необходима информация относно новите опции, най-добрият източник е документацията, съпътстваща пакета с изходен код на BIND версия 8.

Файлове с базата данни на DNS

Главните файлове като *named.hosts*, които се четат от *named*, винаги имат асоциран с тях домейн, който се нарича *начало* (*origin*). Това е името на домейн, което се задава с опциите `cache` и `primary`. В рамките на един главен файл можете да задавате относителни спрямо този домейн имена на хостове и домейни. Име, дадено в един конфигурационен файл, се счита за *абсолютно*, ако завършва с точка, в противен случай се счита за относително спрямо началото. Самото начало може да се посочи със символа (@).

Данните, които се съдържат в един главен файл, са разделени на записи за ресурси (RR – *resource records*). RR са най-малката възможна единица информация, предоставяна от DNS. Всеки запис за ресурс има тип. Записите A, например, дават съответствието между име на хост и IP адрес, а записа CNAME асоциира псевдоним на хост с неговото официално име. Като пример вижте Пример 6-11, който показва главния файл *named.hosts* за Виртуалната пивоварна.

При представянето на записите за ресурси в главните файлове се използва общ формат:

```
[домейн]   [ttl]   [клас] тип rdata
```

Полетата са разделени от интервали или табулации. Един запис може да продължи на няколко реда, ако преди първия символ за нов ред има отваряща скоба и след последното поле се зададе затваряща скоба. Всичко между точка и запетая и символ за нов ред се игнорира. Следва описание на термините на формата:

домейн
: Този термин е името на домейна, за който се прилага този запис. Ако не е посочено никакво име на домейн се приема, че RR се прилага към домейна на предходния RR.

ttl
: За да се укаже на резолвериге, че информацията трябва да се игнорира след определено време, към всеки RR се асоциира "време на живот" (*time to live*, или съкратено *ttl*). Полето *ttl* задава вре-

мето (в секунди) за което информацията е валидна, след като се получи от сървъра. То е десетично число с най-много осем цифри.

Ако не е дадена *ttl* стойност, стойността на полето приема по подразбиране стойността на полето minimum на предхождащия го запис SOA.

клас
: Това е класът на адреса, например IN за IP адреси или HS за обекти от класа на Hesiod. За TCP/IP мрежи трябва да зададете IN.

  Ако не е дадено поле клас, като стойност се приема класа на предишния RR.

тип
: Задава типа на RR. Най-често срещаните типове са A, SOA, PTR и NS. Различните типове RR са описани в следващата част.

rdata
: Съдържа данните, асоциирани с RR. Форматът на това поле зависи от типа на RR. В следващите редове ще опишем поотделно данните за всеки RR.

Следва частичен списък от записи за ресурси, които се използват в главните DNS файлове. Съществуват и някои други RR, които няма да описваме; те са експериментални и обикновено се използват рядко.

SOA

Този RR описва зона на пълномощия (SOA е съкращение от Start of Authority). Той означава, че записите, следващи записа SOA, съдържат достоверна информация за домейна. Всеки главен файл, зададен чрез `primary` конструкция, трябва да съдържа SAO запис за тази зона. Данните за ресурса съдържат следните полета:

origin
: Това поле съдържа каноничното име на първичния сървър за имена за този домейн. Обикновено се дава като абсолютно име.

contact
: Това поле съдържа e-mail адреса на човека, отговорен за поддръжката на домейна, като знакът '*@*' е заменен от точка.

Например, ако отговорният човек във Виртуалната фабрика е **janet**, това поле може да съдържа `janet.vbrew.com.`

serial

Това поле е номера на версията на файла със зоната, зададен като едно десетично число. Всеки път, когато се променят данни във файла със зоната, това число трябва да се увеличава. Общоприето е да се използва число, отговарящо на датата на последната промяна, като към него се прибавя номера на версията, за да се покрие и случая на няколко промени в един и същ ден, например, 2001012600 е промяна 00, направена на 26 Януари 2001 г..

Серийният номер се използва от вторичните сървъри за имена за разпознаване на промени в информацията за зоната. За да предоставят актуална информация, вторичните сървъри изискват на определени интервали записа SOA на първичния сървър и сравняват серийния му номер с този на кеширания запис SOA. Ако номерът се е променил, вторичните сървъри прехвърлят цялата база данни за зоната от първичния сървър.

refresh

Това поле задава интервала в секунди, който вторичните сървъри трябва да изчакат преди следващата проверка на записа SOA на първичния сървър. И тук това е десетично число с най-много осем цифри.

Обикновено, топологията на една мрежа не се променя твърде често, така че това число трябва да задава интервал от около един ден за по-големите мрежи и дори повече за по-малките.

retry

Това число определя интервалите, на които вторичният сървър трябва да се опитва отново да се свърже с първичния сървър, ако запитването или опресняването на зоната се провали. Тези интервали не трябва да са твърде малки, защото една временна повреда на сървъра или мрежов проблем могат да доведат до това, че вторичният сървър хаби ресурсите на мрежата. Час или може би половин час е добър избор.

expire

Това поле задава времето в секунди, след което вторичният сървър трябва да игнорира цялата информация за зоната, ако не е успял да се свърже с първичния сървър. Обикновено

трябва да зададете на това поле поне една седмица (604,800 секунди), но увеличението му на месец, а дори и повече, също е разумно.

`minimum`

Това поле е *ttl* стойността по подразбиране за записите за ресурси, за които не е указана изрично такава стойност. Стойността на ttl определя максималното време, през което другите сървъри за имена, могат да държат RR в своя кеш. Това време се прилага само за нормалните търсения и няма нищо общо с времето, след което един вторичен сървър трябва да се опита да опресни своето копие с информация за зоната.

Ако топологията на вашата мрежа не се променя често, една седмица, а понякога дори повече, е добър избор. Ако единични RR се променят по-често, имате възможност да зададете по-малко ttl конкретно за тях. Ако мрежата ви се променя често, можете да зададете като стойност на `minimum` един ден (86,400 секунди).

A Този запис асоциира един IP адрес с едно име на хост. Полето за данни в ресурса съдържа адреса в десетично-точков формат.

За всяко име на хост трябва да има само един запис A. Името на хоста, използвано в този запис A, се счита за официално или *каноничното* име. Всички други имена на хост са псевдоними и трябва да се свържат с каноничното хост име чрез запис CNAME. Ако каноничното име на нашия хост е **vlager**, трябва да имаме запис A, който да асоциира това хост име с неговия IP адрес. Тъй като може да искаме да имаме и друго име, асоциирано с този адрес, да кажем **news**, трябва да създадем CNAME запис, който да асоциира това алтернативно име с каноничното. Ще поговорим повече за CNAME записите след малко.

NS Записите NS се използват за задаване на първичния и всички вторични сървъри за една зона. Един NS запис сочи към главен сървър за имена на даденатa зона като полето с данни на ресурса съдържа името на хоста – сървър за имена.

Ще срещнете записи NS в две ситуации. Първата е когато делегирате пълномощия за подчинена зона; втората е в базата данни на главната зона в записите за самата подчинена зона. Комплектът от сървъри, зададени в родителската и делегираната зона трябва да съвпадат.

Записът NS задава името на първичния и вторичния сървър за имена на една зона. Тези имена трябва да се преобразуват в адреси, така че да могат да се използват. Понякога сървърите принадлежат на домейна, който обслужват, което води до проблема с "кокошката и яйцето": не можем да разпознаем адреса, докато сървърът за имена не е достъпен, но не можем да се свържем със сървъра, докато не му разпознаем адреса му. За решаването на този проблем можем да конфигурираме специален запис A директно на сървъра за имена на родителската зона. Записът A позволява на сървърите за имена в родителския домейн да разпознаят IP адреса на сървърите за имена на подчинената зона. Тези записи обикновено се наричат свързващи записи, защото осигуряват връзката между родителската и подчинената зона.

CNAME

Този запис асоциира псевдоним с *каноничното име на даден хост*. Той осигурява алтернативно име, чрез което потребителите могат да се обръщат към хоста, чието канонично име е дадено като параметър. Каноничното име е това, за което главният файл съдържа запис A; псевдонимите просто са свързани с това име посредством запис CNAME, но нямат други собствени записи.

PTR

Този тип запис се използва за асоцииране на имена в домейна **in-addr.arpa** с хост имена. Използва се за дефиниране на обратно съответствие между IP адреси и имена на хостове. Зададеното име на хост трябва да е канонично.

MX

Този RR публикува *доставчик на поща* за домейн. Доставчиците на поща се разглеждат в "Mail Routing on the Internet". Синтаксисът на един MX запис е:

```
[домейн] [ttl] [клас] MX предпочитание хост
```

Аргументът `хост` задава името на доставчика на поща за `домейн`. Всеки доставчик на поща има асоциирано с него предпочитание, което е цяло число. Агентът за прехвърляне на поща, който желае да достави поща на *домейн*, опитва да се свърже с всички хостове, които имат MX запис за този домейн, докато успее да намери такъв. Хостът с най-ниска стойност *предпочитание* се проверява първи, след това останалите по нарастваща стойност на *предпочитание*.

HINFO

Този запис предоставя информация за хардуера и софтуера на системата. Синтаксисът му е:

```
[домейн] [ttl] [клас] HINFO хардуер софтуер
```

Полето *хардуер* идентифицира използвания от този хост хардуер. За целта се използват специални конвенции. Списък с валидни "имена на машини" е даден в RFC документа Assigned Numbers (RFC – 1700). Ако полето съдържа интервали, стойността му трябва да се огради с кавички. Полето *софтуер* посочва операционната система, която се използва на машината. Отново трябва да се избере валидно име от Assigned Numbers.

Един HINFO запис, описващ Intel-базирана Linux машина, трябва да изглежда по подобен начин:

```
tao 36500 IN HINFO IBM-PC LINUX2.2
```

HINFO запис за Linux, работещ на Motorola 68000-базирана машина може да изглежда така:

```
cevad   36500 IN HINFO ATARI-104ST LINUX2.0
jedd    36500 IN HINFO AMIGA-3000 LINUX2.0
```

Конфигурация на named само за кеширане

Съществува един специален тип конфигурация на *named*, за която ще поговорим преди да обясним как се прави пълна конфигурация на сървър за имена. Това е така наречената конфигурация *само за кеширане*. На практика тя не обслужва домейн, а работи като ретранслатор за всички DNS запитвания, които се генерират на вашия хост. Предимството на тази схема е, че се изгражда кеш, така че само първото запитване за определен хост в действителност се изпраща до сървъри за имена в Интернет. На всяко повтарящо се запитване се отговаря от кеша на вашия локален сървър за имена. Това може би все още не ви изглежда полезно, но ще се уверите в смисъла от него, когато се свържете към Интернет през телефонна линия, както е описано в Глава 7, *IP през серийна линия* и Глава 8, *Протоколът PPP*.

Файлът *named.boot* за сървър само за кеширане изглежда по следния начин:

```
; Файл named.boot за сървър само за кеширане
directory                       /var/named
primary       0.0.127.in-addr.arpa   named.local ; мрежата localhost
cache         .                      named.ca    ; основните сървъри
```

Като добавка към този файл *named.boot* трябва да създадете файла *named.ca* с валиден списък на основните сървъри за имена. За целта можете да копирате и използвате текста в Пример 6-10. Няма нужда от други файлове при конфигурацията на сървър само за кеширане.

Създаване на главни файлове

В Пример 6-10, Пример 6-11, Пример 6-12 и Пример 6-13 са показани примерни файлове за сървър за имена в Пивоварната, разположен на **vlager**. Поради природата на разглежданата мрежа (една-единствена мрежа), примерът е доста ясен.

Кеш-файлът *named.ca*, представен в Пример 6-10, показва примерни записи с указания за главните сървъри за имена. Един типичен кеш-файл обикновено описва около десетина имена на сървъри. Можете да получите текущия списък на сървърите за имена за основния домейн, като използвате инструмента *nslookup*, който е описан в следващия раздел.[18]

Пример 6-10: Файлът named.ca

```
;
; /var/named/named.ca          Кеш-файл за пивоварната.
;                 Не сме свързани с Интернет, затова не се нуждаем
;                 от основни сървъри За да активирате тези записи,
;                 премахнете символите точка и запетая.
;
;.                          3600000  IN  NS    A.ROOT-SERVERS.NET.
;A.ROOT-SERVERS.NET.        3600000      A     198.41.0.4
;.                          3600000      NS    B.ROOT-SERVERS.NET.
;B.ROOT-SERVERS.NET.        3600000      A     128.9.0.107
;.                          3600000      NS    C.ROOT-SERVERS.NET.
;C.ROOT-SERVERS.NET.        3600000      A     192.33.4.12
;.                          3600000      NS    D.ROOT-SERVERS.NET.
;D.ROOT-SERVERS.NET.        3600000      A     128.8.10.90
;.                          3600000      NS    E.ROOT-SERVERS.NET.
;E.ROOT-SERVERS.NET.        3600000      A     192.203.230.10
;.                          3600000      NS    F.ROOT-SERVERS.NET.
;F.ROOT-SERVERS.NET.        3600000      A     192.5.5.241
;.                          3600000      NS    G.ROOT-SERVERS.NET.
;G.ROOT-SERVERS.NET.        3600000      A     192.112.36.4
```

[18] Забележка: не можете да извършвате запитвания към вашия сървър за имена за основните сървъри, ако нямате инсталирани указания за основен сървър. За да избегнете тази дилема, можете или да направите така, че *nslookup* да използва различен сървър за имена, или да използвате примерния файл от Пример 6-10 като отправна точка и след това да изтеглите пълния списък на валидни сървъри

```
;.                      3600000      NS    H.ROOT-SERVERS.NET.
;H.ROOT-SERVERS.NET.    3600000      A     128.63.2.53
;.                      3600000      NS    I.ROOT-SERVERS.NET.
;I.ROOT-SERVERS.NET.    3600000      A     192.36.148.17
;.                      3600000      NS    J.ROOT-SERVERS.NET.
;J.ROOT-SERVERS.NET.    3600000      A     198.41.0.10
;.                      3600000      NS    K.ROOT-SERVERS.NET.
;K.ROOT-SERVERS.NET.    3600000      A     193.0.14.129
;.                      3600000      NS    L.ROOT-SERVERS.NET.
;L.ROOT-SERVERS.NET.    3600000      A     198.32.64.12
;.                      3600000      NS    M.ROOT-SERVERS.NET.
;M.ROOT-SERVERS.NET.    3600000      A     202.12.27.33
;
```

Пример 6-11: Файлът named.hosts

```
;
; /var/named/named.hosts       Локалните хостове в пивоварната
;                                Началото е vbrew.com
;
@               IN  SOA   vlager.vbrew.com. janet.vbrew.com. (
                          2000012601 ; сериен номер
                          86400      ; опресняване:    веднъж на ден
                          3600       ; повторен опит:  един час
                          3600000    ; изтичане:       42 дни
                          604800     ; минимум:        1 седмица
                          )
                IN  NS    vlager.vbrew.com.
;
; локалната поща се доставя на vlager
                IN  MX    10 vlager
;
; loopback адрес
localhost.      IN  A     127.0.0.1
;
; Ethernet мрежа на Виртуална пивоварна
vlager          IN  A     172.16.1.1
vlager-if1      IN  CNAME vlager
; vlager е освен това сървър за новини
news            IN  CNAME vlager
vstout          IN  A     172.16.1.2
vale            IN  A     172.16.1.3
;
; Ethernet мрежа на Виртуалната винарна
vlager-if2      IN  A     172.16.2.1
vbardolino      IN  A     172.16.2.2
vchianti        IN  A     172.16.2.3
vbeaujolais     IN  A     172.16.2.4
;
```

```
; Ethernet мрежа на Виртуалната спиртоварна (дъщерна компания)
vbourbon          IN  A      172.16.3.1
vbourbon-if1      IN  CNAME  vbourbon
```

Пример 6-12: Файлът named.local

```
;
; /var/named/named.local       Обратно преобразуване на 127.0.0
;                              Началото е 0.0.127.in-addr.arpa.
;
@               IN  SOA   vlager.vbrew.com. joe.vbrew.com. (
                          1          ; сериен номер
                          360000     ; опресняване:   на 100 часа
                          3600       ; повторен опит: един час
                          3600000    ; изтичане:      42 дена
                          360000     ; минимум:       100 часа
                          )

                IN  NS    vlager.vbrew.com.
1               IN  PTR   localhost.
```

Пример 6-13: Файлът named.rev

```
;
; /var/named/named.rev         Обратно преобразуване на нашите IP
адреси
;                              Началото е 16.172.in-addr.arpa.
;
@               IN  SOA   vlager.vbrew.com. joe.vbrew.com. (
                            16         ; сериен номер
                            86400      ; опресняване:   веднъж дневно
                            3600       ; повторен опит: след един час
                            3600000    ; изтичане:      42 дни
                            604800     ; минимум:       1 седмица
                            )
                IN  NS    vlager.vbrew.com.
; пивоварна
1.1             IN  PTR   vlager.vbrew.com.
2.1             IN  PTR   vstout.vbrew.com.
3.1             IN  PTR   vale.vbrew.com.
; винарна
1.2             IN  PTR   vlager-if2.vbrew.com.
2.2             IN  PTR   vbardolino.vbrew.com.
3.2             IN  PTR   vchianti.vbrew.com.
4.2             IN  PTR   vbeaujolais.vbrew.com.
```

Проверка на конфигурацията на сървъра за имена

Инструментът *nslookup* е много добро средство за проверка на работата на вашия сървър за имена. Той може да се използва както интерактивно с въвеждане на команди, така и директно от командния ред с единична команда с непосредствен резултат. Във втория случай просто трябва да го извиквате с:

```
$ nslookup
име-на-хост
```

nslookup запитва сървъра за имена, зададен в *resolv.conf*, за *име-на-хост*. (Ако в този файл са указани няколко сървъра, *nslookup* избира един от тях по произволен начин.)

Интерактивният режим, разбира се, е много по-интересен. Освен да търсите отделни хостове, можете да извършвате запитвания за всякакъв тип DNS записи и да прехвърляте цялата информация за зона от домейн.

Когато се стартира без аргумент, *nslookup* показва сървъра за имена, който използва и влиза в интерактивен режим. След като получите поканата >, можете да въведете името на всеки домейн, за който направите запитване. По подразбиране се търсят записи от клас A, които съдържат IP адреса, свързан с името на домейна.

Можете да зададете друг тип записи с командата:

```
> set type=type
```

където *type* е едно от имената на записите за ресурси описани по-горе, или ANY (произволен).

Следва една примерна сесия на *nslookup*:

```
$ nslookup
Default Server:  tao.linux.org.au
Address:  203.41.101.121

> metalab.unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121


Name:    metalab.unc.edu
Address:  152.2.254.81

>
```

Първо се извежда името на запитвания DNS сървър, а след това резултата от запитването.

Ако се опитате да извършите запитване за име, за което няма асоциран IP адрес, но са били намерени други записи в базата данни на DNS, *nslookup* връща съобщение за грешка., което гласи "`No type A records found`" (не са намерени записи от тип A). Все пак, можете да извършите запитване за други записи, освен тези от тип A, като използвате командата *set type*. За да получите записа SOA на **unc.edu**, трябва да въведете:

```
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

*** No address (A) records available for unc.edu
> set type=SOA
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

unc.edu
        origin = ns.unc.edu
        mail addr = host-reg.ns.unc.edu
        serial = 1998111011
        refresh = 14400 (4H)
        retry   = 3600 (1H)
        expire  = 1209600 (2W)
        minimum ttl = 86400 (1D)
unc.edu name server = ns2.unc.edu
unc.edu name server = ncnoc.ncren.net
unc.edu name server = ns.unc.edu
ns2.unc.edu     internet address = 152.2.253.100
ncnoc.ncren.net internet address = 192.101.21.1
ncnoc.ncren.net internet address = 128.109.193.1
ns.unc.edu      internet address = 152.2.21.1
```

По подобен начин можете да извършвате запитвания за MX записи:

```
> set type=MX
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

unc.edu preference = 0, mail exchanger = conga.oit.unc.edu
unc.edu preference = 10, mail exchanger = imsety.oit.unc.edu
unc.edu name server = ns.unc.edu
unc.edu name server = ns2.unc.edu
unc.edu name server = ncnoc.ncren.net
```

```
conga.oit.unc.edu       internet address = 152.2.22.21
imsety.oit.unc.edu      internet address = 152.2.21.99


ns.unc.edu      internet address = 152.2.21.1
ns2.unc.edu     internet address = 152.2.253.100
ncnoc.ncren.net internet address = 192.101.21.1
ncnoc.ncren.net internet address = 128.109.193.1
```

Използването на тип ANY връща всички записи за ресурси, асоциирани с дадено име.

Практическо приложение на *nslookup*, освен откриването на грешки, е получаването на текущ списък от основните сървъри за имена. Можете да получите този списък като извършите запитване за всички NS записи, асоциирани с основния домейн:

```
> set type=NS
> .
Server:  tao.linux.org.au
Address:  203.41.101.121

Non-authoritative answer:
(root)  name server = A.ROOT-SERVERS.NET
(root)  name server = H.ROOT-SERVERS.NET
(root)  name server = B.ROOT-SERVERS.NET
(root)  name server = C.ROOT-SERVERS.NET
(root)  name server = D.ROOT-SERVERS.NET
(root)  name server = E.ROOT-SERVERS.NET
(root)  name server = I.ROOT-SERVERS.NET
(root)  name server = F.ROOT-SERVERS.NET
(root)  name server = G.ROOT-SERVERS.NET
(root)  name server = J.ROOT-SERVERS.NET
(root)  name server = K.ROOT-SERVERS.NET
(root)  name server = L.ROOT-SERVERS.NET
(root)  name server = M.ROOT-SERVERS.NET

Authoritative answers can be found from:
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
J.ROOT-SERVERS.NET      internet address = 198.41.0.10
K.ROOT-SERVERS.NET      internet address = 193.0.14.129
```

```
L.ROOT-SERVERS.NET      internet address = 198.32.64.12
M.ROOT-SERVERS.NET      internet address = 202.12.27.33
```

За да видите пълния комплект от налични команди използвайте командата *help* в *nslookup*.

Други полезни инструменти

Съществуват още няколко инструмента, които могат да ви помогнат с вашите задачи като администратор на BIND. Ще опишем накратко два от тях. Можете да намерите повече информация за начина, по който те се използват, в съпровождащата ги документация.

Инструментът *hostcvt* ви помага при първоначалното конфигуриране на BIND като преобразува вашия файл */etc/hosts* в главни файлове за *named*. Той генерира и двата вида записи – прав (A) и обратен (PTR), а освен това се грижи и за псевдонимите. Разбира се, той няма да свърши цялата работа вместо вас, тъй като вероятно ще трябва да настроите таймингови стойности в записа SOA или да добавите MX записи. Все пак, това ще ви помогне да спестите някой и друг аспирин. *hostcvt* е част от изходния код на BIND, но може да се намери и като самостоятелен пакет на някои Linux FTP сървъри.

След като настроите вашия сървър за имена, може би ще решите да тествате конфигурацията си. Ето няколко добри инструмента, които опростяват тази задача: първият се нарича *dnswalk*, който е Perl-базиран пакет. Вторият се нарича *nslint*. И двата обхождат вашата DNS база данни, търсейки обичайни грешки и проверяват дали информацията, която намират, е цялостна. Други два полезни инструмента са *host* и *dig*, които са инструменти за извършване на общи запитвания към DNS бази данни. Можете да ги използвате за ръчни проверки и диагностика на състоянието на записите в DNS бази данни.

Много вероятно е да намериге тези инструменти в предварително пакетирана форма. Изходния код на *dnswalk* и *nslint* е достъпен от адрес съответно
http://www.visi.com/~barr/dnswalk/ и *ftp://ftp.ee.lbl.gov/nslint.tar.Z*.
Кодът на *host* и dig може да бъде намерен на
ftp:// ftp.nikhef.nl/pub/network/ и
ftp://ftp.is.co.za /networking/ip/dns/dig/.

ГЛАВА 7

IP ПРЕЗ СЕРИЙНА ЛИНИЯ

Пакетните протоколите като IP или IPX разчитат, че получаващият хост знае къде в потока от данни се намират началото и края на всеки пакет. Механизмът, който се използва за маркиране и откриване на началото и края на пакетите, се нарича *delimitation*. Ethernet протоколът управлява този механизъм в LAN среда, а протоколите SLIP и PPP го управляват при серийни комуникационни линии.

Сравнително ниската цена на нискоскоростните модеми за достъп през телефонна линия и телефонни вериги са причината за голямата популярност на IP протоколите през серийна линия, особено за предоставяне на връзка за крайните потребители до Интернет. Хардуерът, необходим за работата на SLIP или PPP, е прост и може лесно да се осигури. Всичко, от което имате нужда, е модем и сериен порт, снабден с FIFO буфер.

Протоколът SLIP е много прост за реализация и в един момент беше по-използвания от двата протокола. Днес обаче почти всеки използва PPP. Този протокол добавя множество усъвършенствани възможности, което допринася за популярността му в наши дни; по-късно ще разгледаме най-важните от тези възможности.

Linux поддържа базирани на ядрото драйвери за SLIP и PPP. И двата драйвера съществуват от доста време и може да се каже, че са стабилни и надеждни. В тази и в следващата глава ще разгледаме протоколите и начините за тяхното конфигуриране.

Общи изисквания

За да използвате SLIP и PPP, трябва да конфигурирате някои основни мрежови възможности, както беше описано в предишните глави. Трябва да конфигурирате интерфейса-примка и резолвера. Когато се свързвате към Интернет, най-вероятно бихте искали да използвате DNS. Възможностите ви тук са същите както при PPP: можете да изпълнявате DNS заявки през вашата серийна връзка като конфигурирате IP адреса на вашия доставчик на Интернет във файла */etc /resolv.conf*

Работа със SLIP

Сървърите за IP достъп през телефонна линия често предлагат SLIP услуги чрез специални потребителски акаунти. След влизането в такъв акаунт, не се стартира обичайната обвивка; вместо това се изпълнява програма или shell-скрипт, който разрешава SLIP драйвера на сървъра за серийната линия и конфигурира съответния мрежов интерфейс. След това трябва да направите същото и за вашия край на връзката.

При някои операционни системи, SLIP драйверът е програма от потребителското пространство; под Linux той е част от ядрото, което го прави много по-бърз. Тази скорост обаче изисква серийната линия явно да се конвертира в SLIP режим. Тази конверсия се прави чрез специална дисциплина на линия на tty, наречена SLIPDISC. Докато tty използва нормална дисциплина на линия (DISC0), данните се обменят само с потребителски процеси, използвайки обикновените функции *read(2)* и *write(2)*, а SLIP драйверът не може да пише или да чете от tty. При SLIPDISC ролите са обърнати: сега всеки процес от потребителското пространство е блокиран за писане или четене от tty, тъй като всички данни, пристигащи на серийния порт, се предават директно към SLIP драйвера.

Самият SLIP драйвер може да работи с няколко варианта на протокола SLIP. Освен обикновения SLIP, драйверът поддържа CSLIP, в който се извършва така наречената компресия на заглавието от Van Jacobsen (описана в RFC-1144) на изходящите IP пакети. Тази компресия подобрява значително пропускателната способност за интерактивни сесии. Освен това, съществуват шест-битови версии на всеки от тези протоколи.

Един прост начин за конвертиране на серийна линия в SLIP режим е като се използва инструмента *slattach*. Да предположим, че имате модем на */dev/ttyS3* и сте влезли успешно в SLIP сървъра. В този момент изпълнете следното:

```
# slattach /dev/ttyS3 &
```

Този инструмент превключва дисциплината на линия на *ttyS3* към `SLIPDISC` и я свързва към един от мрежовите SLIP интерфейси. Ако това е първата ви активна SLIP връзка, линията ще бъде свързана към *sl0;* втората връзка ще бъде свързана към *sl1* и т.н. По подразбиране, текущите ядра поддържат максимум 256 едновременни SLIP връзки.

Дисциплината на линия, която по подразбиране се избира от *slattach*, е *CSLIP*. Можете да изберете всяка друга дисциплина, като използвате ключа *–p*. За да използвате обикновен SLIP (без компресия), използвайте следната команда:

```
# slattach -p slip /dev/ttyS3 &
```

В Таблица 7-1 можете да намерите списък с всички възможни дисциплини на линията. Съществува специална **псевдо-дисциплина**, наречена `adaptive`, при която ядрото автоматично открива какъв тип SLIP капсулиране се използвана на отдалечения край.

Таблица 7-1. Linux SLIP дисциплини на линия

Дисциплина	Описание
slip	Традиционно SLIP капсулиране.
cslip	SLIP капсулиране с компресия на заглавието от **Van Jacobsen**.
slip6	SLIP капсулиране с шест-битово кодиране. Методът на кодиране е подобен на този, използван от командата *uuencode*, и конвертира SLIP данните в ASCII символи, които могат да се отпечатат. Това конвертиране е полезно, когато нямате серийна връзка, която да е осем бита чиста.
cslip6	SLIP капсулиране с компресия на заглавието от Van Jacobsen и шест-битово кодиране.
adaptive	Това не е действителна дисциплина на линия; при задаването ѝ ядрото се опитва да открие дисциплината на линията на отдалечената машина и да използва същата.

Забележете, че трябва да използвате същото капсулиране, като това на отдалечения край на линията. Например, ако **cowslip** използва CSLIP, вие трябва да използвате същото капсулиране. Ако вашата SLIP връзка не работи, първото нещо, което трябва да направите, е да се уверите, че двата края на линията се съгласуват относно това, дали да се използва компресия на заглавието или не. Ако не сте сигурни какво използва отдалечения край, опитайте се да конфигурирате вашия хост за adaptive slip. Ядрото може да определи правилния тип вместо вас.

slattach ви позволява да разрешите не само SLIP, но и други протоколи, използващи серийната линия, например PPP или KISS (протокол, използван от радиолюбители). Това обаче не е много обичайно и има по-добри инструменти за поддръжката на тези протоколи. За повече подробности прочетете справочната страница на *slattach (8)*.

След като прехвърлите линията на SLIP драйвера, трябва да конфигурирате мрежовия интерфейс. Това отново се прави със стандартните команди *ifconfig* и *route*. Да предположим, че сме се свързали със сървър с име **cowslip** от **vlager**. На **vlager** трябва да изпълните:

```
# ifconfig sl0 vlager-slip pointopoint cowslip
# route add cowslip
# route add default gw cowslip
```

Първата команда конфигурира интерфейса като връзка от тип точка-до точка към **cowslip**, а втората и третата добавят маршрута към ***cowslip*** и подразбиращия се маршрут, използвайки *cowslip* като шлюз.

Две неща не са важни при извикването на *ifconfig*: опцията `pointopoint`, която задава адреса на отдалечения край на връзката точка-до точка и използването на **vlager-slip** като адрес на локалния SLIP интерфейс.

Както вече споменахме, можете да използвате същия адрес, който сте задали на Ethernet интерфейса на **vlager** за вашата SLIP връзка. В този случай, **vlager-slip** може просто да бъде друг псевдоним на адреса **172.16.1.1.** Все пак, възможно е да се наложи да използвате изцяло различен адрес за вашата SLIP връзка. Един такъв случай е когато вашата мрежа използва нерегистриран адрес на IP мрежа, както прави Пивоварната. В следващия раздел ще разгледаме този сценарий по-подробно.

В оставащата част от тази глава винаги ще използваме **vlager-slip**, за да укажем адреса на локалния SLIP интерфейс.

Когато прекъснете SLIP връзката, първо трябва да премахнете всички маршрути през **cowslip**, използвайки *route* с опцията *del*, след това трябва да премахнете интерфейса и да изпратите сигнала hangup (прекъсни работа) към *slattach*. Като използвате отново вашата терминална програма, трябва да затворите линията на модема:

```
# route del default
# route del cowslip
# ifconfig sl0 down
# kill -HUP 516
```

Забележка: трябва да замените 516 с идентификатора на процеса (както се вижда от изхода на командата **ps ax**) на *slattach*, който контролира slip устройството, което искате да спрете.

Работа с частни IP мрежи

Както си спомняте от Глава 5, *Конфигуриране на TCP/IP мрежа*, Виртуалната пивоварна има Ethernet-базирана IP мрежа, използваща нерегистрирани мрежови номера, които са запазени само за вътрешно използване. Пакети от или към една от тези мрежи не се маршрутизират в Интернет; ако трябваше **vlager** да се свърже с **cowslip** и да действа като маршрутизатор за мрежата на Виртуалната пивоварна, хостовете в рамките на Виртуалната пивоварна нямаше да могат да общуват директно с реални Интернет хостове, защото техните пакети просто щяха да бъдат игнорирани от първия основен маршрутизатор.

За да се справим с тази дилема, ще конфигурираме **vlager** да работи като един вид входна точка за получаване на достъп до Интернет услуги. За външния свят той ще изглежда като обикновен SLIP – свързан Интернет хост с регистриран IP адрес (вероятно зададен от мрежовия доставчик, който поддържа **cowslip**). Всеки, който влезе във **vlager**, може да използва текстово-базирани програми като *ftp*, *telnet* или дори *lynx*, за да използва Интернет. По този начин потребителите от мрежата на Виртуалната пивоварна могат да използват telnet, за да влязат във **vlager** и да използва програмите, които се намират там. За някои приложения съществуват решения, с които да се избегне влизането във **vlager**. За WWW потребители, например, можем да стартираме така наречения *proxy-сървър* на **vlager**, който ще препредава всички заявки от вашите потребители към съответните сървъри.

Малко е неудобно, ако трябва да влизате във **vlager**, за да използвате Интернет. Но освен освобождаване от писането на документи (и разходите) за регистриране на една IP мрежа, друго предимство е въз-

можността за инсталиране на защитна стена (firewall). Защитните стени са специално предназначени хостове, които се използват за предоставяне на ограничен достъп до Интернет на потребители от вашата локална мрежа без вътрешните хостове да се излагат на атаки по мрежата от външния свят. Конфигурирането на проста защитна стена е разгледано по-подробно в Глава 9, *Защитна стена за TCP/IP*. В Глава 11, *IP маскиране и транслиране на мрежови адреси*, ще обсъдим една особеност на Linux, наречена "IP маскиране", която предоставя мощна алтернатива на proxy-сървърите.

Да допуснем, че на Пивоварната е бил зададен IP адрес **192.168.5.74** за SLIP достъп. Всичко, което трябва да направите, е да разберете, че конфигурирането, за което ставаше дума по-горе, означава да въведете този адрес във файла */etc/hosts*, като го наречете **vlager-slip**. Процедурата за установяването на самата SLIP връзка остава непроменена.

Използване на dip

Досега, всичко беше много просто. Въпреки това, може би искате да автоматизирате стъпките, които описахме по-горе. Много по-добре би било да има проста команда, която да изпълнява всички необходими стъпки за отваряне на серийното устройство, за указване на модема да набира доставчика, за влизане, разрешаване на SLIP дисциплина на линията и конфигуриране на мрежовия интерфейс. Точно за това се използва програмата *dip*.

Dip е съкращение от *Dialup IP*. Тя е написана от Fred van Kempen и е била доста сериозно модифицирана от много хора. Днес съществува един вариант, който се използва от почти всеки: версията `dip337p-uri`, която е включена в повечето съвременни дистрибуции на Linux или е достъпна от FTP архива **metalab.unc.edu**.

Dip предоставя интерпретатор на прост скрипт-език, който може да управлява модема вместо вас, да конвертира линията в SLIP режим и да конфигурира интерфейсите. Скрипт-езикът е достатъчно мощен, за да върши работа при повечето конфигурации.

За да може да конфигурира SLIP интерфейса, *dip* изисква привилегии на **root**. Примамливо е да се направи *dip* setuid **root**, така че всички потребители да могат да набират SLIP сървъра без да се налага да имат **root** достъп. Това обаче е доста опасно, тъй като задаването на фалшиви интерфейси и подразбиращи се маршрути с *dip* може да внесе смущения при маршрутизацията във вашата мрежа. Дори по-

лошо, това действие ще даде възможност на вашите потребители да установят връзка с *всеки* SLIP сървър и да извършват опасни атаки по вашата мрежа. Ако искате да позволите на вашите потребителите да създават SLIP връзки, напишете малки обвиващи програми за всеки евентуален SLIP сървър и позволете на тези програми да извикват *dip* със специфичния скрипт, който установява връзката. Внимателно написани обвиващи програми след това могат безопасно да бъдат направени **setuid root**[19]. Алтернативен и по-гъвкав подходе да се даде *root* достъп до *dip* на доверените потребители, използвайки програма като *sudo*.

Примерен скрипт

Да предположим, че хостът, към който искаме да направим нашата SLIP връзка е **cowslip**, и че сме написали скрипт за *dip*, наречен *cowslip.dip*, който осъществява нашата връзка. Стартираме *dip* като задаваме името на скрипта като аргумент:

```
# 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 192.168.5.74
#
```

Самият скрипт е показан в Пример 7-1.

Пример 7-1: Примерен dip скрипт

```
# Примерен dip скрипт за набиране на cowslip
# Задаване на локално и отдалечено име и адрес
   get $local vlager-slip
   get $remote cowslip
   port ttyS3          # избиране на сериен порт
   speed 38400         # задаване на максимална скорост
   modem HAYES         # задаване на типа на модема
   reset               # установяваме в изходно положение на модема и tty
   flush               # изчистване на отговора от модема
# Подготовка за набиране.
   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
# OK, вече имаме връзка
   sleep 3
   send \r\n\r\n
```

[19] *diplogin* също трябва да се стартира като setuid **root**. Вижте раздела в края на тази глава.

```
    wait ogin: 10
    if $errlvl != 0 goto error
    send Svlager\n
    wait ssword: 5
    if $errlvl != 0 goto error
    send knockknock\n
    wait running 30
    if $errlvl != 0 goto error
# Влязохме и отдалечената страна стартира SLIP
    print Connected to $remote with address $rmtip
    default  # Правим тази линия наш маршрут по подразбиране
    mode SLIP# Преминаваме в SLIP режим
# в случай на грешка
error:
    print SLIP to $remote failed.
```

След осъществяване на връзка с **cowslip** и разрешаване на SLIP, *dip* ще се отдели от терминала и ще премине във фонов режим. Тогава можете да започнете да използвате обикновените мрежови услуги на SLIP връзката. За да прекратите връзката, просто стартирайте *dip* с опцията *–k*. По този начин се изпраща сигнал за прекъсване до *dip*, използвайки идентификатора на процеса *dip*, намиращ се в */etc/dip.pid:*

```
# dip -k
```

В скрипт-езика на *dip*, ключовите думи предхождани от символа за долар ($) обозначават имена на променливи. *dip* има предварително дефиниран набор от променливи, които ще бъдат изброени по-долу. Например, `$remote` и `$local` съдържат имената на отдалечения и локалния хост, участващи в SLIP връзката.

Първите две конструкции в примерния скрипт са команди *get*, които се използват в *dip* за задаване стойност на променлива. Тук локалните и отдалечените имена на хостове се задават съответно като **vlager** и **cowslip**.

Следващите пет конструкции установяват терминалната линия и модема. `reset` изпраща инициализиращ низ към модема, който го установява в изходно положение. Следващата конструкция изчиства отговора от модема, така че комуникацията за влизане в следващите няколко реда да работи правилно. Тази комуникация е много проста: първо се набира 41988, номера на **cowslip** и се влиза в акаунта `Svlager`, използвайки паролата `knockknock`. Командата `wait` кара dip да чака за низа, зададен като неин пръв аргумент; числото, зададено като втори аргумент, указва времето в секунди, след което чакането ще се прекрати, ако не се получи такъв низ. Докато командата

се изпълнява, използваните в процедурата за влизане `if` команди правят проверка за възникване на грешка.

Последните команди, които се изпълняват след влизането, са `default`, която прави SLIP връзката подразбиращ се маршрут за всички хостове, и `mode`, която разрешава SLIP режим на линията и конфигурира интерфейса и таблицата с маршрути.

Справочник за dip

В този раздел ви даваме справка за повечето команди на *dip*. Можете да получите общ преглед на всичките команди, които предоставя програмата, като стартирате *dip* в тестов режим и въведете командата `help`. За да научите повече за синтаксиса на дадена команда, можете да я въведете без никакви аргументи. Следващият пример илюстрира командата `help`:

```
# dip -t
DIP: Dialup IP Protocol Driver version 3.3.7p-uri (25 Dec 96)
Written by Fred N. van Kempen, MicroWalt Corporation.
Debian version 3.3.7p-2 (debian).

DIP> help
DIP knows about the following commands:
     beep        bootp       break       chatkey     config
     databits    dec         default     dial        echo
     flush       get         goto        help        if
     inc         init        mode        modem       netmask
     onexit      parity      password    proxyarp    print
     psend       port        quit        reset       securidfixed
     securid     send        shell       skey        sleep
     speed       stopbits    term        timeout     wait

DIP> echo
Usage: echo on|off
DIP>
```

В следващият раздел, примерите, които показват поканата за въвеждане *DIP>*, демонстрират начина, по който се въвежда команда в тестов режим и какъв е резултата от нея. Примерите, в които липсва тази покана, трябва да се считат за извадки от скриптове.

Команди за модем

Dip предоставя множество команди, които конфигурират вашата серийна линия и модема. Някои от тях са очевидни, например командата *port*, която избира сериен порт, и `speed`, `databits`, `stopbits` и

`parity`, които задават общите параметри на линията. Командата `modem` избира типа на модема. В момента единственият тип, който се поддържа, е `HAYES` (главните букви са задължителни). Трябва да укажете на *dip* типа на модема, защото в противен случай ще откаже да изпълни командите `dial` и `reset`. Командата *reset* изпраща към модема низ за установяване в изходно състояние; използвания низ зависи от избрания тип модем. За Hayes-съвместими модеми този низ е `ATZ`.

Кодът `flush` може да бъде използван за изчистване на всички отговори, които модемът е изпратил досега. В противен случай, скрипта за влизане, следващ командата `reset`, може да се объркa, ако прочете отговора `OK` от предишни команди.

Командата `init` избира низ за инициализиране, който ще бъде изпратен на модема преди набирането. По подразбиране, за Hayes модеми този низ е "`ATE0 Q0 V1 X1`", което включва ехото на команди, дълги кодове за резултат и избира "сляпо" набиране (без проверка за наличието на сигнал от телефонната централа). По подразбиране, съвременните модеми имат добра подразбираща се конфигурация, така че това е малко излишно, въпреки че няма да навреди.

Командата `dial` изпраща инициализационния низ на модема и набира отдалечената система. По подразбиране, командата за набиране за Hays модеми е `ATD`.

Командата echo

Командата `echo` се използва като помощно средство при отстраняването на грешки. Ако използвате *echo on*, *dip* ще повтори на конзолата всичко, което се изпраща към серийното устройство. Тази възможност може отново да бъде изключена с `echo off`.

Освен това, *dip* ви позволява да напуснете временно скрипт-режима и да влезете в терминален режим. В този режим можете да използвате *dip* като всяка обикновена терминална програма, като записвате символите, които въвеждате в серийната линия, четете данни от серийната линия и показвате символите. За да излезете от този режим, натиснете Ctrl-].

Командата get

Командата `get` се използва за задаване на променливи от *dip*. Най-простата форма е да се зададе константа като стойност на променли-

ва, както направихме в *cowslip.dip*. Можете обаче да укажете, че данните трябва да се въведат от потребителя, като вместо стойност зададете ключовата дума `ask`:

```
DIP> get $local ask
Enter the value for $local: _
```

Трети метод е да се получи стойността от отдалечения хост. На пръв поглед това изглежда странно, но в някои случаи е много полезно. Например, някои SLIP сървъри няма да ви позволят да използвате вашия собствен IP адрес при SLIP връзката, а всеки път, когато се свържете с тях, ще ви задават адрес от пул с адреси, като отпечатват някакво съобщение, което ви информира за дадения ви адрес. Ако съобщението е подобно на "`Your address: 192.168.5.74`", следващият фрагмент с *dip* код ще ви позволи да извлечете този адрес:

```
# завършване на влизането
wait address: 10
get $locip remote
```

Командата print

Това е командата, която се използва за отпечатване на текст на конзолата, от която е била стартирана програмата *dip*. Всяка от променливите на *dip* може да се използва в команди за отпечатване. Ето един пример:

```
DIP> print Using port $port at speed $speed
Using port ttyS3 at speed 38400
```

Имена на променливите

Dip разпознава само предварително дефиниран набор от променливи. Името на една променлива винаги започва със символа за долар и трябва да се записва с малки букви.

Променливите `$local` и `$locip` съдържат името и IP адреса на локалния хост. Когато съхранявате каноничното име на хост в променливата `$local`, *dip* автоматично ще се опита да преобразува името на хоста в IP адрес и да го запише в променливата `$locip`. Подобен, но обратен процес се извършва, когато зададете IP адрес в променливата `$locip`; *dip* ще се опита да извърши обратно търсене, за да идентифицира името на хоста и да го съхрани в променливата `$local`.

Променливите `$remote` и `$rmtip` действат по същия начин, само че за името и адреса на отдалечения хост. `$mtu` съдържа стойността на MTU за връзката.

Тези пет променливи са единствените, на които могат директно да се задават стойности, използвайки командата `get`. Стойностите на множество други променливи се задават като резултат от конфигурационните команди, носещи същото име, но те могат да `$speed` се използват в конструкции `print`; тези променливи са `$modem`, `$port` и. `$errlvl` е променливата, чрез която можете да получите достъп до резултата от последната изпълнена команда. Ниво на грешка 0 означава успех, а ненулева стойност обозначава грешка.

Командите if и goto

Командата `if` е по-скоро условно разклонение, отколкото пълноценна програмна конструкция *if*. Нейният синтаксис е следния:

`if` *променлива оператор число* `goto` *етикет*

Изразът трябва да бъде просто сравнение с използването на една от променливите `$errlvl`, `$locip` и `$rmtip`. Променливата трябва да е цяло число; операторът `op` може да бъде един от ==, !=, <, >, <= или >=.

Чрез командата `goto` изпълнението на скрипта продължава от реда, който следва след обозначения с *етикет* ред. Етикетът трябва да бъде първата дума от реда и непосредствено след нея трябва да следва двоеточие.

send, wait и sleep

Тези команди помагат за реализирането на прости chat-скриптове в *dip*. Командата *send* извежда аргументи си към серийната линия. Тя не поддържа променливи, но разпознава всички последователности от символ и обратно наклонена черта в C-стил като `\n` за нов ред и `\b` за backspace (стъпка назад). Символът тилда (~) може да се използва като съкращение за връщане на каретката/нов ред.

Командата `wait` приема като аргумент дума и чете всички входни данни от серийната линия, докато не открие последователност от символи, която съвпада с тази дума. Самата дума не може да съдържа интервали. Като незадължителен втори аргумент можете да зададете на `wait` стойност за таймаут; ако очакваната дума не бъде получена в рамките на посочените секунди, командата ще върне стойност 1 посредством `$errlvl`. Тази команда се използва за откриване на покана за влизане и други съобщения.

Командата `sleep` може да се използва, за да се зададе определен период от време за изчакване; например, за търпеливото изчакване да завърши всяка login-последователност. Отново, интервалът се задава в секунди.

mode и default

Тези команди се използват за превключване на серийната линия в SLIP режим и за конфигуриране на интерфейса.

Командата `mode` е последната изпълнявана от *dip* команда, преди да се премине в режим демон. Тази команда не връща стойност, освен ако не възникне някаква грешка.

`mode` приема като аргумент име на протокол. В моментa *dip* разпознава `SLIP`, `CSLIP`, `SLIP6`, `CSLIP6`, `PPP` и `TERM` като валидни имена. Текущата версия на *dip* обаче не разпознава adaptive SLIP.

След разрешаване на SLIP режим за серийната линия, *dip* изпълнява *ifconfig*, за да конфигурира интерфейса като връзка от тип точка-до-точка и стартира *route*, за да зададе маршрута до отдалечения хост.

Ако, като допълнение, скриптът изпълни командата `default` преди `mode`, *dip* създава подразбиращ се маршрут, който сочи към SLIP връзката.

Работа в сървърен режим

Настройването на вашия SLIP клиент беше действително трудната задача. Конфигурирането на вашия хост да работи като SLIP сървър е доста по-лесно.

Има два начина за конфигуриране на SLIP сървър. И двата начина обаче изискват да се създаде акаунт за влизане в сървъра за всеки SLIP клиент. Представете си, че предоставяте SLIP услуги на Arthur Dent на **dent.beta.com**. Можете да създадете акаунт **dent**, като добавите следния ред към вашия файл *passwd*:

```
dent:*:501:60:Arthur Dent's SLIP account:/tmp:/usr/sbin/diplogin
```

След това трябва да зададете паролата на **dent**, използвайки инструмента *passwd*.

Командата *dip* може да се използва в сървърен режим, ако я стартирате с името *diplogin*. Обикновено, *diplogin* е връзка към *dip*. Основният й конфигурационен файл е */etc/diphosts*, където се задава какъв IP адрес трябва да бъде зададен на SLIP потребител, когато той/тя

влезе в системата. Като алтернатива, можете да използвате командата *sliplogin* – инструмент, произлизащ от BSD, с по-гъвкава конфигурационна схема, която ви позволява да изпълнявате shell-скриптове винаги, когато един хост се свързва или прекратява връзката.

Когато нашият SLIP потребител **dent** влиза в сървъра, *dip* се стартира като сървър. За да разбере дали той всъщност има право да използва SLIP, dip търси потребителското име в */etc/diphosts*. Този файл съдържа подробности за правата за достъп и параметри на връзката за всеки SLIP потребител. Общият формат на запис от */etc/diphosts* изглежда по следния начин:

```
# /etc/diphosts
user:password:rem-addr:loc-addr:netmask:comments:protocol,MTU
#
```

Всяко от полетата е описано в Таблица 7-2.

Таблица 7-2: Описание на полетата на /etc/diphosts

Поле	Описание
`user`	Потребителското име на клиента, извикващ *dip*, за който ще се приложи този запис.
password	Второто поле на файла */etc/diphosts* се използва за добавяне на допълнителен слой на сигурност на връзката, която се базира на парола. Тук можете да поставите парола в шифрирана форма (също както в */etc/passwd*) и *diplogin* ще изисква от потребителя да въведе паролата преди да получи SLIP достъп. Забележете, че тази парола се използва като допълнение към обикновената базирана на *login* парола, която потребителят ще въведе.
`rem-addr`	Адресът, който ще бъде зададен на отдалечената машина. Този адрес може да бъде зададен или като име на хост, което ще бъде преобразувано, или като IP адрес в стандартния четиризначен формат с точки.
`loc-addr`	IP адресът, който ще бъде използван за този край на SLIP връзката. Може да се зададе и като име на хост, което може да се преобразува, в стандартния четиризначен формат с точки.

Поле	Описание
`netmask`	Мрежовата маска, която ще се използва за целите на маршрутизацията. Много хора се объркват от този елемент. Мрежовата маска не се отнася за самата SLIP връзка, а се използва в комбинация с полето `rem-addr`, за да се създаде маршрут към отдалечения сайт. Мрежовата маска трябва да е такава, че да е използваема от мрежата, поддържана от отдалечения хост.
`comments`	Това поле е текст в свободна форма, който можете да използвате за документирането на файла */etc/diphosts*. Не се използва за други цели.
`protocol`	Това е полето, в което се задава какъв протокол или дисциплина на линията искате да се приложи към тази връзка. Валидните стойности са същите като за аргумента *-p* на командата *slattach*.
`MTU`	Максималната дължина на единица за предаване, която ще преминава през тази връзка. Това поле описва максималния обем от данни, който ще се предават през връзката. Всеки пакет от данни, насочен към SLIP устройството и по-голям от MTU, ще бъде разделен на пакети, не по-големи от тази стойност. Обикновено MTU се конфигурира идентично и в двата края на връзката.

Примерен запис за **dent** може да изглежда така:

```
dent::dent.beta.com:vbrew.com:255.255.255.0:Arthur Dent:CSLIP,296
```

В нашия пример на потребителя **dent** се дава достъп до SLIP без да се изисква допълнителна парола. Потребителят ще получи IP адреса, свързан с **dent.beta.com** с мрежова маска `255.255.255.0`. Неговият подразбиращ се маршрут трябва да бъде насочен към IP адреса на **vbrew.com** и той ще използва CSLIP протокол с MTU от 296 байта.

Когато **dent** влиза в сървъра, *diplogin* извлича информацията за него от файла *diphosts*. Ако второто поле съдържа някаква стойност, *diplogin* ще изиска "парола за допълнителна сигурност". Низът, въведен от потребителя, се шифрира и сравнява с паролата от *diphosts*. Ако те не съвпадат, опитът за влизане се отхвърля. Ако полето password съдържа низа `s/key` и *dip* е била компилирана с поддръжка на S/Key, ще се извърши S/Key удостоверяване на самоличността.

Тази процедура е описана в документацията, придружаваща пакета с изходен код на *dip*.

След успешно влизане, *diplogin* продължава като превключва серийната линия в CSLIP или SLIP режим и конфигурира интерфейса и маршрута. Тази връзка остава докато потребителят не я прекрати и модемът затвори линията. Тогава *diplogin* връща линията в нормална дисциплина и излиза.

diplogin изисква привилегии на супер-потребител. Ако нямате *dip*, работещ със setuid **root**, трябва да направите за *diplogin* отделно копие на *dip* вместо проста връзка. Тогава за *diplogin* може безпроблемно да се направи setuid, без това да окаже влияние върху състоянието на самия *dip*.

ГЛАВА 8

ПРОТОКОЛЪТ PPP

Също като SLIP, PPP е протокол, който се използва за изпращане на дейтаграми през серийна връзка; освен това, той е създаден за преодоляване на някои от недостатъците на SLIP. Първо, PPP може да пренася множество протоколи и по този начин не е ограничен само до протокола IP. Той предоставя възможност за откриване на грешки по самата връзка, докато SLIP приема и препраща повредени дейтаграми, стига грешките да не са в заглавието. Също толкова важно е, че този протокол позволява на комуникиращите страни да договарят опции при стартиране, например IP адреса и максималния размер на дейтаграмата и предоставя възможност за удостоверяване на самоличността на клиента. Това вградено договаряне позволява надеждно автоматизиране на процеса за установяване на връзка, докато удостоверяването на самоличността премахва необходимостта от неудобните потребителски акаунти за влизане, които изисква SLIP. За всяка от тези възможности PPP има отделен протокол. В тази глава ще разгледаме накратко основните изграждащи блокове на PPP. Това далеч не е пълно разглеждане на PPP; ако искате да научите повече за този протокол, съветваме ви да прочетете неговата RFC спецификация и още около дузината свързани с него RFC документи[20]. Освен това книгата *Using & Managing PPP* (издание от O'Reilly) от Andrew Sun съдържа изчерпателна информация по темата.

В основата на PPP е протоколът *HDLC* (*High-Level Data Link Control* – управление от високо ниво на връзката за данни), който определя

[20] Съответните RFC документи са изброени в библиографията на края на тази книга.

границите около отделните PPP кадри и предоставя 16-битова контролна сума[21]. Обратно на по-примитивното SLIP капсулиране, един PPP кадър може да съдържа пакети от протоколи, различни от IP, например IPX на Novell или Appletalk. PPP постига това чрез добавяне на поле за протокол към основния HDLC кадър, което идентифицира типа на пакета, пренасян от кадъра.

Протоколът *LCP* (*Link Control Protocol* – протокол за управление на връзката) се използва върху HDLC за договаряне на опции на връзката за данни. Например, опцията *MRU* (*Maximum Receive Unit* – максимална единица за приемане) определя максималния размер на дейтаграмата, който една от страните на връзката е съгласна да получава.

Важна стъпка в етапа за конфигуриране на една PPP връзка е удостоверяването на самоличността на клиента. Въпреки че не е задължително, за телефонни линии това е необходимо с цел да не се допускат злонамерени потребители. Обикновено, извикващият хост (сървърът) иска от клиента да се оторизира като докаже, че знае някакъв таен ключ. Ако извикващият не успее да отговори с правилната тайна, връзката се прекъсва. При PPP оторизирането става и в двете посоки; извикващият също може поиска от сървъра да удостовери самоличността си. Тези процедури за удостоверяване на самоличността са напълно независими една от друга. Съществуват два протокола за различни видове оторизиране, които ще разгледаме по-долу в тази глава: *PAP (Password Authentication Protocol* – протокол за удостоверяване на самоличността с парола) и *CHAP* (*Challenge Handshake Authentication Protocol* – протокол за удостоверяване на самоличността с отговаряне на предизвикателства).

Всеки мрежов протокол, който се маршрутизира през връзката (например IP и AppleTalk) се конфигурира динамично като се използва съответния NCP протокол (*Network Control Protocol* – управляващ мрежов протокол). За изпращане на IP дейтаграми през връзката, и двете страни, използващи PPP, първо трябва да се договорят за IP адреса, който използва всяка от тях. Управляващият протокол, използван за това договаряне е *IPCP* (*Internet Protocol Control Protocol* – управляващ протокол за Интернет протокола).

---

[21] Всъщност, HDLC е много по-общ протокол, създаден от ISO и освен това е ключов компонент на спецификацията X.25

Освен изпращането на стандартни IP дейтаграми през връзката, PPP поддържа и компресията на заглавието от Van Jacobson за IP дейтаграми. Тази техника свива заглавията на TCP пакетите до три байта. Тя се използва и в CSLIP и разговорно е известна като VJ компресия на заглавието. Използването на компресия също може да се договори при стартиране чрез IPCP.

PPP под Linux

Под Linux, функционалността на PPP е разделена на две части: компонент в ядрото, който управлява протоколите от ниско ниво (HDLC, IPCP, IPXCP и т.н.) и демона от потребителско пространство *pppd*, който предоставя различните протоколи от високо ниво като PAP и CHAP. Настоящата версия на PPP софтуера за Linux съдържа PPP демона *pppd* и програма, наречена *chat*, която автоматизира набирането на отдалечената система.

Драйверът за PPP в ядрото е написан от Michael Callahan и е преработен от Paul Mackerras. *pppd* е производен на свободната реализация на PPP[22] за Sun и 386BSD мишини, която е написана от Drew Perkins и други, и се поддържа от Paul Mackerras. Тя е адаптирана за Linux от Al Longyear. Програмата *chat* е написана от Karl Fox[23].

Както SLIP, PPP е реализиран чрез специална дисциплина на линия. За да използвате серийна линия като PPP връзка, първо, както обикновено, установявате връзката през вашия модем и след това конвертирате линията в PPP режим. В този режим всички входящи данни се предават към PPP драйвера, който проверява входящите HDLC кадри за валидност (всеки HDLC кадър съдържа 16-битова контролна сума), разопакова ги и ги разпределя. В момента PPP може да пренася както протокола IP, като опция с компресия на заглавието от Van Jacobson, така и протокола IPX.

pppd помага на драйвера в ядрото като извършва инициализация и фазата за удостоверяване на самоличността, която е необходима преди през връзката да може да бъде изпратен действителен мрежов трафик. Поведението на *pppd* може да се настрои фино посредством

---

22 Ако имате някакви общи въпроси относно PPP, попитайте хората от пощенския списък Linux-net на адрес **vger.rutgers.edu.**

23 С Karl можете да се свържете на адрес *karl@morningstar.com.*

множество опции. Тъй като PPP е доста сложен, не е възможно всички опции да бъдат обяснени в една глава. Затова, тази книга не може да покрие всички аспекти на *pppd* и ви дава само въведение. Повече информация можете да намерите в *Using & Managing PPP* или справочните страниците за *pppd* и README файловете в дистрибуцията с изходен код за *pppd*, която ще ви помогне да намерите отговори на повечето въпроси, които не са разгледани в тази глава. Документът PPP-HOWTO също може да ви бъде от полза.

Вероятно най-голямата помощ по отношение на конфигурирането на PPP ще получите от други потребители на същата дистрибуция за Linux. Въпросите, свързани с конфигурирането на PPP, са много общи, затова опитайте вашия локален пощенски списък с потребители или IRC канала за Linux. Ако проблемите ви продължат дори след като сте прочели документацията, можете да опитате групата по интереси *comp.protocols.ppp*. Това е мястото, където можете да намерите повечето от хората, които участват в разработването на *pppd*.

Използване на pppd

Когато искате да се свържете с Интернет чрез PPP връзка, трябва да конфигурирате основни мрежови възможности като loopback устройството и резолвера. И двете бяха разгледани в Глава 5, *Конфигуриране на TCP/IP мрежа*, и Глава 6, *Конфигуриране на услугата за имена и резолвера*. Можете просто да конфигурирате сървъра за имена на вашия доставчик на Интернет във файла */etc/resolv.conf*, но това ще означава, че всяка DNS заявка се изпраща през вашата серийна връзка. Тази ситуация не е оптимална; колкото по-близо (по отношение на мрежата) сте до вашия сървър за имена, толкова по-бързо ще се извършват търсенията на имена. Алтернативно решение е да се конфигурира сървър за имена само за кеширане на хост от вашата мрежа. Това означава, че първия път, когато извършвате DNS запитване за конкретен хост, заявката ви ще бъде изпратена през вашата серийна връзка, но на всяка следваща заявка ще се отговаря директно от локалния сървър за имена и работата ви ще бъде доста по-бърза. Това конфигуриране е описано в Глава 6, в раздела Конфигурация на named само за кеширане.

Като въвеждащ пример за начина, по който можете да създадете PPP връзка с *pppd*, да предположим, че оттук сте на **vlager**. Първо, наберете PPP сървъра **c3po** и влезте с акаунта **ppp**. **c3po** ще стартира своя PPP драйвер. След като излезете от програмата за комуникации, която сте използвали за набиране, изпълнете следващата команда, като

замениге името на използваното серийно устройство, което е показано тук:

```
# pppd /dev/ttyS3 38400 crtscts defaultroute
```

Тази команда превключва серийната линия *ttyS3* на PPP дисциплина на линията и договаря IP връзка със **c3po**. Скоростта на прехвърляне, използвана на серийния порт, ще бъде 38400 bps. Опцията `crtscts` включва хардуерно договаряне на порта, което е абсолютно задължително за скорости над 9600 bps.

Първото нещо, което прави *pppd* след стартирането си е да договори няколко характеристики на връзката с отдалечения край, използвайки протокола LCP. Обикновено, подразбиращият се набор от опции, които *pppd* се опитва да договори, ще работят, така че тук няма да се знимаваме с това. Трябва да кажем, че част от това договаряне включва изискване или задаване на IP адреса на всеки край на връзката.

За известно време ще предполагаме също, че **c3po** не изисква никакво удостоверяване на самоличността, така че фазата на конфигуриране е завършена успешно.

След това *pppd* ща договори IP параметрите с другия край на връзката, използвайки IPCP, управляващият протокол за IP. Тъй като порано не зададохме определен IP адрес на *pppd*, той ще се опита да използва адреса, получен от търсенето на името на локалния хост от резолвера. След това и двете страни ще обявят една на друга своите адреси.

Обикновено няма нищо лошо да се използва това подразбиращо се поведение. Дори ако машината ви е в Ethernet мрежа, можете да използвате един и същ IP адрес за Ethernet и за PPP интерфейса. Въпреки това, *pppd* ви позволява да използвате различни адреси или дори да изискате от отсрещната страна да използва някакъв конкретен адрес. Тези опции са разгледани по-долу в раздела "Опции за конфигуриране на IP".

След преминаването през фазата за конфигуриране на IPCP, *pppd* ще подготви мрежовия слой на вашия хост да използва PPP връзката. Той конфигурира първо мрежовия PPP интерфейс като връзка от точка до точка, използвайки *ppp0* за първата активна PPP връзка, *ppp1* за втората и т.н. След това демонът създава запис в таблицата с маршрутите, който сочи към хоста от другата страна на връзката. В предишния пример *pppd* направи така, че подразбиращият се мрежов маршрут да сочи към **c3po**, тъй като му зададохме опцията

`defaultroute`[24]. Подразбиращият се маршрут ви улеснява като изпраща всички IP дейтаграми, насочени към не-локален хост, към **c3po**; това има смисъл, тъй като е единствения начин, по който могат да бъдат достигнати хостовете-получатели. Съществуват множество различни поддържани от *pppd* схеми за маршрутизиране, които ще разгледаме детайлно по-долу в тази глава.

Използване на файлове с опции

Преди *pppd* да анализира своите аргументи от командния ред, той сканира няколко файла за опции по подразбиране. Тези файлове могат да съдържат всякакви валидни аргументи, поставени в произволен брой редове. Символите диез (#) обозначават начало на коментар.

Първият файл с опции е */etc/ppp/options*, който се сканира винаги, когато се стартира *pppd*. Използването му за задаване на някои глобални опции по подразбиране е добра идея, тъй като ви позволява да предпазите потребителите си от извършване на няколко неща, които могат да нарушат сигурността. Например, за да укажете на *pppd* да изисква някакъв тип удостоверяване на самоличността (PAP или CHAP) от отсрещната страна, трябва да добавите опцията *auth* в този файл. Тази опция не може да бъде отменена от потребителя, така че става невъзможно да се създаде PPP връзка с някоя система, която не е във вашата база данни за удостоверяване на самоличността. Трябва обаче да отбележим, че някои опции могат да бъдат отменяни; добър пример за това е низа `connect`.

Другият файл с опции, който се чете след */etc/ppp/options*, е *.ppprc* в личната директория на потребителя. Това позволява на всеки потребител да зададе свой собствен набор от опции по подразбиране.

Един примерен файл */etc/ppp/options* изглежда така:

```
# Глобални опции за pppd, работещ на vlager.vbrew.com
lock            # използване на UUCP-стил за заключване на устройството
auth            # изисква удостоверяване на самолична
usehostname     # използва локалното име на хоста за CHAP
domain
vbrew.com       # нашето име на домейн
```

[24] Подразбиращия се мрежов маршрут се инсталира само ако вече не е зададен такъв.

Ключовата дума `lock` указва на *pppd* да спазва стандартния UUCP метод за заключване на устройства. Според тази конвенция, всеки процес, който използва серийното устройство, например */dev/ttyS3*, създава заключващ файл с име подобно на *LCK..ttyS3* в специална директория със заключващи файлове, за да покаже, че устройството се използва. Това е необходимо като предупредителен сигнал за други програми, като *minicom* или *uucico*, да не отварят серийното устройство, докато то се използва от PPP.
Следващите три опции са свързани с удостоверяването на самоличността и следователно със сигурността на системата. Опциите за удостоверяване на самоличността е най-добре да се поставят в глобалния конфигурационен файл, тъй като те са "привилегировани" и не могат да бъдат отменени от файловете *~/.ppprc* с опции на потребители.

Използване на chat за автоматизиране на набирането

Едно от нещата, които може да са ви направили впечатление като неудобни в предишния пример, е че трябва да създадете връзката ръчно преди да можете да стартирате *pppd*. За разлика от *dip*, *pppd* няма свой собствен скрипт-език за набиране на отдалечената система и влизане в нея, а разчита на външна програма или shell-скрипт да направи това. Командата, която трябва да се изпълни, може да бъде дадена на *pppd* с опцията `connect` от командния ред. *pppd* ще пренасочи стандартния вход и изход на командата към серийната линия.

Софтуерният пакет *pppd* се предоставя с много проста програма, наречена *chat*, която може да бъде използвана за автоматизиране на простите последователности при влизане. Ще разгледаме част от възможностите на тази команда.

Ако вашата последователност при влизане е сложна, ще ви е необходимо нещо по-мощно от *chat*. Една полезна алтернатива, за която можете да помислите, е инструментът *expect*, написан от Don Libes. Той има много мощен език, базиран на Tcl, и е проектиран точно за такъв тип приложения. Тези от вас, чиято последователност при влизане изисква, например, удостоверяване на самоличността чрез предизвикателство/отговор, включващо подобни на калкулатор генератори на ключове, ще открият, че expect е достатъчно мощен да се справи с тази задача. Тъй като има много възможни вариации на тази тема, в тази книга няма да описваме как да разработите подходящ

expect скрипт. Достатъчно е да кажем, че трябва да извикате вашия *expect* скрипт като зададете името му, използвайки опцията `connect` на *pppd*. Освен това е важно да отбележим, че когато скриптът се изпълнява, стандартните вход и изход ще бъдат свързани с модема, а не с терминала, от който е стартиран *pppd*. Ако скриптът ви изисква взаимодействие с потребителя, трябва да го направите като отворите свободен виртуален терминал или използвате някакво друго средство.

Командата *chat* ви позволява да укажете chat скрипт в UUCP-стил. По принцип, един chat скрипт се състои от редуваща се последователност от низове, които очакваме да получим от отдалечената система и отговорите, които трябва да изпратим. Ще ги наричаме съответно *очаквани* и *изпращани* низове. Следва типична извадка от chat скрипт:

```
ogin: b1ff ssword: s3|<r1t
```

Този скрипт указва на *chat* да чака отдалечената система да изпрати покана за влизане и да върне име за влизане **b1ff**. Чакаме само `ogin:`, така че няма значение дали поканата за влизане започва с малка или главна буква l. Следващият низ е друг очакван низ, който указва на *chat* да чака за покана за паролата и да изпрати като отговор нашата парола.

Общо взето, това е предназначението на chat-скриптовете. Един пълен скрипт за набиране на PPP сървър, разбира се, трябва да включва и подходящите команди за модема. Да предположим, че вашият модем разпознава набора от команди на Hayes и телефонният номер на сървъра е 318714. Пълното извикване на *chat* за създаване на връзка със **c3po** ще бъде:

```
$ chat -v " ATZ OK ATDT318714 CONNECT " ogin: ppp word: GaGariN
```

По дефиниция, първо трябва да се зададе очакван низ, но тъй като модемът няма да каже нищо преди да го му пратим нещо, указваме на *chat* да прескочи първия очакван низ като задаваме празен низ. След това изпращаме `ATZ`, командата за установяване в изходно положение на Hayes-съвместими модеми и чакаме отговор (`OK`). Следващият низ изпраща към *chat* командата *dial* заедно с телефонния номер и в отговор очаква съобщение за свързване `CONNECT`. Отново следва празен низ, тъй като сега не искаме да изпращаме нищо, а чакаме поканата за влизане. Останалата част от chat-скрипта работи точно по начина, който описахме. Това описание може би звучи мал-

ко объркващо, но след малко ще се убедите, че има начин, по който можете да направите *chat* скриптовете много лесни за разбиране.

Опцията *–v* указва на *chat* да записва в дневник всички действия чрез инструмента `local2` на демона *syslog*[25].

Задаването на chat скрипт от командния ред носи определен риск, защото потребителите могат да видят командния ред на процеса с командата ps. Може да избегнете този риск, като поставите chat скрипта във файл като *dial-c3po*. По този начин указвате на *chat* да чете скрипта от файла вместо от командния ред, като му задавате опцията *–f*, последвана от името на файла. Допълнителната полза от това действие е, че нашите очаквани *chat* последователности стават полесни за разбиране. За да преобразуваме горния пример, нашият файл *dial-c3po* би трябвало да изглежда по следния начин:

```
''          ATZ
OK          ATDT318714
CONNECT     ''
ogin:       ppp
word:       GaGariN
```

Когато използваме файл с chat-скрипт по този начин, низът, който очакваме да получим, е вляво, а отговорът, който ще изпратим, е вдясно. Те се четат и разбират много по-лесно, когато са представени по този начин.

Пълното заклинание на *pppd* сега ще изглежда така:

```
# pppd connect "chat -f dial-c3po" /dev/ttyS3 38400 -detach \
    crtscts modem defaultroute
```

Освен опцията `connect`, която задава скрипта за набиране, ние сме добавили още две опции към командния ред: `-detach`, която указва на *pppd* да не се отделя от конзолата и да става фонов процес, и ключовата дума `modem`, която му указва да извършва специфични за модема действия на серийното устройство като прекъсване на линията преди и след обаждането. Ако не използвате тази ключова дума, *pppd* няма да наблюдава DCD линията на порта, и следователно, няма да открие дали отдалечения край е прекъснал връзката неочаквано.

Примерите, които показахме са доста прости; *chat* позволява да се пишат доста по-сложни скриптове. Например, могат да се задават

[25] Ако редактирате *syslog.conf* да пренасочва съобщенията за дневника във файл, уверете се, че този файл не е достъпен за четене от всички, защото *chat* записва по подразбиране целия скрипт, включително паролите.

низове, при които изпълнението на скрипта да се прекрати поради грешка. Типични низове за прекратяване са съобщения като `BUSY` или `NO CARRIER`, които вашият модем обикновено генерира, когато набраният номер дава заето или не отговаря. За да укажете на *chat* да разпознава тези съобщения ведната, вместо след определен интервал от време, можете да ги зададете в началото на скрипта, използвайки ключовата дума `ABORT`:

```
$ chat -v ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ...
```

По подобен начин можете да промените стойността таймаут за части от chat скриптовете, като вмъкнете опции `TIMEOUT`.

Освен това, понякога имате нужда и от условно изпълнение на части от chat скрипта: когато не получите поканата за влизане на отдалечения край, можете да искате да изпратите BREAK или връщане на каретката (кодът от натиснат Enter – б.р.). Можете да постигнете това като добавите подскрипт към очакван низ. Подскриптът се състои от последователност от изпращани и очаквани низове, точно както целия скрипт, които са разделени с тирета. Подскриптът се изпълнява всеки път, когато очакваният низ, към който той е добавен, не се получи навреме. В горния пример ще променим chat скрипта по следния начин:

```
ogin:-BREAK-ogin: ppp ssword: GaGariN
```

Когато *chat* не получи от отдалечената система поканата за влизане, подскриптът се изпълнява като първо се изпраща BREAK, а след това огново се чака за покана за влизане. Ако сега поканата се получи, скриптът продължава изпълнението си както обикновено; в противен случай, той ще спре поради грешка.

Опции за конфигуриране на IP

За договаряне на множество IP параметри по време на конфигуриране на връзката се използва IPCP. Обикновено, всеки край на връзката изпраща пакет IPCP Configuration Request (IPCP пакет със заявка за конфигуриране), указващ кои подразбиращи се стойности иска да промени, както и новите стойности. При получаване, отдалеченият край проверява последователно всяка опция и или я потвърждава, или я отхвърля.

pppd ви дава голям контрол върху IPCP опциите, които ще се опита да договори. Можете да го настроите чрез различни опции от командния ред, които ще разгледаме в този раздел.

Избиране на IP адреси

Всички IP интерфейси изискват да им бъдат зададени IP адреси; едно PPP устройство винаги има IP адрес. Комплектът от PPP протоколи предоставя механизъм, който позволява автоматичното задаване на IP адреси на PPP интерфейси. Възможно е за PPP програмата в единия край на връзка от точка до точка да се зададе IP адрес, който да се използва от отдалечения край, или всеки край на връзката да използва свой собствен адрес.

Някои PPP сървъри, които управляват много клиентски сайтове, задават адреси динамично; адресите се задават на системи, само когато те се свързват и се освобождават при излизане. Това позволява броят на необходимите IP адреси да се ограничи до броя на телефонните линии. Макар че ограничението е удобно за мениджърите на PPP сървъра за телефонна линия, често пъти това не е толкова удобно за потребителите, които влизат в него. В Глава 6 вече разгледахме начина, по който имената на хостовете се свързват с IP адресите чрез използване на база данни. За да могат хората да се свързват към вашия хост, те трябва да знаят вашия IP адрес или свързаното с него име на хост. Ако сте потребител на PPP услуга, която динамично ви задава IP адрес, е трудно да знаете това без да се предоставят някои средства, които да позволяват базата данни DNS да бъде обновена, след като вече ви е зададен IP адрес. Такива системи съществуват, но тук няма да ги разглеждаме подробно; вместо това, ще разгледаме по-предпочитания подход, който включва възможността да можете да използвате един и същи IP адрес всеки път, когато установите връзка с мрежата[26].

В предишния пример, *pppd* набираше **c3po** и установяваше IP връзка. Не бяха взети никакви мерки за избиране на определен IP адрес за който и да е край на връзката. Вместо това, оставихме *pppd* да извърши своите действия по подразбиране. Той се опита да преобразува локалното име на хост, в нашия случай **vlager**, в IP адрес, който използва за локалния край, но позволява на отдалечената машина, **c3po**, сама да избере своя адрес. PPP поддържа няколко алтернативи на този алгоритъм.

[26] Повече информация за два механизъма за динамично задаване на хост можете да намерите на адрес *http://www.dynip.com/* и *http://www.justlinux.com/ dynamic_dns.html*

За да поискате определен адрес, обикновено давате на *pppd* следната опция:

`локален_адрес:отдалечен_адрес`

Стойностите `локален_адрес` и `отдалечен_адрес` могат да бъдат зададени с десетично-точков запис или като имена на хостове[27]. Тази опция указва на *pppd* да опита да използва първия зададен адрес като свой собствен IP адрес, а втория – като адрес на другия край на връзката. Ако другата страна отхвърли и двата адреса по време на IPCP договарянето, IP връзка няма да бъде установена[28].

Ако набирате сървър и очаквате той да ви зададе IP адрес, трябва да се уверите, че *pppd* не се опитва сам да договори такъв. За да направите това, използвайте опцията *noipdefault* и оставете полето `local_addr` празно. Опцията *noipdefault* няма да позволи на *pppd* да опита да използва като локален адрес IP адреса, свързан името на хоста.

Ако искате да зададете само локалния адрес и да приемете всеки адрес, който другият край на връзката използва, просто не задавайте стойност за полето `remote_addr`. За да направите така, че **vlager** да използва IP адрес **130.83.4.27** вместо неговия собствен адрес, от командния ред задайте *130.83.4.27*:. Аналогично, за да зададете само адреса на отдалечената страна, оставете полето `local_addr` празно. По подразбиране, *pppd* ще използва адреса, който е свързан с вашето име на хост.

Маршрутизиране през PPP връзка

След конфигурирането на мрежовия интерфейс, *pppd* обикновено ще установи само маршрут към хоста от отсрещната страна. Ако отдалеченият хост е в локална мрежа, определено ще искате да можете да се свързвате с хостове, намиращи се "зад" него; в този случай трябва да се зададе маршрут към мрежа.

---

[27] Използването на имена на хостове в тази опция има последици за удостоверяване на самоличността с CHAP. Вижте раздела "Удостоверяване на самоличността с PPP" по-долу в тази глава.

[28] Опциите *ipcp-accept-local* и *ipcp-accept-remote* инструктират *pppd* да приеме локалния и отдалечения IP адрес, предлагани от отдалечения PPP, дори ако вие сте задали адреси във вашата конфигурация. Ако тези опции не са зададени, вашия *pppd* демон ще отхвърли всеки опит да се договарят използваните IP адреси.

Вече видяхме, че *pppd* може да зададе подразбиращ се маршрут, използвайки опцията *defaultroute*. Тази опция е много полезна, ако PPP сървърът, който набирате, работи като шлюз към Интернет.

Обратният случай, в който вашата система работи като шлюз за един единствен хост, също е сравнително лесно да се направи. Например, вземете някой работник от Виртуалната пивоварна, чиято машина се нарича **oneshot**. Да предположим още, че сме конфигурирали **vlager** като PPP сървър за набиране. Ако сме конфигурирали **vlager** да задава динамично IP адреси, които принадлежат на подмрежата на Пивоварната, тогава можем да използваме опцията *proxyarp* на *pppd*, която ще инсталира прокси ARP вход за **oneshot**. Това автоматично прави **oneshot** достъпен от всички хостове на Пивоварната и Винарната.

Нещата обаче не винаги са толкова прости. Свързването на две локални мрежи обикновено изисква добавяне на специфичен мрежов маршрут, тъй като тези мрежи може да имат свои собствени подразбиращи се маршрути. Освен това, когато и двете страни използват PPP връзката като подразбиращ се маршрут, ще се генерира цикъл, чрез който пакетите до неизвестни местоназначения ще отскачат между двата края на връзката, докато не изтече времето на тяхното съществуване.

Да предположим, че Виртуалната Пивоварна отваря филиал в друг град. Той има собствена Ethernet мрежа, използваща IP мрежов номер **172.16.3.0**, която е подмрежа 3 на мрежата от клас B на Пивоварната. Този филиал иска да се свърже с мрежата на Пивоварната чрез протокола PPP, за да обнови базата данни за клиентите. **Vlager** отново работи като шлюз за мрежата на Пивоварната и ще поддържа PPP връзката; другият край на връзката при новия филиал се нарича **vbourbon** и има IP адрес **172.16.3.1.** Тази мрежа е илюстрирана на Фигура A-2 в Приложение A, *Примерна мрежа: Виртуална пивоварна фабрика*.

Когато **vbourbon** се свързва с **vlager**, той указва насочва подразбиращия се маршрут към **vlager** както обикновено. При **vlager** обаче ще имаме само маршрут от точка до точка към **vbourbon** и ще трябва специално да конфигурираме мрежов маршрут за подмрежа 3, която използва **vbourbon** свой шлюз. Можем да направим това ръчно, използвайки командата *route*, след като PPP връзката е създадена, но това не е много практично решение. За щастие, можем да конфигурираме маршрута автоматично като използваме една възможност на *pppd*, която все още не сме разглеждали – командата *ip-up*. Тази команда е shell-скрипт или програма, намираща се *в /etc/ppp*, която се

изпълнява от *pppd*, след като е конфигуриран PPP интерфейса. Когато съществува, тя се извиква със следните параметри:

```
ip-up iface device speed local_addr remote_addr
```

Следващата таблица обобщава значението на всеки от аргументите (в първата колона показваме номера, който се използва от shell-скрипта за обръщане към всеки аргумент):

Аргумент	Име	Цел
$1	*iface*	Използвания мрежов интерфейс, например, ppp0.
$2	*device*	Името на пътя до използвания файл на серийното устройство *(/dev/tty*, ако се използват stdin/stdout).
$3	*speed*	Скоростта на серийното устройство в битове за секунда.
$4	*local_addr*	IP адресът на отдалечения край на връзката, представен в точкуван четиризначен запис.
$5	*remote_addr*	IP адресът на отдалечения край на връзката, представен в точкуван четиризначен запис.

В нашия случай, *ip-up* скриптът може да съдържа следния фрагмент от код:

```
#!/bin/sh
case $5 in
172.16.3.1)     # this is vbourbon
   route add -net 172.16.3.0 gw 172.16.3.1;;
...
esac
exit 0
```

По подобен начин, */etc/ppp/ip-down* може да се използва за отменяне на всякакви действия на *ip-up*, след като PPP връзката отново е била освободена. Следователно, в нашия */etc/ppp/ip-down* скрипт би трябвало да имаме команда route, която е премахнала маршрута, който създадохме в */etc/ppp/ip-up* скрипта.

Схемата за маршрутизация обаче все още не е завършена. Създадохме записи за таблицата с маршрутите на двата PPP хоста, но до този момент нито един от хостовете на една от двете мрежи не знае нищо за PPP връзката. Това не е голям проблем, ако всички хостове в мрежата на филиала имат свой подразбиращ се маршрут, сочещ към **vbourbon**, а всички хостове на Пивоварната са насочени към **vlager** по подразбиране. Ако случаят обаче не е такъв, единствената ви възможност е да използвате демон за маршрутизиране като *gated*. След създаването на мрежов маршрут на **vlager**, демонът за маршрутизиране изпраща новия маршрут към всички хостове на прикачените подмрежи.

Опции за управление на връзката

Вече споменахме за протокола LCP (Link Control Protocol), който се използва за договаряне на характеристики на връзката и тестване на връзка.

Двете най-важни опции, които се договорят от LCP, са *Asynchronous Control Character Map* и *Maximum Receive Unit*. Съществуват множество други опции за конфигуриране на LCP, но те са твърде специализирани, за да разглеждаме тук.

Опцията Asynchronous Control Character Map, разговорно наричана *async map*, се използва за асинхронни връзки, като телефонни линии, за идентифициране на контролни символи, които трябва да бъдат кодирани (заместени от двусимволна последователност), за да не се допусне да бъдат интерпретирани от устройството, използвано за създаване на връзката. Например, можете да искате да извършите escape (премахване на специалното значение) на символите XON и XOFF, използвани при договаряне на софтуера, тъй като един неправилно конфигуриран модем може да се затрудни при получаване на XOFF. Други възможности включват Ctrl-l (escape-символът за *telnet*). PPP ви позволява да извършите escape на всички символи с ASCII код от 0 до 31, като ги укажете в async map.

async map е 32-битово растерно изображение, изразено в шестнадесетична форма. Най-младшият бит съответства на ASCII символа NULL, а най-старшият бит съответства на ASCII символа 31 десетично. Тези 32 ASCII символа не са контролните символи. Ако бит е зададен в растерното изображение, той сигнализира, че за съответния символ трябва да се извърши escape преди да се предаде през връзката.

За да укажете на другия край на връзката, че не трябва да извършва escape на всички контролни символи, а само на някои от тях, можете да зададете async map на *pppd*, използвайки опцията *asyncmap*. Например, ако само за ^S и ^Q (ASCII символите 17 и 19, обикновено се използват вместо XON и XOFF) трябва да извърши escape, използвайте следващата опция:

```
asyncmap 0x00A00
```

Преобразуването толкова просто, че можете да преминете от двоична в шестнадесетична бройна система. Разпростерете 32 бита пред вас. Най-десният бит отдясно съответства на ASCII символа 00 (NULL), а най-левият съответства на ASCII символа 32 десетично. Установете битовете, съответстващи на символите, за които искате да извършите escape, в единица, а всички останали в нула. За да преобразувате всичко това в шестнадесетично число, каквото *pppd* очаква, просто вземете поотделно всеки четирибита и ги превърнете в шестнадесетична бройна система. Накрая трябва да получите осем шестнадесетични цифри. Обединете ги всички в един низ и прибавете отпред "0x", за да означите, че това е шестнадесетично число и сте готови.

Първоначално, async map е установена на `0xffffffff`, което означава, че ще се извърши escape за всички контролни символи. Това е сигурна стойност по подразбиране, но обикновено е повече, отколкото ви е необходимо. Всеки символ, който се среща в async map има като последица изпращането на два символа през връзката, така че премахването на специалното значение става с цената на повишено използване на връзката и съответстващото на това намаляване на производителността.

В повечето случаи, async map установена на `0x0`, върши добра работа. При нея не се извършва escape.

MRU (Maximum Receive Unit – максимална дължина на единица за приемане), сигнализира на другия край на връзката максималния размер на HDLC кадрите, които искаме да получим. Въпреки че това може да ви напомня за MTU (Maximum Transfer Unit – максимална дължина на единица за прехвърляне), тези две понятия нямат много общи неща. MTU е параметър на мрежовото устройство на ядрото и определя максималния размер на кадъра, който интерфейсът може да предаде. MRU е повече като съвет към отдалечения край да не генерира кадри, които са по-големи от MRU; въпреки всичко интерфейсът трябва да може да получава кадри с размер до 1,500 байта.

Следователно, избирането на MRU не е толкова въпрос на това, какво връзката може да прехвърля, а какво ви дава най-добра пропускателна способност. Ако възнамерявате да изпълнявате интерактивни приложения по връзката, задаването на ниски стойности на MRU от порядъка на 296 е добра идея, така че някой случаен по-голям пакет (да речем от FTP сесия) няма да накара курсора ви "да подскочи". За да укажете на *pppd* да изиска стойност на MRU 296, трябва да му дадете опцията `mru 296`. Малките стойности на MRU обаче имат смисъл, само ако имате VJ компресия на заглавието (тя е разрешена по подразбиране), тъй като в противен случай ще изгубите голяма част пропускателната способност, пренасяйки само IP заглавието на всяка дейтаграма.

Освен това, *pppd* разпознава и няколко опции на LCP, които конфигурират цялостното поведение на процеса на договаряне, например, максималния брой на заявките за конфигуриране, които могат да бъдат обменени преди връзката да се прекъсне. Ако не знаете какво точно правите, трябва да оставите тези опции непроменени.

И накрая, съществуват две опции, които се прилагат към LCP съобщения за *echo*. *PPP* дефинира две съобщения, *Echo Request* и *Echo Response*. *pppd* използва тази възможност за проверка дали дадена връзка все още работи. Можете да я разрешите като използвате опцията *lcp-echo-interval* заедно с време в секунди. Ако в рамките на този интервал не се получат никакви кадри от отдалечения хост, *pppd* генерира Echo Request и очаква другата страна да върне Echo Response. Ако тя не генерира отговор, след изпращането на определен брой заявки връзката се прекъсва. Този брой може да се зададе посредством опцията *lcp-echo-failure*. По подразбиране, тази възможност е изцяло забранена.

Основни съображения за сигурност

Неправилно конфигуриран PPP демон може да бъде пагубен пробив в сигурността. Това може да бъде толкова опасно, колкото да позволите на някого да си закачи машината към вашата Ethernet мрежа (а това може да бъде много лошо). В този раздел ще разгледаме някои мерки, които би трябвало да направят вашата конфигурацията на PPP сигурна.

Привилегия на root се изисква, за да се конфигурира мрежовото устройство и таблицата с маршрутите. Обикновено, ще се справите с това като стартирате *pppd* със setuid **root**. Все пак, *pppd* позволява на потребителите да задават различни опции, свързани със сигурността.

За да се защити против атаки, един потребител може да започне като управлява опциите на *pppd*, така че вие трябва да зададете няколко подразбиращи се стойности в глобалния файл */etc/ppp/options*, като тези, показани в примерния файл в раздела "Използване на файлове с опции", разгледан по-рано в тази глава. Някои от тях, като опциите за удостоверяване на самоличността, не могат да се отменят от потребителите и по този начин предоставят сигурна защита срещу манипулации. Важна опция за защита е опцията `connect`. Ако възнамерявате да позволите на някой от потребителите без `root` привилегии да извикват *pppd*, за да се свързват към Интернет, винаги трябва да добавяте опциите `connect` и `noauth` в глобалния файл с опции */etc/ppp/options*. Ако не успеете да направите това, потребителите ще могат да изпълнят произволни команди с привилегии на `root` като просто зададат командата като тяхна `connect` команда на *pppd* или във файла с техните лични опции.

Друга добра идея е да ограничите потребителите, които могат да изпълняват *pppd*, като създадете група в */etc/group* и добавите в нея само потребителите, които искате да имат възможност да изпълняват PPP демона. След това трябва да смените правото на собственост на групата на *pppd* демона с тази група и да премахнете привилегиите за изпълнение, които може да има всеки потребител. Ако предположим, че сте нарекли групата си **dialout**, за да направите това, можете да използвате нещо подобно:

```
# chown root /usr/sbin/pppd
# chgrp dialout /usr/sbin/pppd
# chmod 4750 /usr/sbin/pppd
```

Разбира се, трябва да се защитите и от системите, с които комуникирате посредством PPP. За да избегнете хостове, представящи се като някой друг, трябва винаги да изисквате някакъв вид удостоверяване на самоличността от другата страна на връзката. Като допълнение, не трябва да позволявате на чуждите хостове да използват всеки IP адрес, който изберат, а да ги ограничите най-много до няколко. В следващия раздел ще разгледаме тези теми в подробности.

Удостоверяване на самоличността с PPP

С PPP всяка система може да изисква от другата страна да удостовери самоличността си, използвайки един от двата протокола за тази цел: PAP (*Password Authentication Protocol*) и CHAP (*Challenge Handshake Authentication Protocol*). Когато е създадена връзка, всяка от страните може да изиска от другата да удостовери самоличността си, без значение далитя извикващата или викана страна. В описанието, което следва, свободно ще говорим за "клиент" и "сървър", когато искаме да направим разлика между системата, която изпраща заявка за удостоверяване на самоличността, и системата, която им отговаря. PPP демон може да поиска от другия край на връзката да удостовери самоличността си като изпрати заявка за конфигуриране на LCP, идентифицирайки желания протокол за проверка удостоверяване на самоличността.

PAP срещу CHAP

PAP, който се предлага от много доставчици на Интернет услуги, по същество работи по същия начин като нормалната процедура за влизане. Клиентът удостоверява самоличността си като изпраща потребителско име и (криптирана по избор) парола към сървъра, която сървърът сравнява с неговата база данни с тайни. Тази техника е уязвима за подслушвачи, които може да се опитат да получат паролата чрез прослушване на серийната линия и да повтарят изпробването за грешки.

CHAP няма тези недостатъци. С този протокол сървърът изпраща произволно генериран низ със запитване към клиента, заедно с неговото име на хост. Клиентът използва името на хоста за търсене на подходящата тайна, комбинира го низа със запитване към клиента и криптира низа, използвайки еднопосочна функция за хеширане. Резултатът се връща на сървъра заедно с името на хоста на клиента. Сега сървърът извършва същото изчисление и изпраща потвърждение на клиента, ако то пристига в същия резултат.

Освен това, CHAP не изисква клиентът да удостоверява самоличността си само при стартиране, а изпраща низове със запитвания на определени интервали, за да се увери, че клиентът не е бил заменен от нарушител, например, чрез превключване на телефонните линии или поради грешка при конфигурирането на модема, поради която

PPP демона не може да забележи, че първоначалната телефонна връзка се е разпаднала и някой друг е набрал номера.

pppd пази тайните ключове за PAP и CHAP в два отделни файла, наречени */etc/ppp/pap-secrets* и */etc/ppp/chap-secrets*. Чрез въвеждане на отдалечения хост в единия или другия файл, ще имате фин контрол върху това кой от двата протокола PAP или CHAP се използва, за да удостоверите самоличността си пред другата страна на връзката или обратното.

По подразбиране, *pppd* не изисква удостоверяване на самоличността от отдалечения хост, но ще се съгласи да удостовери своята самоличност, когато това се изисква от отдалечения хост. Тъй като протокольт CHAP е много по-мощен от PAP, *pppd* се опитва да го използва винаги, когато е възможно. Ако другият край на връзката не го поддържа или ако *pppd* не може да намери CHAP тайна за отдалечената система в своя файл *chap-secrets*, той се обръща към PAP. Ако и за другия край на връзката той няма PAP тайна, напълно отказва да удостовери самоличността си. Като последствие от това, връзката се прекратява.

Можете да промените това поведение по няколко начина. Когато е зададена ключовата дума *auth*, *pppd* изисква от другия край на връзката да удостовери самоличността си. *pppd* се договаря да използва или CHAP, или PAP доколкова, доколкото има тайна за другия край на връзката в своята CHAP или PAP база данни. Съществуват други опции за включване или изключване на определен протокол за удостоверяване на самоличността, но тук няма да ги описвам.

Ако всички системи, с които комуникирате посредством PPP, се договорят да удостоверяват самоличността си, ще трябва да поставите опцията *auth* в глобалния файл */etc/ppp/options* и да дефинирате пароли за всяка система във файла *chap-secrets*. Ако дадена система не поддържа CHAP, добавете запис за нея във файла с *pap-secrets* тайни. По този начин можете да сте сигурни, че система, която не е удостоверила самоличността си, няма да се свърже към вашия хост.

В следващите два раздела са разгледани двата PPP файла с тайни, *pap-secrets* и *chap-secrets*. Те се намират в */etc/ppp* и съдържат групи от тройки, включващи клиент, сървър и парола, които по избор може да са последвани от списък с IP адреси. Интерпретацията на полетата за клиента и сървъра е различна за CHAP и PAP, и освен това, зависи от това дали удостоверяваме самоличността си пред отсрещната страна или дали изискваме сървърът да направи това пред нас.

CHAP файл с тайни

Когато трябва да удостовери своята самоличност пред сървър посредством CHAP, *pppd* претърсва файла *chap-secrets* за запис, в който полето клиент да е равно на локалното име на хост, а полето сървър да съвпада бъде равно на отдалеченото име на хост, изпратено в CHAP запитване. Когато се изисква другата страна на връзката да удостовери самоличността си, ролите просто се разменят: тогава *pppd* търси запис, в който полето за клиента е равно на името на отдалечения хост (което се изпраща в CHAP отговора на клиента), а полето за сървъра е равно на името на локалния хост име.

Следва примерен файл *chap-secrets* за **vlager**:

```
# CHAP тайни за vlager.vbrew.com
#
# клиент            сървър           тайна                  адрес
#-------------------------------------------------------------------------
vlager.vbrew.com   c3po.lucas.com   "Use The Source Luke"  vlager.vbrew.com
c3po.lucas.com     vlager.vbrew.com "arttoo! arttoo!"      c3po.lucas.com
*                  vlager.vbrew.com "TuXdrinksVicBitter"   pub.vbrew.com
```

Когато **vlager** създава PPP връзка с **c3po**, **c3po** изисква от **vlager** да удостовери самоличността си, изпращайки CHAP запитване. След това *pppd* на **vlager** сканира *chap-secrets* за запис, в който полето за клиента е равно на **vlager.vbrew.com**, а полето за сървъра е равно на **c3po. lucas. com**, и открива първия ред, показан в примера. След това генерира CHAP отговора от низа със запитването и тайната (`Use The Source Luke`) и го изпраща към **c3po**.

Освен това, *pppd* съставя CHAP запитване за c3po, съдържащо уникален низ със запитване и неговото пълно квалифицирано име на хост, **vlager. vbrew.com**. **c3po** генерира CHAP отговор по начина, който разгледахме, и го връща към **vlager**. След това *pppd* извлича името на хоста на клиента (**c3po.vbrew.com**) от отговора и претърсва файла *chap-secrets* за ред, съвпадащ с **c3po** като клиент и **vlager** като сървър. Вторият ред удовлетворява това условие, така че *pppd* комбинира CHAP запитването със тайната `arttoo! arttoo!`, криптира ги и сравнява резултата с CHAP отговора на **c3po**.

Четвъртото поле, което не е задължително, изброява IP адресите, които са приемливи за клиента, чието име е посочено в първото поле. Адресите могат да бъдат дадени като точкуван четиризначен запис или като имена на хостове, за които резолвера извършва търсене. Например, ако **c3po** иска да използва IP адрес по време на IPCP договарянето, който не е в този списък, заявката се отхвърля, а IPCP се

прекратява. Следователно, в показания по-горе примерен файл **c3po** е ограничен до ползването на своя собствен IP адрес. Ако полето с адресите е празно, са позволени всякакви адреси; стойност "-" въобще не позволява използването на IP за този клиент.

Третият ред от примерния файл *chap-secrets* позволява всеки хост да установи PPP връзка с **vlager**, тъй като стойността * на полетата клиент или сървър е универсален символ, съответстващ на всяко име на хост. Единствените изисквания са свързващия се хост да знае тайната и да използва IP адреса, свързан с **pub.vbrew.com**. Записи с произволни имена на хостове може да се срещат навсякъде във файла със тайни, тъй като *pppd* винаги ще използва най-доброто съответствие, което може да намери, за двойката клиент/сървър.

pppd може да се нуждае от някаква помощ при формиране на имената на хостове. Както обяснихме преди, отдалеченото име на хост винаги се доставя от другия край на връзката в CHAP запитването или пакета с отговора. Локалното име на хост се получава чрез извикване на функцията *gethostname(2)* по подразбиране. Ако сте задали името на системата на вашето неквалифицирано име на хост, трябва да зададете на *pppd* името на домейна, използвайки опцията *domain*:

```
# pppd ... domain vbrew.com
```

Това задаване добавя името на домейна на Brewery към **vlager** за всички действия, свързани с удостоверяването на самоличността. Други опции, които променят идеята на *pppd* за локалното име на хост са *usehostname* и *name*. Когато давате локалния IP адрес на командния ред, използвайки `local:remote` и `local` като име вместо точкуван четиризначен запис, *pppd* използва това като локално име на хост.

Файл с PAP тайни

Файлът с PAP тайни е много подобен на този на CHAP. Първите две полета винаги съдържат потребителско име и име на сървър; третото поле съдържа PAP тайната. Когато отдалеченият хост изпраща своята информация за удостоверяване на самоличността, *pppd* използва записа, в който полето за сървъра е равно на локалното име на хост, а полето за потребителя е равно на потребителското име, изпратено в заявката. Когато ние е необходимо да изпратим нашите credentials на другата страна, *pppd* използва тайната, при който полето за потребителя е равно на локалното потребителско име, а полето за сървъра е равно на отдалеченото име на хост.

Един примерен файл с PAP тайни изглежда така:

```
# /etc/ppp/pap-secrets
#
# потребител      сървър      тайна          адрес
vlager-pap        c3po        cresspahl      vlager.vbrew.com
c3po              vlager      DonaldGNUth    c3po.lucas.com
```

Първият ред се използва за удостоверяване на нашата самоличност, когато комуникираме с **c3po**. Вторият ред описва начина, по който потребител с име **c3po** трябва да удостовери самоличността си пред нас.

Името `vlager-pap` в първата колона е потребителското име, което изпращаме към **c3po**. По подразбиране, *pppd* взима локалното име на хост като потребителско име, но можете да укажете и различно име като зададете опцията *user*, последвана от това име.

Когато взима запис от файла *pap-secrets*, за да ни идентифицира пред отдалечения хост, *pppd* трябва да знае името на отдалечения хост. Тъй като няма начин да го открие, трябва да го зададете в командния ред, използвайки ключовата дума *remotename*, последвана от името на хоста на другия край на връзката. За да използваме горния запис за удостоверяване на самоличността пред **c3po**, например, трябва да добавим следващата опция към командния ред на *pppd*:

```
# pppd ... remotename c3po user vlager-pap
```

В четвъртото поле на файла с PAP тайни (и всички следващи полета), можете да укажете какви IP адреси са позволени за този определен хост, също както във файла с CHAP тайни. На другата страна на връзката ще бъде позволено да заявява само адреси, намиращи се в този списък. В примерния файл записът, който **c3po** ще използва, когато набира – редът, в който **c3po** е клиента – му позволява да използва само своя реален IP адрес и никакъв друг адрес.

Забележете, че PAP е доста слаб метод за удостоверяване на самоличността, така че вместо него трябва да използвате CHAP винаги, когато това е възможно. Затова тук няма да разглеждаме PAP с по-големи подробности; ако се интересувате от използването му, повече възможности на PAP ще намерите в справочната страница *pppd(8)*.

Дебъгване на инсталацията на PPP

По подразбиране, *pppd* записва в дневник всякакви предупреждения и съобщения за грешки в средството `deamon` на *syslog*. Трябва да добавите запис в *syslog.conf*, който пренасочва тези съобщения към файл или дори към конзолата; в противен случай, *syslog* просто ги отхвърля. Следващият запис изпраща всички съобщения към */var/log/ppp-log*:

```
daemon.*      /var/log/ppp-log
```

Ако вашата PPP инсталация не работи както трябва, трябва да погледнете в този log-файл. Ако log-съобщенията не ви помогнат, можете да включите допълнително дебъване, използвайки опцията *debug*. Този резултат указва на *pppd* да записва в дневник съдържанието на всички контролни пакети, изпратени от данни или получени от *syslog*. Тогава всички съобщения тогава отиват към средството на `deamon`.

Накрая, най-драстичният начин да се провери един проблем е да се активира дебъване на ниво ядро чрез извикване на *pppd* с опцията *kdebug*. Тази опция е следва на числов аргумент, който е сума от следните стойности: 1 за общи дебъг съобщения, 2 за отпечатване на съдържанието на всички входящи HDLC кадри и 4 за указване на драйвера да отпечатва всички изходящи HDLC кадри. За да прехващате дебъг съобщенията на ядрото, трябва или да стартирате *syslogd* демон, който чете от файла */proc/kmsg*, или демона *klogd*. Всеки от тях пренасочва дебъването на ядрото към *syslog* `kernel` facility.

По-разширени конфигурации на PPP

Тъй като конфигурирането на PPP за набиране в мрежа като Интернет е най-често срещаното приложение, някои от вас имат по-разширени изисквания. В този раздел ще разгледаме някои от по-разширените възможни конфигурации на PPP под Linux.

PPP сървър

Стартирането на *pppd* като сървър е просто въпрос на конфигуриране на серийно tty устройство за извикване на *pppd* с подходящи опции, когато се получи извикване на входящи данни. Един от начините да се направи това е да се създаде специален акаунт, да речем **ppp**, и да

му се зададе скрипт или програма като shell за влизане, който извиква *pppd* с тези опции. Друга възможност, ако имате намерение да поддържате удостоверяване на самоличността с PAP или CHAP, е че можете да използвате програмата *mgetty* да поддържа вашия модем и да използва неговата възможност "/AutoPPP/".

За да изградите сървър, използвайки метод за влизане, трябва да добавите подобен ред към вашия файл */etc/passwd*:

```
ppp:x:500:200:Public PPP Account:/tmp:/etc/ppp/ppplogin
```

Ако системата ви поддържа скрити пароли, трябва да добавите и запис към файла */etc/shadow*:

```
ppp:!:10913:0:99999:7:::
```

Разбира се, използваните от вас UID и GID зависят от това кой потребител искате да притежава връзката и как сте го създали. Освен това, трябва да зададете паролата за споменатия акаунт, използвайки командата *passwd*.

Скриптът *ppplogin* може да изглежда по следния начин:

```
# !/bin/sh
# ppplogin - скрипт, който стартира pppd при влизане
mesg n
stty -echo
exec pppd -detach silent modem crtscts
```

Командата *mesg* забранява на други потребители да пишат към tty, използвайки, например, командата *write*. Командата *stty* изключва echoing на символи. Тази команда е необходима; в противен случай, всичко, което другата страна на връзката изпраща ще се връща отново към нея. Най-важната опция на *pppd* е `–detach`, тъй като тя предпазва *pppd* от отделяне от контролиращия tty. Ако не сме задали тази опция, той ще отиде на заден план, указвайки на shell-скрипта да излезе. От своя страна, това ще доведе до прекъсване на серийната линия и до разпадане на връзката. Опцията *silent* указва на *pppd* да чака докато не получи пакет от извикващата система преди той да започне да изпраща. Тази опция предпазва появата на таймаути при предаване, когато викащата система е бавна при стартиране на своя PPP клиент. Опцията `modem` указва на модема да контролира линиите на серийния порт. Винаги, когато използвате *pppd* с модем, трябва да включвате тази опция. Опцията *crtscts* включва договаряне на хардуера.

Освен тези опции, може би ще искате да наложите някакъв вид удостоверяване на самоличността, например, като зададете *auth* от командния ред на *pppd* или в глобалния файл с опции. Справочната страница разглежда по-специфични опции за включване и изключване на отделни протоколи за удостоверяване на самоличността.

Ако искате да използвате *mgetty*, всичко, което трябва да направите, е да конфигурирате *mgetty* да поддържа серийното устройство, към което е свързан вашия модем (за подробности вижте "Конфигуриране на демона mgetty"), да конфигурирате *pppd* за удостоверяване на самоличността чрез PAP или CHAP с подходящи опции в неговия файл с опции, и накрая, да добавите към своя файл */etc/mgetty/login.config* секция, подобна на следната:

```
# Конфигурираме mgetty да открива автоматично входящите PPP извиквания
# и да извиква демона pppd да управлява (манипулира) връзката.
#
/AutoPPP/ -     ppp   /usr/sbin/pppd auth -chap +pap login
```

Първото поле е "специална магия", която се използва за установяване, че едно входящо извикване е PPP извикване. Не трябва да променяте размера на буквите в този низ; той е чувствителен в това отношение. Третата колона е потребителското име, което се появява в *who* списъците, когато някой е влязъл. Останалата част от реда е командата, която се извиква. В нашия пример, се уверихме, че се изисква удостоверяване на самоличността с PAP, забранихме CHAP и указахме, че системният файл *passwd* ще се използва за удостоверяване на самоличността на потребителите. Това вероятно е подобно на това, което вие ще искате. Запомнете, че можете да задавате опциите във файла *options* или от командния ред, ако предпочитате.

Следва малък списък със задачите, които трябва да се изпълнят, и последователността, която трябва да спазвате при изпълнението им, за да можете да укажете на PPP набирането да работи на вашата машина. Уверете се, че всяка от стъпките работи, преди да се преминете към следващата:

1. Конфигурирайте модема за режим на автоматично отговаряне. За Hayes-съвместими модеми това се прави, използвайки команда като `ATS0=3`. Ако ще използвате демона *mgetty*, това не е необходимо.
2. Конфигурирайте серийното устройство с команда от тип *getty*, за да отговаряте на входящите повиквания. Често използван вариант на *getty* е *mgetty*.

3. Обмислете начин за удостоверяване на самоличността. Дали викащите страни ще удостоверят самоличността си, използвайки PAP, CHAP или системно влизане?
4. Конфигурирайте *pppd* като сървър, както беше описано в този раздел.
5. Обмислете начин за маршрутизация. Ще ви се наложи ли да предоставяте мрежов маршрут към извикващите страни? Маршрутизирането може да бъде извършено, използвайки скрипта *ip-up*.

Набиране по желание

Когато има IP трафик, който трябва да бъде пренесен през връзката, набирането по желание указва на вашия телефонен модем да набира и създава връзка с отдалечен хост. Такова набиране е най-удобно, когато не можете да оставите телефонната си линия постоянно включена към вашия Интернет доставчик. Например, може би ще трябва да заплащате локалните разговори за време, така че може окаже по-евтино, ако телефонната линия е включена, само когато имате нужда от това и се прекъсва, когато не използвате Интернет.

Традиционните решения на Linux използваха командата *diald*, която работи добре, но е доста за конфигуриране. Версия 2.3.0 на PPP демона и по-късните версии имат вградена поддръжка за набиране по желание и при тях конфигурирането на тази команда е просто. За да работи това обаче, трябва да използвате ново ядро. Всяко ядро след 2.0 ще работи без проблеми.

За да конфигурирате *pppd* за набиране по желание, трябва само да добавите опции към вашия файл с опции или в командния ред на *pppd*. Следващата таблица обобщава опциите, свързани с набирането по желание:

Опция	**Описание**
`demand`	Тази опция указва, че PPP връзката трябва да бъде поставена в режим на набиране по желание. PPP мрежовото устройство ще бъде създадено, но командата `connect` няма да бъде използвана, докато не се изпрати дейтаграма от локалния хост. Тази опция е задължителна за работата на набирането по желание.

Опция	Описание
`active-filter` *`expression`*	Тази опция ви позволява да укажете кои пакети от данни да се считат за активен трафик. Всеки трафик, който съответства на зададеното правило, ще рестартира demand dial idle timer, като по този начин гарантира, че *pppd* ще изчака отново преди да затвори връзката. Синтаксисът на филтъра е взет от командата *tcpdump*. Подразбиращият се филтър съответства на всички дейтаграми.
`holdoff` *`n`*	Тази опция ви позволява да зададете минималното време в секунди, което трябва да се изчака, преди отново да се направи опит за свързване, ако тази връзка се прекрати. Ако връзката се разпадне докато *pppd* счита, че тя се използва активно, връзката ще бъде създадена отново след изтичане на този интервал от време. Този интервал не се прилага при повторно свързване след празен таймаут.
`idle` *`n`*	Ако тази опция е конфигурирана, *pppd* ще прекратява връзката, винаги когато изтече този интервал от време. Това време се задава в секунди. Всеки нов активен пакет от данни ще върне установи брояча в изходно положение.

Следователно, една проста конфигурация за набиране по желание ще изглежда по следния начин:

```
demand
holdoff 60
idle 180
```

Тази конфигурация ще разреши набиране по желание, ще изчака 60 секунди преди да установи отново прекъсната връзка и ще прекрати връзката след изтичане на 180 секунди, без да се предадат някакви активни данни през връзката.

Постоянно набиране

Постоянно набиране е това, което хора, които имат постоянни връзки за набиране, ще искат да използват. Съществува малка разлика между набирането по желание и постоянното набиране. При постоянното набиране връзката се създава автоматично веднага щом се стартира PPP демона, а постоянният аспект се проявява, когато телефонното обаждане, поддържащо връзката прекъсне. Постоянното набиране гарантира, че връзката ще съществува винаги чрез автоматично ѝ възстановяване, ако тя се разпадне.

Може да имате късмет и да не трябва да си плащате телефонните обаждания; може би те са локални и безплатни или се плащат от вашата компания. В тази ситуация, възможността за постоянно набиране е изключително полезна. Ако трябва обаче да плащате телефонните си обаждания, трябва да бъдете малко по-внимателни. Ако плащате телефонните си разговори в зависимост от времето, почти сигурно е, че постоянното набиране не е това, което ви трябва, освен ако не сте напълно сигурни, че ще използвате връзката плътно по двадесет и четири часа на ден. Ако плащате за обаждания, които не се заплащат в зависимост от времето, трябва да се защитите от ситуации, които могат да укажат на модема ви да набира постоянно. *pppd* демонът предоставя опция, която доведе до намаляване на ефекта от този проблем.

За да разрешите постоянно набиране, трябва да включите опцията `persist` в един от вашите *pppd* файлове с опции. Включването само на тази опция е всичко, което трябва да направите, за да може *pppd* автоматично да извика командата, определена от опцията `connect` за възстановяване на връзката, когато тя се разпадне. Ако се притеснявате, че модемът ще набира твърде бързо (в случай на неизправност на модема или сървъра на другия край на връзката), можете да използвате опцията `holdoff`, за да определите минималното време, което *pppd* трябва да изчака преди да направи опит за повторно свързване. Тази опция няма да реши проблема за неизправност, която ви струва пари в пропиляни телефонни разговори, но поне ще успее да намали ефекта му.

Типичната конфигурация може да има опции за постоянно набиране, които изглеждат по следния начин:

```
persist
holdoff 600
```

Времето, което се изчаква преди да се установи повторно връзката, се задава в секунди. В нашия пример *pppd* изчаква цели пет минути преди да повторното набиране, след като обаждането е било прекъснато.

Възможно е да се комбинират постоянно набиране и набиране по желание, използвайки `idle` за прекъсване на връзката, ако тя не се използва определен период от време. Съмняваме се, че много потребителище са съгласни да направят това, но ако искате да разгледате такава ситуация, можете да използвате справочната страница на *pppd*.

ГЛАВА 9

TCP/IP ЗАЩИТНА СТЕНА

Сигурността става все по-важна както за компаниите, така и за отделните хора. Интернет им предостави мощно средство за разпространение на своя и получаване на информация от други компании, но освен това ги изложи на опасности, каквито по-рано не ги заплашваха. Компютърните престъпления, кражбата на информация и злонамерените вреди са реално съществуващи опасности.

Един неупълномощен и безскрупулен човек, който е получил достъп до компютърна система, може да отгатне системни пароли или да използва грешките и специфичното поведение на определени програми, за да получи работещ акаунт на тази машина. След като получи възможност да влиза в машината, той би могъл да използва информация, с която да навреди на компанията, например поверителна търговска информация като пазарни планове, подробности за нови проекти или бази данни с клиентите на фирмата. Повреждането или изменението на данни от този тип може да причини сериозни загуби на компанията.

Най-сигурният начин за избягване на тези широко разпространени вреди е да се попречи на неупълномощените лица да получат достъп до машината през мрежата. Точно това е ролята на защитните стени.

Изграждането на сигурни защитни стени е изкуство. То изисква добро разбиране на технологията, но също толкова важно е, че изисква разбиране на философията, стояща зад идеята за защитните стени. В тази книга не можем да обхванем всичко, което бихте искали да знаете. Горещо ви препоръчваме да направите допълнително проучване преди да се доверите на архитектурата на дадена защитна стена, включително и на тези, които предоставяме тук.

Съществуват достатъчно материали за конфигуриране и проектиране на защитни стени, които могат да запълнят цяла книга, и всъщност, има няколко добри източника, които можете да прочетете, за да разширите своите познания по въпроса. Два от тях са:

Building Internet Firewalls

от D. Chapman и E. Zwicky (издание на O'Reilly). Ръководство, обясняващо как се проектират и внедряват защитни стени за Unix, Linux и Windows NT и как да конфигурираме Интернет услугите да работят със защитни стени.

Firewalls and Internet Security (Защитни стени и Интернет сигурност)

от W. Cheswick и S. Bellovin (издание на Addison Wesley). Тази книга обхваща философията на проектиране на защитна стена и нейното реализиране.

В тази глава ще се съсредоточим върху специфичните за Linux технически въпроси. По-късно ще представим примерна конфигурация на защитна стена, която да послужи като помощна отправна точка за ваша собствена конфигурация, но като всички свързани със сигурността неща – не се доверявайте на никого. Два пъти проверете проекта, уверете се, че го разбирате, и след това го променете така, че да съответства на вашите изисквания. За да бъдете защитени, бъдете сигурни.

Методи на атака

Като мрежов администратор, за вас е важно да разберете характера на потенциалните атаки, насочени срещу сигурността на компютъра. Ще опишем накратко най-важните видове атаки, за да можете по-добре да разберете от какво именно ще ви предпазва защитната стена

за IP под Linux. Трябва да прочетете и други материали, за да сте сигурни, че сте в състояние да защитите своята мрежа от други видове атаки. Следват някои от най-важните методи за атака и начините да се предпазите от тях.

Непозволен достъп

Това просто означава, че хора, които не би трябвало да използват услугите на вашия компютър, могат да се свързват с него и да използват тези услуги. Например, хора извън вашата компания могат да се опитат да се свържат със счетоводната машина на вашата фирма или с вашия NFS сървър.

Съществуват редица начини за избягване на тази атака чрез внимателно определяне на това, кой може да получи достъп до тези услуги. Можете да забраните достъпа през мрежата за всички, освен за оторизираните потребители.

Използване на известни слабости в програмите

При проектирането на някои програми и мрежови услуги не са предвиждани сериозни мерки за сигурност, затова те поначало са уязвими на атаки. Пример за това са отдалечените услуги на BSD (rlogin, rexec и т.н.).

Най-добрият начин да се предпазите от този тип атаки е да се забранят всички уязвими услуги или да се намерят техни алтернативни решения. При софтуера с отворен код понякога е възможно да се поправят слабостите в софтуера.

Отказ на услуга

Атаките от тип отказ на услуга предизвикват прекратяване на функционирането на услугата или програмата или пречат за тяхното използване от други хора. Те могат да бъдат осъществени в мрежовия слой чрез изпращане на старателно изработени злонамерени дейтаграми, които причиняват невъзможност да се обслужват мрежови връзки. Освен това, те могат да бъдат извършени и в приложния слой, където на програмата се подават внимателно подбрани приложни команди, които я натоварват максимално или спират нейната работа.

Предпазването на хостовете от подозрителния мрежов трафик и предотвратяването на съмнителни програмни команди и заявки са най-добрите начини за намаляване на риска от атаки от тип отказ на услуга. Полезно е детайлното познаване на метода за атака, затова би трябвало да се запознавате с всяка нова атака, веднага щом тя стане обществено достояние.

Мамене

При този тип атака даден хост или приложение да имитира действията на друг. Обикновено атакуващият се преструва на невинен хост, като проследява IP адреси в мрежови пакети. Например, напълно документиран експлойт на BSD услугата rlogin може да използва този метод, за да имитира TCP връзка от друг хост, предполагайки номерата на TCP последователностите.

За да се предпазите от този тип атаки, проверявайте автентичността на дейтаграмите и командите. Не допускайте маршрутизиране на дейтаграми с невалидни изходни адреси. Въведете непредсказуемост в механизмите за контрол на връзката, като промените номерата на TCP последователностите и използвате динамични адреси за портовете.

Подслушване

Това е най-простият тип атака. Даден хост се конфигурира да "слуша" и да прехваща данни, които не са предназначени за него. Старателно написаните подслушващи програми могат да прехванат имената и паролите на потребителите по време на влизането им в мрежата. Broadcast мрежи като Ethernet са особено уязвими на този тип атаки.

За да се предпазите от подобна опасност, избягвайте broadcast мрежови технологии и задължително използвайте шифриране на данните.

Изграждането на защитни стени за IP е много полезно за предотвратяване или ограничаване на непозволения достъп, отказ на услуга в мрежовия слой и при IP мамене. То не е особено полезно при използване на слабостите в мрежовите услуги или програми, както и при подслушване.

Какво представлява защитната стена?

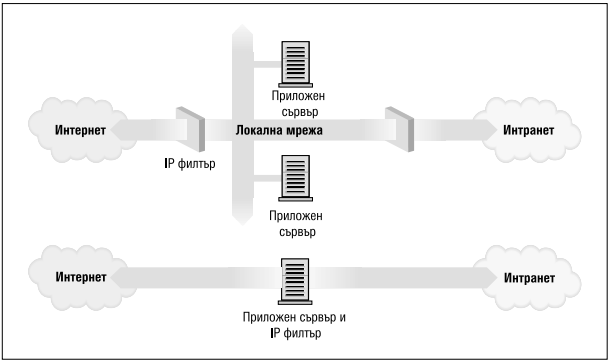
Защитната стена е сигурна и надеждна машина, която се намира между частната и обществената мрежа.* Машината – защитна стена е

* Терминът *защитна стена* (*firewall*) произлиза от името на приспособление, използвано за защита на хората от пожар. Защитната стена е преграда от огнеупорен материал, която се поставя между предполагаемото място за възникване на пожара и хората, които защитава.

конфигурирана с набор от правила, които определят кой мрежов трафик ще бъде допуснат да премине и кой ще бъде блокиран или отхвърлен. При някои големи организации можете да срещнете дори защитна стена, разположена вътре в корпоративната мрежа, за да изолира поверителните зони на организацията от други нейни служители. В много случаи компютърните престъпления се извършват вътре в организацията, а не само отвън.

Защитните стени могат да бъдат конструирани по най-различни начини. Най-усъвършенстваната схема включва множество самостоятелни машини и е известна като *гранична мрежа* (*perimeter network*). Две от машините работят като "филтри", наречени клапи (chokes), които позволяват преминаването само на конкретни типове мрежов трафик, а между тези клапи се намират мрежовите сървъри, например пощенския портал или WWW proxy сървър. Тази конфигурация може да бъде много сигурна и позволява лесно осъществяване на широк контрол над това кой може да се свързва както отвътре навън, така и отвън навътре. Конфигурации от тази категория би трябвало да се използват в големите организации.

Обикновено, обаче, защитните стени са самостоятелни машини, които изпълняват всичките тези функции. Те са малко по-несигурни, защото ако има някаква слабост в самата защитна машина, позволяваща на някого да получи достъп до нея, сигурността на цялата мрежа ще бъде разбита. Все пак, този тип защитни стени са по-евтини и по-лесни за управление в сравнение с усъвършенстваната схема, описана по-горе. На Фигура 9-1 са показани двете най-често срещани конфигурации на защитни стени.



Фигура 9-1. Двата основни класа архитектури за защитни стени.

Ядрото на Linux предоставя множество вградени средства, които му позволяват да работи доста добре като защитна стена за IP. Мрежовата реализация включва код за извършване на IP филтриране по множество различни начини и предоставя механизъм за много прецизно задаване на правилата, които бихте искали да използвате. Защитната стена за Linux е достатъчно гъвкава, за да може да бъде много полезна и при двете конфигурации, показани на Фигура 9-1. Софтуерът – защитна стена за Linux предоставя две други полезни възможности, които ще разгледаме в отделни глави: IP счетоводство (Глава 10, *IP счетоводство*) и IP маскиране (Глава 11, *IP маскиране и транслиране на мрежови адреси*).

Какво представлява IP филтрирането?

IP филтрирането е просто механизъм, който решава кои типове IP дейтаграми ще бъдат обработени нормално и кои ще бъдат пренебрегнати. Под *пренебрегната* дейтаграма ще разбираме, че тя е изгрита и напълно игнорирана, все едно, че никога не е получавана. Можете да приложите множество различни видове критерии, за да определите кои дейтаграми искате да филтрирате; някои примери за това са:

- Тип на протокола: TCP, UDP, ICMP и т.н.
- Номер на гнездото (за TCP/UDP)
- Тип на дейтаграмата: SYN/ACK, данни, ICMP Echo Request и т.н.
- Адрес на изпращача на дейтаграмата: от къде идва
- Адрес на получателя на дейтаграмата: къде отива.

На този етап е важно да разберете, че IP филтрирането е възможност на мрежовия слой. Това означава, че то не разбира нищо от приложението, използващо мрежовите връзки, а само от самите връзки. Например, можете да забраните на потребителите достъп до вашата вътрешна мрежа през подразбиращия се порт за telnet, но ако разчитате само на IP филтриране, не можете да им попречите да използват програмата telnet с порт, който сте разрешили да преминава през вашата защитна стена. Можете да избегнете подобен род проблеми чрез използване на proxy сървъри (сървъри – представители) за всяка услуга, която допускате през вашата защитна стена. Proxy сървърите разбират протокола на приложението, което са проектирани да представят, и поради това могат да предотвратят злоупотреби като използването на програмата telnet за преминаване през защитна стена, използвайки WWW порт. Ако вашата защитна стена поддържа WWW proxy, тяхната telnet връзка винаги ще получава отговор от proxy-сървъра, който ще позволи преминаването само на HTTP заявки. Съществуват голям брой програми – proxy сървъри. Някои от тях са свободен софтуер, а много други са комерсиални продукти. В документа Firewall-HOWTO са разгледани най-популярните от тях, но те са извън обхвата на тази книга.

Наборът от правила за IP филтриране се формира от множество комбинации на критериите, изброени по-горе. Например, нека си представим, че искате да позволите на WWW потребителите от мрежата на Виртуалната пивоварна да имат достъп до Интернет, освен до web-сървърите на други сайтове. Можете да конфигурирате вашата защитна стена да позволява препредаване на:

- дейтаграми с адрес на изпращача от Виртуалната пивоварна, произволен адрес на получателя и порт за получаване 80 (WWW)
- дейтаграми с адрес на получателя от Виртуалната пивоварна и порт на изпращача 80 (WWW) от произволен адрес.

Обърнете внимание, че тук използвахме две правила. Ние позволяваме на нашите данни да излизат навън, но също и на съответните отговори да се връщат обратно. На практика, както ще видим след малко, Linux опростява това и ни позволява да го зададем с една единствена команда.

Конфигуриране на Linux като защитна стена

За да построите защитна стена за IP под Linux, е необходимо да разполагате с ядро, компилирано с поддръжка на IP защитна стена и съответните конфигурационни инструменти. За всички стабилни ядра, разработени преди серията 2.2, трябва да използвате инструмента *ipfwadm*. С ядрата 2.2.x се отбелязва началото на трето поколение IP защитни стени за Linux, наречени *IP Chains* (IP вериги). При IP веригите се използва подобна на *ipfwadm* програма, наречена *ipchains*. Ядрата на Linux с версия 2.3.15 или по-нова поддържат четвърто поколение IP защитни стени, наречени *netfilter*. Кодът на *netfilter* е резултат от мащабно преработване на управлението на потока пакети в Linux. *netfilter* е създание с много лица, осигуряващо директна обратна съвместимост за *ipfwadm* и *ipchains*, както и новата алтернативна команда, наречена *iptables*. В следващите няколко раздела ще поговорим за разликата между тези три команди.

Ядро, конфигурирано за IP защитна стена

Ядрото на Linux трябва да бъде конфигурирано така, че да поддържа IP защитна стена. За целта е достатъчно да изберете съответните опции, когато конфигурирате ядрото с `make menuconfig`.* Можете да видите как се извършва това в Глава 3, *Конфигуриране на мрежовия хардуер*. При ядрата от серия 2.2 трябва да изберете следните опции:

```
Networking options  --->
        [*] Network firewalls
        [*] TCP/IP networking
        [*] IP: firewalling
        [*] IP: firewall packet logging
```

* Опцията Firewall packet logging е специална възможност, която записва върху специално устройство ред с информация за всяка дейтаграма, отговаряща на определено защитно правило, така че да можете да я видите.

При ядрата с версия 2.4.0 или по-нова трябва вместо това да изберете следната опция:

```
Networking options  --->
   [*] Network packet filtering (replaces ipchains)
       IP: Netfilter Configuration  --->
            .
          <M> Userspace queueing via NETLINK (EXPERIMENTAL)
          <M> IP tables support (required for filtering/masq/NAT)
          <M>   limit match support
          <M>   MAC address match support
          <M>   netfilter MARK match support
          <M>   Multiple port match support
          <M>   TOS match support
          <M>   Connection state match support
          <M>   Unclean match support (EXPERIMENTAL)
          <M>   Owner match support (EXPERIMENTAL)
          <M>   Packet filtering
          <M>     REJECT target support
          <M>     MIRROR target support (EXPERIMENTAL)
            .
          <M>   Packet mangling
          <M>     TOS target support
          <M>     MARK target support
          <M>   LOG target support
          <M> ipchains (2.2-style) support
          <M> ipfwadm (2.0-style) support
```

Инструментът ipfwadm

Инструментът *ipfwadm* (IP Firewall Administration – администриране на IP защитна стена) е средство, използвано за създаване на правила за защитни стени за всички ядра до версия 2.2.0. Неговият команден синтаксис може да бъде много объркващ, защото той може да извършва доста сложен кръг от дейности, но ние ще представим някои типични примери, илюстриращи най-важните разновидности от тях.

Инструментът *ipfwadm* е включен в повечето съвременни дистрибуции на Linux, но може би не по подразбиране. Може би за него има специален софтуерен пакет, който трябва да инсталирате. Ако вашата дистрибуция не го включва, можете да изтеглите пакета с изходен код от директорията */pub/linux/ipfwadm/* на **ftp.xos.nl** и да го компилирате.

Инструментът ipchains

Също като *ipfwadm*, първоначалното използване на инструмента *ipchains* може да бъде доста объркващо. Той осигурява цялата гъвкавост на *ipfwadm* с опростен команден синтаксис, а в допълнение осигурява и "верижен" механизъм, който ви позволява да управлявате множество набори от правила и да ги свързвате заедно. Ще разгледаме свързването на правила в отделен раздел към края на главата, защото в повечето случаи това е една усъвършенствана концепция.

Командата *ipchains* се намира в повечето дистрибуции на Linux, базирани на ядрото 2.2. Ако искате да я компилирате самостоятелно, можете да намерите изходния код от сайта на разработчиците ѝ *http://www.rustcorp.com/linux/ipchains/*. В пакета с изходния код е включен и обвиващ скрипт, наречен *ipfwadm-wrapper*, който имитира командата *ipfwadm*, но всъщност извиква командата *ipchains*. С това допълнение прехвърлянето на една съществуваща конфигурация за защитна стена е много по-безболезнено.

Инструментът iptables

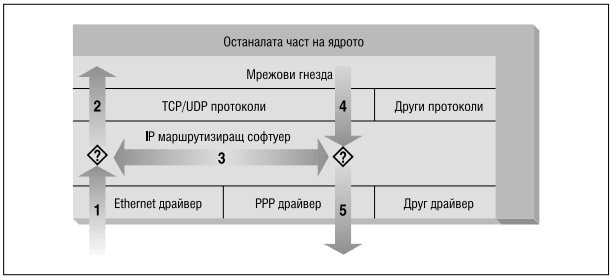
Синтаксисът на инструмента *iptables* е доста близък до синтаксиса на *ipchains*. Промените са подобрения и резултат от преработването на инструмента, за да бъде разширяем чрез споделени библиотеки. Също както за *ipchains*, ще представим еквивалентните за командата *iptables* примери, за да можете да ги сравните и да разграничите нейния синтаксис от този на другите команди.

Инструментът *iptables* е включен в пакета с изходен код на *netfilter*, който е на разположение на адрес *http://www.samba.org/netfilter/*. Освен това, той ще бъде включен в дистрибуциите на Linux, базирани на ядрата от серия 2.4.

За огромната крачка напред в *netfilter* ще поговорим в посветения на него раздел, намиращ се малко по-долу в тази глава.

Три начина, по които извършваме филтриране

Да разгледаме как една Unix машина или всъщност всяка машина, способна да извършва IP маршрутизиране, обработва IP дейтаграми. Основните стъпки, показани на Фигура 9-2, са следните:



Фигура 9-2: Етапи при обработката на IP дейтаграми

- IP дейтаграмата е получена. (1)
- Входящата IP дейтаграма се анализира, за да се определи дали тя е предназначена за процес на тази машина.
- Ако дейтаграмата е за тази машина, тя се обработва локално. (2)
- Ако дейтаграмата не е предназначена за тази машина, извършва се търсене на подходящ маршрут в таблицата с маршрути и дейтаграмата се насочва към съответния интерфейс или се игнорира, ако не се намери маршрут. (3)
- Дейтаграмите от локалните процеси се изпращат на маршрутизиращия софтуер за изпращане към съответния интерфейс. (4)
- Излизащата дейтаграма се анализира, за да се определи дали за нея има валиден маршрут, по който да се изпрати; ако няма такъв, тя се игнорира.
- IP дейтаграмата се предава. (5)

В нашата диаграма потокът 1→3→5 изобразява маршрутизираните от нашата машина данни от хост в нашата Ethernet мрежа към хост, достъпен през нашата PPP връзка. Потоците 1→2 и 4→5 изобразяват входния и изходния поток от данни на мрежова програма, която работи на нашия локален хост. Потокът 4→3→2 представлява потока от данни през loopback връзка. Естествено данните текат както към, така и от мрежовите устройства. Въпросителните знаци на схемата показват точките, в които IP слоят взима решения за маршрутизация.

Защитната стена в ядрото на Linux е способна да прилага филтриране на различни етапи от този процес. Това означава, че можете да

филтрирате IP дейтаграмите, които пристигат във вашата машина, да филтрирате дейтаграмите, които се препращат през вашата машина и да филтрирате дейтаграми, които са готови за изпращане.

При *ipfwadm* и *ipchains*, към поток 1 от схемата се прилага правилото Input, към поток 3 – правилото Forwarding, а към поток 5 – правилото Output. Когато разглеждаме *netfilter* по-късно, ще видим, че точките на пресичане са се променили така, че правилото Input се прилага върху поток 2, а правилото Output – върху поток 4. Това има важни последици за начина, по който ще структурирате вашите набори от правила, но общият принцип остава в сила за всички версии на защитната стена на Linux.

От началото това може да ви изглежда излишно сложно, но то осигурява гъвкавост, която позволява да се изградят някои много по-усъвършенствани и мощни конфигурации.

Оригиналната защитна стена за IP (ядрата 2.0)

Първото поколение поддръжка на IP защитна стена за Linux се появи в ядрата от серията 1.1. Това беше адаптация за Linux на защитната стена ipfw за BSD, написана от Alan Cox. Поддръжката на защитна стена, която се появи в ядрата от серия 2.0, е от второто поколение и бе усъвършенствана от Jos Vos, Pauline Meddelink и други.

Използване на ipfwadm

Командата *ipfwadm* беше конфигурационно средство за второто поколение IP защитна стена за Linux. Може би най-простият начин да се опише използването на командата *ipfwadm* е чрез пример. Да започнем, като кодираме примера, който представихме по-горе.

Един прост пример

Да допуснем, че в нашата организация имаме мрежа и че за свързване на нашата мрежа към Интернет използваме Linux базирана машина-защитна стена. Освен това да предположим, че искаме потребителите от тази мрежа да имат достъп до web-сървърите в Интернет, но да не разрешаваме преминаването на никакъв друг трафик.

Ще поставим подходящо правило за препредаване, позволяващо дейтаграмите с адрес на изпращача от нашата мрежа и адрес на полу ча-

теля с порт 80 да бъдат препредавани навън, а съответните дейтаграми-отговор да бъдат препредавани обратно чрез защитната стена.

Да предположим, че нашата мрежа има 24-битова мрежова маска (Клас C) и адрес 172.16.1.0. Правилата, които можем да използваме, са следните:

```
# ipfwadm -F -f
# ipfwadm -F -p deny
# ipfwadm -F -a accept -P tcp -S 172.16.1.0/24 -D 0/0 80
# ipfwadm -F -a accept -P tcp -S 0/0 80 -D 172.16.1.0/24
```

Аргументът *-F* в командния ред указва на *ipfwadm*, че това е правило за препредаване. Първата команда инструктира *ipfwadm* да изчисти всички съществуващи правила за препредаване. Това ни гарантира, че започваме да добавяме конкретни правила при известно състояние на защитната стена.

Второто правило задава нашата подразбираща се политика при препредаване. Указваме на ядрото да отказва или да не разрешава препредаването на IP дейтаграми. Много важно е да се зададе подразбираща се политика, защото тя описва какво ще се случи с всяка дейтаграма, която не се управлява конкретно от някое друго правило. В повечето конфигурации на защитни стени трябва да зададете подразбираща се политика "отказване", както е показано в този пример, за да сте сигурни, че през вашата защитна стена се препредава само трафикът, който специално сте разрешили да преминава.

Третото и четвъртото правила са тези, които изпълняват нашите изисквания. Третата команда позволява на нашите дейтаграми да излизат, а четвъртото правило позволява на отговорите да се връщат обратно.

Да разгледаме всеки от аргументите:

-F Това е правило за препредаване (от Forwarding).

-a accept

Добави това правило с политика "accept" (приема се), което означава, че ще препредаваме всички дейтаграми, които съответстват на правилото.

-P tcp

Това правило се отнася за tcp дейтаграми (за разлика от UDP или ICMP)

-S 172.16.1.0/24

Първите 24 бита от адреса на изпращача трябва да съвпадат с битовете на адреса на мрежа 172.16.1.0.

-D 0/0 80

Адресът на получателя трябва да има нула бита, съвпадащи с адреса 0.0.0.0. Това всъщност е краткия запис на "всеки". Числото 80 е порта на получателя, в този случай WWW сървър. Освен този запис, можете да използвате всяка дефиниция във файла */etc/services*, за да укажете порта, така че записа `-D 0/0 www` също би работил много добре.

ipfwadm възприема мрежови маски по начин, който може би не ви е познат. Означението /nn е средство да се опише колко бита от зададения адрес са значещи или какъв е размерът на маската. Битовете винаги се броят отляво надясно; в Таблица 9-1 са показани някои често срещани маски.

Таблица 9-1. Брой на значещите битове в най-често срещаните мрежови маски

Мрежова маска	Битове
255.0.0.0	8
255.255.0.0	16
255.255.255.0	24
255.255.255.128	25
255.255.255.192	26
255.255.255.224	27
255.255.255.240	28
255.255.255.248	29
255.255.255.252	30

По-горе споменахме, че *ipfwadm* позволява малък трик, който прави добавянето на правила от този вид по-лесно. Този трик е опция, наречена *–b*, която прави командата двупосочно правило.

Двупосочният флаг позволява да обединим нашите две правила в едно по следния начин:

```
# ipfwadm -F -a accept -P tcp -S 172.16.1.0/24 -D 0/0 80 -b
```

Важно уточнение

Разгледайте по-внимателно нашия набор от правила. Забелязвате ли, че там все още има един метод за атака, който някой отвън би могъл да използва, за да пробие нашата защитна стена?

Нашият набор от правила позволява да преминават всички дейтаграми, идващи извън нашата мрежа, за които портът на изпращача е 80. Това включва и дейтаграмите с вдигнат бит SYN! Битът SYN е това, което декларира TCP дейтаграмата като заявка за връзка. Ако човек извън нашата мрежа има привилегирован достъп до хост, той би могъл да осъществи връзка през нашата защитна стена с всеки от нашите хостове, при условие, че от своята страна използва порт 80. Това не е, каквото имахме предвид.

За щастие този проблем има решение. Командата *ipfwadm* предоставя друг флаг, който ни позволява да изграждаме правила, съответстващи на дейтаграмите с вдигнат бит SYN. Да променим нашия пример, за да включим такова правило:

```
# ipfwadm -F -a deny -P tcp -S 0/0 80 -D 172.16.10.0/24 -y
# ipfwadm -F -a accept -P tcp -S 172.16.1.0/24 -D 0/0 80 -b
```

Флагът *-y* указва, че правилото съответства на дейтаграмата само, ако е вдигнат флагът SYN. Нашето ново правило гласи: "Отказвай всички TCP дейтаграми, предназначени за нашата мрежа, идващи откъдето и да е и които имат порт на изпращача 80 и вдигнат бит SYN," или "Отказвай всички заявки за връзка от хостове, идващи от порт 80."

Защо поставихме това специално правило *преди* основното правило? Правилата на защитната стена за IP работят така, че първото открито съответствие автоматично става правилото, което ще се използва. И двете правила съответстват на дейтаграмите, които искаме да спрем, затова трябва да сме сигурни, че правилото `deny` (отхвърли) е преди правилото `accept` (приеми).

Преглед на правилата

След като въведем правилата, можем да поискаме от *ipfwadm* да ги изброи с командата:

```
# ipfwadm -F -l
```

Тази команда ще отпечата всички конфигурирани правила за препредаване. Изходът би трябвало да изглежда подобно на този:

```
# ipfwadm -F -l
IP firewall forward rules, default policy: accept
type  prot source              destination         ports
deny  tcp  anywhere            172.16.10.0/24      www -> any
acc   tcp  172.16.1.0/24       anywhere            any -> www
```

Командата *ipfwadm* ще се опита да преобразува номера на порта в име на услуга, използвайки файла */etc/services*, ако в него има запис за този номер порт.

В подразбиращия се изход липсват някои важни за нас подробности. От него не можем да видим ефекта от аргумента `-y`. Командата *ipfwadm* може да покаже и по-подробен списък, ако зададете и аргумента `-e` (от extended output – разширен изход). Тук няма да показваме целия изход, защото той е твърде широк, за да се събере на страницата, но той включва колоната `opt` (опции), която показва опцията -y за управление на SYN пакети:

```
# ipfwadm -F -l -e
IP firewall forward rules, default policy: accept
 pkts bytes type  prot opt  tosa tosx ifname ifaddress source...
    0     0 deny  tcp  --y- 0xFF 0x00 any    any       anywhere...
    0     0 acc   tcp  b--- 0xFF 0x00 any    any       172.16.1.0/24...
```

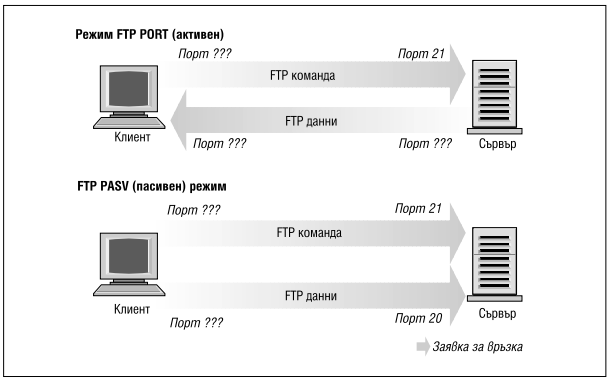
Един по-сложен пример

Предният пример беше прост. Не всички мрежови услуги са толкова лесни за конфигуриране като услугата WWW; на практика, една типична конфигурация на защитна стена би била доста по-сложна. Нека разгледаме един друг общ пример, този път FTP. Искаме потребителите на нашата вътрешна мрежа да могат да влизат в FTP сървъри в Интернет, в които да четат и записват файлове. Но не искаме потребители от Интернет да могат да влизат в нашите FTP сървъри.

Знаем, че FTP използва два TCP порта – порт 20 (ftp данни) и порт 21 (ftp), така че:

```
# ipfwadm -a deny -P tcp -S 0/0 20 -D 172.16.1.0/24 -y
# ipfwadm -a accept -P tcp -S 172.16.1.0/24 -D 0/0 20 -b
#
# ipfwadm -a deny -P tcp -S 0/0 21 -D 172.16.1.0/24 -y
# ipfwadm -a accept -P tcp -S 172.16.1.0/24 -D 0/0 21 -b
```

Така ли е? Ами, не винаги. FTP сървърите могат да работят в два различни режима: пасивен режим и активен режим.* В пасивен режим FTP сървърът очаква връзка от клиента. В активен режим всъщност сървърът прави връзката към клиента. Активният режим обикновено е включен по подразбиране. Различията са показани на Фигура 9-3.



Фигура 9-3. Режими на FTP сървъра

Когато работят в активен режим, много FTP сървъри създават своята връзка за данни през порт 20, което за нас улеснява донякъде нещата, но за нещастие не всички FTP сървъри го правят.*

Но по какъв начин ни засяга това? Погледнете нашето правило за порт 20, т.е. порта за FTP данни. Правилото, каквото е сега, предполага, че връзката ще бъде направена от нашия клиент към сървъра. Това ще работи, ако използваме пасивен режим. Но ще ни бъде много трудно да конфигурираме задоволително правило, позволяващо активен FTP режим, защото може предварително да не знаем кои портове ще се използват. Ако отворим нашата защитна стена като

* Активният режим FTP някак си неинтуитивно се задава с командата *PORT*. Пасивният режим FTP се задава с командата *PASV*.

* Един добър пример за FTP сървър, който не прави това, е демонът ProFTPd, или поне ранните му версии.

разрешим входящи връзки на всеки порт, ще изложим нашата мрежа на атаки чрез всички услуги, които приемат връзки.

Дилемата се решава най-сигурно чрез изискването нашите потребители да работят в пасивен режим. Повечето FTP сървъри и много FTP клиенти могат да работят по този начин. Популярният клиент *ncftp* също поддържа пасивен режим, но може да се изисква малки промени в конфигурацията, за да бъде зададен пасивен режим по подразбиране. Много WWW браузъри като Netscape, също поддържат пасивен режим FTP, така че няма да е много трудно да се намери подходящ софтуер за използване. Като алтернатива, можете напълно да избегнете проблема чрез използването на FTP proxy сървър, който приема връзката от вътрешната мрежа и установява връзки към външната мрежа.

При изграждането на защитна стена вероятно ще се сблъскате с редица подобни проблеми. Винаги трябва внимателно да обмисляте как в действителност работи дадена услуга, за да сте сигурни, че сте задали подходящия набор правила за нея. Една истинска конфигурация на защитна стена може да бъде доста сложна.

Преглед на аргументите на ipfwadm

Командата *ipfwadm* има много различни аргументи, отнасящи се до конфигурирането на защитна стена за IP. Общият ѝ синтаксис е:

```
ipfwadm категория команда параметри [опции]
```

Да разгледаме всеки един аргумент.

Категории

Трябва да бъде зададена точно една от дадените по-долу категории. Категорията указва на защитната стена какъв вид защитно правило конфигурирате:

-I Правило за входящи пакети

-O Правило за изходящи пакети

-F Правило за препредавани пакети

Команди

Трябва да се укаже поне една от следващите команди, като тя се прилага само към онези правила, които се отнасят към зададената ка-

тегория. Командата указва на защитната стена какво действие да предприеме.

-a [политика]

Добавя ново правило

-i [политика]

Вмъква ново правило

-d [политика]

Изтрива съществуващо правило

-p политика

Задава подразбиращата се политика

-l Извежда списък на всички съществуващи правила

-f Изчиства всички съществуващи правила

Политиките, отнасящи се до IP защитната стена и техните значения, са:

accept

Позволява съответстващите дейтаграми да бъдат получавани, препредавани или изпращани

deny

Блокира съответстващите дейтаграми, за да не бъдат получавани, препредавани или изпращани

reject

Не позволява съответстващите дейтаграми да бъдат получавани, препредавани или изпращани и връща ICMP съобщение за грешка на хоста, изпратил дейтаграмата.

Параметри

Трябва да се зададе поне един от следващите параметри. Параметрите се използват, за да се определи към кои дейтаграми се прилага това правило:

-P протокол

Аргументът може да бъде TCP, UDP, ICMP или all (всички). Пример:

```
-P tcp
```

-S address[/mask] [port]

Задава IP адреса на изпращача, на който ще съответства това правило. Ако не зададете мрежова маска, подразбира се маска "/32". Като опция можете да зададете за кои портове ще се прилага това правило. За да работи това правило, трябва да зададете и протокола с аргумента *-P*, както беше описано по-горе. Ако не зададете порт или интервал от портове, предполага се съответствие за всички портове. Ако желаете, можете да зададете портовете чрез име с помощта на записите във файла */ets/services*. Ако зададете протокола ICMP, полето port се използва за означаване типа на ICMP дейтаграмите. Могат да се описват интервали от портове; използвайте общия синтаксис: *долен-порт:горен-порт*. Ето един пример:

```
-S 172.29.16.1/24 ftp:ftp-data
```

-D address[/mask] [port]

Задава IP адреса на получателя, на който ще съответства това правило. Адресът на получателя се кодира със същите правила, описани по-горе за адреса на изпращача. Ето един пример:

```
-D 172.29.16.1/24 smtp
```

-V address

Задава адреса на мрежовия интерфейс, на който пакетът се приема (*-I*) или изпраща (*-O*). Това позволява да създаваме правила, приложими само за определени мрежови интерфейси на нашата машина. Ето един пример:

```
-V 172.29.16.1
```

-W name

Задава името на мрежовия интерфейс. Този аргумент работи по същия начин като аргумента *-V*, с тази разлика, че подавате името на устройството вместо неговия адрес. Ето един пример.:

```
-W ppp0
```

Незадължителни аргументи

Тези аргументи понякога са много полезни:

-b Този аргумент се използва за двупосочен (bi-directional) режим. Този флаг съответства на трафик, преминаващ в произволна посока между зададените изпращач и получател. Това спестява създаването на две правила: едно за правата и друго за обратната посока.

-o Този аргумент разрешава записването на съвпадащите дейтаграми в дневника на ядрото. Всяка дейтаграма, която съответства на това правило, ще бъде отбелязана като съобщение на ядрото. Това е полезно, защото ви позволява да откривате неупълномощен достъп.

-y Използва се за обозначаване на TCP connect дейтаграми. Опцията указва правилото да съответства само на дейтаграми, които се опитват да създадат TCP връзки. Ще съответстват само дейтаграмите, които имат вдигнат бит SYN и свален бит ACK. Това е полезно за филтриране на опитите за създаване на TCP връзки и се игнорира от другите протоколи.

-k Този аргумент се използва за обозначаване на TCP acknowledgement дейтаграми. Тази опция указва правилото да съответства само на дейтаграми, които са потвърждения към пакети, опитващи се да създадат TCP връзки. Ще съответстват само дейтаграми с вдигнат бит ACK. Това е полезно за филтриране на опитите за TCP връзки и се игнорира от другите протоколи.

Типове ICMP дейтаграми

Всяка от командите за конфигуриране на защитна стена дава възможност за определяне на типове ICMP дейтаграми. За разлика от TCP и UDP портовете, не съществува удобен конфигурационен файл, който съдържа списък на типовете дейтаграми и техните значения. Типовете ICMP дейтаграми са дефинирани в документа RFC-1700, наречен Assigned Numbers (зададени номера). Типовете ICMP дейтаграми са изброени и в един от заглавните файлове на стандартната библиотека на C. Файлът */usr/include/netinet/ip_icmp.h*, който е част от стандартния пакет с библиотеката на GNU и се използва от програмистите на C, когато пишат мрежов софтуер, използващ протокола ICMP, дефинира типовете ICMP дейтаграми. За ваше удобство изброяваме типовете ICMP дейтаграми в Таблица 9-2. Интерфейсът на командата *iptables* позволява да се задават ICMP типовете чрез имена, затова поместваме и мнемоничните имена, които използва тази команда.

Таблица 9-2. Типове ICMP дейтаграми

Номер на типа	Мнемоника за iptables	Описание на типа
0	echo-reply	Отговор на заявка echo
3	destination-unreachable	Получателят е недостижим
4	source-quench	Подателят е изключен
5	redirect	Пренасочване
8	echo-request	Заявка echo
11	time-exceeded	Превишаване на времето
12	parameter-problem	Проблем с параметър
13	timestamp-request	Заявка за маркер на времето
14	timestamp-reply	Отговор с маркер за времето
15	-	Заявка за информация
16	-	Отговор с информация
17	address-mask-request	Заявка за маска на адрес
18	address-mask-reply	Отговор с маска на адрес

Вериги за IP защитна стена (ядрата 2.2)

Повечето аспекти на Linux се развиват, за да посрещнат нарастващите изисквания на неговите потребители; защитната стена за IP не е изключение. Традиционната реализация на защитна стена за IP е добра за повечето приложения, но може да бъде неудобна и неефективна за конфигуриране в сложни среди. За да се разреши този проблем, беше разработен нов метод за конфигуриране на IP защитна стена и свързаните с нея възможности. Този нов метод беше наречен "Вериги за IP защитна стена" (IP Firewall Chains) и за първи път беше предоставен за общо използване с ядрото на Linux 2.2.0.

Поддръжката на вериги за IP защитна стена бе разработена от Paul Russell и Michael Neuling.* Paul документира софтуера, свързан с веригите за IP защитна стена в документа IPCHAINS-HOWTO.

* С Paul можете да се свържете на адрес *Paul.Russel@rustcorp.com.au*.

Веригите за IP защитна стена позволяват разработването на класове от правила за защитни стени, към които след това можете да добавяте и премахвате хостове или мрежи. В резултат на създаването на вериги за защитни стени може да се разширят възможностите на защитната стена при конфигурации с голям брой правила.

Веригите за IP защитна стена се поддържат от ядрата от серия 2.2 и освен това са достъпни като patch към ядрата 2.0. HOWTO документа описва от къде може да се намери този patch и дава много полезни съвети за начина, по който ефективно да използвате конфигурационния инструмент *ipchains*.

Използване на ipchains

Има два начина, по които можете да използвате инструмента *ipchains*. Първият начин е да използвате shell-скрипта *ipfwadm-wrapper*, който е преди всичко заместител на *ipfwadm*, управляващ в действителност програмата *ipchains*. Ако желаете да направите само това, няма смисъл да четете по-нататък. Вместо това прочетете отново предните раздели, описващи командата *ipfwadm*, и вместо нея използвайте *ipfwadm-wrapper*. Това ще работи, но няма гаранция, че скриптът ще бъде поддържан и няма да можете да се възползвате от нито една от усъвършенстваните възможности, които предлагат веригите за IP защитна стена.

Вторият начин да се използва инструментът *ipchains* е да се научи неговия нов синтаксис и да се променят всички съществуващи конфигурации, използвайки новия синтаксис вместо стария. С внимателно обмисляне ще откриете, че можете да оптимизирате своята конфигурация докато я конвертирате. Сингаксисът на *ipchains* е по-лесен за научаване от този на *ipfwadm*, така че това е добра възможност.

За целите на конфигурирането на защитни стени *ipfwadm* работи с три набора правила. С веригите за IP защитна стена можете да създадете произволен брой набори от правила, всеки от които е свързан с някой друг, но има три набора от правила, които винаги са налице. Стандартните набори от правила са директни еквиваленти на използваните в *ipfwadm* правила, с изключение на това, че имат имена `input`, `forward` и `output`.

Нека най-напред разгледаме общия синтаксис на командата *ipchains*, след което ще разберем как да използваме *ipchains* вместо *ipfwadm*, без да засягаме да разглеждаме усъвършенстваните възможности на веригите. Ще направим това като преработим нашите предишни примери.

Синтаксис на командата ipchains

Синтаксисът на командата *ipchains* е лесно разбираем. Сега ще разгледаме най-важното за него. Общият синтаксис на повечето команди *ipchains* е:

```
ipchains команда задаване-на-правило опции
```

Команди

Съществуват редица начини за обработка на правила и набори от правила с помощта на *ipchains*. Командите, свързани с изграждането на IP защитни стени, са:

-A верига

Добавя едно или повече правила в края на посочената верига. Ако е зададено име на хост на изпращач или получател, което се свежда до повече от един IP адрес, за всеки от адресите ще се прибави правило.

-I верига номер-на-правило

Вмъква едно или повече правила в началото на посочената верига. Отново, ако в спецификацията на правилото е зададено име на хост, ще бъде добавено правило за всеки от разпознатите адреси.

-D верига

Изтрива едно или повече правила от посочената верига, съответстващи на спецификацията на правилото.

-D верига номер-на-правило

Изтрива правилото, намиращо се на място `номер-на-правило` в посочената верига. Местата на правилата започват от едно за първото правило във веригата.

-R верига номер-на-правило

Заменя правилото, намиращо се на място `номер-на-правило` в посочената верига, със зададена спецификация на правило.

-C верига

Проверява дали дейтаграмата, описана от специфицираното правило, съответства на зададената верига. Тази команда ще върне съобщение, описващо как дейтаграмата се обработва от веригата. Това е много полезно за тестване на конфигурацията на защитна стена и малко по-късно ще се спрем на него по-подробно.

-L [верига]

Извежда списък с правилата в посочената верига или във всички вериги, ако не е посочена конкретна верига.

-F [верига]

Изчиства правилата в посочената верига или във всички вериги, ако не е посочена конкретна верига.

-Z [верига]

Нулира броячите на дейтаграми и байтове за всички правила в посочената верига или във всички вериги, ако не е посочена конкретна верига.

-N верига

Създава нова верига с посоченото име. Не трябва да съществува друга верига със същото име. По този начин се създават дефинирани от потребителя вериги.

-X [верига]

Изтрива посочената дефинирана от потребителя верига, или всички дефинирани от потребителя вериги, ако не е зададена такава. За успешното изпълнение на тази команда, не трябва да има връзки към посочената верига от никоя друга верига правила.

-P верига политика

Задава посочената политика като поведение по подразбиране за посочената верига. Валидни политики при изграждането на защитна стена са `ACCEPT`, `DENY`, `REJECT`, `REDIR` или `RETURN`. `ACCEPT`, `DENY` и `REJECT` имат същите значения, както за традиционната реализация на IP защитна стена. `REDIR` указва, че дейтаграмата трябва прозрачно да се пренасочи към порт на хоста-защитна стена. При `RETURN` кодът на IP защитната стена се връща към веригата на защитната стена, която е извикала веригата, съдържаща това правило, и продължава с обработката на правилото след извикващото правило.

Параметри за задаване на правило

Много от параметрите на *ipchains* създават спецификация на правилото чрез определяне на какви типове пакети му съответстват. Ако някой от тези параметри е изпуснат от спецификацията на правилото, се приемат стойности по подразбиране.

-p [!]протокол

Задава протокола на дейтаграмата, съответстваща на това правило. Валидни имена на протоколи са `tcp`, `udp`, `icmp` или `all`. Тук можете да посочите и номер на протокол, за да укажете други протоколи. Например, бихте могли да използвате числото 4 за задаване на капсулиращия протокол `ipip`. Ако е добавен символа !, правилото е негативно и дейтаграмата ще съответства на всеки протокол, различен от зададения. Ако този параметър не е подаден, ще се подразбират всички протоколи (`all`).

-s [!]адрес[/маска] [!] [порт]

Задава адреса и порта на подателя на дейтаграмата, съответстваща на това правило. Адресът може да бъде зададен като име на хост, име на мрежа или IP адрес. Незадължителният параметър маска е мрежовата маска, която да се използва и може да бъде зададена или в традиционна форма (т.е., /255.255.255.0) или с модерната форма (т.е. /24). Незадължителната опция порт задава TCP или UDP порта, или типа на ICMP дейтаграма, които съответстват на това правило. Можете да подавате спецификация за порт само, ако сте задали параметъра *–p* с един от протоколите `tcp`, `udp` или `icmp`. Портовете могат да се задават като обхват, чрез определяне на горната и долната граници на обхвата, с двоеточие като разделител. Например, `20:25` означава всички портове в границите от 20 до 25 включително. И тук знакът ! може да се използва, за да се инвертират стойностите.

-d [!]адрес[/маска] [!] [порт]

Задава адреса и порта на получателя на дейтаграмата, съответстваща на това правило. Кодирането на този параметър е същото като при параметъра -s.

-j цел

Задава действието, което да се предприеме при откриго съответствие с това правило. За този параметър можете да мислите като за "скок към". Валидни цели са `ACCEPT`, `DENY`, `REJECT`, `REDIR` и `RETURN`. Значението на всеки от тези обекти е обяснено по-горе. Все пак, можете да зададете името на потребителски дефинирана верига, където обработката да продължи. Ако този параметър е пропуснат, върху съответстващите на правилото дейтаграми не се извършват никакви други действия, освен да се актуализират броячите на дейтаграми и байтове.

-I [!]име-на-интерфейс

Задава интерфейса, по който дейтаграмата е получена или трябва да бъде предадена. И тук знакът ! инвертира резултата от съвпадението. Ако името на интерфейса завършва с +, тогава ще съответства всеки интерфейс, който започва с подадения низ. Например, `-I ppp+` съответства на всяко PPP мрежово устройство, а `-I ! eth+` съответства на всички интерфейси, различни от Ethernet устройствата.

[!] -f

Указва, че това правило се прилага към всички, освен към първия фрагмент от фрагментирана дейтаграма.

Опции

Следващите опции на *ipchains* са по-общи по характер. Някои от тях управляват по-неясните възможности на софтуера на IP вериги:

-b Указва на командата да генерира две правила. Едното правило съответства на подаваните параметри, а другото правило добавя съответствие със съответните параметри с инвертирана стойност.

-v Указва на *ipchains* да отпечатва подробна информация.

-n Указва на *ipchains* да показва IP адресите и портовете като номера, без да се опитва да разпознае съответните им имена.

-l Разрешава ядрото да записва в дневник съвпадащите дейтаграми. Всяка дейтаграма, която съответства на правилото, ще бъде отбелязана и записана от ядрото в log-файл, като се използва неговата функция *printk()*, която обикновено се управлява от програмата *sysklogd*. Това е полезно за разглеждане на някои необикновени дейтаграми.

-o[maxsize]

Указва на софтуера за IP веригите да копира всяка дейтаграма, съответстваща на правилото, в устройството от потребителско пространство "netlink". Аргументът maxsize ограничава броя на байтовете от всяка дейтаграма, които се подават на устройството netlink. Тази опция е полезна предимно от софтуерните разработчици, но може да бъде използвана в бъдеще и от софтуерни пакети.

-m markvalue

Указва съответстващите дейтаграми да бъдат *маркирани* със стойност. Маркиращите стойности са 32-битови числа без знак. В съществуващите реализации това не прави нищо, но в някой бъдещ момент, те могат да определят как дейтаграмата се обработва от друг софтуер, например маршрутизиращ код. Ако маркиращата стойност започва с + или -, тя се добавя към или изважда от съществуващата маркираща стойност.

-t andmask xormask

Дава възможност да променяте битовете "тип на услугата" в IP заглавието на всяка дейтаграма, която съответства на това правило. Битовете за типа на услугата се използват от интелигентните маршрутизатори, за да категоризират дейтаграмите преди да ги ретранслират. Маршрутизиращият софтуер на Linux е в състояние да използва такъв вид приоритети. Стойностите `andmask` и `xormask` представляват битови маски, с които ще се извърши съответно логическа AND или OR операция с битовете за типа на услугата в дейтаграмата. Това е една перспективна възможност, която е описана по-подробно в IPCHAINS-HOWTO.

-x Указва всички числа в изхода на *ipchains* да бъдат разширени до тяхната точна стойност без закръгляване.

-y Указва правилото да съответства на TCP дейтаграми с вдигнат бит SYN и свалени битове ACK и FIN. Това се използва за филтриране на заявки за TCP връзки.

Преразглеждане на нашия прост пример

Нека отново предположим, че в нашата организация имаме мрежа и че използваме защитна стена на базата на Linux машина, за да позволим на нашите потребители достъп до WWW сървъри в Интернет, без обаче да позволяваме преминаването на друг трафик.

Ако нашата мрежа има 24-битова мрежова маска (клас C) и има адрес 176.16.1.0, бихме използвали следните правила на *ipchains*:

```
# ipchains -F forward
# ipchains -P forward DENY
# ipchains -A forward -s 0/0 80 -d 172.16.1.0/24 -p tcp -y -j DENY
# ipchains -A forward -s 172.16.1.0/24 -d 0/0 80 -p tcp -b -j ACCEPT
```

Първата от командите изчиства всички правила от веригата `forwards`, а втората команда задава подразбираща се политика `DENY` за набора от правила `forwards`. Накрая, третата и четвъртата команди извършват специфичното филтриране, което искаме. Четвъртата команда позволява на дейтаграми към и от web сървърите извън нашата мрежа да преминават, а третата предотвратява входящи TCP връзки с номер на изходящия порт 80.

Ако искаме да добавим правила, които да позволяват само пасивен режим на достъп до FTP сървъри във външна мрежа, би трябвало да ги добавим следното:

```
# ipchains -A forward -s 0/0 20 -d 172.16.1.0/24 -p tcp -y -j DENY
# ipchains -A forward -s 172.16.1.0/24 -d 0/0 20 -p tcp -b -j ACCEPT
# ipchains -A forward -s 0/0 21 -d 172.16.1.0/24 -p tcp -y -j DENY
# ipchains -A forward -s 172.16.1.0/24 -d 0/0 21 -p tcp -b -j ACCEPT
```

Показване на правилата с ipchains

За да получим списък на правилата с *ipchains*, ще използваме неговия аргумент *–L*. Също както при *ipfwadm*, има аргументи, които управляват степента на детайлност на изхода. В своята най-проста форма, *ipchains* създава изход, който изглежда така:

```
# ipchains -L -n
Chain input (policy ACCEPT):
Chain forward (policy DENY):
target     prot opt     source               destination          ports
DENY       tcp  -y----  0.0.0.0/0            172.16.1.0/24        80 ->  *
ACCEPT     tcp  ------  172.16.1.0/24        0.0.0.0/0            * ->   80
ACCEPT     tcp  ------  0.0.0.0/0            172.16.1.0/24        80 ->  *
ACCEPT     tcp  ------  172.16.1.0/24        0.0.0.0/0            * ->   20
ACCEPT     tcp  ------  0.0.0.0/0            172.16.1.0/24        20 ->  *
ACCEPT     tcp  ------  172.16.1.0/24        0.0.0.0/0            * ->   21
ACCEPT     tcp  ------  0.0.0.0/0            172.16.1.0/24        21 ->  *

Chain output (policy ACCEPT):
```

Ако не подадете име на веригата, за която да се генерира списък, *ipchains* ще покаже списък на всички правила във всички вериги. Аргументът `–n` в нашия пример инструктира *ipchains* да не се опитва да преобразува адресите или портовете в имена. Представената информация би следвало да е ясна сама по себе си.

Подробната форма, указана от опцията *–v*, предоставя много повече детайли. Нейният изход включва полета за броячите на дейтаграми и байтове, тип на услуга, *AND* и *XOR* флагове, името на интерфейса, маркера и размерът на изхода.

Всички правила, създадени с *ipchains* имат броячи на дейтаграмите и байтовете, асоциирани с тях. Това е начина, по който е реализирано IP счетоводството, което е разгледано подробно в Глава 10. По подразбиране тези броячи се представят в закръглен вид, използвайки суфиксите `K` и `M` за отбелязване съответно на хиляди и милиони. Ако е зададен аргумента `–x`, броячите се разширяват до тяхната пълна незакръглена форма.

Пълноценно използване на веригите

Сега вече знаете, че командата *ipchains* е заместител на *ipfwadm* и е с по-прост синтаксис на командния ред и някои интересни подобрения, но без съмнение вече искате да научите къде и как бихте могли да използвате потребителски дефинираните вериги. Също така, вероятно бихте искали да знаете как да използвате поддържащите скриптове, които съпровождат командата *ipchains* в нейния софтуерен пакет. Сега ще разгледаме тези теми и ще се спрем на възникналите въпроси.

Дефинирани от потребителя вериги

Трите набора от правила на традиционния код за IP защитна стена осигуряваха механизъм за изграждане на конфигурации на защитни стени, които бяха твърде прости за разбиране и използване за малки мрежи с прости изисквания за защита. Когато изискванията на конфигурацията не са прости, се появяват множество проблеми. Най-напред, големите мрежи често изискват много по-голям брой правила за защита в сравнение с малките, които сме виждали до сега; появяват се необикновени нужди, които изискват добавянето на правила за защита за покриване на различни сценарии. С нарастване на правилата, ефективността на защитната стена спада, тъй като трябва да се извършват все повече и повече тестове върху всяка дейтаграма и управляемостта се превръща в проблем. Второ, не е възможно автоматично да се активират и изключват набори от правила, поради което сте принудени да се излагате на възможност за атака, когато сте на средата на преработването на вашето правило.

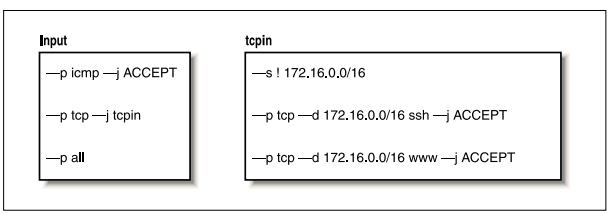
Архитектурата на веригите за IP защитна стена помага да се облекчат тези проблеми, давайки възможност на мрежовия администратор да създава произволни набори от правила за защитни стени, които могат да се свържат към трите вградени набори от правила. Можем да използваме опцията `–N` на *ipchains*, за да създадем нова верига с произволно име от до осем знака. (Препоръчваме ви да използвате

само малки букви в името.) Опцията *–j* конфигурира действието, което трябва да се извърши, когато дейтаграмата съответства на зададеното правило. Опцията *–j* указва, че ако дейтаграмата съответства на правилото, трябва да се извършат следващи тестове, зададени в потребителски дефинирана верига. Ще илюстрираме това с диаграма.

Разгледайте следните команди *ipchains*:

```
ipchains -P input DENY
ipchains -N tcpin
ipchains -A tcpin -s ! 172.16.0.0/16
ipchains -A tcpin -p tcp -d 172.16.0.0/16 ssh -j ACCEPT
ipchains -A tcpin -p tcp -d 172.16.0.0/16 www -j ACCEPT
ipchains -A input -p tcp -j tcpin
ipchains -A input -p all
```

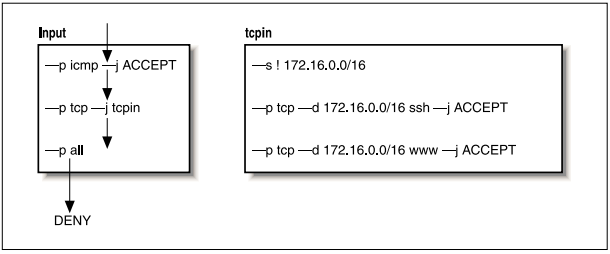
Първо задаваме подразбираща се политика `deny` за входна верига `input`. Втората команда създава потребителска верига, наречена "`tcpin`". Третата команда добавя правило към веригата `tcpin`, което съответства на всяка дейтаграма, чийто изпращач е извън нашата локална мрежа; правилото не предизвиква никакво действие. Това правило е отчетно правило и ще бъде разгледано по-подробно в Глава 10. Следващите две правила съответстват на всяка дейтаграма, която е насочена към нашата локална мрежа за един от портовете `ssh` или `www`; дейтаграмите съответстващи на това правило се приемат. Третото правило е мястото, където започва истинската магия в *ipchains*. То указва на софтуера за защитна стена да проверява всяка дейтаграма от протокола TCP чрез потребителски дефинираната верига `tcpin`. Накрая добавяме към нашата входна верига правило, което съответства на всяка дейтаграма; това е още едно счетоводно правило. По този начин създаваме веригите за защитна стена, показани на Фигура 9-4.



Фигура 9-4. Прост набор от правила за IP верига.

Веригите `input` и `tcpin` са запълнени с нашите правила. Обработката на дейтаграмите започва винаги от една от вградените вериги. Ще видим как нашата потребителски дефинирана верига се извиква за изпълнение, следвайки пътя на обработка на различните типове дейтаграми.

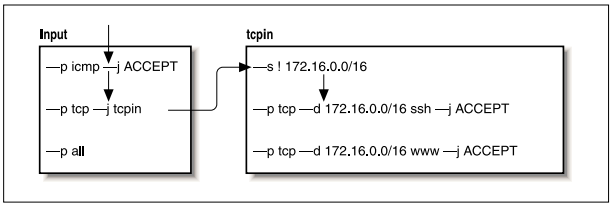
Първо, нека погледнем какво се случва, когато се получава UDP дейтаграма за един от нашите хостове. Фигура 9-5 илюстрира потока през правилата.



Фигура 9-5. Последователност на извършването на тестове по правилата за получена UDP дейтаграма.

Дейтаграмата е получена от веригата `input` и преминава през първите две правила, защото те изискват съответно ICMP и TCP протокол. Тя съответства на третото правило във веригата `input`, но в него не е зададена цел, така че съответните броячи на байтове и дейтаграми се актуализират, но други действия не се предприемат. Дейтаграмата достига края на входната верига, съпоставя се на нейната стандартна политика и се отхвърля.

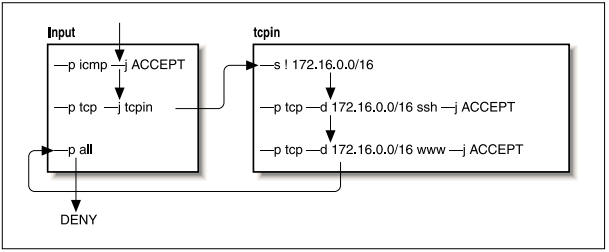
За да видим как работи нашата потребителски дефинирана верига, нека сега разгледаме какво се случва, когато получаваме TCP дейтаграма за `ssh` порт на един от нашите хостове. Последователността е показана на Фигура 9-6.



Фигура 9-6. Потокът на правилата за получена TCP дейтаграма за ssh.

Този път второто правило на веригата `input` съответства на дейтаграмата и задава връзка към `tcpin`, дефинираната от нас потребителска верига. Определянето на потребителски дефинирана верига като получател (цел), означава, че дейтаграмата трябва да бъде тествана по правилата в тази верига, така че следващото правило за проверка е първото правило от веригата `tcpin`. Първото правило съответства на всяка дейтаграма, която има адрес на изпращач извън нашата локална мрежа и не задава цел, поради което това също е едно счетоводно правило и тестването продължава със следващото правило от веригата. Второто правило от нашата верига `tcpin` съответства на дейтаграмата и задава цел `ACCEPT` (приемане). По този начин пристигнахме до целта, след което не се извършват повече обработки от защитната стена. Дейтаграмата е приета.

Накрая да видим какво се случва, когато достигнем до края на дефинираната от потребителя верига. За да видим това, ще проследим потока за TCP дейтаграма, насочена към порт, различен от двата, които следим специално, както е показано на Фигура 9-7.



Фигура 9-7. Потокът на правилата за получена TCP дейтаграма за telnet.

Потребителски дефинираните вериги нямат действие по подразбиране. Когато всички правила в потребителски дефинираната верига са проверени и никое от тях не съответства, кодът на защитната стена действа като при открито правило `RETURN`, така че, ако това не е което искате, би трябвало да си осигурите в края на потребителски дефинираната верига преход към правило, извършващо онова, което ви е необходимо. В нашия пример, проверките продължават с връщане към правилото от набора `input`, непосредствено следващо правилото, което ни е изпратило към потребителски дефинираната верига. Евентуално достигаме до края на веригата `input`, която има действие по подразбиране и нашата дейтаграма не се приема.

Примерът е много прост, но илюстрира идеята ни. Реалното използване на IP вериги може бъде по-сложно. Малко по-сложен пример е даден в следния списък от команди.

```
#
# Задава REJECT като подразбираща се политика за препредаване
ipchains -P forward REJECT
#
# създава нашата потребителска верига
ipchains -N sshin
ipchains -N sshout
ipchains -N wwwin
ipchains -N wwwout
#
# гарантира отхвърлянето на връзки, идващи по грешен път
ipchains -A wwwin -p tcp -s 172.16.0.0/16 -y -j REJECT
ipchains -A wwwout -p tcp -d 172.16.0.0/16 -y -j REJECT
ipchains -A sshin -p tcp -s 172.16.0.0/16 -y -j REJECT
ipchains -A sshout -p tcp -d 172.16.0.0/16 -y -j REJECT
#
```

```
# Гарантира, че всичко достигнало до края на потребителската верига, се
# отхвърля
ipchains -A sshin -j REJECT
ipchains -A sshout -j REJECT
ipchains -A wwwin -j REJECT
ipchains -N wwwout -j REJECT
#
# Отклонява услугите www и ssh към съответната потребителска верига
ipchains -A forward -p tcp -d 172.16.0.0/16 ssh -b -j sshin
ipchains -A forward -p tcp -s 172.16.0.0/16 -d 0/0 ssh -b -j sshout
ipchains -A forward -p tcp -d 172.16.0.0/16 www -b -j wwwin
ipchains -A forward -p tcp -s 172.16.0.0/16 -d 0/0 www -b -j wwwout
#
# Вмъква нашите правила за съответствие с хостовете на позиция две
# от нашите потребителски вериги
ipchains -I wwwin 2 -d 172.16.1.2 -b -j ACCEPT
ipchains -I wwwout 2 -s 172.16.1.0/24 -b -j ACCEPT
ipchains -I sshin 2 -d 172.16.1.4 -b -j ACCEPT
ipchains -I sshout 2 -s 172.16.1.4 -b -j ACCEPT
ipchains -I sshout 2 -s 172.16.1.6 -b -j ACCEPT
#
```

В този пример използвахме подбор от потребителски дефинирани вериги както за да улесним управлението на конфигурацията на защитна стена, така и за да подобрим ефективността на нашата защитна стена в сравнение с решението, използващо само вградени вериги.

Този пример създава потребителски дефинирани вериги за всяка от услугите `ssh` и `www` във всяко направление за свързване. Веригата, наречена `wwwout`, е мястото, където поставяме правила за хостове, които имат право да извършват изходящи WWW връзки, а `sshin` е мястото, където дефинираме правила за хостове, на които искаме да разрешим входящи `ssh` връзки. Допуснахме, че имаме изискване да разрешаваме и да отказваме на отделни хостове в нашата мрежа възможността да правят или да получават `ssh` и `www` връзки. Това опростяване се получава, защото потребителски дефинираните вериги ни дават възможност удобно да групираме правилата за пристигащите и изходящите разрешения за хостовете, вместо да ги смесваме. Подобрението на ефективността се получава, защото за всяка определена дейтаграма намалихме средния брой на необходимите тестове, преди да се намери целта. Ефективността нараства с добавянето на повече хостове. Ако не бяхме използвали потребителски вериги, за да определим какво действие да предприемем с всяка получена дейтаграма, би трябвало да търсим в целия списък от правила. Дори ако предположим, че всяко от правилата в нашия списък съответства на еднаква част от общия брой на обработените дейтаграми, пак бихме търсили

средно в половината от списъка. Погребителските вериги ни позволяват да избегнем тестването на голям брой правила, ако тестваната дейтаграма не съответства на някое просто правило от вградената верига, което препраща към тях.

Скриптове за поддръжка на ipchains

Софтуерният пакет *ipchains* се разпространява с три поддържащи скрипта. Първият от тях вече разгледахме накратко, докато останалите два осигуряват лесно и удобно средство за запазване и възстановяване на конфигурацията на защитната стена.

Скриптът *ipfwadm-wrapper* емулира синтаксиса на командния ред на командата *ipfwadm*, но за построяване на правилата на защитната стена използва командата *ipchains*. Това е удобен начин да пренесете конфигурацията на съществуваща защитна стена към ядрото или една алтернатива на изучаването на синтаксиса на *ipchains*. Скриптът *ipfwadm-wrapper* се държи различно от командата *ipfwadm* в две неща: първо, понеже командата *ipchains* не поддържа задаване на интерфейс по адрес, скриптът *ipfwadm-wrapper* приема аргумент *–V*, но се опитва да го превърне в еквивалента на *–W*, търсейки име на интерфейс, конфигуриран с предоставения адрес. Скриптът *ipfwadm-wrapper* винаги ще подава предупреждение, когато използвате опцията *–V*, за да напомни за това. Второ, правилата за отчетност на фрагменти не се транслират правилно.

Скриптовете *ipchains-save* и *ipchains-restore* правят създаването и изменението на конфигурацията на защитната стена много по-просто. Командата *ipchains-save* чете текущата конфигурация на защитната стена и я записва в опростена форма на стандартния изход. Командата *ipchains-restore* чете данни от изходния формат на командата *ipchains-save* и конфигурира IP защитната стена с тези правила. Ползата от използването на тези команди пред директното модифициране на конфигурационния скрипт за защитната стена и тестване на конфигурацията, е възможността за еднократно динамично изграждане на конфигурацията и нейното запазване след това. По-късно можете да възстановите тази конфигурация, да я модифицирате и ако искате да я запишете отново.

Като използвате скриптовете, трябва да въведете следното за запазване на вашата текуща конфигурация:

```
ipchains-save >/var/state/ipchains/firewall.state
```

Бихте могли да я възстановите, вероятно по време на стартиране, чрез:

ipchains-restore `</var/state/ipchains/firewall.state`

Скриптът *ipchains-restore* проверява дали съществува някоя от потребителски дефинираните вериги, изброени в неговия вход. Ако сте подали аргумента `-f`, той автоматично ще изчисти правилата от потребителската верига, преди да конфигурира правилата от входа. Поведението по подразбиране е да пита дали да прескочи тази верига или да я изчисти.

Мрежов филтър и IP таблици (ядрата 2.4)

Докато разработваше Веригите за IP защитна стена, Paul Russell реши, че защитната стена за IP би трябвало да бъде по-лесна за използване; скоро след това той започна да се занимава със задачата да се опростят аспектите на обработката на дейтаграмите в кода за защитна стена в ядрото и създаде схема за филтриране, която е както по-ясна, така и по-гъвкава. Той нарече новата схема *netfilter* (мрежов филтър).

По времето на изготвянето на тази книга разработването на *netfilter* още не беше стабилизирано. Надяваме се да ни извините за всички грешки в описанието на *netfilter* или свързаните с него средства за конфигуриране, които се дължат на промените, настъпили след подготвянето на този материал. Счетохме работата на *netfilter* за достатъчно важна, за да решим включването на този материал, независимо от това, че част от него все още се обмисля. Ако имате някакви съмнения, съответните HOWTO документи съдържат най-подробната и съвременна информация по всички въпроси, свързани с конфигурирането на *netfilter*.

И така, какво не е наред при IP веригите? Те рязко подобриха ефективността и управлението на правилата за защитни стени. Но начинът, по който те обработват дейтаграмите, все още е сложен, особено във връзка със свързаните със защитни стени възможности като IP

маскирането (разгледано в Глава 11) и други форми на транслиране на адреси. Част от тази сложност се дължи на това, че IP маскирането и транслирането на мрежови адреси бяха разработени независимо от кода на IP защитните стени и интегрирани по-късно, вместо да бъдат проектирани като неразделна част от кода на защитната стена от самото начало. Ако разработчикът иска да добави още нови възможности в последователността на обработката на дейтаграми, той ще има трудности с намирането на мястото, където да вмъкне кода и ще бъде принуден да направи изменения в ядрото, за да постигне това.

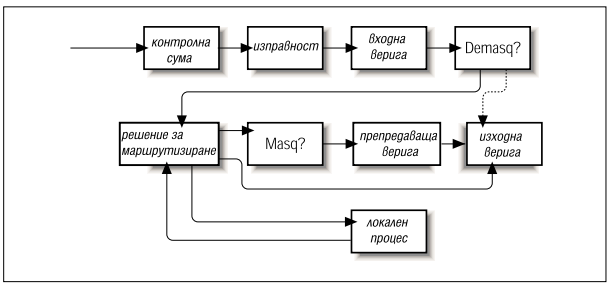
Освен това, има и други проблеми. В частност, веригата "input" описва входа към мрежовото IP ниво като цяло. Входната верига съдържа както дейтаграмите които *са насочени* към този хост, така и дейтаграмите, които ще бъдат *маршрутизирани* от този хост. Това някак си е неинтуитивно, защото обърква функционирането на входната верига с това на веригата за препредаване, която се прилага само за дейтаграми, които са за препредаване, но винаги минават през входната верига. Ако искате да обработвате дейтаграмите за този хост отделно от тези за препредаване, трябва да изградите сложни правила, които се изключват едно друго. Същият проблем съществува и за изходната верига.

Неизбежно част от тази сложност се отразява върху работата на системния администратор, защото тя влияе върху начина, по който трябва да се разработят тези правила. Нещо повече, всички разширения на филтрирането налагат директни изменения на ядрото, защото всички политики за филтриране са внедрени там и няма начин да се осигури прозрачен интерфейс към тях. Новият *netfilter* се стреми да намали както сложността, така и тромавостта на старите решения, въвеждайки обща схема в ядрото, която опростява начина, по който се обработват дейтаграмите и осигурява възможност за разширяване на политиките за филтриране, без да се налага да се променя ядрото.

Да разгледаме две от направените ключови изменения. Фигура 9-8 показва как се обработват дейтаграмите при реализацията на IP вериги, а Фигура 9-9 показва как те се обработват при реализацията на *netfilter*. Ключовите разлики са премахването на маскиращите функции от кода на ядрото и промяната в разполагането на веригите *input* и *output*. За да се реализират тези промени, беше създаден нов и разширяващ се инструмент, наречен *iptables*.

При IP веригите входната верига се прилага към всички дейтаграми, получавани от хоста, независимо от това дали са предназначени за локалния хост или се маршрутизират към някой друг хост. При *netfilter*,

входната верига се прилага само към дейтаграмите, насочени към локалния хост, а препредаващата верига се използва само за дейтаграми, насочени към друг хост. Аналогично, в IP веригите, изходната верига се използва за всички дейтаграми, напускащи локалния хост, независимо от това дали дейтаграмата е генерирана в локалния хост или е подадена от някой друг хост. В *netfilter*, изходната верига се използва само за дейтаграми, генерирани в този хост и не се прилага към дейтаграми, маршрутизирани от друг хост. Дори само тази промяна предлага огромно опростяване на много конфигурации на защитни стени.



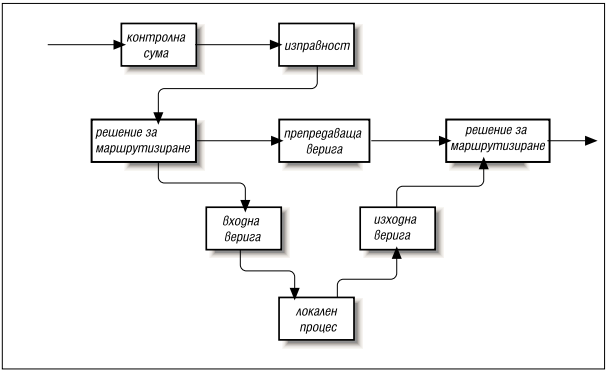
Фигура 9-8. Верига за обработване на дейтаграми в IP веригите.

На Фигура 9-8 елементите, означени като “demasq” и “masq”, са отделни компоненти на ядрото, отговарящи за входната и изходната обработка на маскираните дейтаграми. Тяхната нова реализация е като модули на *netfilter*.

Представете си случая на конфигурация, при която подразбиращата се политика за входящата, препредаващата и изходящата верига е `deny`. При IP веригите са необходими шест правила, за да се позволи която и да е сесия през хоста-защитна стена: по две за входящата, препредаващата и изходящата вериги (едното правило покрива правата, а другото – обратната посока за всеки маршрут). Можете да си представите как конфигурацията бързо може да стане особено сложна и трудна за управление, когато искате да смесите сесиите, които могат да бъдат маршрутизирани, и сесиите, които трябва да бъдат свързани с локалния хост, без да се пренасочват. IP веригите ви позволяват да създавате вериги, които биха опростили тази задача доня-

къде, но проектирането не е очевидно и изисква определено ниво на опит.

При реализацията на *netfilter* с *iptables*, тази сложност изчезва напълно. За да бъде маршрутизирана една услуга през хоста-защитна стена, без обаче да завършва в локалния хост, са необходими само две правила: по едно за правата и обратната посока в препредаващата верига. Това е очевидния начин за проектиране на правила за защитна стена и ще допринесе извънредно много за опростяване на проектирането на конфигурации за защитни стени.



Фигура 9-9. Верига за обработване на дейтаграми в netfilter.

Справочникът PACKET-FILTERING HOWTO предлага подробен списък на извършените изменения, затова нека тук ще обърнем внимание на по-практични аспекти.

Обратна съвместимост с ipfwadm и ipchains

Забележителната гъвкавост на *netfilter* за Linux се илюстрира от възможността за емулиране на интерфейсите на *ipfwadm* и *ipchains*. Емулацията прави преминаването към новата генерация на софтуер за защитна стена малко по-лесна.

Двата модула на *netfilter* в ядрото, наречени *ipfwadm.o* и *ipchains.o* осигуряват обратна съвместимост за *ipfwadm* и *ipchains*. Можете в

даден момент да заредите само един от тези модули и то само, ако модулът *ip_tables.o* не е зареден. Когато съответният модул е зареден, *netfilter* работи точно както предишната реализация на защитна стена.

netfilter имитира интерфейса на *ipchains* със следните команди:

```
rmmod ip_tables
modeprobe ipchains
ipchains …
```

Използване на iptables

Инструментът *iptables* се използва за конфигуриране на филтриращите правила на *netfilter*. Неговият синтаксис заимства много от командата *ipchains*, но се различава в едно много значително отношение: той е *разширяем*. Това означава, че неговата функционалност може да бъде разширена без да се налага прекомпилиране. Този ефект се постига чрез използването на споделени библиотеки. Съществуват стандартни разширения и след малко ще разгледаме някои от тях.

За да можете да използвате командата *iptables*, трябва да заредите модула *netfilter*, който осигурява поддръжка за нея. Най-лесният начин да направите това е като се използва командата *modprobe* по следния начин:

```
modprobe ip_tables
```

Командата *iptables* се използва за конфигуриране както на IP филтрирането, така и на транслирането на мрежови адреси. За улесняване на това има две таблици с правила, наречени *filter* и *nat*. Таблицата filter се подразбира, ако не сте задали опцията *-t*, за да изберете друга таблица. Освен това съществуват пет вградени вериги. За таблицата `filter` могат да се използват веригите `INPUT` и `FORWARD`, за таблицата nat могат да се използват веригите `PREROUTING` и `POSTROUTING`, а и за двете таблици е налична веригата `OUTPUT`. В тази глава ще разгледаме само таблицата *filter*. Таблицата *nat* е описана в Глава 11.

Общият синтаксис на повечето команди *iptables* е следния:

```
iptables команда спецификация-на-правило разширения
```

Сега ще разгледаме подробно някои опции, след което ще покажем някои примери.

Команди

Има много начини, по които можем да управляваме правилата и наборите от правила с командата *iptables*. Отнасящите се за IP защитната стена са:

-A верига

Добавя едно или повече правила към края на указаната верига. Ако е подадено име на хост като изпращач или като получател и то съответства на повече от един IP адрес, правилото ще бъде добавено за всеки адрес.

I верига номер-на-правило

Вмъква едно или повече правила в началото на указаната верига. Отново, ако в спецификацията на правилото е подадено име на хост, ще бъде добавено правило за всеки адрес, на който съответства това име.

-D верига

Изтрива едно или повече правила от зададената верига, съответстващо на спецификацията на правилото.

-D верига номер-на-правило

Изтрива правилото, намиращо се на позиция `номер-на-правило` в указаната верига. Позицията на правилото започва от 1 за първото правило във веригата.

-R верига rulenum

Заменя правилото, намиращо се на позиция `номер-на-правило` в дадената верига с дефинираната спецификация на правило.

-C верига

Проверява дали дейтаграмата, описана от зададеното правило, съответства на определена верига. Тази команда ще върне съобщение, описващо как веригата обработва дейтаграмата. Това е много полезна възможност за тестване на конфигурацията на вашата защитна стена и ще я разгледаме подробно по-долу.

-L [верига]

Генерира списък на правилата в посочената верига или във всички вериги, ако не е посочена верига.

-F [верига]

Изчиства правилата в посочената верига или във всички вериги, ако не е посочена верига.

-Z [верига]

Нулира броячите на дейтаграми и байтове за всички правила в посочената верига или във всички вериги, ако не е посочена верига.

-N [верига]

Създава нова верига с посоченото име. Верига със същото име не трябва да съществува. Така се създават потребителско-дефинирани вериги.

-X [верига]

Изтрива посочената потребителска верига или всички потребителски вериги, ако не е посочена верига. За да бъде успешна тази команда, не трябва да има препратки към посочените вериги в която и да е друга верига от правила.

-P верига политика

Задава подразбиращата се политика за посочената верига като указаната политика. Валидните политики за защитна стена са `ACCEPT`, `DROP`, `QUEUE` и `RETURN`. `ACCEPT` позволява на дейтаграмата да премине. При `DROP` дейтаграмата се игнорира. Политиката `QUEUE` указва дейтаграмата да се изпрати в потребителско пространство за по-нататъшна обработка. При `RETURN` кодът на IP защитната стена се връща към веригата от защитната стена, която е извикала веригата, съдържаща това правило и продължава обработката от правилото, следващо извикващото правило.

Параметри за задаване на правила

Има множество параметри на *iptables*, които съставят спецификацията на правило. Където е необходима спецификация на правило, трябва да се подаде всеки от тези параметри или ще се възприемат техните стойности по подразбиране.

-p[!]протокол

Определя протокола на дейтаграмата, която ще съответства на това правило. Валидните имена на протоколи са `tcp`, `udp`, `icmp` или число, ако знаете номера на IP протокола[29]. Например, можете да използвате числото `4`, за да посочите съответствие с кап-

[29] Можете да намерите имената и номерата на протоколите във файла */etc/protocols*

сулиращия протокол `ipip`. Ако се използва знака !, правилото се инвертира и дейтаграмата ще съответства на всеки протокол, различен от посочения. Ако този параметър не е подаден, по подразбиране ще се възприемат всички протоколи.

-s [!]адрес[/маска]

Определя изходния адрес на дейтаграмата, която съответства на това правило. Адресът може да бъде подаден като име на хост, име на мрежа или IP адрес. Незадължителната `маска` е мрежовата маска, която да се използва и може да се подаде или в традиционната форма (например 255.255.255.0) или в модерната форма (например /24).

-d [!]address[/mask]

Задава адреса и порт на получателя на дейтаграмата, която съответства на това правило. Кодирането на този параметър е същото както при параметъра *-s*.

-j цел

Задава какво действие трябва да се предприеме, при откриване на съответствие с това правило. За този параметър можете да мислите като за команда "иди на". Валидни цели са `ACCEPT`, `DROP`, `QUEUE` и `RETURN`. Вече обяснихме значението на всяка от тях в частта "Команди". Също така можете да посочите и името на потребителски дефинирана верига, където да продължи обработката. Освен това, можете да зададете името на целта чрез разширение. Ще поговорим за разширенията след малко. Ако този параметър е пропуснат, при съответствие на дейтаграмата не се предприемат никакви действия, освен да се актуализират броячите на дейтаграми и байтове за това правило.

-i [!]име-на-интерфейс

Задава интерфейса, на който е получена дейтаграмата. Както и преди, знакът ! инвертира резултата от съответствието. Ако името на интерфейса завършва с "+", тогава всеки интерфейс, започващ с подадения низ, ще съответства. Например, `-i ppp+` съответства на всяко мрежово PPP устройство, а `-i ! eth+` съответства на всички интерфейси, различни от ethernet устройствата.

-o [!]име-на-интерфейс

Задава интерфейса. на който ще бъде предадена дейтаграмата. Този аргумент се кодира по същия начин като аргумента *-i*.

[!] -f

Указва, че това правило се прилага само към втория и следващите фрагменти от фрагментирана дейтаграма, но не и за първия фрагмент.

Опции

Следващите опции на *iptables* са по-общи по-характер. Някои от тях управляват по-екзотичните възможности на софтуера *netfilter*:

-v указва на *iptables* да генерира по-подробен изход; това ще предостави повече информация.

-n указва на *iptables* да показва IP адреса и портовете като номера, без да се опитва да ги свърже с техните съответстващи имена.

-x указва всички числа в изхода на *iptables* да бъдат разширени до тяхната пълна стойност без закръгляване.

--line-numbers

указва при изброяване на правилата да бъдат показвани номерата на редовете. Номерът на реда ще съответства на позицията на правилото във веригата.

Разширения

По-горе казахме, че инструментът *iptables* е разширяем чрез възможността за използване на модули-споделени библиотеки. Съществуват някои стандартни разширения, които осигуряват част от възможностите, предлагани от *iptables*. За да се използва едно разширение, трябва да се зададе неговото име чрез аргумента на *iptables -m име*. Следващият списък показва опциите *–m* и *–p*, които определят контекста на разширението, както и опциите, предоставяни от това разширение.

TCP разширения: използвани с -m tcp -p tcp

--sport [!] [порт[:порт]]

Задава порта, който трябва да използва от изпращача на дейтаграмата, за да съответства на това правило. Портовете могат да бъдат посочени като интервал, чрез посочването на горна и долна граници на интервала, разделени с помощта на двоеточие. Например, `20:25` посочва всичките портове с номера от 20 до 25 включително. И тук знакът ! може да се използва за инвертиране на стойностите.

--dport [!] [порт[:порт]]

Задава порта, който трябва да се използва от получателя на дейтаграмата, за да съответства на това правило. Аргументът се кодира също като при опцията *–sport*.

--tcp-flags [!] маска comp

Указва, че това правило ще съответства, когато TCP флаговете в дейтаграмата съвпадат с посочените от `маска` и `comp`. *Маската* е списък от разделени със запетая флагове, които трябва да бъдат проверени при извършване на теста. `comp` е списък от разделени със запетая флагове, които трябва да бъдат подадени, за да съвпада правилото. Валидни флагове са: *SYN*, *ACK*, *FIN*, *RST*, *URG*, *PSH*, *ALL* и *NONE*. Това е възможност за напреднали: вижте подробното описание на протокола TCP, например RFC-793, за описание на значението и влиянието на всеки от тези флагове. Знакът ! инвертира правилото.

[!] --syn

Указва, че правилото съответства само на дейтаграми с вдигнат бит `SYN` и свалени битове `ACK` и `FIN`. Дейтаграмите с тези опции се използват за отваряне на TCP връзки, затова тази възможност може да бъде използвана за управление на заявките за връзки. Тази опция е кратък запис на:

```
--tcp-flags SYN,RST,ACK SYN
```

Когато използвате инвертиращ оператор, правилото ще съответства на дейтаграми, които нямат вдигнати нито `SYN`, нито `ACK` бит.

UDP разширения: използвани с -m udp -p udp

-sport [!] [порт[:порт]]

Определя порта, който трябва да се използва от изпращача на дейтаграмата, за да съответства на това правило. Портовете могат да бъдат зададени като интервал чрез посочването на горна и долна граници на интервала с помощта на двоеточие като разделител. Например, `20:25` посочва всичките портове с номера от 20 до 25 включително. И тук знакът ! може да се използва за инвертиране на стойностите.

-dport [!] [port[:port]]

Определя порта, който трябва да се използва от получателя на дейтаграмата, за да съвпада с това правило. Аргументът се кодира също както при опцията *–sport*.

ICMP разширения: използвани с `-m icmp -p icmp`

-icmp-type [!] име-на-тип

Определя типа на ICMP съобщението, на което трябва да съответства това правило. Типът може да бъде зададен чрез номер или чрез име. Някои валидни имена са: `echo-request`, `echo-reply`, `source-quench`, `time-exceeded`, `destination-unreachable`, `network-unreachable`, `host-unreachable`, `protocol-unreachable` и `host-unreachable`.

MAC разширения: използвани с `-m mac`

-mac-source [!] адрес

Задава Ethernet адреса на хоста, който е предал дейтаграмата, за да съответства на това правило. Това има смисъл само в правило във входящата или препредаващата верига, защото ще предаваме всяка дейтаграма, която премине изходящата верига.

Още едно преразглеждане на нашия прост пример

За да реализирате нашия прост пример с помощта на *netfilter*, можете просто да заредите модула *ipchains.o* и да симулирате, че това е версия на *ipchains*. Ние обаче ще го реализираме отново с помощта на *iptables*, за да покажем колко е подобен на предишния.

Още веднъж да предположим, че имаме мрежа в нашата организация и че използваме базирана на Linux машина за защитна стена, за да позволим на нашите потребители достъп до WWW услуги в Интернет, но да не позволяваме да преминава друг трафик.

Ако нашата мрежа има 24-битова мрежова маска (Клас C) и адрес 172.16.1.0, тогава бихме използвали следните правила на *iptables*:

```
# modprobe ip_tables
# iptables -F FORWARD
# iptables -P FORWARD DROP
# iptables -A FORWARD -m tcp -p tcp -s 0/0--sport 80 -d 172.16.1.0/24 /
    --syn -j DROP
# iptables -A FORWARD -m tcp -p tcp -s 172.16.1.0/24 --sport /
    80 -d 0/0 -j ACCEPT
# iptables -A FORWARD  -m tcp -p tcp -d 172.16.1.0/24 --dport80-s 0/0 -j/
    ACCEPT
```

В този пример командите *iptables* се интерпретират по същия начин като еквивалентните команди *ipchains*. Основната разлика е, че трябва да се зареди модула *ip_tables.o*. Забележете, че *iptables* не поддържа опцията *–b*, поради което трябва да зададем правило за всяко направление.

Управление на битовете TOS

Битовете TOS (Type Of Service – тип на услугата) представляват набор от четири флага в заглавието на IP. Когато някой от тези битови флагове е вдигнат, маршрутизаторите могат да обработват дейтаграмата различно от дейтаграмите, на които тези флагове не са вдигнати. Всеки от четирите бита има различно предназначение, но само един от тях може да бъде вдигнат в даден момент, така че комбинациите не са допустими. Флаговете се наричат битове за типа на услугата, защото позволяват на приложенията, предаващи данни, да указват на мрежата типа на мрежовата услуга, която изискат.

Наличните класове на мрежови услуги са:

Минимално забавяне

> Използва се, когато времето, необходимо за преминаване на дейтаграмата от изпращащия хост до получаващия хост (забавянето) е най-важно. Мрежовият доставчик може, например, да използва както оптически кабели, така и сателитни мрежови връзки. Данните, пренасяни по сателитните връзки имат по-голям път и поради това тяхното забавяне обикновено е по-голямо отколкото това по наземните връзки между две еднакви крайни точки. Мрежовият доставчик би могъл да не предава по сателит дейтаграмите с искан този тип услуга.

Максимална пропускателна способност

> Използва се, когато е важен обема на предаваните данни във всеки момент. Има много видове приложения, за които пропускателната способност на мрежата е особено важна, докато забавянето практически не е; например прехвърлянето на големи файлове. Мрежовият доставчик може да реши да насочи дейтаграмите, изискващи такъв тип услуга, по маршрути с по-голямо забавяне, но с по-голяма пропускателна способност, каквито са сателитните връзки.

Максимална надеждност

Използва се, когато е важно да имате определена сигурност, че данните ще пристигнат на местоназначението си, без да е необходимо препредаване. IP протоколът може да се пренася през произволен брой носещи среди. Макар че SLIP и PPP са адекватни протоколи за предаване на данни, те не са толкова надеждни като пренасянето на IP по друга мрежа, например X.25. Мрежовият доставчик би могъл да осигури алтернативна мрежа, предлагаща висока надеждност, за да пренесе IP при посочен такъв тип на услуга.

Минимална цена

Използва се, когато е важно да се минимизират разходите по предаването на данни. Наемането на честотна лента на сателит за междуконтинентално предаване обикновено е по-евтино от наемането на връзка по оптически кабели за същото разстояние. Ето защо доставчикът може да осигури и двете възможности и да ви таксува различно, според това каква връзка сте използвали. При този сценарий, при използването на бита за услуга "минимална цена", вашите дейтаграми могат да бъдат насочени през по-евтините сателитни връзки.

Задаване на TOS битовете с ipfwadm или ipchains

Командите *ipfwadm* и *ipchains* работят с TOS битовете по почти един и същ начин. И в двата случая задавате правило, което съответства на дейтаграмите с вдигнати определени TOS битове и използвате аргумента *-t*, за да укажете измененията, които искате да направите.

Измененията се задават с помощта на двубитови маски. С първата от тези битови маски се извършва логически AND с полето за IP опции на дейтаграмата, а с втората се извършва логически eXclusive-OR с него. Ако това ви звучи сложно, след малко ще дадем необходимите предписания, за да можете да разрешите желания от вас тип обслужване.

Битовите маски се задават с помощта на осембитови шестнадесетични стойности. Както *ipfwadm*, така и *ipchains* използват един и същ синтаксис на аргумента:

```
-t and-маска xor-маска
```

За щастие същите аргументи за маските могат да бъдат използвани всеки път, когато искате да зададете определен тип на услуга, което спестява необходимостта да ги разработвате отново. Те са представени в Таблица 9-3 с някои възможни приложения.

Таблица 9-3. Предложения за използване на TOS битови маски

TOS	AND маска	XOR маска	Подходящо за
Минимално забавяне	0x01	0x10	ftp, telnet, ssh
Максимална скорост	0x01	0x08	ftp-data, www
Максимална надеждност	0x01	0x04	snmp, dns
Минимален разход	0x01	0x02	nntp, smtp

Задаване на TOS битовете с iptables

Инструментът *iptables* ви дава възможност да задавате правила, които с помощта на опцията *-m* tos прехващат само дейтаграмите с TOS битове, съвпадащи с предварително дефинирана стойност, а с прехода *-j* `TOS` да задавате TOS битовете на IP дейтаграмите, съответстващи на правилото. Можете да поставяте TOS битове само на веригите `FORWARD` и `OUTPUT`. Проверката за съответствие и поставянето се извършват практически независимо. Можете да конфигурирате всякакъв вид интересни правила. Например, можете да конфигурирате правило, което отхвърля всички дейтаграми с определена комбинация на TOS битовете или правило, което проверява TOS битовете на дейтаграмите само от определени хостове. Най-често ще използвате правила, които ще съдържат проверка на съответствието и задаване на стойност, за да осъществите транслация на TOS битовете, както при *ipfwadm* и *ipchains*.

Вместо сложното двумасково конфигуриране на *ipfwadm* и *ipchains*, *iptables* използва по-прост подход за ясно определяне с какво трябва да съвпадат TOS битовете или как трябва да бъдат зададени TOS битовете. Освен това, вместо да се налага да помните шестнадесетични стойности, можете да зададете TOS битовете с помощта на по-дружелюбната мнемоника, посочена в следващата таблица.

Общият синтаксис, използван за проверка на съответствието на TOS битове е следния:

```
-m tos -tos мнемоника [други-аргументи] -j цел
```

Общият синтаксис, използван за задаване на TOS битове е:

```
[други-аргументи] -j TOS --set мнемоника
```

Както си спомняте, тези неща обикновено трябва да се използват заедно, но могат да се използват и съвсем независимо, ако вашата конфигурация налага това.

Мнемоника	Шестнадесетично
Normal-Service	0x00
Minimize-Cost	0x02
Maximize-Reliability	0x04
Maximize-Throughput	0x08
Maximize-Delay	0x10

Тестване на конфигурацията на защитната стена

След като сте разработили конфигурацията на дадена защитна стена, важно е да проверите дали тя върши наистина това, което сте искали да върши. Един начин да се направи това е да се използва тестов хост извън вашата мрежа, като се опитате да проникнете през вашата защитна стена; това обаче може да бъде много неудобно и бавно и освен това е ограничено до тестването само на онези адреси, които действително можете да използвате.

Има по-бърз и по-лесен начин при внедряването на защитни стени за Linux. Той ви дава възможност ръчно да генерирате тестове и да ги пускате през конфигурацията на защитната стена, точно както ако извършвате изпитанията с реални дейтаграми. Всички варианти на софтуера на защитна стена в ядрата на Linux – *ipfwadm*, *ipchains* и *iptables* – осигуряват поддръжка на този начин за тестване. Използването включва работа със съответната команда *check*.

Общата процедура за тестване е следната:

1. Проектиране и конфигуриране на защитната стена с помощта на *ipfwadm*, *ipchains* или *iptables*.
2. Проектиране на серии от тестове, които ще определят дали защитната стена действително работи както очаквате. За тези тестове можете да използвате произволни адреси на податели и получатели, така че изберете някои комбинации от адреси, които да бъдат приети и други, които да бъдат отхвърлени. Ако позволявате

или забранявате само определен обхват от адреси, една добра идея е да тествате адреси и от двете страни на границите на обхвата – един адрес непосредствено от вътрешната страна и един адрес непосредствено от външната страна на диапазона. Това ще помогне да гарантирате, че сте определили правилно границите, защото е много лесно в конфигурацията да се определят неправилни мрежови маски. Ако извършвате филтриране по протокол и номер на порт, вашият тест би трябвало да проверява също всички важни комбинации на тези параметри. Например, ако искате да получавате само TCP при определени обстоятелства, проверете дали UDP дейтаграмите се отхвърлят.

3. Разработете *ipfwadm*, *ipchains* и *iptables* правила, за да реализирате всеки тест. Вероятно ще ви струва да напишете всички правила в скрипт, така че да можете лесно да тествате повторно, след като сте поправили грешки или сте извършили изменения в проекта. Тестовете използват почти същия синтаксис, както задаването на правила, но аргументите имат малко по-различни значения. Например, аргументът за адрес на подателя в дефинирането на правило посочва адреса на изпращача, който трябва да имат дейтаграмите, съответстващи на това правило. Аргументът за адреса на подателя в синтаксиса на теста, за разлика от преди, определя адреса на подателя на тестовата дейтаграма, която ще бъде генерирана. За *ipfwadm*, трябва да използвате опцията *-c*, за да укажете, че тази команда е тест, докато за *ipchains* и *iptables*, трябва да използвате опцията *-C*. Във всички случаи *винаги* трябва да задавате адрес на подателя, адрес на получателя, протокол и интерфейс, които да бъдат използвани за теста. Възможни са и други аргументи, например номера на портове или стойности на TOS битове.

4. Изпълнете всяка тестова команда и вижте резултата. Изходът от всеки тест ще бъде една дума, показваща крайният изход за дейтаграмата след пропускането ѝ през конфигурацията на защитна

та стена – това е, където приключва обработката. За *ipchains* и *iptables* в допълнение на вградените, ще бъдат тествани и потребителските вериги.

5. Сравнете изхода от всеки тест с очаквания резултат. Ако има някакви различия, трябва да анализирате вашия набор от правила, за да определите къде сте допуснали грешка. Ако сте написали вашите команди в скрипт файл, можете лесно да повторите теста след поправката на грешките в конфигурацията на защитната стена. Добра практика е да изчиствате вашият набор от правила изцяло и да ги изграждате отначало, вместо да правите промените динамично. Това помага да се уверите, че активната конфигурация, която изследвате, действително отразява набора от команди във вашия конфигурационен скрипт.

Да хвърлим бърз поглед как би изглеждал един ръчен тест на нашия прост пример с *ipchains*. Спомнете си, че локалната мрежа в примера беше 172.16.1.0 с мрежова маска 255.255.255.0, а ние трябваше да позволим изходящи TCP връзки към web сървъри в мрежата. През нашата верига за препредаване не трябваше да преминава нищо друго. Започнете с предаването, което знаем, че би следвало да работи – връзка от локален хост към външен web сървър:

```
# ipchains -C forward -p tcp -s 172.16.1.0 1025 -d 44.136.8.2 80 -i
# eth0
accepted
```

Забележете аргументите, които трябва да бъдат подадени и начина, по който те се използват, за да се опише една дейтаграма. Изходът от командата показва, че дейтаграмата е приета за препредаване, което е това, което сме се надявали да стане.

Сега опитайте друг тест, този път с адрес на изпращача, който не принадлежи на нашата мрежа. Тази дейтаграма трябвала да бъде отказана:

```
# ipchains -C forward -p tcp -s 172.16.2.0 1025 -d 44.136.8.2 80 -i
# eth0
denied
```

Пробвайте още няколко други теста, този път със същите детайли като първия тест, но с различни протоколи. Те би трябвало също да бъдат отказани:

```
# ipchains -C forward -p udp -s 172.16.1.0 1025 -d 44.136.8.2 80 -i
# eth0
denied
# ipchains -C forward -p icp -s 172.16.1.0 1025 -d 44.136.8.2 80 -i
# eth0
denied
```

Пробвайте друг порт на получателя, който също очаквате да бъде отхвърлен:

```
# ipchains -C forward -p tcp -s 172.16.1.0 1025 -d 44.136.8.2 23 -i
# eth0
denied
```

Ако проектирате серия от изчерпателни тестове, ще изминете дълъг път до постигането на спокойствие на ума си. Макар че понякога това може да бъде толкова трудно, както създаването на самата конфигурация на защитна стена, то е много добър начина да научите, че вашата разработка осигурява сигурността, която очаквате от нея.

Примерна конфигурация на защитна стена

Дотук разгледахме основите на конфигурацията на защитна стена. Сега нека видим как може да изглежда в действителност една конфигурация на защитна стена.

Конфигурацията в този пример е проектирана така, че да бъде лесно разширена и приспособена за вашите изисквания. Ние осигурихме три версии. Първата версия е реализирана с командата *ipfwadm* (или скрипта *ipfwadm-wrapper*), втората използва *ipchains*, а третата – *iptables*. Примерът не се опитва да използва потребителски дефинирани вериги, но ще покаже сходствата и разликите между синтаксиса на старите и новите средства за конфигуриране на защитна стена.

```
#!/bin/bash
##############################################################################
# ВЕРСИЯ С IPFWADM
# Тази примерна конфигурация е за конфигурация на защитна стена от един
# хост
# без поддръжка на услуги, работещи на самия хост-защитна машина
##############################################################################

# КОНФИГУРИРАНА ОТ ПОТРЕБИТЕЛЯ ЧАСТ

# Името и разположението на инструмента ipfwadm.
# Използва ipfwadm-wrapper за ядрата 2.2.*
IPFWADM=ipfwadm

# Пътят до изпълнимата програма ipfwadm.
PATH="/sbin"

# Адресното пространство на нашата вътрешна мрежа и
# поддържащото го мрежово устройство.
```

```
OURNET="172.29.16.0/24"
OURBCAST="172.29.16.255"
OURDEV="eth0"

# Външният адрес и мрежовото устройство, което го поддържа.
ANYADDR="0/0"
ANYDEV="eth1"

# TCP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
TCPIN="smtp www"
TCPOUT="smtp www ftp ftp-data irc"

# UDP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
UDPIN="domain"
UDPOUT="domain"

# ICMP услугите, които искаме да преминават;
#   празно "" означава всички портове
# виж /usr/include/netinet/ip_icmp.h за номерата на типовете
# забележка: разделителят е интервал
ICMPIN="0 3 11"
ICMPOUT="8 3 11"

# Записване в дневник; махнете знака за коментар от следващия ред,
# за да разрешите записване на дейтаграмите, които са блокирани от
# защитната стена.
# LOGGING=1

# КРАЙ НА КОНФИГУРИРАНАТА ОТ ПОТРЕБИТЕЛЯ ЧАСТ
############################################################################

# Изчиства правилата във входната таблица
$IPFWADM -I -f

# Искаме по подразбиране входящия достъп да се отказва.
$IPFWADM -I -p deny

# Мамене (spoofing)
# Не трябва да приемаме отвън никаква дейтаграма с адрес на изпращача,
#   съвпадащ с нашите, затова ги отказваме
$IPFWADM -I -a deny -S $OURNET -W $ANYDEV

# SMURF
# Не допуска ICMP към нашия broadcast адрес, за да спре "Smurf" атака.
$IPFWADM -I -a deny -P icmp -W $ANYDEV -D $OURBCAST
```

```
# TCP
# Ще приемем всички TCP дейтаграми, принадлежащи към съществуваща
# връзка
#  преминаването.
# Това би следвало да прехване повече от 95% от всички валидни TCP
# пакети.
$IPFWADM -I -a accept -P tcp -D $OURNET $TCPIN -k -b

# TCP - ВХОДЯЩИ ВРЪЗКИ
# Ще приемем заявки за връзка отвън само за разрешените TCP портове
$IPFWADM -I -a accept -P tcp -W $ANYDEV -D $OURNET $TCPIN -y

# TCP - ИЗХОДЯЩИ ВРЪЗКИ
# Приемаме всички заявки за изходящи TCP връзки за разрешените TCP
# портове.
$IPFWADM -I -a accept -P tcp -W $OURDEV -D $ANYADDR $TCPOUT -y

# UDP - ВХОДЯЩИ
# Ще позволим входящи UDP дейтаграми за разрешените портове
$IPFWADM -I -a accept -P udp -W $ANYDEV -D $OURNET $UDPIN

# UDP - ИЗХОДЯЩИ
# Ще позволим изходящи UDP дейтаграми от разрешените портове
$IPFWADM -I -a accept -P udp -W $OURDEV -D $ANYADDR $UDPOUT

# ICMP - ВХОДЯЩИ
# Ще позволим входящи ICMP дейтаграми от разрешените типове
$IPFWADM -I -a accept -P icmp -W $ANYDEV -D $OURNET $ICMPIN

# ICMP - ИЗХОДЯЩИ
# Ще позволим изходящи ICMP дейтаграми от разрешените типове.
$IPFWADM -I -a accept -P icmp -W $OURDEV -D $ANYADDR $ICMPOUT

# По подразбиране и записване в дневник
# Всички други дейтаграми се обработват от подразбиращото се правило
# и се отхвърлят. Те ще се запишат, ако сте конфигурирали променливата
# LOGGING по-горе
#
if [ "$LOGGING" ]
then
   # Записване на недопустимите TCP пакети
   $IPFWADM -I -a reject -P tcp -o

   # Записване на недопустимите UDP пакети
   $IPFWADM -I -a reject -P udp -o

   # Записване на недопустимите ICMP пакети
   $IPFWADM -I -a reject -P icmp -o
fi
```

```
#
# край.
```

Сега ще реализираме защитната стена отново, използвайки командата *ipchains*:

```
#!/bin/bash
##############################################################################
# ВЕРСИЯ С IPCHAINS
# Тази примерна конфигурация е за конфигурация на защитна стена от един
# хост
# без поддръжка на услуги, работещи на самия хост-защитна машина
##############################################################################

# КОНФИГУРИРАНА ОТ ПОТРЕБИТЕЛЯ ЧАСТ

# Името и разположението на инструмента ipchains.
IPCHAINS=ipchains

# Пътят до изпълнимия файл ipchains.
PATH="/sbin"

# Адресното пространство на нашата вътрешна мрежа и
# поддържащото го мрежово устройство.
OURNET="172.29.16.0/24"
OURBCAST="172.29.16.255"
OURDEV="eth0"

# Външният адрес и мрежовото устройство, което го поддържа.
ANYADDR="0/0"
ANYDEV="eth1"

# TCP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
TCPIN="smtp www"
TCPOUT="smtp www ftp ftp-data irc"

# UDP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
UDPIN="domain"
UDPOUT="domain"

# ICMP услугите, които искаме да преминават;
#   празно "" означава всички портове
# виж /usr/include/netinet/ip_icmp.h за номерата на типовете
# забележка: разделителят е интервал
ICMPIN="0 3 11"
```

```
ICMPOUT="8 3 11"

# Записване в дневник; махнете знака за коментар от следващия ред,
# за да разрешите записване на дейтаграмите, които са блокирани от
# защитната стена.
# LOGGING=1

# КРАЙ НА КОНФИГУРИРАНАТА ОТ ПОТРЕБИТЕЛЯ ЧАСТ
################################################################################

# Изчиства правилата във таблицата input
$IPCHAINS -F input

# Искаме по подразбиране входящия достъп да се отказва.
$IPCHAINS -P input deny

# Мамене (spoofing)
# Не трябва да приемаме отвън никаква дейтаграма с адрес на
#   изпращача, съвпадащ с нашите, затова ги отказваме
$IPCHAINS -A input -s $OURNET -i $ANYDEV -j deny

# SMURF
# Не допуска ICMP към нашия broadcast адрес, за да спре "Smurf" атака.
$IPCHAINS -A input -p icmp -w $ANYDEV -d $OURBCAST -j deny

# Трябва да приемаме фрагменти; в ipchains това се указва изрично
$IPCHAINS -A input -f -j accept

# TCP
# Ще приемем всички TCP дейтаграми, принадлежащи към съществуваща
# връзка (т.е. с вдигнат бит ACK) за TCP портове, за които сме
# разрешили преминаването.
# Това би следвало да прехване повече от 95% от всички валидни TCP
# пакети.
$IPCHAINS -A input -p tcp -d $OURNET $TCPIN ! -y -b -j accept

# TCP - ВХОДЯЩИ ВРЪЗКИ
# Ще приемем заявки за връзка отвън само за разрешените TCP портове
$IPCHAINS -A input -p tcp -i $ANYDEV -d $OURNET $TCPIN -y -j accept

# TCP - ИЗХОДЯЩИ ВРЪЗКИ
# Приемаме всички заявки за изходящи TCP връзки за разрешените TCP
# портове.
$IPCHAINS -A input -p tcp -i $OURDEV -d $ANYADDR $TCPOUT -y -j accept

# UDP - ВХОДЯЩИ
# Ще позволим входящи UDP дейтаграми за разрешените портове
$IPCHAINS -A input -p udp -i $ANYDEV -d $OURNET $UDPIN -j accept

# UDP - ИЗХОДЯЩИ
```

```
# Ще позволим изходящи UDP дейтаграми от разрешените портове
$IPCHAINS -A input -p udp -i $OURDEV -d $ANYADDR $UDPOUT -j accept

# ICMP - ВХОДЯЩИ
# Ще позволим входящи ICMP дейтаграми от разрешените типове
$IPCHAINS -A input -p icmp -w $ANYDEV -d $OURNET $ICMPIN -j accept

# ICMP - ИЗХОДЯЩИ
# Ще позволим изходящи ICMP дейтаграми от разрешените типове.
$IPCHAINS -A input -p icmp -i $OURDEV -d $ANYADDR $ICMPOUT -j accept

# По подразбиране и записване в дневник
# Всички други дейтаграми се обработват от подразбиращото се правило
# и се отхвърлят. Те ще се запишат, ако сте конфигурирали променливата
# LOGGING по-горе
#
if [ "$LOGGING" ]
then
   # Записване на недопустимите TCP пакети
        $IPCHAINS -A input -p tcp -l -j reject

   # Записване на недопустимите UDP пакети
        $IPCHAINS -A input -p udp -l -j reject

   # Записване на недопустимите ICMP пакети
        $IPCHAINS -A input -p icmp -l -j reject
fi
#
# край.
```

В нашия пример с *iptables* преминаваме към използване на набора от правила `FORWARD`, поради разликата в значението на набора `INPUT` в реализацията на *netfilter*. Това има значение за нас; то означава, че никое от правилата не защитава самия хост-защитна стена. За да имитираме акуратно нашия пример с *ipchains*, би трябвало да повторим всяко от нашите правила във веригата `INPUT`. За яснота, вместо това игнорираме всички входящи дейтаграми, получени от нашия външен интерфейс.

```
#!/bin/bash
##############################################################################
# ВЕРСИЯ С IPTABLES
# Тази примерна конфигурация е за конфигурация на защитна стена от един
# хост
# без поддръжка на услуги, работещи на самия хост-защитна машина
##############################################################################

# КОНФИГУРИРАНА ОТ ПОТРЕБИТЕЛЯ ЧАСТ
```

```
# Името и разположението на инструмента iptables.
IPTABLES=iptables
# Пътят до изпълнимия файл iptables.
PATH="/sbin"

# Адресното пространство на нашата вътрешна мрежа и
# поддържащото го мрежово устройство.
OURNET="172.29.16.0/24"
OURBCAST="172.29.16.255"
OURDEV="eth0"

# Външният адрес и мрежовото устройство, което го поддържа.
ANYADDR="0/0"
ANYDEV="eth1"

# TCP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
TCPIN="smtp www"
TCPOUT="smtp www ftp ftp-data irc"

# UDP услугите, които искаме да преминават;
#   празно "" означава всички портове
# забележка: разделителят е интервал
UDPIN="domain"
UDPOUT="domain"

# ICMP услугите, които искаме да преминават;
#   празно "" означава всички портове
# виж /usr/include/netinet/ip_icmp.h за номерата на типовете
# забележка: разделителят е интервал
ICMPIN="0 3 11"
ICMPOUT="8 3 11"

# Записване в дневник; махнете знака за коментар от следващия ред,
# за да разрешите записване на дейтаграмите, които са блокирани от
# защитната стена.
# LOGGING=1

# КРАЙ НА КОНФИГУРИРАНАТА ОТ ПОТРЕБИТЕЛЯ ЧАСТ
##############################################################################

# Изчиства правилата във входящата таблица
$IPTABLES -F FORWARD

# Искаме по подразбиране да отказваме входящ достъп.
$IPTABLES -P FORWARD deny

# Игнорира всички получени отвън дейтаграми, насочени към този хост
```

```
$IPTABLE -A INPUT -i $ANYDEV -j DROP

# Мамене (spoofing)
# Не трябва да приемаме отвън никаква дейтаграма с адрес на
#   изпращача, съвпадащ с нашите, затова ги отказваме
$IPTABLES -A FORWARD -s $OURNET -i $ANYDEV -j DROP

# SMURF
# Не допуска ICMP към нашия broadcast адрес, за да спре "Smurf" атака.
$IPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $OURNET -j DENY

# Трябва да приемаме фрагменти; в iptables това се указва изрично
$IPTABLES -A FORWARD -f -j ACCEPT

# TCP
# Ще приемем всички TCP дейтаграми, принадлежащи към съществуваща
# връзка (т.е. с вдигнат бит ACK) за TCP портове, за които сме
# разрешили преминаването.
# Това би следвало да прехване повече от 95% от всички валидни TCP
# пакети.
$IPTABLES -A FORWARD -m multiport -p tcp -d $OURNET --dports $TCPIN /
    ! --tcp-flags SYN,ACK ACK -j ACCEPT
$IPTABLES -A FORWARD -m multiport -p tcp -s $OURNET --sports $TCPIN /
    ! --tcp-flags SYN,ACK ACK -j ACCEPT

# TCP - ВХОДЯЩИ ВРЪЗКИ
# Ще приемем заявки за връзка отвън само за разрешените TCP портове
$IPTABLES -A FORWARD -m multiport -p tcp -i $ANYDEV -d $OURNET $TCPIN /
    --syn -j ACCEPT

# TCP - ИЗХОДЯЩИ ВРЪЗКИ
# Приемаме всички заявки за изходящи TCP връзки за разрешените TCP
# портове.
$IPTABLES -A FORWARD -m multiport -p tcp -i $OURDEV -d $ANYADDR /
    --dports $TCPOUT --syn -j ACCEPT

# UDP - ВХОДЯЩИ
# Ще позволим входящи UDP дейтаграми за разрешените портове
$IPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -d $OURNET /
    --dports $UDPIN -j ACCEPT
$IPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -s $OURNET /
    --sports $UDPIN -j ACCEPT

# UDP - ИЗХОДЯЩИ
# Ще позволим изходящи UDP дейтаграми от разрешените портове
$IPTABLES -A FORWARD -m multiport -p udp -i $OURDEV -d $ANYADDR /
    --dports $UDPOUT -j ACCEPT
$IPTABLES -A FORWARD -m multiport -p udp -i $OURDEV -s $ANYADDR /
    --sports $UDPOUT -j ACCEPT
```

```
# ICMP - ВХОДЯЩИ
# Ще позволим входящи ICMP дейтаграми от разрешените типове
$IPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $OURNET /
    --dports $ICMPIN -j ACCEPT

# ICMP - ИЗХОДЯЩИ
# Ще позволим изходящи ICMP дейтаграми от разрешените типове.
$IPTABLES -A FORWARD -m multiport -p icmp -i $OURDEV -d $ANYADDR /
    --dports $ICMPOUT -j ACCEPT

# По подразбиране и записване в дневник
# Всички други дейтаграми се обработват от подразбиращото се правило
# и се отхвърлят. Те ще се запишат, ако сте конфигурирали променливата
# LOGGING по-горе
#
if [ "$LOGGING" ]
then
   # Записване на недопустимите TCP пакети
   $IPTABLES -A FORWARD -m tcp -p tcp -j LOG

   # Записване на недопустимите UDP пакети
   $IPTABLES -A FORWARD -m udp -p udp -j LOG

   # Записване на недопустимите ICMP пакети
   $IPTABLES -A FORWARD -m udp -p icmp -j LOG
fi
#
# край.
```

В много прости ситуации, за да използвате горния образец, трябва просто да редактирате раздела, озаглавен "Конфигурирана от потребителя част", за да определите кои протоколи и типове дейтаграми искате да пропускате навътре и навън. За по-сложни конфигурации ще се наложи да редактирате и раздела в края. Не забравяйте, че това е прост пример, затова ако го използвате в своята реализация на защитна стена, разгледайте го колкото се може по-внимателно, за да сте сигурни, че върши това, което искате.

ГЛАВА 10

IP СЧЕТОВОДСТВО

В днешния свят на комерсиални Интернет услуги става все по-важно да знаете колко данни предавате и получавате по вашите мрежови връзки. Ако сте доставчик на Интернет услуги и вашите клиенти заплащат според обема на прехвърлените данни, това би било от първостепенно значение за вашия бизнес. Ако сте клиент на доставчик на Интернет услуги, при който заплащането е според обема на данните, ще ви бъде от полза да събирате собствена информация, за да сте сигурни в правилността на вашата сметка за Интернет.

Отчитането на обема данни по мрежата има и други приложения, които нямат нищо общо с доларите и сметките. Ако управлявате сървър, който предлага голям брой различни мрежови услуги, вероятно ще ви бъде полезно да знаете точно колко данни генерира всяка една услуга. Такъв вид информация може да ви помогне при вземането на някои решения, като например какъв хардуер да закупите или колко сървъра да използвате.

Ядрото на Linux осигурява средство, което ви позволява да събирате всевъзможна полезна информация за мрежовия трафик, който минава през него. Тази възможност се нарича *IP счетоводство*.

Конфигуриране на ядрото за IP счетоводство

Възможността за IP счетоводство на Linux е много тясно свързана със софтуера, реализиращ защитна стена за Linux. Местата, от които искате да събирате отчетни данни, са същите места, на които бихте искали да осъществите филтриране със защитната стена: на входа и на изхода на мрежовия хост и в софтуера, който маршрутизира дейтаграмите. Ако не сте прочели главата за защитната стена, може би сега е подходящият момент да направите това, защото ще използваме някои от концепциите, описани в Глава 9, *Защитна стена за TCP/IP*.

За да активирате IP счетоводството на Linux, първо трябва да видите дали ядрото на вашия Linux е конфигурирано за това. Проверете дали съществува файлът */proc/net/ip_acct*. Ако го има, вашето ядро вече поддържа IP счетоводство. Ако го няма, трябва да създадете ново ядро, като за ядрата от серии 2.0 и 2.2 отговорите с "Y" на следните опции:

```
Networking options --->
   [*] Network firwalls
   [*] TCP/IP networking
       …
   [*] IP accounting
```

а за ядрата от серия 2.4 отговорите с "Y" на опцията:

```
Networking options --->
   [*] Network packet filtering (replace ipchains)
```

Конфигуриране на IP счетоводство

Тъй като IP счетоводството е тясно свързано със защитната стена за IP, за конфигурирането на IP счетоводство се използва същият инструмент, като за защитна стена. Така че за конфигурирането на IP счетоводство се използват *ipfwadm*, *ipchains* или *iptables*. Командният синтаксис е много близък до този на правилата за защитната стена, затова няма да се съсредоточаваме върху него, но ще обсъдим какво можете да откриете за характера на вашия мрежов трафик, използвайки тази възможност.

Общият синтаксис за IP счетоводство с *ipfwadm* е:

```
# ipfwadm -A [посока] [команда] [параметри]
```

Аргументът посока е нов. Той се кодира просто като `in`, `out` или `both` (навътре, навън или в двете направления). Тези посоки са от гледна точка на самата Linux машина, затова in означава данни, влизащи в машината през мрежовата връзка, а `out` означава данни, които се предават от този хост през мрежовата връзка. Направлението `both` е обединение от влизащото и излизащото направления.

Общият команден сингаксис за *ipchains* и *iptables* е:

```
# ipchains -A верига спецификация-на-правило
# iptables -A верига спецификация-на-правило
```

Командите *ipchains* и *iptables* ви позволяват да определите посоките по един по-близък начин до правилата за защитна стена. IP Firewall Chains не ви позволява да конфигурирате правило, което обединява две посоки, но позволява формулирането на правила в препредаващата верига, което по-старите реализации не можеха. Ще разгледаме произтичащите от това разлики чрез някои примери малко по-късно.

Командите до голяма степен са като правилата за защитна стена, като изключим, че тук не се прилагат правила на политиката. Можем да добавяме, вмъкваме, изтриваме и преглеждаме правилата за счетоводство. В случая на *ipchains* и *iptables*, всички валидни правила са счетоводни правила и всяка команда, за която не е зададена опцията *–j*, извършва само счетоводство.

Параметрите за задаване на правила за IP счетоводство са същите, като тези за IP защитни прегради. Те се използват, когато определяме точно какъв мрежов трафик искаме да анализираме.

Счетоводство по адрес

Ще използваме един пример, за да илюстрираме как може да се използва IP счетоводството.

Представете се, че имате базиран на Linux маршрутизатор, който обслужва два отдела във Виртуалната Пивоварна. Маршрутизаторът има две Ethernet устройства - *eth0* и *eth1*, всяко от които обслужва един отдел, и PPP устройството *ppp0*, което през високоскоростна серийна линия ни свързва към основния корпус на Университета Groucho Marx.

Освен това, за да разпределим бюджета си, искаме да знаем общия трафик, генериран от всеки от отделите по серийната линия, а за целите на управлението искаме да знаем общия трафик, генериран между двата отдела.

В следващата таблица са показани адресите на устройствата, които ще използваме в нашия пример:

интерфейс	адрес	мрежова маска
eth0	172.16.3.0	255.255.255.0
eth1	172.16.4.0	255.255.255.0

За да отговорим на въпроса "Колко данни генерира всеки отдел по PPP линията?", можем да използваме правило, което изглежда по следния начин:

```
# ipfwadm -A both -a -W ppp0 -S 172.16.3.0/24 -b
# ipfwadm -A both -a -W ppp0 -S 172.16.4.0/24 -b
```

или:

```
# ipchains -A input -i ppp0 -d 172.16.3.0/24
# ipchains -A output -i ppp0 -s 172.16.3.0/24
# ipchains -A input -i ppp0 -d 172.16.4.0/24
# ipchains -A output -i ppp0 -s 172.16.4.0/24
```

а ако използваме *iptables*:

```
# iptables -A FORWARD -i ppp0 -d 172.16.3.0/24
# iptables -A FORWARD -o ppp0 -s 172.16.3.0/24
# iptables -A FORWARD -i ppp0 -d 172.16.4.0/24
# iptables -A FORWARD -o ppp0 -s 172.16.4.0/24
```

Първата половина на всеки от тези комплекти правила означава: "Преброй всички данни, преминаващи и в двете посоки през интерфейса, наречен ppp0, с адрес на източника или получателя `172.16.3.0/24`" (спомнете си функцията на флага *-b* в *ipfwadm* и *iptables*). Втората половина на всеки комплект правила е също като първата, но се отнася за втората Ethernet мрежа в нашия сайт.

За да отговорим на втория въпрос "Колко данни преминават между двата отдела?", се нуждаем от правило, което изглежда например така:

```
# ipfwadm -A both -a -S 172.16.3.0/24 -D 172.16.4.0/24 -b
```

или:

```
# ipchains -A forward -s 172.16.3.0/24 -d 172.16.4.0/24 -b
```

или:

```
# iptables -A FORWARD -s 172.16.3.0/24 -d 172.16.4.0/24
# iptables -A FORWARD -s 172.16.4.0/24 -d 172.16.3.0/24
```

Тези правила ще броят всички дейтаграми с изгочник, принадлежащ на мрежата на единия от отделите и получател, принадлежащ на мрежата на другия отдел.

Счетоводство по порт на услуга

Добре, да предположим, че искаме освен това да имаме по-добра представа за това какъв е трафика, който се пренася през нашата PPP връзка. Можем, например да се интересуваме каква част от връзката изразходват FTP, smtp и WWW услугите.

Един скрипт от правила, даващ възможност за събиране на такава информация, би могъл да изглежда по следния начин:

```
#! bin/sh
# Събира статистика на обема FTP, smtp и www данните,
# пренасяни през нашата PPP връзка, използвайки ipfwadm
#
ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 ftp ftp-data
ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 smtp
ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 www
```

или:

```
#! bin/sh
# Събира статистика на обема FTP, smtp и www данните,
# пренасяни през нашата PPP връзка, използвайки ipchains
#
ipchains -A input -i ppp0 -p tcp -s 0/0 ftp-data:ftp
ipchains -A output -i ppp0 -p tcp -d 0/0 ftp-data:ftp
ipchains -A input -i ppp0 -p tcp -s 0/0 smtp
ipchains -A output -i ppp0 -p tcp -d 0/0 smtp
ipchains -A input -i ppp0 -p tcp -s 0/0 www
ipchains -A output -i ppp0 -p tcp -d 0/0 www
```

или:

```
#! bin/sh
# Събира статистика на обема FTP, smtp и www данните,
# пренасяни през нашата PPP връзка, използвайки iptables
#
iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport ftp-data:ftp
iptables -A FORWARD -o ppp0 -m tcp -p tcp --dport ftp-data:ftp
iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport smtp
iptables -A FORWARD -o ppp0 -m tcp -p tcp --dport smtp
iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport www
iptables -A FORWARD -o ppp0 -m tcp -p tcp --dport www
```

Тази конфигурация има две интересни свойства. Първо, в нея сме указали протокол. Когато задаваме портове в нашите правила, трябва да укажем и протокола, защото TCP и UDP предоставят независими набори от портове. Тъй като всички тези услуги са TCP–базирани, ние ги задаваме като протокол. Второ, зададохме двете услуги `ftp` и

`ftp-data` с една команда. Командата *ipfwadm* ви позволява да задавате единични портове, серия от портове или произволен списък от портове. Командата *ipchains* позволява да задавате или единичен порт, или интервал от портове, което и направихме тук. Синтаксисът "`ftp-dat:ftp`" означава "портовете от ftp-data (20) до ftp (21)" и това бе начинът, по който кодирахме серия от портове както в *ipchains*, така и в *iptables*. Когато имате списък от портове в едно счетоводно правило, това означава, че обема на всички данни, получени през кой да е от портовете от списъка, ще бъде добавен към сумата за това правило. Както си спомнихме, че услугата FTP използва два порта – порт за команди и порт за пренасяне на данни, ние ги добавихме заедно към общия FTP трафик. И накрая, зададохме адреса на източника като "`0/0`", което е специален запис, съответстващ на всички адреси и се изисква при задаването на портове както от *ipfwadm*, така и от *ipchains*.

Можем да разширим малко втората точка, за да придобием друга представа за данните по нашата връзка. Нека сега си представим, че класифицираме FTP, SMTP и WWW трафика като основен трафик, а всичкия останал трафик като допълнителен. Ако ни интересува съотношението между основния и допълнителния трафик, бихме могли да направим нещо като това:

```
# ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 ftp ftp-data smtp www
# ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 1:19 22:24 26:79 81:32767
```

Ако сте разгледали внимателно вашия файл */etc/service*, вече сте видели, че второто правило съответства на всички портове освен `ftp`, `ftp-data`, `smtp` и `www`.

Как да направим това с командите *ipchains* и *iptables*, след като те допускат само един аргумент при спецификацията на порта? При счетоводството можем да използваме дефинирани от потребителя вериги също толкова лесно, както и при правилата за защитна верига. Вижте следния подход:

```
# ipchains -N a-essent
# ipchains -N a-noness
# ipchains -A a-essent -j ACCEPT
# ipchains -A a-noness -j ACCEPT
# ipchains -A forward -i ppp0 -p tcp -s 0/0 ftp-data:ftp -j a-essent
# ipchains -A forward -i ppp0 -p tcp -s 0/0 smtp -j a-essent
# ipchains -A forward -i ppp0 -p tcp -s 0/0 www -j a-essent
# ipchains -A forward -j a-noness
```

Тук създаваме две потребителски вериги – едната наречена `a-essent`, където поставяме отчетните данни за основните услуги, а

другата наречена `a-noness`, където събираме данните за допълнителните услуги. След това добавяме към веригата за препращане правила, които съответстват на нашите основни услуги и правим скок към веригата `a-essent`, където вече имаме само едно правило, което приема всички дейтаграми и ги изброява. Последното правило в нашата верига за препращане е правило, което прескача към нашата верига `a-noness`, където пак имаме само едно правило, което приема всички дейтаграми и ги изброява. Правилото, което прескача към веригата `a-noness`, няма да бъде достигнато от нито една от нашите основни услуги, тъй като те ще бъдат приети в тяхната собствена верига. Следователно нашите пресмятания за основните и допълнителните услуги ще са налице в правилата, съдържащи се в тези вериги. Това е само един подход, който бихте могли да използвате; има и други. Реализацията с *iptables* на същия подход би изглеждала така:

```
# iptables -N a-essent
# iptables -N a-noness
# iptables -A a-essent -j ACCEPT
# iptables -A a-noness -j ACCEPT
# iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport ftp-data:ftp -j
# a-essent
# iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport smtp -j a-essent
# iptables -A FORWARD -i ppp0 -m tcp -p tcp --sport www -j a-essent
# iptables -A FORWARD -j a-noness
```

Това изглежда достатъчно просто. За съжаление, има малък, но неизбежен проблем, когато се опитваме да извършим отчитане по тип на услугата. Сигурно си помните, че в една по-предна глава обсъждахме ролята, която играе MTU в TCP/IP мрежите. MTU определя размера на най-голямата дейтаграма, която може да бъде предавана през дадено мрежово устройство. Когато маршрутизатора получи дейтаграма, която е по-голяма от MTU на интерфейса, който трябва да я препрати, маршрутизаторът използва една възможност, наречена *фрагментация*. Маршрутизаторът разбива големите дейтаграми на малки парчета, не по-дълги от MTU на устройството, и след това предава тези парчета. Маршрутизаторът създава нови заглавия, които поставя в началото на всяко от тези парчета, а отдалечената машина ги използва, за да възстанови оригиналните данни. За съжаление, по време на процеса на фрагментация портът е загубен за всинки, освен за първия фрагмент. Това означава, че IP счетоводството не може правилно да преброи фрагментираните дейтаграми. Той може надеждно да изброи само първия фрагмент или нефрагментираните дейтаграми. Има един малък трик, който позволява *ipfwadm* и който ни гарантира, че макар че не сме в състояние да определим точно от кой порт са вторият и следващите фрагменти, можем поне да ги

преброим. Една по-ранна версия на счетоводния софтуер на Linux задаваше на фрагментите фалшив номер на порт 0xFFFF, така че да можем да ги броим. За да сме сигурни, че сме прехванали втория и следващите фрагменти, можем да използваме правило като следното:

```
# ipfwadm -A both -a -W ppp0 -P tcp -S 0/0 0xFFFF
```

Реализацията на IP веригите има малко по-сложно решение, но резултатът е съвсем същия. Ако използваме командата *ipchains*, бихме използвали:

```
# ipchains -A forward -i ppp0 -p tcp -f
```

а с *iptables*:

```
# iptables -A FORWARD -i ppp0 -m tcp -p tcp -f
```

Тези команди не ни показват кой е бил първоначалният порт на данните, но поне имаме възможност да видим колко от нашите данни са фрагментирани и сме в състояние да определим обема на трафика, който те генерират.

В ядрата 2.2 по време на компилиране можете да изберете опция, която премахва тези проблеми, ако вашата Linux машина работи като единствена точка за достъп за мрежата. Ако по време на компилирането на вашето ядро разрешите опцията IP: *allways defragment*, всички получавани дейтаграми ще бъдат повторно събрани от маршрутизатора на Linux преди маршрутизиране и препредаване. Тази опция се изпълнява преди защитния и счетоводния софтуер да видят дейтаграмата и поради това няма да има нужда да се занимавате с фрагменти. В ядрата 2.4 компилирайте и заредете модула *forward-fragment* на *netfilter*.

Преброяване на ICMP дейтаграми

Протоколът ICMP не използва номера на портове на услугата и поради това е малко по-тежко събирането на подробности за него. Този протокол използва голям брой различни типове дейтаграми. Много от тях са безобидни и нормални, докато други могат да бъдат забелязани само при специални условия. Понякога хора, разполагащи с твърде много време, се опитват злонамерено да прекъснат мрежовия достъп на даден потребител, като генерират голям брой ICMP съобщения. Това обикновено се нарича *наводняване с ping* (*ping flooding*). Макар че IP счетоводството не може да направи нищо за предотвратяването на този проблем (IP защитната стена все пак може да по-

могне!), можем поне да поставим правила за счетоводство на място, което ще ни покаже дали някой се опитва да направи това.

ICMP не използва портове, както е при TCP и UDP. Вместо това ICMP има типове на ICMP съобщения. Можем да създадем счетоводни правила за всеки тип ICMP съобщение. За да направим това, в счетоводните команди на *ipfwadm* вместо полето на порта поставяме ICMP съобщението и номера на типа. Типовете на ICMP съобщенията описахме в раздела "Типове ICMP дейтаграми", така че ако искате да си ги припомните, препрочетете този раздел.

Едно IP счетоводно правило за събиране на информация относно обема на ping-данните, които са изпратени към вас или които сте генерирали, би могло да изглежда така:

```
# ipfwadm -A both -a -P icmp -S 0/0 8

# ipfwadm -A both -a -P icmp -S 0/0 0
# ipfwadm -A both -a -P icmp -S 0/0 0xff
```

или с *ipchains*:

```
# ipchains -A forward -p icmp -s 0/0 8
# ipchains -A forward -p icmp -s 0/0 0
# ipchains -A forward -p icmp -s 0/0 -f
```

или с *iptables*:

```
# iptables -A FORWARD -m icmp -p icmp --sports echo-request

# iptables -A FORWARD -m icmp -p icmp --sports echo-reply
# iptables -A FORWARD -m icmp -p icmp -f
```

Първото правило събира информация за дейтаграми "ICMP Echo Request" (ping заявки), а второто правило събира информация за дейтаграми "ICMP Echo Reply" (ping отговори). Третото правило събира информация за фрагменти от ICMP дейтаграми. Това е похват, подобен на този, описан за фрагментирани TCP и UDP дейтаграми.

Ако зададете начален и/или краен адрес в правилата, можете да проследите откъде пристигат ping-заявките, като например дали те се генерират във вашата мрежа или извън нея. След като установите откъде идват измамните дейтаграми, можете да решите дали искате като предпазна мярка да поставите подходящи защитни правила или ще предприемете някакви други действия, например да се свържете със собственика на мрежата-нарушител, за да го уведомите за проблема, или може би дори ще прибегнете до юридически действия, ако проблемът е злонамерен акт.

Счетоводство по протокол

Сега нека си представим, че сме заинтересовани каква част от трафика по нашата линия е TCP, UDP или ICMP. Ще използваме следните правила:

```
# ipfwadm -A both -a -W ppp0 -P tcp -D 0/0
# ipfwadm -A both -a -W ppp0 -P udp -D 0/0
# ipfwadm -A both -a -W ppp0 -P icmp -D 0/0
```

или

```
# ipchains -A forward -i ppp0 -p tcp -d 0/0
# ipchains -A forward -i ppp0 -p udp -d 0/0
# ipchains -A forward -i ppp0 -p icmp -d 0/0
```

или

```
# iptables -A FORWARD -i ppp0 -m tcp -p tcp
# iptables -A FORWARD -o ppp0 -m tcp -p tcp
# iptables -A FORWARD -i ppp0 -m udp -p udp
# iptables -A FORWARD -o ppp0 -m udp -p udp
# iptables -A FORWARD -i ppp0 -m icmp -p icmp
# iptables -A FORWARD -o ppp0 -m icmp -p icmp
```

С такива правила на подходящото място може да бъде анализиран целият трафик, преминаващ през интерфейса `ppp0`, за да се определи дали това е TCP, UDP или ICMP трафик, а съответстващите на всеки трафик броячи ще бъдат актуализирани. Примерът за *iptables* разделя входящия от изходящия поток, защото неговият синтаксис изисква това.

Използване на резултатите от IP счетоводство

Прекрасно е да се събира цялата тази информация, но как в действителност можем да я видим? За да разгледаме събраните отчетни данни и конфигурираните правила за счетоводство, използваме нашите команди за конфигуриране на защитната стена, като искаме от тях да покажат списъка с нашите правила. Броячите на пакети и байтове се отпечатват в изхода за всяко от правилата.

Командите *ipfwadm*, *ipchains* и *iptables* се различават по това как работят с отчетните данни, затова ще ги разгледаме поотделно.

Извеждане на счетоводни данни с ipfwadm

Основният начин за извеждане на счетоводни данни с командата *ipfwadm* е следният:

```
# ipfwadm -A -l
IP accounting rules
 pkts bytes dir prot source               destination          ports
 9833 2345K i/o all  172.16.3.0/24      anywhere             n/a
56527   33M i/o all  172.16.4.0/24      anywhere             n/a
```

Това ще ни даде броя на пакетите, изпратени във всяка посока. Ако използваме разширен формат на изхода с опцията *-e* (тук не е показан, защото изходът е много широк за страницата), ще получим също така опциите и имената на използвания интерфейс. Смисълът на повечето от полетата в изведения текст се разбира от само себе си, но все пак бихме искали да поясним следните полета:

dir Посоката, в която се прилага правилото. Възможните стойности тук са `in`, `out` и `i/o` (навътре, навън и в двете посоки).

prot

Протоколите, за които се прилага правилото.

opt Кодирана форма на опциите, които използваме, когато извикваме *ipfwadm*.

ifname

Името на интерфейса, за който се прилага правилото.

ifaddress

Адресът на интерфейса, за който се прилага правилото.

По подразбиране *ipfwadm* показва броячите на пакети и байтовете в съкратен вид, като показанията са закръглени до най-близките хиляда (K) или милион (M) единици. Можем да поискаме показване на събраните данни с повишена точност с помощта на опцията за разширена информация:

```
# ipfwadm -A -l -e -x
```

Извеждане на счетоводни данни с ipchains

Командата *ipchains* няма да ни покаже счетоводните данни (стойностите на броячите на пакети и байтове), ако не зададем аргумента `-v`.

Най-простият начин за извеждане на отчетените данни с *ipchains* е следния:

```
# ipchains -L -v
```

Както при *ipfwadm*, и тук можем да покажем броячите на пакети и байтове, като използваме разширен формат за изхода. За целта командата *ipchains* използва аргументът –x:

```
# ipchains -L -v -x
```

Извеждане на счетоводни данни с iptables

Командата *iptables* работи по същия начин, както командата *ipchains*. И тук трябва да използваме *–v*, за да видим стойностите на броячите, когато извеждаме правилата. За да изведем счетоводните данни трябва да използваме:

```
# iptables -L -v
```

Също както при командата *ipchains*, можете да използвате аргумента –x, за да покажете изхода в разширен формат с по-точни стойности.

Нулиране на броячите

При IP счетоводството броячите ще се препълнят, ако ги оставим достатъчно дълго време. Ако те се препълнят, ще имате трудности при определянето на стойностите, на които те отговарят в действителност. За да избегнете този проблем, трябва периодично да анализирате счетоводните данни, да ги записвате и след това да нулирате броячите, започвайки събирането на счетоводна информация за следващия период на отчитане.

Командите *ipfwadm* и *ipchains* ви предоставят съвсем просто начин за осъществяване на това:

```
# ipfwadm -A -z
```

или:

```
# ipchains -Z
```

или:

```
# iptables -Z
```

Можете дори да комбинирате действията по извеждане на информацията и нулиране на броячите, за да сте сигурни, че няма да загубите счетоводни данни междувременно:

```
# ipfwadm -A -l -z
```

или:

```
# ipchains -L -Z
```

или:

```
# iptables -L -Z -v
```

Тези команди първо ще изведат отчетените данни и веднага след това ще нулират броячите, за да започнат броенето отново. Ако сте заинтересовани от регулярното събиране и използване на тази информация, вероятно ще поставите тези команди в скрипт, който записва и съхранява някъде изхода, и периодично ще изпълнявате този скрипт с помощта на командата *cron*.

Изчистване на набора от правила

Една последна команда, която може да бъде полезна, ви позволява да изчистите всички IP счетоводни правила, които сте конфигурирали. Това е полезно най-вече, когато искате радикално да промените вашия набор от правила без да рестартирате машината.

Аргументът `-f` в комбинация с командата *ipfwadm* ще изчисти всички правила от типа, който сте задали. Командата *ipchains* поддържа аргумента `-F`, който прави същото:

```
# ipfwadm -A -f
```

или:

```
# ipchains -F
```

или:

```
# iptables -F
```

Това изчиства всички конфигурирани от вас IP счетоводни правила, като премахва всички правила и ви спестява необходимостта да ги премахвате едно по едно. Забележете, че изчистването на правилата с *ipchains* не предизвиква премахването на потребителски дефинираните вериги, а само на правилата в тях.

Пасивно събиране на счетоводни данни

Един последен похват, който може би ще ви хареса: ако вашата Linux машина е свързана към Ethernet, можете да приложите счетоводни правила към всички данни от сегмента, а не само към онези, които машината ви предава или са предназначени за нея. Вашата машина пасивно ще слуша всички данни в сегмента и ще ги отчита.

Най-напред трябва да изключите IP препредаването на вашата Linux машина, така че тя да не се опитва да пренасочва дейтаграмите, които получава.* В ядрата 2.0.36 и 2.2 това се постига чрез:

```
# echo 0 >/proc/sys/net/ipv4/ip_forward
```

След това с помощта на командата *ifconfig* трябва да разрешите режима promiscuous на Ethernet интерфейса. Сега можете да създадете счетоводни правила, които позволяват събирането на информация за дейтаграмите, преминаващи през Ethernet, без изобщо да включвате вашата Linux машина в маршрута.

* Това не е много добра идея, ако вашата машина изпълнява ролята на маршрутизатор. Ако забраните IP препредаването, тя ще престане да маршрутизира! Правете това само на машина с един физически мрежов интерфейс.

ГЛАВА 11

IP МАСКИРАНЕ И ТРАНСЛИРАНЕ НА МРЕЖОВИ АДРЕСИ

Не е необходимо да имате добра памет, за да си припомните времето, когато само големите организации можеха да си позволят да имат няколко компютъра, свързани в локална мрежа (LAN). Днес цените за мрежова технология спаднаха толкова много, че се случиха две неща. Първо, в момента локалните мрежи са нещо обикновено, дори и в домашни условия. Без съмнение много потребители на Linux имат два или повече компютъра, свързани с някаква Ethernet мрежа. Второ, мрежовите ресурси, и особено IP адресите, сега са дефицитни и докато по-рано се използваха свободно, днес те се купуват и продават.

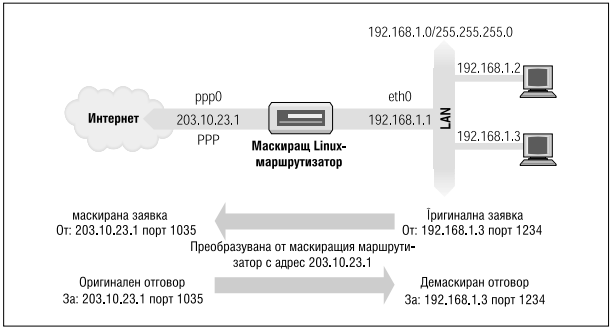
Повечето хора, работещи в локална мрежа, вероятно ще искат и всеки компютър от мрежата да има връзка с Интернет. Правилата за IP маршрутизация са много стриктни за това как се постъпва в такава ситуация. Традиционните решения на този проблем изискват резервиране на IP адрес на мрежа, може би от клас C за малки сайтове, определяне на адрес в тази мрежа за всеки хост от LAN и използването на маршрутизатор за свързването на локалната мрежа към Интернет.

В комерсиализираната Интернет среда това е едно твърде скъпо решение. Първо, ще се наложи да платите за мрежовия адрес, който ви се преотстъпва. Второ, вероятно ще трябва да платите на своя доставчик на Интернет за привилегията да се поддържа подходящ маршрут за достъп до вашата мрежа, така че останалата част от Интернет да знае как да достигне до вас. Това може би е приемливо за по-големите компании, но за домашни инсталации обикновено цената не е оправдана.

За щастие Linux предлага отговор на тази дилема. Този отговор включва един елемент от групата съвременни мрежови възможности, наречен *транслиране на мрежов адрес* (NAT – *Network Address Translation*). NAT описва процеса на модифициране на мрежовите адреси, съдържащи се в заглавната част на дейтаграмите, докато тези дейтаграми пътуват. В началото може да ви прозвучи странно, но ние ще ви покажем, че това е идеално решение на проблема, който описахме току-що и с който мнозина са се сблъсквали. Един от начините за транслиране на мрежов адрес се нарича IP маскиране, позволяващо на всички хостове в една частна мрежа да използват Интернет на цената на един-единствен IP адрес.

IP маскирането ви позволява да използвате частен (запазен) адрес на IP мрежа за вашата LAN, като вашият Linux-базиран маршрутизатор извършва в реално време някакво умело преобразуване на IP адреси и портове. Когато маршрутизаторът получи дейтаграма от компютър от локалната мрежа, той проверява какъв тип е дейтаграмата – "TCP," "UDP," "ICMP" и т.н., и я модифицира така, че тя изглежда като генерирана от самия маршрутизатор (едновременно с това, той запомня, че е направил това). След това дейтаграмата се изпраща в Интернет, като се използва единственият наличен IP адрес. Когато хостът-получател приеме тази дейтаграма, той вярва, че тя пристига от хоста-маршрутизатор и изпраща всички отговори на неговия адрес. Когато маскиращият Linux-маршрутизатор получи дейтаграма от своята Интернет връзка, той преглежда своята таблица с установени маскирани връзки, за да види дали всъщност тази дейтаграма е предназначена за компютър от локалната мрежа и ако е така, извършва обратната промяна на направената преди това и изпраща дейтаграмата на локалния компютър.

На Фигура 11-1 е илюстриран един прост пример.



Фигура 11-1. Типична конфигурация за IP маскиране

Имаме малка Ethernet мрежа, използваща един от запазените мрежови адреси. Мрежата има Linux-базиран маскиращ маршрутизатор, осигуряващ достъп до Интернет. Една от работните станции в мрежата (192.168.1.3) иска да установи връзка с отдалечения хост 209.1.106.178 на порт 8888. Работната станция изпраща своята дейтаграма на маскиращия маршрутизатор, който установява, че тази заявка за връзка изисква маскиране. Маршрутизаторът приема дейтаграмата и ѝ отделя работен порт (1035), поставя своите собствени адрес и номер на порта на мястото на тези на изпращащия хост и предава дейтаграмата на хоста, за който е предназначена. Хостът-получател смята, че е получил заявка за връзка от маскиращия Linux-хост и генерира дейтаграма-отговор. При получаване на тази дейтаграма маскиращият хост открива съответствието в своята таблица на маскираните връзки и извършва обратното заместване на това, което е направил в изходната дейтаграма. След това той хоста-маршрутизатор изпраща дейтаграмата-отговор на инициирлия връзката хост.

Локалният хост вярва, че е говорил директно с отдалечения хост. Отдалеченият хост не знае нищо за локалния хост и смята, че е комуникирал с маскиращия Linux-хост. Маскиращият Linux-хост знае, че тези два хоста говорят един с друг и на кои портове става това и извършва необходимите преобразувания на адреси и портове, за да се осъществи връзката.

Този метод може би изглежда малко объркващ и наистина може да бъде такъв, но работи и всъщност е доста прост за конфигуриране. Така че, не се тревожете, ако все още не сте разбрали всички подробности.

Странични ефекти и допълнителни ползи

Възможността за IP маскиране има свои собствени странични ефекти, някои от които са полезни, а други могат да бъдат неприятни.

Никой от хостовете от поддържаната мрежа зад маскиращия маршрутизатор не се вижда директно; оттук следва, че се нуждаете само от един валиден и маршрутизируем IP адрес, позволяващ на всички хостове да осъществяват мрежови връзки навън в Интернет. Това обаче има и обратна страна; никой от тези хостове не е видим от Интернет и не можете директно да се свържете с него от Интернет; единственият хост в маскираната мрежа, който се вижда, е самата маскираща машина. Това е важно, когато предвиждате услуги като поща или FTP. Трябва да определите какви услуги могат да се осигурят от маскиращия хост и на какви услуги той би могъл да бъде представител (proxy) или да третира специално по някакъв друг начин.

Второ, тъй като никой от маскиращите хостове не е видим, те са относително защитени от външни атаки; това може да опрости или дори да премахне необходимостта от конфигуриране на защитна стена на маскиращия хост. Все пак не бива да разчитате твърде много на това. Вашата мрежа ще бъде точно толкова защитена, колкото е защитен вашият маскиращ хост, така че, ако сте загрижени за сигурността, трябва да използвате защитна стена, за да го защитите.

Трето, IP маскирането ще има известно влияние върху работата на вашата мрежа. В типичните конфигурации това може би ще бъде едва доловимо. Обаче, ако имате голям брой активни маскирани сесии, ще установите, че обработката в маскиращата машина започва да влияе на производителността на вашата мрежа. За IP маскирането трябва да се извърши значително повече работа за всяка дейтаграма в сравнение с процеса на обикновеното маршрутизиране. Добрата стара 386SX16-ца, която планирате да използвате като маскираща машина за връзка към Интернет през телефонна линия, вероятно ще ви свърши работа, но не очаквайте много, ако решите да я използвате като маршрутизатор във вашата корпоративна мрежа с Ethernet скорости.

И накрая, някои мрежови услуги просто не искат да работят при маскиране, или поне не и без много помощ от ваша страна. Обикновено това са услуги, които разчитат на входящи сесии, за да работят, например такива са някои видове канали за директна комуникация (DCC – Direct Communications Channels), IRC възможности или определени типове видео и аудио услуги за много получатели. За някои от тези услуги са разработени специални модули на ядрото, осигуряващи решение на проблема; ще поговорим за това след малко. За други е възможно да не намерите поддръжка, така че трябва да знаете, че маскирането не е подходящо за всички случаи.

Конфигуриране на ядрото за IP маскиране

За да използвате възможността за IP маскиране, вашето ядро трябва да бъде компилирано с поддръжка на маскиране. Когато конфигурирате ядро от серия 2.2, трябва да изберете следните опции:

```
Networking options  --->
   [*] Network firewalls
   [*] TCP/IP networking
   [*] IP: firewalling
   [*] IP: masquerading
   --- Protocol-specific masquerading support will be built as modules.
     [*] IP: ipautofw masq support
     [*] IP: ICMP masquerading
```

Забележете, че част от поддръжката за маскиране се предоставя само като модули за ядрото. Това означава, че при компилиране на ядрото трябва да изпълните не само традиционното "`make zImage`", а и "`make modules`".

Ядрата от серия 2.4 вече не предлагат поддръжка на IP маскирането като опция по време на компилиране на ядрото. Вместо това, трябва да изберете опцията за филтриране на мрежови пакети:

```
Networking options --->
[M] Network packet filtering (replaces ipchains)
```

При ядрата от серия 2.2 по време на компилиране на ядрото се създават множество специфични за протокола помощни модули. Някои протоколи започват с изходяща заявка от един порт, а след това очакват входяща връзка на друг порт. Обикновено те не могат да бъдат маскирани, защото няма начин за асоцииране на втората връзка с първата без анализ на самите протоколи. Помощните модули правят точно това; всъщност те гледат вътре в дейтаграмите и позволяват

маскирането да работи за поддържаните протоколи, които в противен случай би било невъзможно да се маскират. Поддържаните протоколи са следните:

Модул	Протокол
ip_masq_ftp	FTP
ip_masq_irc	IRC
ip_masq_raudio	RealAudio
ip_masq_cuseeme	CU-See-Me
ip_masq_vdolive	VDO Live
ip_masq_quake	Quake на IdSoftware

За да използвате тези модули, трябва да ги заредите ръчно с командата *insmod*. Забележете, че тези модули не могат да бъдат заредени с помощта на демона *kemeld*. Всеки от тях приема аргумент, задаващ кои портове ще следи. Например, за модула RealAudio™ бихте могли да използвате:*

```
# insmod ip_masq_raudio.o ports=7070,7071,7072
```

Портовете, които трябва да зададете, зависят от протокола. В mini-HOWTO документа за IP маскиране, написан от Ambrose Au, се разглеждат по-подробно модулите за IP маскиране и начина, по който да ги конфигурирате.†

Пакетът *netfilter* включва модули, които изпълняват подобни функции. Например, за да се осъществи проследяване на връзките за FTP сесии, трябва да заредите и използвате модулите *ip_conntrack_ftp* и *ip_nat_ftp.o*.

Конфигуриране на IP маскиране

Ако вече сте прочели главите за защитната стена и счетоводството, вероятно няма да се изненадате, че командите *ipfwadm*, *ipchains* и *iptables* се използват също и при конфигуриране правилата за IP маскиране.

* RealAudio е запазена марка на Progressive Network Corporation.

† Можете да се свържете с Ambrose на адрес *ambrose@writeme.com*.

Правилата за маскиране са специален клас филтриращи правила. Можете да маскирате само дейтаграми, които са получени от един интерфейс и ще бъдат маршрутизирани към друг интерфейс. За да конфигурирате правило за маскиране, трябва да съставите правило, което е много близко до правилото за препредаване при защитна стена, но има специални опции, които указват на ядрото да маскира дейтаграмата. С командата *ipfwadm* се използва опцията *–m*, с *ipchains* се използва `–j MASQ`, а с *iptables* се използва `–j MASQUERADE`, за да се укаже, че дейтаграмите, съответстващи на спецификациите в правилото, трябва да бъдат маскирани.

Да разгледаме един пример. Един студент по компютърни науки от Университета Groucho Marx има у дома си няколко компютъра, свързани в малка локална Ethernet мрежа. Той е избрал за своята мрежа един от запазените частни адреси на мрежи. Той има съквартиранти, всеки от които иска да използва Internet. Тъй като студентите живеят в много скромни условия, те не могат да си позволят използването на непрекъсната Интернет връзка, затова вместо нея използват обикновена PPP връзка до Интернет през телефонна линия. Всичките биха искали да е възможно да използват връзката съвместно, за да чатват в IRC, да сърфират в Web и да теглят файлове директно чрез FTP на всеки от своите компютри. Решението е IP маскиране.

Студентът първо конфигурира една Linux машина, която поддържа комутируема връзка и работи като маршрутизатор за локалната мрежа. IP адресът, който получава тази машина при свързване през телефонната линия, не е важен. Студентът конфигурира Linux маршрутизатора с IP маскиране и използва един от частните адреси за своята локална мрежа: `192.168.1.0`. За всеки от хостовете в локалната мрежа се задава подразбиращ се път към Linux маршрутизатора.

Всичко, което се изисква да се направи, за да работи маскирането при описаната конфигурация, са следните команди на *ipfwadm*:

```
# ipfwadm -F -p deny
# ipfwadm -F -a accept -m -S 192.168.1.0/24 -D 0/0
```

или с *ipchains*:

```
# ipchains -P forward -j deny
# ipchains -A forward -s 192.168.1.0/24 -d 0/0 -j MASQ
```

или с *iptables*:

```
# iptables -t nat -P POSTROUTING DROP
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Сега вече, когато който и да е от локалните хостове се опита да се свърже към услуга на отдалечен хост, неговите дейтаграми ще бъдат автоматично маскирани от маскиращия Linux-маршрутизатор. Първото правило във всеки пример предпазва Linux машината от маршрутизиране на всякакви други дейтаграми и освен това добавя известна сигурност.

За да получите списък с правилата за маскиране, които сте създали, използвайте аргумента `-l` в командата *ipfwadm*, както описахме погоре при обсъждането на защитните стени.

За да изведем правилото, което сме създали по-рано, използваме:

```
# ipfwadm -F -l -e
```

което ще покаже на екрана приблизително нещо такова:

```
ipfwadm -F -l -e
IP firewall forward rules, default policy: accept
 pkts bytes type  prot opt  tosa tosx ifname  ifaddress   ...
    0     0 acc/m all  ---- 0xFF 0x00 any     any         ...
```

Текстът "*/m*" в изхода означава, че това е маскиращо правило.

За да получим списък с правилата за маскиране с командата *ipchains*, използваме аргумента *–L*. Ако изведем правилото, което по-рано сме създали с *ipchains*, резултатът ще изглежда по следния начин:

```
# ipchains -L
Chain input (policy ACCEPT):
Chain forward (policy ACCEPT):
target     prot opt     source               destination          ports
MASQ       all  ------  192.168.1.0/24       anywhere             n/a

Chain output (policy ACCEPT):
```

Всички правила с цел `MASQ` са правила за маскиране.

И накрая, за да получите списък на правилата с помощта на *iptables*, трябва да използвате:

```
# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
Chain POSTROUTING (policy DROP)

target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere            MASQUERADE
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

И тук маскиращите правила се появяват с цел `MASQUERADE`.

Задаване на времеви параметри за IP маскиране

При установяването на всяка нова връзка, софтуерът за IP маскиране създава в паметта асоциация между хостовете, включени във връзката. Можете да видите тези асоциации по всяко време като погледнете файла */proc/net/ip_masquerade*. Все пак, тези асоциации се премахват автоматично след изтичане на определено време без активност по тях.

Можете да зададете дължината на периода преди изключване с помощта на командата *ipfwadm*. Общият синтаксис е следния:

```
ipfwadm -M -s <tcp> <tcpfin> <udp>
```

а за командата *ipchains*:

```
ipchains -M -S <tcp> <tcpfin> <udp>
```

Реализацията с *iptables* използва по подразбиране по-дълги таймери и не позволява те бъдат настройвани.

Всяка от горните стойности съответства на таймер, използван от маскиращия IP софтуер, като показанията са в секунди. Следващата таблица обобщава таймерите и техните значения:

Име	Описание
tcp	Таймаут за TCP сесия. Колко дълго TCP връзката може да остане неизползвана, преди асоциацията за нея да бъде премахната.
tcpfin	Таймаут за TCP след FIN. Колко дълго ще се запази асоциацията след като TCP връзката бъде прекъсната.
udp	Таймаут за UDP сесия. Колко дълго UDP връзката може да остане неизползвана, преди асоциацията за нея да бъде премахната.

Управление на заявки към сървъра за имена

Управлението на заявки към сървъра за имена от хостовете в локална мрежа с IP маскиране винаги е представлявало проблем. Съществуват два начина за използване на DNS в условията на маскиране. Можете да укажете на всеки от хостовете да използва същия DNS, който използва Linux маршрутизатора, и да оставите IP маскирането да си върши работата и при DNS заявки. Другата възможност е в Linux машината да стартирате кеширащ сървър за имена, така че всеки от хостовете в локалната мрежа да използва Linux машината като свой DNS сървър. Въпреки че това е по-агресивния вариант, той може би е по-добрият избор, защото намалява обема на минаващия през Интернет връзката DNS трафик и повечето заявки ще се изпълняват малко по-бързо, понеже ще се обслужват от кеша. Обратната страна на медала при тази конфигурация е, че тя е по-сложна. Разделът "Конфигурация само за кеширане на named" в Глава 6, описва как да конфигурирате кеширащ сървър за имена.

Още за транслирането на мрежови адреси

Софтуерът *netfilter* е способен да извършва много различни видове транслиране на мрежови адреси. IP маскирането е само едно (при това просто) негово приложение.

Възможно е, например, да се изградят правила за транслиране на мрежови адреси (NAT правила), които транслират само определени адреси или серии от адреси и оставят всички други недокоснати, или правила за транслиране на адреси до стойност, взета от пул с адреси вместо транслирането само до един-единствен адрес, както става при маскиране. Всъщност, можете да използвате командата *iptables* за да генерирате NAT правила, задаващи практически всякакви преобразувания, в комбинации със съответствия, използващи произволен стандартен атрибут – адрес на изпращача, адрес на получателя, тип на протокола, номер на порта и т.н.

В документацията на *netfilter* транслирането на адреса на изпращача на дейтаграмата се обозначава като "Source NAT", или `SNAT`. Транслирането на адреса на получателя на дейтаграмата е известно като "Destination NAT", или `DNAT`. Транслирането на TCP или UDP порт е

известно с термина `REDIRECT`. `SNAT`, `DNAT` и `REDIRECT` са цели, които можете да използвате с командата *iptables*, за да построите по-сложни и по-съвършени правила.

Темата за транслирането на мрежови адреси и неговото използване изисква най-малко една отделна глава.* За съжаление в тази книга няма достатъчно място, за да обхванем темата в по-голяма дълбочина. Ако искате да научите повече за начина, по който можете да използвате транслирането на мрежови адреси, прочетете IPTABLES-HOWTO.

* ... а може би дори и цяла книга!

ГЛАВА 12

ВАЖНИ МРЕЖОВИ ВЪЗМОЖНОСТИ

След успешната настройка на IP и резолвера, трябва да решите какви услуги искате да предоставите през мрежата. Тази глава описва конфигурирането на някои прости мрежови приложения, включително сървъра *inetd* и програмите от фамилията *rlogin*. Освен това ще разгледаме накратко интерфейса за отдалеченото извикване на функции RPC, върху който се базират услуги като Мрежовата файлова система (NFS) и Мрежовата информационна система (NIS). Конфигурирането на NFS и NIS обаче е по-сложно и е описано в отделни глави, както и електронната поща и новините в мрежата.

Разбира се, в тази книга не можем да обхванем всички мрежови приложения. Ако искате да инсталирате някое приложение, което не е обсъдено тук, например *talk*, *gopher* или *http*, моля, обърнете се към справочните страници на сървъра за подробности.

Супер сървърът inetd

Програмите, които осигуряват приложни услуги през мрежата се наричат мрежови *демони*. Демонът е програма, която отваря порт, най-често добре известен (стандартен) порт за услуга и чака заявки на него. Ако се появи такава, демонът създава дъщерен процес, който приема връзката, а родителят продължава да чака следващи заявки. Този механизъм работи добре, но има някои недостатъци – например, поне един екземпляр от всички възможни услуги, които бихте желали да осигурите, трябва да е зареден в паметта по всяко време. Освен това, софтуерните функции, които реализират слушането и управлението на портовете, трябва да бъдат копирани във всеки мрежов демон.

За да се преодолеят тези недостатъци, повечето инсталации на UNIX използват специален мрежов демон, за който можете да мислите като за "супер сървър". Този демон създава гнезда (сокети), съответстващи на множество услуги и слуша на всичките тях едновременно. Когато на някое от гнездата се получи заявка за връзка, супер сървърът я приема и стартира копие на сървъра за съответния порт, като му предава гнездото за обработка. След това супер сървърът се връща към слушането.

Най-популярният супер сървър се нарича *inetd*, Интернет демонът. Той се стартира при зареждането на системата и взема списъка от услуги, които трябва да управлява, от стартов файл, наречен */etc/inetd.conf*. Освен тези сървъри, има няколко тривиални услуги, изпълнявани от самия *inetd*, които се наричат *вътрешни услуги*. Те включват *chargen*, която просто генерира символен низ, и *daytime*, която връща часа от деня според системния часовник.

Всеки запис в този файл се състои от един ред със следните полета:

```
услуга тип протокол изчакване потребител сървър
команден-ред
```

Всяко от тези полета е описано в следващия списък:

услуга
: Задава името на услугата. Това име трябва да бъде преобразувано в номер на порт, който се извлича от файла */etc/services*. Този файл ще бъде описан по-долу в тази глава в раздела "Файлове за услугите и протоколите".

тип
: Задава типа на гнездото, който може да бъде или `stream` (за протоколи с гарантирана доставка), или `dgram` (за дейтаграмни протоколи). Затова за базираните на TCP услуги трябва винаги да се използва `stream`, а базираните на UDP – `dgram`.

протокол
: Задава име на протокола за пренос, използван от услугата. То трябва да бъде валидно име на *протокол* от файла за протоколите, описан по-долу.

изчакване
: Тази опция се отнася само за гнезда от тип `dgram`. Възможните стойности са само `wait` и `nowait`. Ако е зададена стойност `wait`, при връзка към порта *inetd* изпълнява само един сървър за

зададения порт. В противен случай, супер сървъра веднага се връща към слушането на порта след като задейства съответния сървър.
Това е полезно за "еднонишковите" сървъри, които четат всички входящи дейтаграми, а когато те свършат, се изключват. Повечето RPC сървъри са от този тип и затова трябва да им бъде зададена опцията `wait`. Противоположният вид сървъри – "многонишковите", позволяват неограничен брой свои представители да работят едновременно. За тези сървъри трябва да бъде зададена опцията `nowait`.

За гнездата от тип `stream` трябва винаги да се използва `nowait`.

потребител
Това е login-идентификатора на потребителя, който ще бъде собственик на изпълнявания процес. Често това е потребителят **root**, но някои услуги могат да използват други акаунти. Много добра идея е тук да приложите принципа на най-ниската привилегия, според който не трябва да изпълнявате команда в привилегирован акаунт, ако програмата не го изисква за правилното си функциониране. Например, сървърът за новини NNTP работи с правата на потребителя **news**, а услуги, които могат да доведат до риск за сигурността (например *tftp* и *finger*), често работят като **nobody**.

сървър
Задава пълния път до програмата-сървър, която трябва да бъде изпълнена. Вътрешните услуги се маркират с ключовата дума `internal`.

команден-ред
Това е командният ред, който трябва да бъде предаден на сървъра. Той започва с името на сървъра, който трябва да бъде изпълнен, и може да включва всякакви аргументи, които трябва да му бъдат предадени. Ако използвате TCP обвивка (wrapper), тук трябва да зададете пълния път до сървъра. Ако не, трябва просто да зададете името на сървъра, както бихте искали да се появи в списъка с процесите. Ще поговорим за обвивката TCP след малко.

За вътрешните услуги това поле се оставя празно.

Един примерен файл *inetd.conf* е показан на Пример 12-1. Услугата *finger* е поставена в коментар и затова не е достъпна. Често това се прави от съображения за сигурност, защото тази услуга може да се

използва от злонамерени потребители, за да се сдобият с имената и други детайли на потребителите на вашата система.

Пример 12-1: Примерен файл /etc/inetd.conf

```
#
# услуги на inetd
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 /boot/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
#
#    вътрешни услуги на 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
```

Демонът *tftp* също е поставен в коментар. *tftp* реализира протокола TFTP (*Trivial File Transfer Protocol* – тривиален протокол за прехвърляне на файлове), който позволява на всеки да изтегля всички достъпни за четене от всеки файлове от вашата система, без да се извършва проверка на паролата. Това е особено опасно за файла */etc/passwd* и е дори още по-опасно, ако не използвате скриги пароли.

TFTP обикновено се използва от бездискови клиенти и X-терминали за изтегляне на необходимия им код от сървър за начално зареждане. Ако трябва да използвате *tftpd* поради тази причина, уверете се, че сте ограничили обсега на демона само до директориите, от които клиентите ще теглят файлове; трябва да добавите имената на тези директории към командния ред на *tftpd*. Това е показано в примера на втория ред с *tftp*.

Инструментът tcpd за управление на достъпа

Тъй като отварянето на компютъра за достъп през мрежата го излага на рискове, свързани със сигурността, създадени са приложения, които го предпазват от няколко типа атаки. В някои детайли на сигурността, обаче, може да има пропуски (най-драстичният пример е интернет-червеят RTM, който използва дупка в няколко програми, включително стари версии на демона за изпращане на поща sendmail), или да не се различават сигурните хостове, чиито заявки за дадена услуга трябва да бъдат приети, и несигурните хостове, чиито заявки трябва да бъдат отхвърлени. Вече се спряхме за кратко на услугите *finger* и *tftp*. Мрежовият администратор би искал да ограничи достъпа до тези услуги само за "доверени хостове", което е невъзможно при обичайното конфигуриране, в което *inetd* предоставя тази услуга или на всички клиенти, или на нито един.

Полезен инструмент за управление на достъпа според хоста-клиент е *tcpd*, често наричан демон-"обвивка".* За TCP услугите, които искате да наблюдавате или защитите, той се стартира вместо програмата-сървър. *tcpd* проверява дали е позволено отдалечения хост да използва услугата, и само ако това е така, стартира истинската програма-сървър. Освен това, *tcpd* съобщава за заявката на системния демон за поддръжка на дневник *syslog*. Забележете, че тази възможност не работи за услуги, базирани на UDP.

Например за да обвиете демона *finger*, трябва да промените съответния ред в *inetd.conf* от:

```
# необвит демон finger
finger stream tcp nowait bin /usr/sbin/fingerd in.fingerd
```

със следното:

```
# обвиване на демона finger
finger stream tcp nowait root /usr/sbin/tcpd in.fingerd
```

Без да добавяте контрол на достъпа, това ще изглежда на клиента като обичайната конфигурация на *finger*, с изключение на това, че всички заявки ще се отбелязват в дневника чрез интерфейса *auth* на *syslog*.

* Този демон е написан от Wietse Venema, *wietse@wzv.win.tue.nl*.

Управлението на достъпа се извършва чрез двата файла */etc/hosts.allow* и */etc/hosts.deny*. Те съдържат данни, които разрешават или отказват достъп до определени услуги и хостове. Когато *tcpd* обработва заявка за услуга като *finger* от хоста-клиент с име **biff.foobar.com**, той преглежда *hosts.allow* и *hosts.deny* (в този ред) за запис, съответстващ на услугата и хоста-клиент. Ако се открие съвпадение във файла *hosts.allow*, достъпът се разрешава и *tcpd* не сканира файла *hosts.deny*. Ако няма съответствие в *hosts.allow*, но има в *hosts.deny*, заявката се отхвърля като връзката се прекъсва. Ако изобщо не бъде открито съвпадение, заявката се приема.

Записите във файловете за достъп изглеждат по следния начин:

```
servicelist: hostlist [:shellcmd]
```

`servicelist` е списък от имена на услуги от */etc/services* или ключовата дума `ALL`. За да зададете всички услуги с изключение на *finger* и *tftp*, използвайте `ALL EXCEPT finger, tftp`.

`hostlist` е списък от имена на хостове, IP адреси или ключовите думи `ALL`, `LOCAL`, `UNKNOWN` или `PARANOID`. `ALL` съответства на всеки хост, а `LOCAL` – на всеки хост, чието име не съдържа точка.[+] `UNKNOWN` съответства на всички хостове, чиито имена или адреси не могат да бъдат открити. `PARANOID` съответства на всеки хост, за който при разпознаването на името не се получава неговия IP адрес.[#] Име, започващо с точка, съответства на всички хостове, чийто домейн е същия като това име. Например **.foobar.com** съответства на **biff.foobar.com**, но не и на **nurks.fredsville.com**. Шаблон, завършващ с точка, съответства на хостовете, чиито IP адреси започват със зададения шаблон, например **172.16.** съответства на **172.16.32.0**, но не и на **172.15.9.1**. Шаблон във формата `n.n.n.n/m.m.m.m` се разглежда като IP адрес и мрежова маска, затова с него можем да запишем предния пример като **172.16.0.0/255.255.0.0**. И накрая, всеки шаблон, започващ със сим-

[+] Обикновено само имената на локални хостове, получени чрез сканиране на файла */etc/hosts*, не съдържат точки

[#] Макар че името предполага, че това е една доста крайна мярка, ключовата дума `PARANOID` всъщност е добра стойност по подразбиране, защото може да ви предпази от злонамерени хостове, представящи се за някой, който всъщност не са. Не всички *tcpd* се разпространяват с компилирана опция `PARANOID`; ако вашият демон няма тази възможност, трябва да прекомпилирате *tcpd*, за да можете да я използвате.

вола "/" ви позволява да посочите файл, за който се предполага, че съдържа списък с имена на хостове или шаблони на IP адреси, за всеки от които може да се получи съответствие. Например при срещане на шаблон, изглеждащ по следния начин: */var/access /trustedhosts*, демонът *tcpd* ще прочете този файл, като проверява дали някой от редовете в него съответства на свързващия се хост.

За да забраните достъпа до *finger* и *tftp* на всички хостове, с изключение на локалните, поставете в */etc/hosts.deny* следното, като оставите */etc/ hosts.allow* празен:

```
in.tftpd, in.fingerd: ALL EXCEPT LOCAL, .вашия.домейн,
```

Незадължителното поле *shellcmd* може да съдържа команда на обвивката, която трябва да се изпълни, ако се получи съответствие в някой запис. Това е полезно, ако искате да поставите капани за потенциалните нападатели. Следващият пример създава log-файл за следене на потребителя и хоста, от който идва опита за връзка, и ако хоста не е **vlager.vbrew.com**, прибавя накрая изхода от командата *finger* за този хост:

```
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
```

Аргументите `%h` и `%d` се заменят от *tcpd* съответно с името на хоста-клиент и името на услугата. За повече подробности прочетете справочната страница *hosts_access(5)*.

Файлове за услугите и протоколите

Номерата на портовете, на които се предлагат някои "стандартни" услуги, са дефинирани в RFC-документа Assigned Numbers. За да позволите на сървърните и клиентските програми да превръщат имената на услугите в тези кодове, поне част от списъка трябва да се пази върху всеки хост; той се записва във файл, наречен */etc/services*. Записите в него се създават по следния начин:

услуга порт/протокол [*псевдоними*]

В полето *услуга* се задава името на услугата, *порт* дефинира порта, на който се предлага услугата, а *протокол* задава кой транспорген протокол се използва. Обикновено стойността в последното поле е *udp* или *tcp*. Възможно е някоя услуга да се предоставя с повече от

един протокол, както и предлагането на различни услуги на един и същи порт, стига протоколите да са различни. Полето *псевдоними* ви позволява да зададете алтернативни имена за една и съща услуга.

Обикновено не се налага да променяте файла с услугите, който се разпространява с мрежовия софтуер за вашата Linux система. Въпреки това, ще ви покажем малка извадка от този файл в Пример 12-2.

Пример 12-2: Примерен файл /etc/services

```
# Файл за услугите
#
# стандартни услуги
echo        7/tcp                   # Exo
echo        7/udp                   #
discard     9/tcp sink null         # Отказ
discard     9/udp sink null         #
daytime    13/tcp                   # Време на деня
daytime    13/udp                   #
chargen    19/tcp ttytst source     # Генератор на символи
chargen    19/udp ttytst source     #
ftp-data   20/tcp                   # Протокол за прехвърляне на файлове
                                    # (данни)
ftp        21/tcp                   # Протокол за прехвърляне на файлове
                                    # (контрол)
telnet     23/tcp                   # Виртуален терминален протокол
smtp       25/tcp                   # Прост протокол за прехвърляне на поща
nntp      119/tcp readnews          # Протокол за прехвърляне на мрежови
                                    # новини
#
# UNIX услуги
exec       512/tcp                  # отдалечено изпълнение (BSD rexec)
biff       512/udp comsat           # известие за получена поща
login      513/tcp                  # отдалечено влизане
who        513/udp whod             # отдалечено извикване на who и uptime
shell      514/tcp cmd              # отдалечена команда без използване на
                                    # парола
syslog     514/udp                  # отдалечено добавяне в дневник
printer    515/tcp spooler          # отдалечено изпращане към принтера
route      520/udp router routed    # протокол за информация за
                                    # маршрутизиране
```

Обърнете внимание, че услугата *echo* се предлага на седми порт както за TCP, така и за UDP, а порт 512 се използва за 2 различни услуги: отдалечено изпълнение (*rexec*) чрез TCP и демона *COMSAT*, който уведомява потребителите за получена нова поща чрез UDP (вж. *xbiff(1x)*).

Аналогично на файла с услугите, мрежовата библиотека се нуждае от начин да превежда имената на протоколите – например тези, използвани във файла с услугите – в такива, които се разбират от IP слоя на другите хостове. Това се прави като се търси името на протокола във файла */etc/protocols*. Той съдържа по един запис на ред, всеки от които се състои от име на протокол и съответният му номер. Почти изключено е да се наложи да промените нещо в този файл, това е по-малко вероятно дори от променянето на */etc/services*. Един такъв файл е показан в Пример 12-3.

Пример 12-3: Примерен файл /etc/protocols

```
#
# Интернет (IP) протоколи
#
ip    0    IP        # интернет протокол, псевдо-номер на протокола
icmp  1    ICMP      # протокол за управляващи съобщения
igmp  2    IGMP      # протокол за изпращане на съобщения до група
tcp   6    TCP       # протокол за управление на предаването
udp   17   UDP       # протокол за потребителски дейтаграми
raw   255  RAW       # RAW IP интерфейс
```

Отдалечено извикване на процедури

Общият механизъм за клиент-сървър приложения се осигурява от пакета RPC (*Remote Procedure Call* – отдалечено извикване на процедури). RPC бе разработен от Sun Microsystems и представлява колекция от инструменти и библиотечни функции. Важни приложения, изградени върху RPC, са NIS (*Network Information System* – мрежова информационна система – виж Глава 13, *Мрежова информационна система*) и NFS (*Network File System* – мрежова файлова система – виж Глава 14, *Мрежова файлова система*); и двете приложения са описани в тази книга.

Един RPC сървър се състои от колекция от процедури, които могат да бъдат извиквани от клиента чрез изпращане на RPC заявка до сървъра, заедно с параметрите на процедурата. Сървърът ще извика зададената процедура от името на клиента и ще му предаде върнатата стойност, ако има такава. За да бъдат машинно-независими, всички данни, обменяни между клиента и сървъра, се преобразуват във формат *XDR* (*External Data Representation* – външно представяне на данни) от изпращача и отново се преобразуват във вида на представяне в съответния компютър от получателя. RPC използва стандартните UDP и TCP гнезда за пренос на данни в XDR формат до отдале-

чения хост. Sun бяха така любезни да направят RPC публично достояние; пакета е описан в серия от RFC-документи.

Понякога усъвършенстванията в дадено RPC приложение водят до несъвместими промени на интерфейса за извикване на процедури. Разбира се, ако просто промените сървъра, всички приложения, които разчитат на първоначалното поведение, ще спрат да работят. Затова, на RPC програмите се задава номер на версията, като обикновено се започва с 1 и при всяка нова версия на RPC интерфейса брояч ът се увеличава. Често даден сървър може да предлага няколко версии едновременно; тогава потребителите указват чрез номера на версията в заявката си, коя реализация на услугата искат да използват.

Общуването между RPC сървърите и клиентите е малко особено. RPC сървърът предлага един или повече комплекта процедури, всеки от които се нарича *програма* и еднозначно се определя от *номер на програма*. Списък, който свързва името на услугите с номера на програмата, обикновено се пази във файла */etc/rpc*, извадка от който е показана в Пример 12-4.

Пример 12-4: Примерен файл /etc/rpc

```
#
# /etc/rpc - различни услуги, базирани на 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
yppasswdd     100009 yppasswd
bootparam     100026
ypupdated     100028 ypupdate
```

В TCP/IP мрежите, авторите на RPC трябва да решат проблема за намиране на съответствие между номера на програма и обща мрежова услуга. Те проектират всеки сървър да предоставя както TCP, така и UDP порт за всяка програма и всяка версия. Обикновено RPC приложенията използват UDP, когато изпращат данни, и се връщат към TCP само, когато данните за изпращане не се събират в един UDP пакет.

Разбира се, клиентските програми трябва да открият кой номер на програма на кой порт съответства. Използването на конфигурационен файл за тази цел би било твърде негъвкаво решение; RPC приложенията не използват запазени портове и затова няма гаранция, че порт, първоначално предназначен да бъде използван от нашето приложение за бази от данни, няма да бъде зает от друг процес. Затова RPC приложенията избират произволен свободен порт и го регистрира със специална програма, наречена *portmapper демон* (буквално демон за съответствието с порта). Този демон играе ролята на посредник за услугите на всички RPC сървъри, работещи на неговата машина. Клиент, който иска да извика услуга с даден програмен номер, трябва първо да изпрати запитване до portmapper демона на хоста със сървъра, който връща като резултат номерата на TCP и UDP портовете, през които може да бъде достигната услугата.

Този метод има само един недостатък, който много прилича на аналогичния проблем при демона *inetd* за стандартните услуги на Бъркли. В случая, обаче, проблема е дори малко по-голям, защото когато демона portmapper умре, цялата информация за RPC портовете ще бъде загубена; това обикновено означава, че трябва да рестартирате ръчно всички RPC сървъри или да рестартирате цялата машина.

В Linux portmapper демона се нарича */sbin/portmap* или понякога */usr/sbin/rpc.portmap*. Не е нужно да извършвате каквото и да е конфигуриране на демона, освен да се уверите, че се зарежда от стартовите скриптове за поддръжка на мрежа.

Конфигуриране на отдалеченото влизане и изпълнение на команди

Често е много полезно да изпълните команда на отдалечен хост и да пренасочите входа и изхода от тази команда към мрежова връзка.

Традиционните команди, използвани за изпълнение на команди на отдалечени хостове, са *rlogin*, *rsh* и *rcp*. Видяхме пример на командата *rlogin* в Глава 1, *Въведение в работата в мрежа*, в частта "Въведение в TCP/IP мрежите". Накратко обсъдихме въпросите на сигурността, свързани с тази команда в "Сигурност на системата" и предложихме замяната и със *ssh*. Пакетът *ssh* предоставя заместители, наречени *slogin*, *ssh* и *scp*.

Всяка от тези команди поражда нов процес-обвивка върху отдалечения хост и позволява на потребителя да изпълнява команди. Разбира

се, клиентът трябва да има акаунт на отдалечения хост, където ще се изпълнява командата. По този начин, всяка от тези команди ще премине през процес за удостоверяване на самоличността. *r*-командите използват проста обмяна без кодиране на потребителско име и парола между хостовете, така че всеки, който следи комуникационния канал, може лесно да прехване паролите. Командите от сюитата *ssh* осигуряват по-високо ниво на сигурност: те използват техника, наречена *Криптография с публичен ключ*, която предоставя удостоверяване на самоличността и шифриране между хостовете, за да гарантира, че нито паролите, нито данните от сесията могат лесно да се прехванат от други хостове.

Възможно е още повече да намалите проверките за автентичност за определени потребители. Например, ако често ви се налага да влизате в други машини от локалната ви мрежа, вероятно бихте искали да ви допускат, без да се налага да всеки път да въвеждате паролата си. Това винаги е било възможно с *r*-командите, но комплекта *ssh* ви позволява да го правите малко по-лесно. Все пак, това не е много добра идея, защото ако някой злонамерен човек проникне в акаунт в даден компютър, може да получи достъп до всички други акаунти, за които съответния потребител е избрал да влиза без парола, но тъй като тази възможност е много удобна, хората я използват.

Нека сега поговорим за премахването на *r*-командите и заменянето им със *ssh*.

Забраняване на r*-командите*

Започнете с премахването на *r*-командите, ако са инсталирани. Най-лесният начин да забраните старите *r*-команди е да поставите в коментар (или да изтриете) записите им във файла */etc/inetd.conf*. Съответните записи биха изглеждали приблизително по следния начин:

```
# Shell, login, exec и talk са BSD протоколи
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
```

Може да ги коментирате като поставите символа # в началото на всеки ред, или да изтриете реда напълно. Не забравяйте, че за да влязат промените в сила, трябва да рестартирате демона *inetd*. Най-добре би било да премахнете и самите демони.

Инсталиране и конфигуриране на ssh

OpenSSH е свободна версия на пакета програми ssh; адаптацията им за Linux може да бъде открита на адрес *http://violet.ibs.com.au/openssh/* и в повечето модерни дистрибутори на Linux.*

Тук няма да се спираме на компилирането; в изходния код са включени достатъчно добри указания. Ако имате възможност, инсталирайте софтуера от предварително компилиран пакет.

Във всяка *ssh*-сесия участват две програми. Едната е *ssh*-клиент, който трябва да конфигурирате и стартирате на локалния хост, а другата – *ssh*-демон, който трябва да работи на отдалечения хост.

ssh демон

Демонът *sshd* е програма, която очаква мрежови връзки от *ssh*-клиенти, управлява проверката на самоличността и изпълнява заявената команда. Демонът използва един основен конфигурационен файл, наречен */etc/ssh/sshd_config* и специален файл, съдържащ ключ, който се използва от процесите за удостоверяване на самоличността и шифриране на връзката, за да представи конкретния хост. Всеки хост и всеки клиент имат свой собствен ключ.

Пакетът съдържа програма, наречена *ssh-keygen*, която се използва за генериране на случаен ключ. Най-често това се прави веднъж, по време на инсталирането, за да се генерира ключ на хоста, и обикновено системният администратор записва този ключ във файл, наречен */etc/ssh/ssh_host_key*. Ключовете могат да бъдат с произволна дължина, но не по-малка от 512 бита. По подразбиране *ssh-keygen* генерира ключове с дължина 1024 бита и повечето хора използват този модел. За да генерирате случаен ключ, трябва да въведете командата *ssh-keygen* по следния начин:

```
# ssh-keygen -f /etc/ssh/ssh_host_key
```

Ще ви бъде поискано да въведете тайна фраза. Тъй като ключовете за хостовете не трябва да съдържат такава фраза, просто натиснете клавиша Return, без да въвеждате нищо. Изходът от програмата ще изглежда по следния начин:

* OpenSSH бе разработена от проекта OpenBSD и е чудесен пример за ползата от свободния софтуер.

```
Generating RSA keys: ......ooooo0.............................ooooo0
Key generation complete.
Enter passphrase (празно, ако не искате да въвеждате фраза):
Enter same passphrase again:
Your identification has been saved in /etc/ssh/ssh_host_key
Your public key has been saved in /etc/ssh/ssh_host_key.pub
The key fingerprint is:
1024 3a:14:78:8e:5a:a3:6b:bc:b0:69:10:23:b7:d8:56:82 root@moria
```

Накрая ще видите, че са създадени два файла. Първият се нарича личен ключ (private key), който трябва да бъде пазен в тайна и се намира във файла */etc/ssh/ssh_host_key*. Вторият се нарича публичен ключ (public key) и е информацията, която можете да публикувате; той ще бъде запазен във файла */etc/ssh/ssh_host_key.pub*.

След като се въоръжите с два ключа за комуникация с *ssh*, трябва да създадете конфигурационен файл. Пакетът *ssh* е много мощен и конфигурационния файл може да съдържа много опции. Тук ще ви представим прост пример, с който можете да започнете; обърнете се към документацията на *ssh*, за да разрешите и други възможности. Следващият програмен код показва един сигурен и минимален конфигурационен файл на *sshd*. Останалата част от конфигурационните опции са описани подробно в справочната страница *sshd(8)*:

```
# /etc/ssh/sshd_config
#
# IP адресите, от които се очаква опит за свързване. 0.0.0.0. означава
# всички локални адреси
ListenAddress 0.0.0.0

# TCP портът, на който се очаква опит за свързване По подразбиране 22
Port 22

# Името на файла с ключа на хоста
HostKey /etc/ssh/ssh_host_key

# Дължината на ключа в битове
ServerKeyBits 1024

# Да разрешим ли влизане на root чрез ssh?
PermitRootLogin no

# Да разрешим ли на демона ssh да проверява дали правата за достъп до
# личната директория на потребителя и файловете му са сигурни,
# преди да се разреши влизане?
StrictModes yes
```

```
# Да разрешим ли старите методи за проверка на самоличността
# ~/.rhosts и /etc/hosts.equiv?
RhostsAuthentication no

# Да разрешим ли чисто RSA удостоверяване на самоличността?
RSAAuthentication yes
# Да разрешим ли удостоверяване на самоличността с парола?
PasswordAuthentication yes

# Да разрешим ли комбинирано удостоверяване на самоличността
# с RSA и /etc/hosts.equiv?
RhostsRSAAuthentication no
#Да игнорираме ли файловете ~/.rhosts?
IgnoreRhosts yes

# Да позволим ли влизането в акаунти с празни пароли?
PermitEmptyPasswords no
```

Важно е да се уверите, че правата за достъп до конфигурационния файл са коректни, за да гарантирате сигурността на системата. Използвайте следните команди:

```
# chown -R root:root /etc/ssh
# chmod 755 /etc/ssh
# chmod 600 /etc/ssh/ssh_host_key
# chmod 644 /etc/ssh/ssh_host_key.pub
# chmod 644 /etc/ssh/sshd_config
```

Последната фаза на администрацията на демона *sshd* е да го стартирате. Обикновено трябва да създадете *rc*-скрипт за него или да го прибавите към вече съществуващ скрипт, за да се изпълнява автоматично при зареждане на системата. Демонът работи самостоятелно и не се нуждае от добавяне на запис във файла */etc/inetd.conf*. Демонът трябва да се стартира с привилегиите на `root`. Синтаксисът е много прост:

```
/usr/sbin/sshd
```

При стартирането си демонът *sshd* автоматично ще се конфигурира като фонов процес. Сега вече сте готови да приемате *ssh*-връзки.

ssh клиент

Има няколко клиентски *ssh* програми: *slogin*, *scp* и *ssh*. Всички те четат един и същи конфигурационен файл, обикновено наречен */etc/ssh/ssh_config*. Освен това, те четат и конфигурационните файлове от поддиректорията *.ssh* на личната директория на потребителя, който ги изпълнява. Най-важните от тези файлове са *.ssh/config*, който

може да съдържа опции, отменящи посочените във файла */etc/ssh/ssh_config*, файлът *.ssh/identity*, съдържащ личния ключ и съответния файл *.ssh/identity.pub*, в който се намира публичния ключ на потребителя. Други важни файлове са *.ssh/known_hosts* и *.ssh/authorized_keys*; ще поговорим за тях по-късно в раздела "Използване на ssh". Най-напред нека създадем глобалния конфигурационен файл и файла с ключа на потребителя.

Файлът */etc/ssh/ssh_config* е много близък до конфигурационния файл за сървъра. В него също има много възможности, които можете да конфигурирате, но една минална конфигурация изглежда като в Пример 12-5. Останалата част от конфигурационните опции са описани подробно в справочната страница *sshd (8)*. Можете да добавяте секции, които съответстват на определени хостове или групи от хостове. Параметърът в конструкцията "`Host`" трябва да е или пълното име на хоста, или да бъде зададен с маска като тези, които използвахме в нашия пример, за да означим всички хостове. Например, можем да създадем запис, съдържащ `Host *.vbrew.com`, за да посочим всеки хост в домейна `vbrew.com`.

Пример 12-5: Примерен конфигурационен файл за ssh-клиент

```
# /etc/ssh/ssh_config
# Опции по подразбиране, когато се свързваме с отдалечен хост
Host *
    # Да се компресират ли данните от сесията?
    Compression yes
    # Ниво на компресия? (1-бърза/недобро, 9 - бавна/добро)
    CompressionLevel 6

    # Да се опита ли rsh, ако не може да се осъществи сигурна    връзка?
    FallBackToRsh no

    # Да се изпращат ли поддържащи връзката съобщения?
    # Полезно е, ако използвате маскиране на IP адреса.
    KeepAlive yes

    # Да се опита ли удостоверяване на самоличността чрез RSA?
    RSAAuthentication yes
    # Да се опита ли RSA-удостоверяване в комбинация с rhosts?
    RhostsRSAAuthentication yes
```

В секцията за конфигуриране на сървъра споменахме, че всеки хост и потребител притежават ключ. Ключът на потребителя се записва в неговия файл *~/.ssh/indentity*. За да генерирате този ключ, използвайте същата команда *ssh-keygen*, която използвахме, за да генерираме ключ на хоста, само че този път не е нужно да посочвате име на фай-

ла, в който да се запише ключа. По подразбиране *ssh-keygen* използва коректното място, но ви дава възможност да запишете файла и на друго място. Понякога е полезно да имате няколко файла за идентичност, така че *ssh* позволява това. Също като преди, *ssh-keygen* ще ви помоли да въведете тайна фраза. Тайните фрази добавят още едно ниво на сигурност, затова е добра идея да ги използвате. Тайната фраза няма да бъде отпечатвана на екрана, когато я въвеждате.

Не съществува начин да възстановите забравена тайна фраза. Измислете такава фраза, която можете да запомните, но както при паролите, не използвайте нещо очевидно като обикновено съществително или вашето име. За да бъде тайната фраза истински ефективна, тя трябва да е с дължина от 10 до 30 символа и да не е обикновено изречение на английски език. Опитайте се да поставите някои необичайни символи в нея. Ако забравите тайната си фраза, ще се наложи да генерирате нов ключ.

Трябва да помолите всеки ваш потребител да изпълни командата *ssh-keygen* само веднъж, за да бъдете сигурни, че неговият файл с ключа е създаден коректно. *ssh-keygen* ще създаде директориите *~/.ssh/* със съответните права за достъп и личния и публичен ключ на потребителя съответно в *.ssh/identity* и *.ssh/identity.pub*. Една такава примерна сесия изглежда последния начин:

```
$ ssh-keygen
Generating RSA keys: .....ooooo0.....................................
Key generation complete.
Enter file in which to save the key (/home/maggie/.ssh/identity):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/maggie/.ssh/identity.
Your public key has been saved in /home/maggie/.ssh/identity.pub.
The key fingerprint is:
1024 85:49:53:f4:8a:d6:d9:05:d0:1f:23:c4:d7:2a:11:67 maggie@moria
$
```

Сега *ssh* е готов за работа.

Използване на ssh

Сега програмата *ssh* и съответните ѝ програми трябва да са инсталирани и готови за работа. Нека видим как да можем да ги използваме.

Най-напред ще се опитаме да влезем в отдалечен хост. Можем да използваме програмата *slogin* по същия начин, по който показахме използването на *rlogin* по-рано в книгата. Първият път, когато се опитате да се свържете с някой хост, *ssh*-клиентът ще извлече неговия публичен ключ и ще поиска да потвърдите идентичността му, като ви покаже съкратен вариант на публичния ключ, наречен отпечатък (*fingerprint*).

Администраторът на отдалечения хост трябва предварително да ви снабди с отпечатък на публичния ключ, който вие трябва да добавите във файла *.ssh/known_hosts*. Ако отдалеченият администратор не ви е предоставил подходящия ключ, можете да се свържете с отдалечения хост, но *ssh* ще ви предупреди, че не притежава ключа и ще ви попита дали искате да приемете този, предлаган от отдалечения хост. Ако приемем, че сте сигурни, че никой не е фалшифицирал DNS информацията и вие действително разговаряте с търсения хост, отговорете с "да" на запитването. Съответният ключ се записва автоматично във вашия файл *ssh/known_hosts* и повече няма да бъдете питани за него. Ако при бъдещ опит за връзка, публичния ключ, получен от този хост, не съвпада със записания във файла, ще бъдете предупредени, защото това представлява потенциално нарушение на сигурността.

Първото влизане в отдалечен хост ще изглежда приблизително така:

```
$ slogin vchianti.vbrew.com
The authenticity of host 'vchianti.vbrew.com' can't be established.
Key fingerprint is 1024
7b:d4:a8:28:c5:19:52:53:3a:fe:8d:95:dd:14:93:f5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vchianti.vbrew.com,172.16.2.3' to the list
   of/known hosts.
maggie@vchianti.vbrew.com's password:
Last login: Tue Feb 1 23:28:58 2000 from vstout.vbrew.com
$
```

Ще бъдете запитан за парола и трябва да въведете паролата за отдалечения акаунт, а не тази за локалния. Паролата не се извежда на екрана, докато я пишете.

Ако не зададете специални аргументи, *slogin* ще се опита да влезе със същия потребителски идентификатор, който се използва на ло-

калния компютър. Можете да промените това, като използвате аргумента `-l`, с който задавате алтернативно име за влизане в отдалечения хост. Точно това направихме в нашия пример по-рано в книгата.

Можем да копираме файлове от и върху отдалечения хост като използваме програмата *scp*. Синтаксисът ѝ е подобен на обикновената команда *cp* с изключение на това, че можете да зададете име на хоста преди името на файла, което означава, че файлът се намира на посочения хост. Следващият пример илюстрира синтаксиса на *scp* като копира локален файл, наречен */tmp/fred*, в директорията */home /maggie/*, намираща се на отдалечения хост **chianti.vbrew.com**:

```
$ scp /tmp/fred vchianti.vbrew.com:/home/maggie/
maggie@vchianti.vbrew.com's password:
fred          100% |*****************************| 50165  00:01 ETA
```

Отново ще трябва да въведете парола. По подразбиране, командата *scp* показва полезна информация за прогреса. Можете да копирате файл от отдалечения хост със същата лекота; просто посочете името на хоста и пътя до файла като първи параметър, а локалния път – като втори. Възможно е дори да копирате файл от отдалечен хост на някой друг отдалечен хост, но това е нещо, което обикновено не бихте искали да правите, тъй като цялата информация минава през вашия хост.

Можете да изпълнявате команди върху отдалечения хост като използвате командата *ssh*. Нейният синтаксис също е много прост. Нека помолим нашия потребител **maggie** да ни покаже основната директория на отдалечения хост **vchianti.vbrew.com**. Тя би го направила с командата:

```
$ ssh vchianti.vbrew.com ls -CF /
maggie@vchianti.vbrew.com's password:
bin/   console@ dos/    home/   lost+found/ pub@  tmp/ vmlinuz@
boot/  dev/     etc/    initrd/ mnt/        root/ usr/ vmlinuz.old@
cdrom/ disk/    floppy/ lib/    proc/       sbin/ var/
```

Можете да поставите *ssh* в команден канал и да пренасочите нейния вход и изход точно както при другите команди, с изключение на това, че входът и изходът се изпращат към или от отдалечения хост през *ssh*-връзката. Ето един пример, как можете да използвате тази възможност в комбинация с командата *tar*, за да копирате цяла директория заедно с поддиректориите и файловете от отдалечен хост върху локалния хост:

```
$ ssh vchianti.vbrew.com "tar cf - /etc/" | tar xvf -
maggie@vchianti.vbrew.com's password:
```

```
etc/GNUstep
etc/Muttrc
etc/Net
etc/X11
etc/adduser.conf
..
..
```

Тук заградихме в кавички командата, която ще изпълним, за да посочим ясно кое се предава като аргумент на *ssh* и кое се използва от локалната среда. Тази команда изпълнява командата *tar* върху отдалечения хост, за да архивира директорията */etc/* и да запише резултата в стандартния изход. Този резултат пренасочихме през копие на програмата *tar*, работещо върху локалния хост, което го разархивира, като чете от стандартния си вход.

Отново трябваше да въведем парола. Вече можете да разберете защо ви насърчаваме да конфигурирате *ssh* така, че да не ви пита за парола през цялото време! Нека сега конфигурираме нашия локален *ssh*-клиент по такъв начин, че да не ви пита за парола, когато се свързвате с хоста vchianti.vbrew.com. По-горе споменахме файла. *ssh/authorized_keys*; вече стигнахме до мястото, където той се използва. Файлът *.ssh/authorized_keys* съдържа *публичните* ключове за всеки отдалечен потребителски акаунт, в който искаме да влизаме автоматично. Можете да настроите автоматичното влизане като копирате съдържанието на *.ssh/identity.pub* от *отдалечения* акаунт в нашия локалния файл *.ssh/authorized_keys*. От особена важност е, че правата за достъп до файла *.ssh/authorized_keys* позволяват само на вас да четете и пишете в него; в противен случай всеки може да открадне и използва ключовете ви и да влезе в този отдалечен акаунт. За да сте сигурни, че правата за достъп са правилни, променете *.ssh/authorized_keys* по следния начин:

```
$ chmod 600 ~/.ssh/authorized_keys
```

Публичните ключове представляват *един* дълъг ред от обикновен текст. Ако използвате copy и paste, за да запишете ключа във вашия локален файл, премахнете символите за край на ред, които евентуално сте добавили. Файлът *.ssh/authorized_keys* може да съдържа много такива ключове, всеки от които е на отделен ред.

Пакетът инструменти *ssh* е много мощен и има и други полезни възможности и опции, които ще представляват интерес за вас. За повече информация, прочетете справочните страници и другата документация, която се разпространява с пакета.

ГЛАВА 13

МРЕЖОВА ИНФОРМАЦИОННА СИСТЕМА

Когато управлявате локална мрежа, обикновено главната ви цел е да осигурите на потребителите среда, която прави мрежата прозрачна. Едно от най-важните условия за постигането на тази цел е да запазите жизненоважните данни като информацията за потребителските акаунти синхронизирана между всички хостове. Това дава на потребителите свободата да се преместват от машина на машина без неудобството да помнят различни пароли и да копират данни от един компютър на друг. Данните, които се съхраняват на централно място, не е нужно да се синхронизират при положение, че има някакъв удобен начин за достъп до тях от всеки хост, свързан към мрежата. Чрез централното съхраняване на важната административна информация можете да гарантирате съгласуваност на данните, да увеличите гъвкавостта на потребителите като им позволите да се преместват от хост на хост по прозрачен начин и да направите живота на системния администратор много по-лесен, защото му се налага да поддържа само едно-единствено копие на информацията.

По-горе обсъдихме един важен пример за тази концепция, използвана в Интернет – системата DNS (Domain Name System – система за имена в домейн). DNS предоставя ограничено количество информация, най-важната от която е съответствието между името на хоста и неговия IP адрес. За останалите типове информация няма такава специализирана услуга. Освен това, ако управлявате само една малка

локална мрежа, която не е свързана с Интернет, конфигурирането на DNS може би не си струва усилията.

Ето защо Sun разработи *Мрежовата информационна система* (NIS – *Network Information System*). NIS осигурява възможност за общ достъп до база данни, която може да се използва за разпространение, например, на информацията, съдържаща се във файловете *passwd* и *groups* до всички хостове във вашата мрежата. По този начин мрежата изглежда като една-единствена система с едни и същи акаунти за всички хостове. По подобен начин можете да използвате NIS за разпространение на информацията от файла */etc/hosts* с имената на хостовете до всички машини в мрежата.

NIS е базирана на RPC и се състои от сървър, библиотека от страната на клиента и няколко инструмента за администриране. Първоначално NIS се е наричала *Жълти страници* (*Yellow Pages* или YP) и това название все още се използва за обозначаването ѝ. За съжаление, то е запазена марка на British Telecom, която задължи Sun да се откаже от него. Но както често става с имената на някои хора, които винаги остават свързани с тях, така и YP продължава да се използва като префикс в имената на повечето свързани с NIS команди, например *ypserv* и *ypbind*.

Днес NIS е достъпна практически за всички Unix-и и съществуват дори няколко безплатни реализации. BSD Net-2 съдържа една такава реализация, която води началото си от публично достъпна справочна реализация, предоставена от Sun. Клиентският код библиотеката на тази реализация дълго време беше в основната библиотека *libc* на Linux, а програмите за администриране бяха адаптирани за Linux от Swen Thummler.* В справочната реализация обаче липсва NIS сървър.

Peter Eriksson разработи нова реализация, наречена NYS.* Тя поддържа както обикновената NIS, така и доста разширената NIS+ на Sun. NYS осигурява не само набор от NIS инструменти и сървър, но добавя и цял нов набор от библиотечни функции, които трябва да се компилират във вашата библиотека *libc*, ако искате да ги използвате.

---

* Със Swen може да се свържете на адрес *swen@uni-paderborn.de*. NIS клиентите са на разположение в архива *yp-linux.tar.gz* на сървъра **metalab.unc.edu** в директорията *system/Network*.

* С Peter можете да се свържете на адрес *pen@lysator.liu.se*. Настоящата версия на NYS е 1.2.8.

Това включва нова конфигурационна схема за разпознаване на имена на хостове, която заменя текущата схема, използваща *host.conf*.

Библиотеката *libc* на GNU, известна в Linux средите като *libc6*, включва актуализирана версия на традиционната поддръжка на NIS, разработена от Thorsten Kukuk.* Тя поддържа всички библиотечни функции, които осигурява NYS, а освен това използва и подобрената конфигурационна схема на NYS. Все още се нуждаете от инструменти и сървър, но използвайки библиотеката *libc* на GNU, ще си спестите неудобството да прилагате patch-ове и да прекомпилирате библиотеката.

В тази глава ще наблегнем повече на поддръжката на NIS, включена в GNU *libc*, отколкото на другите два пакета. Ако наистина искате да използвате някой от тези пакети, инструкциите в тази глава може да са, но може да не са достатъчни. За допълнителна информация се обърнете към документа NIS-HOWTO или към някоя добра книга като *Managing NFS and NIS* от Hal Stern (издание на O'Reilly).

Запознаване с NIS

NIS съхранява информацията от базата данни във файлове, наречени *карти* (*maps*), които съдържат двойки ключ-стойност. Пример за една такава двойка е името за влизане на потребител и шифрирания вид на паролата му за достъп. Картите се съхраняват в централен хост, на който работи NIS сървър и от който клиентите могат да извлекат информацията с помощта на различни RPC функции. Доста често картите се съхраняват в DBM файлове.*

Самите карти обикновено се генерират от главните текстови файлове като */etc/hosts* или */etc/passwd*. За някои файлове се създават няколко карти – по една за всеки тип ключове за търсене. Например, може да претърсите файла *hosts* както за име на хост, така и за IP адрес. Съответно от него се извличат две NIS карти, наречени *hosts.byname* и *hosts.byaddr*.

---

* С Thorsten може да се свържете на адрес *kukuk@uni-paderborn.de*.

* DBM е библиотека за управление на опростена база данни, която използва хеш-техники за увеличаване скоростта на операциите за търсене. Съществува безплатна реализация на DBM от проекта GNU, наречена *gdbm*, която е част от повечето дистрибуции на Linux.

Таблица 13.1 съдържа най-често срещаните карти и файловете, от които те са генерирани.

Таблица 13.1: Някои стандартни NIS карти и съответстващите им файлове

Главни файлове	Карти	Описание
/etc/hosts	*hosts.byname*, *hosts.byaddr*	Съответствия между IP адреси и имена на хостове
/etc/networks	*networks.byname*, *networks.byaddr*	Съответствия между IP адреси на мрежи и имена на мрежи
/etc/passwd	*passwd.byname*, *passwd.byuid*	Съответствия между шифрирани пароли и потребителски имена за влизане
/etc/group	*group.byname*, *group.bygid*	Съответствия между идентификатори на групи и имена на групи
/etc/services	*services.byname*, *services.bynumber*	Съответствия между описания на услуги и имена на услуги
/etc/rpc	*rpc.byname*, *rpc.bynumber*	Съответствия между номера на RPC услуги на Sun и имена на RPC услуги
/etc/protocols	*protocols.byname*, *protocols.bynumber*	Съответствия между номера на протоколи и имена на протоколи
/usr/lib/aliases	*mail.aliases*	Съответствия между пощенски псевдоними и имена на пощенски псевдоними

Можете да намериге поддръжка за останалите файлове и карти в други NIS пакети. Обикновеното съдържат информация за приложения, които не обсъждаме в тази книга, например за картата *bootparams*, която се използва от сървъра *bootparamd* на Sun.

За някои карти хората обикновено използват *прякори*, които са по-къси и следователно по-лесни за въвеждане. Забележете, че тези прякори се разпознават само от *ypcat* и *ypmatch* – два инструмента за про-

верка на вашата конфигурация на NIS. За да получите пълен списък с прякорите, разпознавани от тези инструменти, използвайте следната команда:

```
$ ypcat -x
Use "passwd" for "passwd.byname"
Use "group" for "group.byname"
Use "networks" for "networks.byaddr"
Use "hosts" for "hosts.byaddr"
Use "protocols" for "protocols.bynumber"
Use "services" for "services.byname"
Use "aliases" for "mail.aliases"
Use "ethers" for "ethers.byname"
```

Софтуерът NIS сървър обикновено се нарича *ypserv*. За една мрежа със средни размери обикновено е достатъчен един сървър. В големите мрежи могат да се използват няколко сървъра на различни машини и различни сегменти от мрежата, за да се облекчи натоварването на сървърните машини и маршрутизаторите. Тези сървъри се синхронизирани, като един от тях се използва като *главен сървър* (*master server*), а останалите са *подчинени сървъри* (*slave servers*). Карти се създават само на хоста на главния сървър. От там те се разпространяват до всички подчинени сървъри.

Досега говорихме много неопределено за "мрежи". В NIS съществува специфичен термин, който се отнася за съвкупността от всички хостове, споделящи част от данните за конфигурацията на своите системи чрез NIS: *NIS домейн*. За съжаление NIS домейните нямат нищо общо с домейните, с които се срещнахме при DNS. Затова, за да избегнем всякакво двусмислие в тази глава, винаги ще указваме кой тип домейн имаме предвид.

NIS домейните имат чисто административна функция. В повечето случаите са невидими за потребителите, с изключение на съвместното използване на пароли от всички машини в домейна. Затова името, дадено на NIS домейн, е от значение само за администраторите. Обикновено всяко име върши работа, при положение, че е различно от останалите имена на NIS домейни в локалната ви мрежа. Например, администраторът на Виртуалната пивоварна може да реши да създаде два NIS домейна – един за самата пивоварна и друг за винарната, които да нарече съответно **brewery** и **winery**. Друг често срещан метод е просто да се използва името на DNS домейна и за име на NIS домейна.

За да зададете и покажете на екрана името на NIS домейна на вашия хост, можете да използвате командата *domainname*. Когато се изпълнява без аргументи, тази команда отпечатва текущото име на NIS домейна. За да зададете име на домейн, трябва да влезете като суперпотребител:

```
# domainname brewery
```

NIS домейните определят към кой NIS сървър ще се обръща дадено приложение. Например, програмата *login* на хост от Винарната трябва, разбира се, да се обръща за информация относно потребителската парола само към NIS сървъра на Винарната (или ако те са няколко, към един от тях), докато приложение, работещо на хоста в Пивоварната, трябва да използва сървъра на Пивоварната.

Остава да се разреши една загадка: как клиентът разбира към кой сървър трябва да се свърже? Най-простият подход е да се използва конфигурационен файл, в който се указва името на хоста, на който се намира сървъра. Този подход обаче е твърде негъвкав, защото не позволява на клиентите да използват различни сървъри (разбира се, от един и същи домейн) в зависимост от тяхната наличност. Поради тази причина реализациите на NIS разчитат на специален демон, наречен *ypbind*, за да открият подходящ NIS сървър в своя NIS домейн. Преди да направи някакво запитване, приложението първо разбира от *ypbind* кой сървър да използва.

ypbind търси сървъри чрез изпращане на broadcast пакети в локалната IP мрежа. Приема се, че първият отговорил сървър е най-бързият и той се използва при всички следващи NIS запитвания. След като измине определен интервал от време или ако сървърът стане недостъпен, *ypbind* отново проверява за активни сървъри.

Динамичното свързване е полезно само, когато мрежата виосигурява повече от един NIS сървър. Динамичното свързване обаче създава проблем със сигурността на данните. *ypbind* сляпо вярва на всеки, който отговори, независимо дали е скромен NIS сървър или злонамерен нарушител. Излишно е да споменаваме, че това става особено опасно, ако управлявате своите бази данни с пароли под NIS. За да ви предпази от тази опасност, програмата *ypbind* за Linux ви предоставя възможност да претърсвате локалната мрежа за локалния NIS сървър или да конфигурирате името на хоста с NIS сървъра в конфигурационен файл.

NIS срещу NIS+

Общото между NIS и NIS+ е почти само целта и името. NIS+ е структурирана напълно различно от NIS. Вместо плоско пространство от имена с несвързани NIS домейни, NIS+ използва йерархично пространство от имена, подобно на това в DNS. Вместо карти се използват т.нар. *таблици*, съставени от редове и колони, в които всеки ред представя обект от базата данни на NIS+, а колоните съдържат свойства на обектите, които са важни и познати на NIS+. Всяка таблица за даден NIS+ домейн включва таблиците на родителските си домейни. Освен това, запис от таблица може да съдържа връзка към друга таблица. Тези възможности позволяват структурирането на информацията да става по много начини.

Като допълнение, NIS+ поддържа сигурно и шифрирано използване на RPC, което до голяма степен помага при решаването на проблема със сигурността в NIS.

Традиционната система NIS има RPC версия 2, докато NIS+ е с версия 3. Докато писахме тази книга, все още нямаше добре работеща реализация на NIS+ за Linux, затова тук не разглеждаме NIS+.

Клиентската страна на NIS

Ако сте запознати с писането или адаптирането на мрежови приложения, може да забележите, че повечето от изброени по-горе NIS карти съответстват на библиотечните функции от библиотеката на C. Например, за да получите информация от *passwd*, обикновено използвате функциите *getpwnam* и *getpwuid*, които връщат информация за акаунта, асоцииран съответно с даденото потребителско име или числов потребителски идентификатор. При нормални обстоятелства тези функции извършват заявеното търсене в някой стандартен файл, например */etc/passwd*.

Една съгласувана с NIS реализация на тези функции, обаче, модифицира това поведение, като извиква RPC функция на NIS сървъра, която търси потребителското име или потребителския идентификатор. Това става по прозрачен за приложението начин. Функцията може да третира NIS данните като че са били прибавени към истинския файл *passwd*, така че и двете множества от информация са достъпни на приложението и могат да се използват, или NIS данните напълно са заменили файла, така че информацията в локалния *passwd* се игнорира и се използват само NIS данните.

При традиционните реализации на NIS съществуваха някои конвенции за това кои картите се заместват и кои се добавят към оригиналната информация. Някои карти като *passwd* изискваха направени набързо изменения на файла *passwd*, което, ако се извърши некоректно, ще отвори дупки в сигурността. За да избегнат тези капани, NYS и GNU *libc* използват обща конфигурационна схема, която определя дали определена група клиентски функции използва оригиналните файлове, NIS или NIS+ и в какъв ред. Тази схема ще бъде описана по-късно в тази глава.

Използване на NIS сървър

След толкова много теоретични техно-приказки, време е да си поиграпаме ръцете с истинска работа по конфигурирането. В този раздел ще разгледаме конфигурирането на NIS сървър. Ако в мрежата ви работи NIS сървър, няма нужда да инсталирате ваш собствен. В този случай можете спокойно да прескочите този раздел.

Забележка: ако просто ще експериментирате със сървъра, уверете се, че не го конфигурирате с име на NIS домейн, което вече се използва в мрежата ви. Това може да разстрои цялата работа на мрежата и да направи доста хора много нещастни и много ядосани.

Възможни са две конфигурации на NIS сървъра: главен и подчинен. Конфигурирането на подчинен сървър осигурява машина за моментално възстановяване на работата, в случай на проблеми с главния сървър. Тук ще разгледаме конфигурирането само на главния сървър. В случай че искате да конфигурирате подчинен сървър, ще намерите разликите в документацията на сървъра.

В момента съществуват два свободни NIS сървъра за Linux: единият се съдържа в пакета *yps* на Tobias Reber, а другият – в пакета *ypserv* на Peter Eriksson. Няма никакво значение кой от двата ще използвате.

След инсталирането на програмата-сървър (*ypserv*) в */user/sbin*, трябва да създадете директорията, в която ще се съхраняват файловете-карти, разпространявани от сървъра ви. Когато зададете NIS домейн за домейна **brewery**, картите ще се намират в */var/yp/brewery*. Сървърът установява дали обслужва даден NIS домейн, като проверява дали съществува директорията за карти. Ако забраните обслужването на някой NIS домейн, уверете се, че сте премахнали и директорията.

Картите обикновено се съхраняват в DBM файлове, за да се ускори търсенето. Тези файлове се създават се от главните файлове с по-

мощта на програми, наречени *makedbm* (за сървъра на Tobias) или *dbmload* (за сървъра на Peter).

Преобразуването на главен файл във формат, който *dbmload* може да анализира, обикновено изисква някаква магия с *awk* и *sed* – нещо, което е малко досадно за писане и трудно за запомняне. Поради това, пакетът *ypserv* на Peter Eriksson съдържа make-файл (наречен *ypMakefile*), който управлява преобразуването на най-често използваните главни файлове. Трябва да го инсталирате като *Makefile* във вашата директория за карти и да го редактирате, за да съответства на картите, които искате да предоставя вашият NIS сървър. В началото на файла ще откриете правилото `all`, което изброява услугите, предлагани от *ypserv*. По подразбиране този ред изглежда приблизително така:

```
all: ethers hosts networks protocols rpc services passwd group netid
```

Ако не искате да създавате, например, картите *ethers.byname* и *ethers.byaddr*, просто махнете условието `ethers` от това правило. За да тествате настройката си, можете да започнете само с една или две карти, например с картите *services.**.

След редактирането на *Makefile*, докато сте в директорията за карти, въведете командата **`make`**. Така автоматично ще генерирате и инсталирате посочените карти. Уверете се, че при смяна на основните файлове картите се актуализират. В противен случай, промените ще останат невидими за мрежата.

В раздела "Настройка на NIS клиент с GNU libc" ще обясним как се конфигурира клиентския код за NIS. Ако конфигурацията ви не работи, трябва да се опитате да разберете дали в сървъра ви пристигат заявки. Ако зададете флага от командния ред *–debug* на *ypserv*, той ще отпечатва debug-съобщения на конзолата за всички входящи NIS запитвания и връщаните резултати. Това би трябвало да ви подскаже до къде е проблема. Сървърът на Tobias няма такава опция.

Сигурност на NIS сървъра

NIS имаше един значителен пропуск в сигурността: вашия файл с пароли оставяше достъпен за четене практически за всеки в Интернет – нещо, което допринася за съществуването на доста голям брой потенциални нарушители. Щом някой злонамерен човек научи името на вашия NIS домейн и адреса на сървъра ви, той може просто да му изпрати заявка за картата *passwd.byname* и моментално да получи

всичките шифрирани пароли за системата ви. С бърза програма за откриване на пароли като *crack* и добър речник, отгатването поне на няколко от потребителските пароли рядко е проблем.

Именно затова съществува опцията *securenets*. Тя просто ограничава достъпа до вашия NIS сървър за определени хостове, базирайки се на техния IP адрес и номер на мрежа. Последната версия на ypserv реализира тази възможност по два начина. Първият разчита на специален конфигурационен файл, наречен */etc/ypserv.securenets*, а вторият използва конвенционалните файлове */etc/hosts.allow* и */etc/hosts.deny*, с които вече се срещнахме в Глава 12, *Важни мрежови възможности*.* По този начин, за да се ограничи достъпът само до вътрешни хостове на Пивоварната, нейният мрежов администратор трябва да добави към *hosts.allow* следния ред:

```
ypserv: 172.16.2.
```

Това ще даде на всички хостове от IP мрежата **172.16.2.0** достъп до NIS сървъра. За да се изключат всички останали хостове, съответният запис в *hosts.deny* трябва да бъде:

```
ypserv: ALL
```

IP адресите не са единственият начин, чрез който могат да се задават хостове или мрежи в *hosts.allow* и *hosts.deny*. За повече подробности се обърнете към справочната страница *hosts_access(5)* в системата ви. Трябва да знаете, обаче, че *не можете* да използвате име на хост или домейн, намиращо се на `ypserv`. Ако зададете име на хост, сървърът се опитва да го разпознае, но резолвера на свой ред извиква *ypserv* и така попадате в безкраен цикъл.

За да конфигурирате защита със `securenets`, използвайки метода */etc/ypserv.securenets*, трябва да създадете неговия конфигурационен файл */etc/ypserv.securenets*. Този конфигурационен файл има проста структура. Всеки ред описва хост или мрежа от хостове, на които ще им бъде разрешен достъп до сървъра. На всеки адрес, който не е описан чрез запис в този файл, ще му бъде отказан достъп. Ред, който започва с #, ще се третира като коментар. Пример 13-1 показва как ще изглежда един прост файл */etc/ypserv.securenets*:

* За да е възможно използването на метода */etc/hosts.allow*, може да се наложи да прекомпилирате сървъра. Моля, прочете инструкциите във файла *README*, включен в дистрибуцията.

Пример 13.1: Примерен файл ypserv.securenets

```
# позволява връзки от локалния хост -- необходимост
host 127.0.0.1
# еквивалентно на 255.255.255.255 127.0.0.1
#
# позволява връзки от всички хостове на Виртуална пивоварна
255.255.255.0   172.16.1.0
#
```

Първият запис във всеки ред е мрежовата маска, която да се използва за записа, като `host` се третира като специална ключова дума, означаваща "мрежова маска 255.255.255.255". Вторият запис във всеки ред е IP адресът, към който се прилага мрежовата маска.

Третата опция е свързана с използването на защитен portmapper демон, вместо опцията *securenets* в ypserv. Защитеният portmapper (*portmap-5.0*) използва и схемата *hosts.allow*, но предлага защита не само на *ypserv*, но и на всички RPC сървъри.* Все пак, не трябва едновременно с опцията *securenets* да използвате и защитен portmapper поради допълнителното натоварване, което внася това удостоверяване на самоличността.

Настройка на NIS клиент с GNU libc

Сега ще опишем и обсъдим конфигурирането на NIS клиент с помощта на поддръжката от библиотеката GNU libc.

С първата си стъпка трябва да укажете на GNU libc кой сървър да използва NIS клиента за NIS услуги. По-горе споменахме, че *ypbind* за Linux ви дава възможност да конфигурирате NIS сървъра, който искате да използвате. Поведението на клиента по подразбиране е да запитва сървъра в локалната мрежа. Ако хостът, който конфигурирате, може да се премества от един домейн в друг, например преносим компютър, оставете файла */etc/yp.conf* празен и библиотеката ще извършва запитване за NIS сървър в локалната мрежа, в която се намирате.

Повечето хостове могат да се конфигурират по-защитено, като се зададе името на сървъра в конфигурационния файл */etc/yp.conf*. Един

* Защитеният portmapper е достъпен с анонимен FTP достъп от сървъра **ftp.win. tue.nl**, директорията */pub/security/*.

много прост файл за хост от мрежата на Винарната може да изглежда по следния начин:

```
# yp.conf - YP конфигурация за библиотеката GNU libc.
#
ypserver vbardolino
```

Конструкцията `ypserver` указва на вашия хост да използва хоста, зададен като NIS сървър за локалния домейн. В този пример зададохме NIS сървъра като **vbardolino**. Разбира се, IP адресът, който съответства на **vbardolino**, трябва да бъде посочен във файла *hosts*. Като алтернатива можете да използвате самия IP адрес чрез аргумента `server`.

При показания в примера формат, командата *ypserver* указва на *ypbind* да използва споменатия сървър независимо от това, какъв е настоящият NIS домейн. Ако, обаче, често местите машината си между различни NIS домейни, във файла *yp.conf* можете да съхранявате информация за няколко домейна. В *yp.conf* можете да разполагате с информация за сървърите в различни NIS домейни като задавате тази информация с конструкцията `domain`. Например, можете да промените горния примерен файл по следния начин, за да бъде подходящ за преносим компютър:

```
# yp.conf - YP конфигурация за библиотеката GNU libc.
#
domain winery server vbardolino
domain brewery server vstout
```

Това позволява използването на машината ви и в двата домейна просто чрез задаване на желания NIS домейн с командата *domainname* по време на зареждане. Тогава NIS клиентът ще използва всеки сървър, съответстващ на текущия домейн.

Съществува и трета опция, която може би ще искате да използвате. Тя се използва в случай, че не знаете името или IP адреса на сървъра, който се използва в даден домейн, и въпреки това искате да използвате конкретен сървър в определени домейни. Представете си, че непременно трябва да използваме точно определен сървър, когато работим в домейна на Винарната, но искаме да търсим достъпен сървър, когато сме в домейна на Пивоварната. Би трябвало отново да модифицираме файла *yp.conf*, за да изглежда вече така:

```
# yp.conf - YP конфигурация за библиотеката GNU libc.
#
domain winery server vbardolino
domain brewery broadcast
```

Ключовата дума `broadcast` указва на *ypbind* да използва NIS сървъра, който намери за домейна.

След създаването на този базов конфигурационен файл и след като се уверите, че той е достъпен за четене от всеки, трябва да направите първия си опит за връзка с вашия сървър. Уверете се, че сте избрали карта, която се разпространява от вашия сървър, например *hosts.byname*, и се опитайте да я извлечете като използвате инструмента *ypcat*:

```
# ypcat hosts.byname
172.16.2.2      vbeaujolais.vbrew.com     vbeaujolais
172.16.2.3      vbardolino.vbrew.com      vbardolino
172.16.1.1      vlager.vbrew.com          vlager
172.16.2.1      vlager.vbrew.com          vlager
172.16.1.2      vstout.vbrew.com          vstout
172.16.1.3      vale.vbrew.com            vale
172.16.2.4      vchianti.vbrew.com        vchianti
```

Получения резултат трябва да е подобен на току-що показания. Ако вместо това получите съобщение за грешка, което гласи: `Can't bind to server which serves domain` (не мога да се свържа със сървъра, който обслужва домейна), тогава или за зададеното име на NIS домейн няма дефиниран съответен сървър в *yp.conf*, или поради някаква причина сървърът не е достъпен. Във втория случай се уверете, че след извършване на *ping* към хоста получавате положителен резултат и че наистина на този хост работи NIS сървър. Потвърждение за последното можете да получите като използвате командата *rpcinfo*, която ще покаже следния изход:

```
# rpcinfo -u хост-на-сървъра ypserv
program 100004 version 1 ready and waiting
program 100004 version 2 ready and waiting
```

Избор на подходящи карти

След като се убедите, че можете да достигнете до NIS сървъра, трябва да решите кои конфигурационни файлове да замените или да допълните с NIS карти. Обикновено се използват NIS карти за функциите, търсещи хост или парола. Първата е особено полезна, ако не разполагате с услугата за имена BIND. Търсенето на парола позволява на всички потребители да влизат в своите акаунти от всяка система в NIS домейна. Това обикновено е съпроводено със споделяне на централна директория */home* между всички хостове посредством NFS. Картата за паролите е описана подробно в следващия раздел.

Други карти като *services.byname* не предлагат толкова впечатляващи придобивки, но наистина ви спестяват доста редакторска работа. Картата *services.byname* е полезна, ако сте инсталирали мрежови приложения, които използват име на услуга, което не е указано в стандартния файл *services*.

Обикновено, трябва да имате някакъв избор, когато една функция за търсене използва локалните файлове, извършва запитвания към NIS сървъра или използва други сървъри, например DNS. GNU libc ви позволява да конфигурирате последователността, в която функция използва тези услуги. Тази възможност се управлява чрез файла */etc/nsswitch.conf*, чието име е съкращение от *Name Service Switch* (смяна на услугата за имена), но разбира се, не се ограничава само с обслужване на имената. За всяка поддържана от GNU libc функция за търсене на данни, този файл съдържа ред, в който са изброени възможните услуги.

Правилната последователност на услугите зависи от типа на данните, които всяка услуга предлага. Малко вероятно е картата *services.byname* да съдържа записи, различаващи се от записите в локалния файл *services*; тя може да съдържа само допълнителни записи. Така че най-често е разумно да се обръщате първо към локалните файлове и да проверявате NIS само, ако името на услугата не е намерено. От друга страна, информацията за имената на хостовете може да се променя доста често, така че DNS или NIS сървърът трябва винаги да имат възможно най-точните данни, докато локалният файл *hosts* се пази само като резерва, в случай че услугите DNS и NIS станат недостъпни. Затова за имена на хостове по правило трябва да проверявате локалния файл най-накрая.

Следващият пример показва как да укажете на *gethostbyname* и *gethostbyaddr* да проверят в NIS и DNS преди файла *hosts* и как функцията *getservbyname* да търси в локалните файлове, преди да се обърне към NIS. Тези функции на резолвера ще опитат последователно всяка от изброените услуги. Ако дадено търсене успее се връща резултат; в противен случай, функциите ще изпробват следващата услуга от списъка. Настройките във файла за тези приоритети е:

```
# Малък примерен файл /etc/nsswitch.conf
#
hosts:      nis dns files
services:  files nis
```

Следва пълен списък на услугите и адресите, които могат да се използват в запис от файла *nsswitch.conf*. Конкретните карти, файлове,

сървъри и обекти, към които се извършва запитване, зависят от името на записа. Отдясно на двоеточието могат да се използват следните аргументи:

`nis`
: Използвай NIS сървъра на текущия домейн. Местоположението на сървъра, към който сме е запитването, е конфигурирано във файла *yp.conf* по начина, показан в предишния раздел. Вследствие на `hosts`-запис се осъществява запитване към картите *hosts.byname* и *hosts.byaddr*.

`nisplus` *или* `nis+`
: Използвай NIS+ сървър за този домейн. Местоположението на сървъра се получава от файла */etc/nis.conf*.

`dns`
: Използвай DNS сървър за имена. Този тип услуга е полезна само в запис `hosts`. Сървърите за имена, към които е запитването, се определят както обикновено от стандартния файл *resolv.conf*.

`files`
: Използвай локалния файл, например файла */etc/hosts* в запис `hosts`.

`compat`
: Съвместимост с по-старите файлови формати. Тази опция може да се приложи, когато се използва NYS или glibc 2.x за NIS или NIS+ търсене. Макар че тези версии обикновено не могат да интерпретират по-старите NIS записи във файловете *passwd* и *group*, опцията `compat` им позволява да работят с тези формати.

`db`
: Търси информацията в DBM файлове, намиращи се в директорията */var/db*. За този файл се използва съответното име на NIS карта.

В момента поддръжката на NIS в GNU libc се отнася за следните бази данни на *nsswitch.conf*: `aliases`, `ethers.group`, `hosts`, `netgroup`, `network`, `passwd`, `protocols`, `publickey`, `pc`, `services` и `shadow`. Вероятно ще бъдат добавени още типове.

Пример 13.2 показва един по-завършен пример, който ни запознава с друга възможност на *nsswitch.conf*. Ключовата дума `[NOTFOUND = return]` в записа `hosts` указва на NIS клиента да спре търсенето,

ако желаният елемент не може да бъде намерен в NIS или DNS базата данни. По този начин NIS клиентът ще продължи да претърсва локалните файлове *само*, ако запитването на NIS и DNS сървърите се провали по някаква друга причина. В този случай локалните файлове ще се използват само по време на зареждане и като резервно копие, когато NIS сървъра не работи.

Пример 13.2: Примерен файл nsswitch.conf

```
# /etc/nsswitch.conf
#
hosts:      nis dns [NOTFOUND=return] files
networks:   nis [NOTFOUND=return] files
services:   files nis
protocols:  files nis
rpc:        files nis
```

GNU libc предоставя и някои други възможности, които са описани в справочната страница на *nsswitch*.

Използване на картите passwd и group

Едно от основните приложения на NIS е синхронизацията на информацията за потребителите и акаунтите във всички хостове на NIS домейна. Затова обикновено трябва да пазите само малък локален файл */etc/passwd*, който се допълва от глобалната за сайта информация от NIS картите. Все пак, обикновено не е достатъчно просто да разрешите в *nsswitch.conf* възможността за NIS претърсвания с тази услуга.

Когато разчитате на предоставяната от NIS информация за паролите, първо трябва да се уверите, че числовият потребителски идентификатор на всеки от потребителите в локалния файл *passwd* съответства на представата на NIS сървъра за идентификатора на този потребител. Съвместимостта на потребителските идентификатори е важна и за други цели, например за монтиране на NFS томове от други хостове в мрежата ви.

Ако някой от числовите идентификатори в */etc/passwd* или */etc/group* се различава от идентификатора в картите, трябва да настроите собствеността за всички файлове, които принадлежат на този потребител. Първо трябва да замените всички идентификатори на потребители (uid) и идентификатори на групи (gid) във файловете *passwd* и *group* с техните нови стойности, а след това да откриете всички файлове, които принадлежат на току-що променените потребители и да

смените тяхната собственост. Да допуснем, че **news** е имал потребителски идентификатор 9, а **okir** е имал потребителски идентификатор 103, и стойността на тези идентификатори се е променила. В такъв случай, може да изпълните следните команди като root:

```
# 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
```

Важно е да изпълните тези команди с инсталиран новия файл *passwd* и да откриете всички имена на файлове, преди да промените собствеността на който и да е от тях. За да актуализирате асоциираната група на файловете, използвайте подобен подход, като вместо uid напишете gid и вместо chown използвате chgrp.

След като направите това, числовите идентификаторите на потребители и групи в системата ви ще се съгласуват с тези във всички останали хостове от вашия NIS домейн. Следващата стъпка е добавянето на конфигурационни редове във файла *nsswitch.conf*, което разрешава NIS търсения на информация за потребителите и групите:

```
# /etc/nsswitch.conf - обработка на passwd и group
passwd: nis files
group:  nis files
```

Тази конфигурация оказва влияние върху командата *login* и приятели, търсещи информация за потребители. Когато един потребител се опита да влезе в системата, *login* първо извършва запитване към NIS картите и ако това търсене не успее, се връща към локалните файлове. Обикновено трябва да премахнете почти всички потребители от локалните файлове, като оставите в тях записи само за **root** и универсални акаунти като **mail**. Тези акаунти са необходими, защото може да се наложи някои жизненоважни системни задачи да установяват съответствието между идентификатори на потребители и техните имена или обратно. Например, административните *cron* задачи могат да изпълнят командата *su*, за да получат временно идентификатора на **news**, или UUCP подсистемата може да изпрати писмо с отчет на състоянието си. Ако **news** и **uucp** нямат записи в локалния *passwd* файл, тези процеси ще се провалят безславно преди инициализирането на NIS.

И накрая, ако използвате старата реализация на NIS (поддържана от режима `compat` във файловете *passwd* и *group* в NYS) или реализациите glibc, ще трябва да включите в тях странните специални записи. Тези записи показват къде в базата данни трябва да бъдат вмъкнати получените от NIS записи. Записите могат да се поставят навсякъде, но обикновено се добавят в края. Записът, който се добавя към файла */etc/passwd* е следния:

```
+:::::
```

а към файла */etc/groups*:

```
+:::
```

Както при glibc 2.x, така и при NYS можете да предефинирате параметри в потребителски записи, получени от NIS сървъра, като създадете записи с поставен "+" пред името за влизане, както и да изключвате определени потребители чрез създаването на записи с поставяне на "-" пред името. Например, запишете:

```
+stuart:::::/bin/jacl
-jedd::::::
```

ще отменят зададената обвивка за потребителя **stuart**, предоставена от NIS сървъра, и ще забранят на потребителя **jedd** да влиза в тази машина. Във всяко оставено празно поле се използва информацията, предоставена от NIS сървъра.

Тук трябва да направим две големи предупреждения. Първо, описаната дотук конфигурация работи само с login програми, които не използват скрити пароли. Ще разгледаме проблемите при използване на скрити пароли с NIS в следващия раздел. Второ, login програмите не са единствените, които имат достъп до файла *passwd* – вземете например командата *ls*, която повечето хора едва ли не постоянно използват. Всеки път, когато се създава подробен списък на файловете в директория, *ls* показва символните имена на потребителите и групите, асоциирани с файлове. Т.е., за всеки uid и gid, който срещне, *ls* трябва да направи запитване до NIS сървъра. Едно запитване към NIS продължава малко по-дълго от еквивалентното търсене в локален файл. Можете дори да откриете, че споделянето на информацията от *passwd* и *group* с помощта на NIS води до забележително намаляване на времето за изпълнение на някои програми, които използват често тази информация.

И все пак, това не е цялата история. Представете си какво се случва, ако потребител иска да смени своята парола. Обикновено той стар-

тира командата *passwd*, която прочита новата парола и актуализира локалния файл *passwd*. При използването на NIS това обаче е невъзможно, тъй като този файл вече не е локално достъпен. Влизането на потребителите в NIS сървъра всеки път, когато искат да променят паролата си, също не е най-добрият начин. Ето защо NIS предоставя заместител на *passwd*, наречен *yppasswd*, който управлява смяната на паролите под NIS. За да смени паролата в хоста-сървър, тя се свързва с демона *yppasswdd* на този хост чрез RPC и предоставя актуализираната информация за паролата. Обикновено трябва да инсталирате *yppasswd* върху нормална програма, например по следния начин:

```
# cd /bin
# mv passwd passwd.old
# ln yppasswd passwd
```

По същото време трябва да инсталирате демона *rpc.yppasswdd* на сървъра и да го стартирате от мрежов скрипт. По този начин ще скриете от вашите потребители всички особености на NIS.

Използване на NIS с поддръжка на скрити пароли

Използването на NIS съвместно с файловете за скрити пароли е малко проблематично. Първо малко лоши новини: използването на NIS обезсмисля целта на скритите пароли. Системата със *shadow* (скрити) пароли бе разработена, за да попречи на потребители с ограничени права за достъп да получат достъп до шифрирания формат на паролите за влизане. Използването на NIS за споделяне на *shadow* данни неизбежно прави шифрираните пароли достъпни за всеки потребител, който може да чуе отговорите на NIS сървъра в мрежата. Много по-добре е да се следва политика на насърчаване на потребителите да избират "добри" пароли, отколкото да се опитвате да криете пароли в NIS среда. Да хвърлим един бърз поглед на начина, по който се прави това, в случай че решите да продължите нататък.

В библиотеката libc5 не съществува реално решение за споделяне на *shadow* данни при работа с NIS. Единственият начин за разпространение на информация за потребители и пароли чрез NIS е с помощта на стандартните *passwd.** карти. Ако имате инсталирани скрити пароли, най-лесният начин за споделянето им е с генериране на съответния *passwd* файл от */etc/shadow* с помощта инструменти като *pwuncov* и създаване на NIS карти от този файл.

Разбира се, съществуват някои техники, които са необходими за едновременното използване на скрити пароли и NIS. Един пример е инсталирането на файла */etc/shadow* на всеки хост от мрежата, докато информацията за потребителите се разпространява чрез NIS. Тази техника обаче наистина е примитивна и противоречи на целта на NIS да облекчи системното администриране.

Поддръжката на NIS в библиотеката GNU libc (libc6) предоставя възможност за работа с бази данни със скрити пароли. Това не предлага никакво реално решение на проблема с достъпността на паролите, но наистина опростява управлението на пароли в условия, в които се изисква да използвате NIS със скрити пароли. За да използвате тази поддръжка, трябва да създадете базата данни *shadow.byname* и да добавите към */etc/nsswitch.conf* следния ред:

```
# Поддръжка на скрити пароли
shadow: compat
```

Ако използвате скрити пароли заедно с NIS, трябва да се опитате да поддържате някаква сигурност, като ограничите достъпа до вашата база данни на NIS. Виж раздела "Сигурност на NIS сървъра" в началото на тази глава.

ГЛАВА 14

МРЕЖОВА ФАЙЛОВА СИСТЕМА

Мрежовата файлова система (NFS – Network File System) вероятно е най-известната мрежова услуга, която използва RPC. Тя ви дава достъп до файлове в отдалечени хостове по абсолютно същия начин, по който осъществявате достъп до локални файлове. Това е възможно благодарение на комбинацията от поддръжката на ядрото и демони в потребителското пространство от страна на клиента и NFS сървър от страна на сървъра. Този достъп до файлове е напълно прозрачен за клиента и работи с най-разнообразни архитектури на сървъри и хостове.

NFS предлага редица полезни свойства:

- Данните, използвани от всички потребители, могат да се съхраняват на централен хост, като клиентите монтират съответните директории по време на начално зареждане на системата. Например, можете да пазите акаунтите на всички потребители на един хост и във всички хостове от мрежата да монтирате неговата директория */home*. Ако освен NFS е инсталирана и NIS, потребителите могат да влизат във всяка система и да работят със същия набор от файлове.
- Данни, които заемат големи обеми дисково пространство, могат да се съхраняват на един-единствен хост. Например, всички файлове и програми, свързани с LaTeX и METAFONT, могат да се съхраняват и поддържат на едно място.

- Административните данни могат да се съхраняват на един-единствен хост. Не е нужно да използвате rcp, за да инсталирате един и същи досаден файл на 20 различни машини.

Конфигурирането на базовата работа с NFS не е никак трудно както на клиента, така и на сървъра; тази глава описва как става това.

Linux NFS е разработен предимно от Rick Sladkey, който написа кода за ядрото на NFS и голяма част от NFS сървъра.* NFS сървърът е производен на NFS сървъра от потребителското пространство **unfsd**, първоначално написан от Mark Shand, и сървъра **hnfs** Harris NFS, написан от Donald Becker.

Да разгледаме как работи NFS. Първо, клиентът се опитва да монтира директория от отдалечен хост в локална директория по същия начин, по който прави това с едно физическо устройство. Синтаксисът, който се използва за задаване на отдалечената директория, обаче, е различен. Например, за да монтира */home* от хоста **vlager** в */users* на **vale**, администраторът въвежда следната команда на **vale**:†

```
# mount -t nfs vlager:/home /users
```

mount ще се опита да се свърже с демона за монтиране *rpc.mountd* на **vlager** чрез RPC. Сървърът ще провери дали на **vale** е разрешено да монтира въпросната директория и ако е така, ще върне файлов манипулатор. Този манипулатор ще се използва при всички следващи заявки за файлове в */users*.

Когато някой поиска достъп до файл през NFS, ядрото отправя RPC заявка към *rpc.nfsd* (NFS демона) на машината-сървър. Тази заявка приема като параметри файловия манипулатор, името на файла, до който се иска достъп, и идентификаторите на потребителя и неговата група. Тези параметри се използват за определяне правата за достъп до указания файл. За да се попречи на неупълномощени потребители да четат или модифицират файлове, по правило идентификаторите на потребителя и на групата трябва да бъдат едни и същи и на двата хоста.

При повечето реализации на Unix функционалността на NFS клиента и сървъра се реализират като демони на ниво ядро, които се старти-

---

* Можете да се свържете с Rick на адрес *jrs@world.std.com*.

† Всъщност, аргументът `-t nfs` може да се изпусне, защото *mount* от двоеточието разбира, че е зададен NFS том.

рат от потребителското пространство при начално зареждане на системата. Такива са демоните *NFS Daemon* (*rpc.nfsd*) в хоста-сървър и *Block I/O Daemon* (*biod*) в хоста-клиент. За увеличаване на производителността *biod* извършва асинхронен вход/изход, използвайки предварително четене и отложен запис. Освен това, обикновено едновременно работят няколко демона *rpc.nfsd*.

Настоящата реализация на NFS за Linux малко се различава от класическата NFS по това, че кодът на сървъра се изпълнява изцяло в потребителското пространство, така че едновременната работа на няколко копия е по-сложна. Настоящата реализация на *rpc.nfsd* съдържа едно експериментална възможност, която позволява ограничена поддръжка на няколко сървъра. Олаф Кирх разработи базирана на ядрото поддръжка на NFS сървър, включена в ядрата на Linux версия 2.2. Работата ѝ е значително по-добра от съществуващата реализация за работа в потребителското пространство. Ще опишем тази поддръжка по-долу в тази глава.

Подготовка на NFS

Преди да започнете да използвате NFS, независимо дали като сървър или клиент, трябва да се уверите, че ядрото ви е компилирано с поддръжка на NFS. За целта по-новите ядра имат прост интерфейс във файловата система *proc* – файлът */proc/filesystems*, който можете да отпечатате с помощта на *cat*:

```
$ cat /proc/filesystems
    minix
    ext2
    msdos
nodev proc
nodev nfs
```

Ако в този списък липсва `nfs`, ще трябва да компилирате свое собствено ядро с разрешена NFS или може би ще трябва да заредите модула за ядрото, в случай че поддръжката на NFS е компилирана като модул. Конфигурирането на мрежовите опции на ядрото е описано в раздела “Конфигуриране на ядрото” на Глава 3, *Конфигуриране на мрежовия хардуер*.

Монтиране на NFS том

Монтирането на NFS томове много наподобява монтирането на обикновените файлови системи. Стартирайте mount като използвате следния синтаксис:*

```
# mount -t nfs nfs_том локална_директория опции
```

Форматът на параметъра *nfs_том* е *отдалечен_хост: отдалечена_директория*. Тъй като това означение е уникално за файловите системи NFS, можете да изпуснете опцията *-t nfs*.

Съществуват голям брой допълнителни опции, които можете да зададете на *mount* при монтиране на NFS том. Те могат да бъдат зададени или в командния ред след ключа *-o*, или в полето за опции на записа за тома във файла */etc/fstab*. И в двата случая отделните опции се разделят със запетаи, като не трябва да се използват никакви интервали. Опции, зададени в командния ред, винаги отменят опциите от файла *fstab*.

Ето един примерен запис от */etc/fstab*:

```
# том                 място за монтиране   тип   опции
news:/var/spool/news  /var/spool/news      nfs   timeo=14,intr
```

Този том може да бъде монтиран с командата:

```
# mount news:/var/spool/news
```

Ако липсва запис в *fstab*, извикването на командата NFS *mount* изглежда доста по-грозно. Да допуснем например, че монтирате личните директории на потребителите си от машина, наречена **moonshot**, която за операциите четене/запис по подразбиране използва размер на блока от 4K. За по-добра производителност може да увеличите размера на блока до 8K чрез командата:

```
# mount moonshot:/home /home -o rsize=8192,wsize=8192
```

Пълният списък на всички валидни опции е даден на справочната страница на *nfs(5)*. Следва частичен списък на опции, които може да ви се наложи да използвате:

* Томовете не се наричат "файлова система", защото те не са истински файлови системи.

rsize=n и wsize=n

Задават размера на дейтаграмите, използвани от NFS клиентите съответно при заявки за четене/запис. Стойността по подразбиране зависи от версията на ядрото, но обикновено е 1024 байта.

timeo=n

Задава времето (в десети от секундата), през което NFS клиентът ще чака заявката да се изпълни. Стойността по подразбиране е 7 (0.7 секунди). Какво става след изтичане на това време, зависи от това дали ще използвате опцията *hard* или *soft*.

hard

Изрично маркира този том като твърдо монтиран. Тази опция се използва по подразбиране. Тя указва на сървъра да изпрати съобщение на конзолата при дълга липса на отговор (т.нар. major timeout) и продължава опитите за изпълнение на заявката безкрайно.

soft

Гъвкаво монтиране на драйвера (противоположно на твърдото монтиране). Тази опция води до съобщение за I/O грешка до процеса, който се опитва да извършва действие с файл, когато дълго време няма отговор.

intr

Разрешава сигнали за прекъсване на NFS заявка. Тази опция е полезна за прекратяване на действието, когато сървърът не отговаря.

С изключение на *rsize* и *wsize*, всички останали опции се отнасят за поведението на клиента, в случай че сървърът стане временно недостъпен. Те работят съвместно по следния начин: Всеки път, когато клиентът изпраща заявка до NFS сървъра, той очаква, че операцията ще е свършила след определен интервал от време (зададен с опцията за *timeout*). Ако през това време не се получи потвърждение, настъпва т.нар. *малък таймаут* (*minor timeout*) и операцията се повтаря с удвоен интервал за *timeout*. След достигане на максималния *timeout* от 60 секунди, настъпва голям таймаут (*major timeout*).

По подразбиране при голям таймаут клиента да отпечата на конзолата съобщение и започва всичко отначало, като този път началният интервал за timeout е удвоен. По принцип това може да продължи

вечно. Томове, за които упорито се опитват операции, докато сървърът не стане отново достъпен, се наричат *твърдо монтирани*. Противоположният тип, наречен *гъвкаво монтиране*, генерира I/O грешка до извикващия процес всеки път, когато настъпи голям таймаут. Поради отложеното писане, съществуващо заради буферната кеш памет, това условие за грешка не се предава на самия процес до следващия път, когато той извика функцията *write*. Така че една програма никога не може да бъде сигурна дали операция за запис в гъвкаво монтиран том е завършила с истински успех.

Дали ще извършвате твърдо или гъвкаво монтиране на том, зависи до известна степен от предпочитанията ви, но също и от типа на информацията, която искате да използвате от даден том. Например, ако монтирате вашите X програми чрез NFS, със сигурност не бихте искали X сесията ви да полудее, само защото някой внезапно е претоварил мрежата, стартирайки едновременно седем копия на Doom или издърпвайки за момент Ethernet кабела. Когато извършвате твърдо монтиране на директорията, съдържаща тези програми, се уверете, че компютърът ви чака до момента, в който може отново да установи връзка с вашия NFS сървър. От друга страна, некритична информация като NFS-монтираните дялове за новини или FTP архиви могат да се монтират и гъвкаво, така че ако отдалечената машина временно е недостъпна или изключена, това няма да провали вашата сесия. Ако мрежовата ви връзка със сървъра понякога се разпада или минава през натоварен маршрутизатор, можете или да увеличите началния таймаут чрез опцията *timeo* или да монтирате томовете твърдо. По подразбиране NFS томовете са монтирани твърдо.

Твърдото монтиране представлява проблем, защото по подразбиране файловите операции не могат да се прекъсват. Затова, ако един процес се опита, например, да пише в отдалечен сървър и този сървър е недостъпен, потребителското приложение увисва и потребителят не може да направи нищо, за да прекрати операцията. Ако използвате опцията *intr* заедно с твърдото монтиране, всички сигнали, получени от процеса, прекъсват NFS заявката, така че потребителите все пак да могат да прекъснат увисналата файлова операция и да продължат да работят (макар и без да запишат файла).

Обикновено по един или друг начин демонът *rpc.mountd* следи директориите, монтирани от различните хостове. Можете да отпечатате тази информация с помощта на програмата *showmount*, която също е включена в пакета на NFS сървъра:

```
# showmount -e moonshot
Export list for localhost:
/home <anon clnt>

# showmount -d moonshot
Directories on localhost:
/home

# showmount -a moonshot
All mount points on localhost:
localhost:/home
```

NFS демоните

Ако искате да предоставите NFS услуга на други хостове, трябва да стартирате на машината си демоните *rpc.nfsd* и *rpc.mountd*. Като RPC-базирани програми, те не се управляват от *inetd*, а се стартират по време на зареждане на системата и се регистрират чрез portmapper. По тази причина трябва да сте сигурни, че стартирате демоните едва след като *rpc.portmap* вече работи. Обикновено в някой от мрежовите скриптове за начално зареждане би трябвало да използвате нещо, подобно на следния пример:

```
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
```

Информацията за собствеността на файловете, която демона NFS предоставя на клиентите си, обикновено съдържа само числови идентификатори на потребители и групи. Ако клиентът и сървърът асоциират едни и същи имена на потребители и групи с тези числови идентификатори, казваме, че те споделят едно uid/gid пространство. Такъв например е случаят, когато използвате NIS за разпространяването на информацията в *passwd* до всички хостове във вашата локална мрежа.

В някои случаи, обаче, идентификаторите в различните хостове не си съответстват. Вместо да актуализирате идентификаторите на клиента, така че да съответстват на тези на сървъра, можете да използвате демонът за съответствия *rpc.ugidd*, за да заобиколите това несъответствие. Използвайки опцията *map_deamon*, която описваме малко по-

долу, можете да укажете на *rpc.nfsd* да преобразува *uid/gid*-пространството на сървъра в *uid/gid*-пространство на клиента с помощта на *rpc.ugidd* на клиента. За съжаление не всички съвременни дистрибуции на Linux предоставят демона *rpc.ugidd*, така че, ако той ви е необходим, а не разполагате с него, ще трябва сами да го компилирате от изходния код.

rpc.ugidd е RPC-базиран сървър, който се стартира от мрежовите скриптове за начално зареждане, точно както *rpc.nfsd* и *rpc.mountd*:

```
if [ -x /usr/sbin/rpc.ugidd ]; then
        /usr/sbin/rpc.ugidd; echo -n " ugidd"
fi
```

Файлът exports

Сега ще разглеждаме начина за конфигуриране на NFS сървъра. По-специално, ще обърнем внимание на начина, по който указваме на NFS сървъра каква файлова система трябва да направи достъпна за монтиране, както и различните параметри, управляващи достъпа, който клиентите ще имат до файловата система. Сървърът определя типа на разрешения достъп до файловете на сървъра. Във файлът */etc/exports* се описват файловите системи, които сървърът ще направи достъпни за монтиране и използване от клиентите.

По подразбиране *rpc.mountd* забранява всякакво монтиране на директории, което е едно доста разумно решение. Ако искате да позволите на един или повече хоста да монтират NFS директория, трябва да я *предоставите*, т.е. да я зададете във файла *exports*. Един такъв примерен файл може да изглежда така:

```
# exports file for vlager
/home             vale(rw) vstout(rw) vlight(rw)
/usr/X11R6        vale(ro) vstout(ro) vlight(ro)
/usr/TeX          vale(ro) vstout(ro) vlight(ro)
/                 vale(rw,no_root_squash)
/home/ftp         (ro)
```

Всеки ред дефинира директория и хостовете, на които е разрешено монтирането ѝ. Едно име на хост обикновено е зададено като пълно домейн име, но може допълнително да съдържа маските * и ?, които имат същият смисъл, както и в обвивката Bourne shell. Например, `lab*.foo.com` съответства както на **lab01.foo.com**, така и на **laboratory.foo.com.** Хостът може да бъде зададен и с помощта на интервал от IP адреси във формат *адрес/мрежова маска*. Ако не е за-

дадено име на хост, какъвто е случаят с директорията */home/ftp* от предния пример, всеки хост отговаря на критерия и му е разрешено да монтира директорията.

Когато проверява за клиентския хост във файла *exports*, *rpc.mountd* търси името на хоста на клиента с помощта на функцията *gethostbyaddr*. При DNS тази функция връща каноничното име на хоста на клиента, затова трябва да сте сигурни, че в *exports* не използвате псевдоними. В NIS среда върнатото име е първото съответствие от базата данни *hosts*, а без DNS или NIS върнатото име е първото име на хост, намерено във файла *hosts*, което съответства на адреса на клиента.

Името на хоста е последвано от незадължителен, заграден в скоби списък от флагове, разделени със запетаи. Някои от стойностите, които тези флагове могат да приемат, са:

secure

Изисква заявките да се правят от запазен изходен порт, т.е. такъв, чийто номер е по-малък от 1024. Този флаг се задава по подразбиране.

insecure

Този флаг обръща действието на флага *secure*.

ro Указва, че NFS директорията трябва да се монтира само за четене. Този флаг е включен по подразбиране.

rw Тази опция монтира файлова йерархия за четене-запис.

root_squash

Тази функция за сигурност отказва всякакви специални права за достъп на суперпотребителите от зададените хостове, като преобразува заявките от uid 0 в клиента на uid 65534 (т.е., -2) в сървъра. Този идентификатор трябва да се асоциира с потребителя **nobody**.

no_root_squash

Не преобразувай заявки от uid 0. Тази опция е зададена по подразбиране, така че суперпотребителите имат достъп на суперпотребител до предоставените от вашата система директории.

link_relative

Тази опция преобразува абсолютни символни връзки (в които съдържанието на връзката започва с наклонена надясно черта) в относителни връзки. Тази опция има смисъл, само когато е монтирана цялата файлова система на хост. В противен случай някои от връзките може да не сочат наникъде, или дори по-лошо - да сочат към файлове, за които въобще не са били предназначени. Тази опция е включена по подразбиране.

link_absolute

Тази опция оставя всички символни връзки такива, каквито са (нормалното поведение на предоставяните от Sun NFS сървъри).

map_identity

Тази опция указва на сървъра да приеме, че клиентът използва същите идентификатори на потребители и групи като сървъра. Тази опция е включена по подразбиране.

map_daemon

Тази опция указва на NFS сървъра да приеме, че клиентът и сървърът не използват едно и също uid/gid пространство. В такъв случай *rpc.nfsd* изгражда списък, който преобразува идентификаторите между клиента и сървъра чрез запитване до демона *rpc.ugidd* на клиента.

map_static

Тази опция ви позволява да зададете името на файл, който съдържа статична карта на идентификаторите uid и gid. Например, `map_static=/etc/nfs/vlight.map` ще зададе файла */etc/nfs/vlight.map* като uid/gid карта. Синтаксисът на файла-карта е описан в справочната страница на *exports(5)*.

map_nis

Тази опция указва на NIS сървъра да извършва преобразуването на идентификаторите на потребители и групи.

anonuid и anongid

Тези опции ви позволяват да зададете идентификатор на потребител и идентификатор на група за анонимния акаунт. Тя е полезна, ако имате предоставен том за публично монтиране.

Всяка грешка при анализа на файла *exports* се съобщава на инструмента `daemon` на *syslogd* с ниво `notice` всеки път, когато се стартира *rpc.nfsd* или *rpc.mountd*.

Забележете, че имената на хостовете се получават от IP адреса на клиента чрез обратно преобразуване, така че резолверът трябва да бъде конфигуриран както трябва. Ако използвате BIND и сте много чувствителни на тема сигурност, трябва да разрешите проверката за замене във файла *host.conf*. Тази тема е обсъдена в Глава 6, *Конфигуриране на услугата за имена и резолвера*.

Базирана на ядрото поддръжка на NFSv2 сървър

NFS сървъра в потребителското пространство, който по традиция се използва в Linux, работи надеждно, но при претоварване се появяват проблеми с производителността. Това се получава преди всичко поради допълнителното натоварване, което интерфейсът на викане на системни функции добавя към работата на NFS сървъра и поради факта, че сървърния процес трябва да се конкурира за процесорно време с други, потенциално по-маловажни процеси в потребителското пространство.

Ядрото с версия 2.2.0 поддържа експериментален базиран на ядрото NFS сървър, създаден от Олаф Кирх и доразработен от H.J. Lu, G. Allan Morris и Trond Myklebust. Базираната на ядрото поддръжка на NFS дава значително по-голяма производителност на сървъра.

В настоящите дистрибуции можете да откриете инструментите на сървъра, достъпни в предварително пакетирана форма. Ако не ги откриете в дистрибуцията си, можете да ги намерите на адрес *http://csua.berkeley.edu/~gam3/knfsd/*. За да използвате тези инструменти, трябва да компилирате ядро 2.2.0, в което е включен базиран на ядрото NFS демон. Можете да разберете дали в ядрото ви е включен NFS демон като проверите дали съществува файла */proc/sys/sunrpc/nfsd_debug*. Ако няма такъв файл, може би трябва да заредите модула *rpc.nfsd* с помощта на инструмента *modprobe*.

Базираният на ядрото NFS демон използва стандартния конфигурационен файл */etc/exports*. Пакетът предоставя заместващи версии на демоните *rpc.mountd* и *rpc.nfsd*, които можете да стартирате по начин, много подобен на стартирането на съответните им демони в потребителското пространство.

Базирана на ядрото поддръжка на NFSv3 сървър

Версията на NFS, която се използваше най-често, беше NFS версия 2. Технологията се разви и започна да показва слабости, които биха могли да се преодолеят само с преработване на протокола. Версия 3 на Мрежовата файлова система поддържа по-големи файлове и файлови системи, предоставя значително по-добра система за сигурност и предлага голям брой подобрения на производителността, които ще са полезни на повечето потребители.

Олаф Кирх и Trond Myklebust разработват експериментален NFSv3 сървър. Той е включен в разработваната версия 2.3 на ядрата и съществува patch за изходния код на ядрата с версия 2.2. Този NFSv3 сървър е създаден на основата на версия 2 на базирания на ядрото NFS демон.

Patch-пакетите са на разположение в страницата на базирания на ядрото на Linux NFS сървър на адрес *http://csua.berkeley.edu/ ~gam3/knfsd/*.

ГЛАВА 15

IPX И ФАЙЛОВАТА СИСТЕМА NCP

Много преди Microsoft да научи за работата в мрежа и дори преди Интернет да стане известен извън академичните кръгове, в корпоративните среди се споделяха файлове и принтери с помощта на сървъри за файлове и печат, базирани на операционната система Novell NetWare* и асоциираните с нея протоколи. Много от тези корпоративни потребители все още имат наследени мрежи, използващи тези протоколи, и се нуждаят от интегриране на поддръжка за тях с тяхната нова поддръжка на TCP/IP.

Linux поддържа не само TCP/IP протоколи, но и група протоколи, използвани от операционната система NetWare на Novell Corporation. Тези протоколи са далечни братовчеди на TCP/IP и въпреки че изпълняват подобен род функции, се различават в много отношения и за съжаление са несъвместими.

За Linux има свободни и платени софтуерни предложения за осигуряване на поддръжка за интегрирането с продукти на Novell.

В тази глава ще дадем кратко описание на самите протоколи, но ще обърнем внимание и на това как се конфигурира и използва свободния софтуер, позволяващ на Linux да взаимодейства с продукти на Novell.

* Novell и NetWare са търговски марки на Novell Corporation.

Xerox, Novell и история

Първо, нека видим откъде идват протоколите и как изглеждат. В края на 70-те години Xerox Corporation разработи и публикува отворен стандарт, наречен XNS (Xerox Network Specification – мрежова спецификация на Xerox). Той описваше серия от протоколи, проектирани за осигуряване на работа от общ характер в свързани мрежи, като силно се наблягаше на използването на локални мрежи. Бяха включени два основни мрежови протокола: IDP (Internet Datagram Protocol – междумрежов протокол с дейтаграми), който осигуряваше ненадеждно и без гарантирана доставка пренасяне на дейтаграми от един хост до друг, и SPP (Sequenced Packet Protocol – протокол за последователни пакети), който беше базираща се на връзки модификация на IDP и предлагаше гарантирана доставка. Дейтаграмите в XNS мрежата се адресираха индивидуално. Схемата за адресиране използваше комбинация от 4-байтов IDP адрес на мрежа (който се задаваше уникално за всеки Ethernet LAN сегмент) и 6-байтов адрес на възел (адресът на мрежовата карта). Маршрутизаторите бяха устройства, които комутират дейтаграми между две или повече отделни IDP мрежи. В IDP нямаше дефиниция за подмрежа. Всяка нова група хостове изисква задаването на адрес на мрежа. Адресите на мрежи се избират така, че да бъдат уникални в разглежданите свързани мрежи. Понякога администраторите разработват конвенции, според които във всеки байт се кодира някаква друга информация, например географското място, така че адресите на мрежи да са систематично разпределени. Това обаче не е изискване на протокола.

Novell Corporation предпочете да базира своята собствена мрежова сюита на спецификацията XNS. Novell направи малки подобрения на IDP и SPP и ги преименува съответно на IPX (Internet Packet eXchange – размяна на пакети в свързани мрежи) и SPX (Sequenced Packet eXchange – последователна размяна на пакети). Novell добави нови протоколи като NCP (NetWare Core Protocol – базов протокол на NetWare), който осигурява услуги за споделяне на файлове и принтери, работещи под IPX, и протокола SAP (Service Advertisement Protocol – протокол за обявяване на услуги), който позволи на хостовете в мрежа на Novell да научават кой хост какви услуги осигурява.

На Таблица 15-1 е показана връзката между групи протоколи XNS, Novell и TCP/IP от функционална гледна точка. Връзките са само приблизителни, но би трябвало да ви помогнат да разберете какво става, когато по-късно споменаваме тези протоколи.

Таблица 15-1: Връзки между протоколите XNS, Novell и TCP/IP

XNS	Novell	TCP/IP	Възможности
IDP	IPX	UDP/IP	Без гарантирана доставка, ненадеждно пренасяне
SPP	SPX	TCP	Базирано на връзка, гарантирана доставка
	NCP	NFS	Файлови услуги
	RIP	RIP	Обмен на информация за маршрутизиране
	SAP		Обмен на информация за предоставяне на услуги

IPX и Linux

Alan Cox* създаде първата поддръжка на IPX за ядрото на Linux през 1995 г. В началото тя беше полезна почти само за маршрутизирането на IPX дейтаграми. Оттогава и други хора, особено Greg Page*, осигуряват допълнителна поддръжка. Greg разработи програми за конфигуриране на IPX, които ще използваме в тази глава за конфигуриране на нашите интерфейси. Volker Lendecke* разработи поддръжка за файловата система NCP, позволяваща на Linux да монтира томове в свързани към мрежата файлови сървъри NetWare. Той създаде и инструменти, които дават възможност за печат на и от Linux машини. Ales Dryak и Martin Stover* независимо един от друг разработиха демони за NCP файлов сървър под Linux, които позволяват на свързаните в мрежата NetWare клиенти да монтират Linux директории, предоставяни като NCP томове, точно както демона NFS дава възможност на Linux да обслужва файлови системи на клиенти, използвайки протокола NFS. Caldera Systems, Inc.* предлага комерсиални и напълно лицензирани NetWare клиент и сървър, поддържащи най-новите стандарти на Novell, включително поддръжка на NDS (NetWare Directory Service – директорийна услуга за NetWare).

---

* С Alan Cox можете да се свържете на адрес *alan@lxorguk.ukuu.org.uk*.

* С Greg Page може да се свържете на адрес *gpage@sovereign.org*.

* С Volker Lendecke може да се свържете на адрес *lendecke@namu01.gwdg.de*.

* С Ales може да се свържете на адрес *A.Dryak@sh.cvut.cz*, а с Martin на адрес *mstover@freeway.de*.

* Информация за Caldera може да бъде намерена на адрес *http://www.caldera.com/*.

И така, днес Linux поддържа голямо разнообразие от услуги, които позволяват на системите да се интегрират с съществуващи базирани на Novell мрежи.

Поддръжка от Caldera

Въпреки че в тази глава няма да се спираме подробно на поддръжката на NetWare от Caldera, важно е да кажем някои неща за нея. Caldera е основана от Ray Noorda – бившият CEO на Novell. Поддръжката на NetWare от Caldera е търговски продукт и изцяло се поддържа от Caldera. Caldera осигурява поддръжката на NetWare като част от собствената си дистрибуция на Linux, наречена Caldera OpenLinux. Решението на Caldera е идеален начин за въвеждането на Linux в среди, които изискват комерсиална поддръжка и възможност за интегриране със съществуващи или нови мрежи на Novell.

Поддръжката на NetWare от Caldera е напълно лицензирана от Novell, което осигурява висока степен на гаранция, че продуктите на двете компании ще са съвместими. Двете изключения на тази гаранция са "чистите IP" операции за клиента и NDS сървъра, въпреки че нито една от тях не е достъпна по време на писането на тази книга. На разположение са NetWare клиент и NetWare сървър. Осигурен е също комплект от инструменти за управление, който може да улесни управлението не само на вашите Linux-базирани NetWare машини, но също и на вашите Novell NetWare машини, използвайки за целта мощта на скрипт-езиците на Unix. Повече информация за Caldera можете да намерите в сайта на компанията.

Повече за поддръжката на NDS

Заедно с версия 4 на NetWare, Novell въведе възможност, наречена NDS (NetWare Directory Service – директорийна услуга за NetWare). Спецификациите на NDS не са достъпни без споразумение за неразкриване на търговската тайна – ограничение, което спъва развитието на свободната поддръжка. Само версия 2.2.0 или по-новите версии на пакета *ncpfs*, който ще обсъдим по-късно, имат някаква поддръжка на NDS. Тази поддръжка беше разработена чрез обратен инженеринг на протокола NDS. Поддръжката изглежда работеща, но официално все още се счита за експериментална. При работа с NetWare 4 сървъри можете да използвате инструменти, които не използват NDS, при условие, че на тези сървъри е разрешен "режим за емулация на базата данни bindery".

Софтуерът на Caldera има пълна поддръжка за NDS, защото неговата реализация е лицензирана от Novell. Тази реализация, обаче, не е свободна. Затова няма да имате достъп до изходния код и няма да можете свободно да копирате и разпространявате софтуера.

Конфигуриране на ядрото за IPX и NCPFS

Конфигурирането на ядрото за IPX и файловата система NCP е просто въпрос на подбор на подходящите опции по време на компилирането на ядрото. Също както при много други негови части, компонентите на ядрото за IPX и NCPFS могат да бъдат вградени в ядрото или компилирани като отделни модули и зареждани при необходимост с помощта на командата *insmod*.

Ако искате да имате поддръжка под Linux и да маршрутизирате протокола IPX, трябва да изберете следните опции:

```
General setup  --->
   [*] Networking support
Networking options  --->
   <*> The IPX protocol
Network device support  --->
   [*] Ethernet (10 or 100Mbit)
       ... и подходящите драйвери за Ethernet устройства
```

Ако искате Linux да поддържа файловата система NCP, така че да може да монтира отдалечени NetWare томове, освен горните, трябва да изберете и следните опции:

```
Filesystems  --->
   [*] /proc filesystem support
   <*> NCP filesystem support (to mount NetWare volumes)
```

След като компилирате и инсталирате новото ядро, вече сте готови да работите с IPX.

Конфигуриране на IPX интерфейси

Също както при TCP/IP, и тук трябва да конфигурирате вашите IPX интерфейси, за да можете да ги използвате. Протоколът IPX има някои специфични изисквания, в резултат на които е разработен специален набор от инструменти за конфигуриране. Ще използваме тези инструменти, за да конфигурираме нашите IPX интерфейси и маршрути.

Мрежови устройства, поддържащи IPX

Протоколът IPX допуска, че всяка група от хостове, които могат да си предават дейтаграми един на друг без маршрутизация, принадлежи на една и съща IPX мрежа. Всички хостове, които са свързани към един Ethernet сегмент, принадлежат на една и съща IPX мрежа. По подобен (но по-малко интуитивен) начин, и двата хоста, поддържащи PPP-базирана серийна връзка, трябва да принадлежат на IPX мрежа, която е самата серийната връзка. В Ethernet среда съществуват голям брой различни типове кадри, които могат да се използват за пренос на IPX дейтаграми. Типовете кадри представляват различни Ethernet протоколи и описват различни начини за пренос на множество протоколи през една и съща Ethernet мрежа. Най-разпространените типове кадри, които ще срещнете, са `802.2` и `ethernet_II`. Ще поговорим повече за типовете кадри в следващия раздел.

Мрежовите устройствата в Linux, които в момента поддържат протокола IPX, са Ethernet и PPP драйверите. Ethernet и PPP интерфейсите трябва да са активни, преди да могат да се конфигурират за работа с IPX. Обикновено трябва да конфигурирате Ethernet устройство както с IP, така и с IPX, затова устройството вече съществува, но ако мрежата ви е само IPX, ще се наложи да използвате *ifconfig*, за да активирате Ethernet устройството по следния начин:

```
# ifconfig eth0 up
```

Инструменти за конфигуриране на IPX интерфейс

Greg Page разработи набор от конфигурационни инструменти за IPX интерфейси, който е включен в съвременните дистрибуции като предварително компилиран пакет; изходния код може да бъде получен с анонимен FTP достъп от *ftp://metlab.unc.edu/*, файла */pub/Linux/system/filesystems/ncpfs/ipx.tgz*.

Един *rc*-скрипт обикновено стартира IPX инструментите по време на зареждането на системата. Вашата дистрибуция вероятно прави това за вас, ако имате инсталиран предварително пакетирания софтуер.

Командата ipx_configure

Всеки IPX интерфейс трябва да знае към коя IPX мрежа принадлежи и кой тип кадри да използва за IPX. Всеки хост, поддържащ IPX, има

поне един интерфейс, известен като *главен* интерфейс, който останалата част от мрежата ще използва, за да се обръща към хоста. Поддръжката на IPX в ядрото на Linux осигурява възможност за автоматично конфигуриране на тези параметри. Командата *ipx_configure* разрешава или забранява тази възможност за автоматично конфигуриране.

Изпълнена без аргументи, командата *ipx_configure* показва текущата настройка на флаговете за автоматично конфигуриране:

```
# ipx_configure
Auto Primary Select is OFF
Auto Interface Create is OFF
```

И двата флага Auto Primary и Auto Interface по подразбиране са изключени. За да ги зададете и активирате автоматичното конфигуриране, просто задайте аргументи като тези:

```
# ipx_configure --auto_interface=on --auto_primary=on
```

Когато стойността на аргумента `--auto_primary` е on, ядрото автоматично ще осигури поне един активен интерфейс да работи като главен интерфейс за хоста.

Когато стойността на аргумента `--auto_interface` е on, драйверът за IPX в ядрото ще следи всички кадри, получени от активните мрежови интерфейси, и ще се опита да определи адреса на IPX мрежата и използвания тип кадри.

Механизмът за автоматично откриване работи добре при правилно управлявани мрежи. Понякога обаче мрежовите администратори съкращават процедурата и нарушават правилата, което може да доведе до проблеми за кода в Linux за автоматично откриване. Най-често срещания пример за това е, когато една IPX мрежа е конфигурирана за работа в една и съща Ethernet мрежа с множество типове кадри. Това технически е невалидна конфигурация, тъй като един **802.2** хост не може директно да комуникира с Ethernet-II хост и следователно те не могат да бъдат в една и съща IPX мрежа. Мрежовият IPX софтуер на Linux следи сегмента за предадени IPX дейтаграми. От тях той се опитва да установи кои адреси на мрежи се използват и кой тип кадри се асоциира с всеки от тях. Ако един и същи адрес на мрежа се използва с множество типове кадри или на множество интерфейси, кодът в Linux възприема това като колизия на адреса на мрежа и не може да определи кой е правилният тип кадри. Ще разберете, че това се случва, ако в дневника на системата си видите съобщения, които изглеждат така:

```
IPX: Network number collision 0x3901ab00
eth0 etherII and eth0 802.3
```

Ако срещнете този проблем, забранете възможността за автоматично откриване и конфигурирайте ръчно интерфейсите, като използвате командата *ipx_interface*, описана в следващия раздел.

Командата ipx_interface

Командата *ipx_interface* се използва за ръчно добавяне, модифициране и изтриване на поддръжката на IPX за съществуващо мрежово устройство. Трябва да използвате *ipx_interface*, когато току-що описаният метод за автоматично конфигуриране не работи, или ако не искате да оставите конфигурирането на вашия интерфейс на случайността. *ipx_interface* ви позволява да задавате адреса на IPX мрежата, състоянието на главния интерфейс и типа на кадрите за IPX, които ще използва мрежовото устройство. Ако създавате множество IPX интерфейси, ще ви трябва по една команда *ipx_interface* за всеки един от тях.

Синтаксисът на командата за добавяне на IPX към съществуващо устройство е лесно разбираем и най-добре може да се обясни с пример. Да добавим IPX към съществуващо Ethernet устройство:

```
# ipx_interface add -p eth0 etherII 0x32a10103
```

Параметрите означават последователно:

-p Този параметър задава, че този интерфейс трябва да бъде главен интерфейс. Този параметър не е задължителен.

eth0

Това е името на мрежовото устройство, към което добавяме поддръжка на IPX.

etherII

Този параметър показва типа кадри, който в случая е Ethernet-II. Тази стойност може да бъде кодирана и като `802.2`, `802.3` или `SNAP`.

0x32a10103

Това е адреса на IPX мрежата, към която принадлежи този интерфейс.

Следващата команда премахва IPX от даден интерфейс:

```
# ipx_interface del eth0 etherII
```

И накрая, за да покажете текущата конфигурация на IPX за дадено мрежово устройство, използвайте:

```
# ipx_interface check eth0 etherII
```

Командата *ipx_interface* е обяснена по-пълно в нейната справочна страница.

Конфигуриране на IPX маршрутизатор

От кратката ни дискусия за използваните в една IPX среда протоколи вероятно си спомняте, че IPX е протокол за маршрутизиране и че протоколът RIP (Routing Information Protocol – протокол за информация за маршрутизиране) се използва за разпространяване информацията за маршрутизиране. IPX версията на RIP е много подобна на версията за IP. Те работят практически по един и същ начин. Маршрутизаторите перио-дично изпращат до всички данните, съдържащи се в техните таблици за маршрутизация, а останалите маршрутизатори се учат от тях, като слушат и интегрират информацията, която получават. На хостовете е необходимо да знаят само коя е тяхната локална мрежа и да са сигурни, че дейтаграмите за всички други направления се изпращат през техния локален маршрутизатор. Маршрутизаторът се грижи за преноса на тези дейтаграми и препредаването им за следващото ретранслиране от маршрута.

В IPX среда в мрежата трябва да се разпространява информация от още един клас. Протоколът SAP (Service Advertisement Protocol – протокол за обявяване на услуги) пренася информация за това, кои услуги от кои хостове са достъпни в мрежата. Например, SAP е този, който дава възможност на потребителите да получат списък с файловите сървъри или сървърите за печат в мрежата. Протоколът SAP работи благодарение на хостове, които предоставят услуги за периодично разпространение на списък с предлаганите от тях услуги. Мрежовите маршрутизатори за IPX събират тази информация и я разпространяват по мрежата заедно с информацията за маршрутизиране. За да сте отговарящ на стандартите IPX маршрутизатор, трябва да разпространявате и RIP, и SAP информация.

Точно като IP, IPX за Linux предоставя маршрутизиращ демон, наречен *ipxd*, който изпълнява задачите, свързани с управлението на маршрутизацията. И отново, точно както при IP, в действителност ядрото управлява придвижването на дейтаграми между мрежовите IPX интерфейси. То обаче извършва това съгласно набор от правила, наречен IPX таблица за маршрутизация. *ipxd* демонът поддържа този набор от правила актуален, слушайки на всеки от активните мрежови интерфейси и анализирайки кога е необходима промяна в маршрутите. Демонът *ipxd* отговаря и на заявки от хостове от директно свързана мрежа, които искат информация за маршрутизиране.

Командата *ipxd* е на разположение като предварително пакетирана програма в някои дистрибуции или в изходен код с анонимен FTP достъп до *ftp://metalab.unc.edu/* във файла */pub/Linux/system/filesystems/ncpfs/ipxripd-x.xx.tgz*.

Демонът *ipxd* не се нуждае от конфигуриране. Когато започне да работи, той автоматично управлява маршрутизирането между IPX устройствата, които са били конфигурирани. Важното е да се увериге, че вашите IPX устройства са конфигурирани правилно, като използвате командата *ipx_interface* преди да стартирате *ipxd* . Макар че автоматичното откриване може да работи, когато изпълнявате маршрутизиращи функции, по-добре е да не поемате рискове, затова конфигурирайте интерфейсите ръчно и ще си спестите мъките от неприятностите при маршрутизация. На всеки 30 секунди *ipxd* проверява отново всички локално свързани IPX мрежи и автоматично ги управлява. Това предоставя начин за управление на мрежи през интерфейси, които може да не са активни през цялото време, каквито са например PPP интерфейсите.

Обикновено *ipxd* трябва да се стартира по време на начално зареждане от *rc*-скрипт по следния начин:

```
# /usr/sbin/ipxd
```

Не е необходимо да се използва символът &, защото *ipxd* по подразбиране преминава във фонов режим. Въпреки че демонът *ipxd* е най-полезен в машини, които работят като IPX маршрутизатори, той е полезен и за хостове на сегменти, където са достъпни множество маршрутизатори. Когато зададете аргумента `-p`, *ipxd* ще работи пасивно, като следи за информация за маршрутизиране от сегмента и актуализира таблиците за маршрутизиране, но самият той няма да предава никаква информация за маршрутизиране. По този начин даден хост може да поддържа актуални своите таблици с маршрути,

без да се налага да търси маршрут всеки път, когато иска да се свърже с отдалечен хост.

Статична IPX маршрутизация с използване на командата ipx_route

Има случаи, в които бихме искали да зададем фиксиран IPX маршрут. Както при IP, можем да направим това и с IPX. Командата *ipx_route* записва маршрут в таблицата за маршрутизация на IPX, без да е необходимо демонът за маршрутизация *ipxd* да го е научил. Синтаксисът за маршрутизация е много прост (понеже IPX не поддържа подмрежови) и изглежда например така:

```
# ipx_route add 203a41bc 31a10103 00002a02b102
```

Показаната команда ще добави маршрут към отдалечената IPX мрежа **203a41bc** през маршрутизатора в локалната ни мрежа **31a10103** с адрес на възел **00002a02b102**.

Можете да намерите адреса на възела на даден маршрутизатор като използвате по подходящ начин командата *tcpdump* с аргумента `-e`, за да покажете заглавните части от мрежово ниво и потърсите трафик от маршрутизатора. Ако маршрутизаторът е Linux машина, можете просто да използвате командата *ifconfig*, за да видите това.

Можете да изтриете маршрут като използвате командата *ipx_route*:

```
# ipx_route del 203a41bc
```

Можете да видите списък с маршрутите, които са активни в ядрото, като погледнете във файла */proc/net/ipx_route*. До тук нашата таблица за маршрутизация изглежда така:

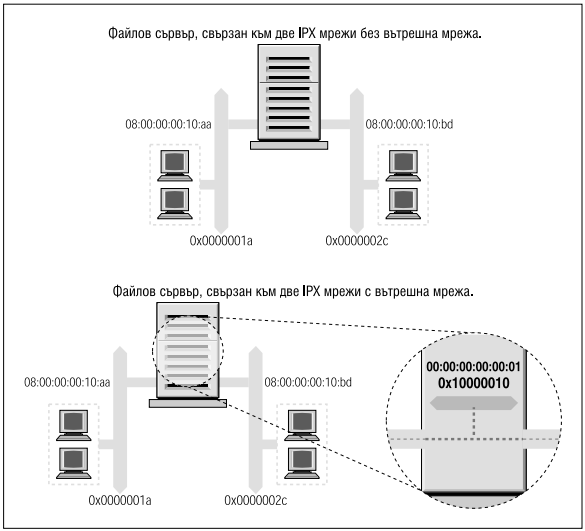
```
# cat ipx_route
Network     Router_Net   Router_Node
203A41BC    31A10103     00002a02b102
31A10103    Directly     Connected
```

Маршрутът към мрежата **31A10103** беше създаден автоматично, когато конфигурирахме IPX интерфейса. Всяка от нашите локални мрежи ще бъде представена с подобен запис в */proc/net/ipx_route*. Естествено, ако нашата машина играе ролята на маршрутизатор, тя ще се нуждае от поне още един интерфейс.

Вътрешни IPX мрежи и маршрутизатори

IPX хостовете с повече от един IPX интерфейс имат уникална комбинация за адреса на мрежа/възел за всеки от своите интерфейси. За да се свържете с такъв хост, можете да използвате всяка от тези комбинации на адреса на мрежа/възел. Когато SAP обявява услуги, той предоставя адреса на мрежата/възела, асоцииран с предлаганата услуга. При хостове с множество интерфейси това означава, че един от интерфейсите трябва да бъде избран за разпространяване; това е функцията на флага за главен интерфейс, за който говорихме порано. Това обаче създава проблем: маршрутът до този интерфейс може не винаги да е оптимален, а ако се появи проблем, който изолира тази мрежа от останалите мрежи, хостът ще остане недостъпен, въпреки че има други *възможни* маршрути през останалите интерфейси. Другите маршрути никога не се знаят от останалите хостовете, защото те никога не се разпространяват и дрогото няма как да разбере, че трябва да избере друг интерфейс за главен. За избягване на този проблем е разработено устройство, което дава възможност един IPX хост да бъде известен чрез един-единствен независим от маршрута адрес на мрежа/възел за целите на SAP разпространението. Това решава проблема ни, защото този нов адрес на мрежа/възел е достъпен през всички интерфейси на хоста и е адресът на мрежа/възел, който се обявява от SAP.

За да илюстрираме проблема и неговото решение на Фигура 15-1 показваме сървър, който е свързан към две IPX мрежи. Първата мрежа няма вътрешна мрежа, но втората има. Хостът от Фигура 15-1 ще избере един от интерфейсите си за главен интерфейс, да кажем **0000001a : 0800000010aa**, и това е, което ще бъде обявено като негова точка за достъп до услуги. Това работи безпроблемно за хостове в мрежата **0000001a**, но означава, че ако потребителите в мрежа **0000002c** са открили сървъра чрез SAP предаванията, те ще минават през първата мрежа, за да достигнат този порт, въпреки че сървърът има директна връзка към тяхната мрежа.



Фигура 15-1: вътрешна IPX мрежа

Този проблем се решава благодарение на възможността такива хостове да имат виртуална мрежа с виртуални адреси на хостовете, които са напълно софтуерна конструкция. Можете да си представите най-добре тази виртуална мрежа, като мислите за нея, че се намира *вътре* в IPX хоста. Тогава е необходимо да се разпространи SAP информация само за тази виртуална комбинация на адрес на мрежа/възел. Тази виртуална мрежа е известна като *вътрешна мрежа*. Но как другите хостове разбират как да достигнат до тази вътрешна мрежа? Отдалечените хостове определят маршрута до вътрешната мрежа чрез директно свързаните към хоста мрежи. Това означава, че ще видите записи за маршрути, които сочат към вътрешната мрежа за хостове, поддържащи множество IPX интерфейси. От тези маршрути трябва да се избере оптималният, достъпен за момента, и ако някой маршрут се провали, маршрутизацията автоматично се актуализира със следващия най-добър интерфейс и маршрут. На Фигура 15-1 конфигурирахме вътрешна IPX мрежа с адрес **0x10000010**, а за адрес на хоста използвахме **00:00:00:00:00:01**. Това е адресът, който ще бъде нашият главен интерфейс и който ще бъде обявен чрез SAP.

Нашата маршрутизация ще отразява тази мрежа като достижима през *всеки* от реалните портове на мрежата ни, така че хостовете винаги ще използват най-добрия мрежов маршрут за връзка с нашия сървър.

За да създадете тази вътрешна мрежа, използвайте командата *ipx_internal_net*, включена в пакета на Greg Page с IPX инструменти. Отново ще дадем един елементарен пример, демонстриращ използването ѝ:

```
# ipx_internal_net add 10000010 00000000001
```

Тази команда ще създаде вътрешна IPX мрежа с адрес **10000010** и адрес на възела **000000000001**. Адресът на мрежата, точно както всеки друг адрес на IPX мрежа, трябва да бъде уникален за вашата мрежа. Адресът на възела е напълно произволен, тъй като обикновено в мрежата има само един възел. Всеки хост може да има само една вътрешна IPX мрежа и ако е конфигурирана, вътрешната мрежа винаги ще бъде главната мрежа.

За да премахнете вътрешна IPX мрежа, използвайте:

```
# ipx_internal_net del
```

Вътрешна IPX мрежа ви е необходима само, ако вашият хост предоставя услуга *и* има повече от един активен IPX интерфейс.

Монтиране на отдалечен том на NetWare

Протоколът IPX обикновено се използва за монтиране на NetWare томове във файловата система на Linux. Това дава възможност за базирано на файлове споделяне на данни между Linux и други операционни системи. Volker Lendecke разработи NCP клиента за Linux и набор от помощни инструменти, които правят възможно споделянето на данни.

В NFS среда за монтиране на отдалечена файлова система бихме използвали Linux командата *mount*. За съжаление, файловата система NCP има уникални изисквания, които я правят неудобна за вграждане по нормалния начин в командата *mount*. Linux разполага с команда *ncpmount*, която ще използваме вместо *mount*. Командата *ncpmount* е един от инструментите в пакета *ncpfs* на Volker, който е включен в предварително пакетирана форма в повечето съвременни дистрибуции или може да бъде намерен в изходен код от **ftp.gwdg.de** в дирек-

торията */pub/linux/misc/ncpfs/*. Актуалната версия в момента на писането на тази книга е 2.2.0.

За да можете да монтирате отдалечени томове на NetWare, трябва да се уверите, че вашият мрежов IPX интерфейс е конфигуриран правилно (както бе описано по-горе). Освен това трябва да знаете подробностите за влизане в NetWare сървъра, от който искате да монтирате. Това включва идентификатора на потребителя и неговата парола. И накрая, трябва да знаете кой том искате да монтирате и в коя локална директория ще правите това.

Прост пример с командата ncpmount

Един прост пример за използването на командата *ncpmount* изглежда така:

```
# ncpmount -S ALES_F1 -U rick -P d00-b-gud /mnt/brewery
```

Тази команда монтира всички томове от файловия сървър `ALES_F1` в директорията */mnt/brewery*, като за влизане в NetWare се използва идентификатора `rick` с паролата `d00-b-gud`.

Командата *ncpmount* обикновено е конфигурирана setuid **root** и следователно може да се използва от всеки Linux потребител. По подразбиране този потребител притежава връзката и само той или потребителя **root** могат да я демонтират.

В NetWare се използва понятието *том*, което е аналогично на файлова система в Linux. Един NetWare том е логическо представяне на файлова система на NetWare, която може да бъде един дял от диск или разпределена върху много дялове. По подразбиране Linux поддръжката на NCPFS третира томовете като поддиректории на по-голяма логическа файлова система, съответстваща на целия файлов сървър. При използването на командата *ncpmount* всеки от NetWare томовете на монтирания файлов сървър се появява като поддиректория в точката на монтиране. Това е удобно, ако желаете достъп до целия сървър, но поради сложни технически причини няма да можете да предоставите тези директории от вашата машина като използвате NFS, ако искате да постъпите така. След малко ще обсъдим една по-сложна алтернатива, която заобикаля този проблем.

По-подробно за командата ncpmount

Командата *ncpmount* има голям брой опции, които ви позволяват доста голяма гъвкавост при извършване на NCP монтиране. Най-важните от тях са описани в Таблица 15-2.

Таблица 15-2: Аргументи на командата ncpmount

Аргумент	Описание
-S *server*	Името на файловия сървър, който ще се монтира.
-U *user_name*	NetWare идентификатора на потребителя, който ще се използва за влизане във файловия сървър.
-P *password*	Паролата, която ще се използва за влизане в NetWare.
-n	Тази опция трябва да се използва за NetWare идентификатори, за които не е зададена парола.
-C	Този аргумент забранява автоматичното преобразуване на пароли в главни букви.
-c *client_name*	Тази опция ви позволява да зададете кой притежава връзката към файловия сървър. Това е полезно за печат под NetWare, който ще обсъдим по-подробно малко по-късно.
-u *uid*	Linux идентификатора на потребителя, който трябва да бъде посочен като собственик на файлове в монтираната директория. Ако тази стойност не е зададена, по подразбиране ще се използва идентификатора на потребителя, който извиква командата *ncpmount*.
-g *gid*	Linux идентификатора на групата, която трябва да бъде посочена за собственик на файлове в монтираната директория. Ако тази стойност не е зададена, по подразбиране ще се използва идентификатора на групата на потребителя, който извиква командата *ncpmount*.

Аргумент	Описание
-f *file_mode*	Тази опция ви позволява да зададете файловия режим (правата за достъп) за файловете в монтираната директория. Стойността трябва да се зададе в осмична бройна система, например `0664`. Правата за достъп, които всъщност ще получите, са правата за достъп от файловия режим, зададени с тази опция, наложени върху правата за достъп до файловете във файловия сървър, които притежава потребителя, като който сте влезли в NetWare. Трябва да имате права на сървъра и права, зададени с тази опция, за да имате достъп до файла. Стойността по подразбиране се получава от текущата маска `umask`.
-d *dir_mode*	Тази опция ви позволява да зададете правата за достъп до директории в монтираната директория. Тя работи по същия начин като опцията *–f*, освен ако правата за достъп по подразбиране не са получени от текущата `umask`. Правото за достъп execute се дава там, където е позволен достъп за четене.
-V *volume*	Тази опция ви позволява да зададете името на един NetWare том, който да се монтира в мястото на монтиране, вместо да монтирате всички томове от зададения сървър. Тази опция е необходима, ако искате да предоставите монтиран NetWare том чрез NFS.
-t *time_out*	Тази опция ви позволява да зададете времето, през което NCPFS клиента ще чака отговор от сървъра. Стойността по подразбиране е 60ms, като timeout се задава в стотни от секундата. Ако имате някакви проблеми със стабилността при монтиране на NCP томове, трябва да се опитате да увеличите тази стойност.
-r *retry_count*	Програмният код на NCP клиента се опитва неколкократно да изпрати дейтаграми към сървъра, преди да реши, че връзката се е разпаднала. Тази опция ви позволява да промените броя на опитите, който по подразбиране е 5.

Скриване на паролата ви за влизане в NetWare

Поставянето на парола в командния ред, както направихме при командата *ncpmount*, е малко рисковано от гледна точка на сигурността. Останалите активни, работещи паралелно с вас потребители биха могли да видят паролата, ако случайно работят с програма като *top* или *ps*. За да се намали рискът от откриването и откраднването на пароли за влизане в NetWare, командата *ncpmount* е в състояние да чете определени детайли от файл в личната директория на потребителя. В този файл потребителят съхранява името за влизане и паролата за всеки от файловите сървъри, които възнамерява да монтира. Файлът се нарича *~/.nwclient* и трябва да има права за достъп `0600`, за да е сигурно, че останалите не могат да го четат. Ако правата за достъп не са коректни, командата *ncpmount* ще откаже да използва този файл.

Файлът има много елементарен синтаксис. Всички редове, които започват със символа # се смятат за коментари и се игнорират. Останалите редове имат следния синтаксис:

```
файлов-сървър/идентификатор-на-потребителя парола
```

`Файлов-сървър` е името на файловия сървър, поддържащ томовете, които желаете да монтирате. *`Идентификатор-на-потребителя`* е името за влизане във вашия акаунт за този сървър. Полето *`парола`* не е задължително. Ако то не е попълнено, командата *ncpmount* показва покана за въвеждане на паролата, когато потребителите се опитват да монтират. Ако в полето *`парола`* е зададен знакът `-` (минус), тогава не се използва парола; това е еквивалентно на използването на аргумента `-n` в командния ред.

Можете да зададете произволен брой записи, но полето за името на файловия сървър трябва да е уникално. Първият запис за файлов сървър има специално значение. Командата *ncpmount* използва аргумента от командния ред `-S`, за да определи кой от записите във файла *~/.nwclient* да използва. Ако не е зададен сървър с аргумента `-S` се използва сървърът от първия запис във файла *~/.nwclient* и се третира като ваш предпочитан сървър. Трябва на първо място в този файл да поставите файловия сървър, който най-често монтирате.

По-сложен пример с ncpmount

Да разгледаме един по-сложен пример с командата *ncpmount*, който включва много от описаните възможности. Първо да създадем един прост файл *~/.nwclient*:

```
# Детайли за влизане в NetWare на Виртуалната пивоварна и Винарната
#
# Влизане в Пивоварната
ALES_F1/MATT staoic1
#
# Влизане във Винарната
EDS01/MATT staoic1
#
```

Уверете се, че правата за достъп до този файл са коректни:

```
$ chmod 600 ~/.nwclient
```

Сега нека монтираме том от сървъра на Винарната в поддиректория на споделена директория, като зададем правата за достъп до файлове и директории така, че другите потребители да могат да използват съвместно данни от там:

```
$ ncpmount -S REDS01 -V RESEARCH -f 0664 -d 0775/usr/share/winery/data/
```

Тази команда в комбинация с показания файл *~/.nwclient* би трябвало да монтира в директорията */usr/share/winery/data/* тома `RESEARCH` от сървъра `REDS01`, като за влизане в NetWare използва идентификатора `MATT` и паролата, получената от файла *~/.nwclient*. Правата за достъп до монтираните файлове са `0664`, а правата за достъп до директорията са `0775`.

Изследване на някои от останалите IPX инструменти

Пакетът *ncpfs* съдържа голям брой полезни инструменти, които все още не сме описали. Много от тези инструменти емулират инструментите, предоставяни с NetWare. В този раздел ще разгледаме най-полезните от тях.

Списък на сървърите

Командата *slist* извежда списък на всички достъпни за хоста файлови сървъри. Всъщност, информацията се извлича от най-близкия IPX маршрутизатор. Вероятно първоначалното предназначение на тази команда е било да позволи на потребителите да видят кои файлови сървъри могат да монтират. Тя обаче стана полезна като инструмент за диагностика на мрежата, позволяващ на мрежовите администратори да видят къде се разпространява SAP информация:

```
$ slist
NPPWR-31-CD01                    23A91330   000000000001
V242X-14-F02                     A3062DB0   000000000001
QITG_284ELI05_F4                 78A20430   000000000001
QRWMA-04-F16                     B2030D6A   000000000001
VWPDE-02-F08                     35540430   000000000001
NMCS_33PARK08_F2                 248B0530   000000000001
NCCRD-00-CD01                    21790430   000000000001
NWGNG-F07                        53171D02   000000000001
QCON_7TOMLI04_F7                 72760630   000000000001
W639W-F04                        D1014D0E   000000000001
QCON_481GYM0G_F1                 77690130   000000000001
VITG_SOE-MAIL_F4R                33200C30   000000000001
```

Командата *slist* не приема аргументи. Изходът ѝ показва името на файловия сървър, адреса на IPX мрежата и адреса на хоста.

Изпращане на съобщения до NetWare потребители

NetWare поддържа механизъм за изпращане на съобщения до влезлите в системата потребители. Командата *nsend* реализира тази възможност под Linux. За да изпращате съобщения, трябва да сте влезли в сървъра, затова е необходимо в командния ред да зададете името на файловия сървър и детайли за влизане в него, както и потребителя-получател и съобщението, което се изпраща:

```
# nsend -S vbrew_f1 -U gary -P j0yj0y supervisor
    "Ела да изпием по бира, преди да се захванем с опашките за печат!"
```

Тук потребител на име gary изпраща примамлива покана, като използва акаунта `supervisor` на файловия сървър `ALES_F1`. Ако не зададем нашия файлов сървър и параметрите за влизане, ще бъдат използвани тези по подразбиране.

Преглеждане и управление на базата данни bindary

Всеки файлов сървър NetWare поддържа база данни с информация за своите потребители и конфигурация. Тази база данни се нарича *bindery*. Linux поддържа комплект инструменти, с помощта на които можете да четете от нея, а ако имате супервайзорски права на сървъра, можете и да я настройвате и премахвате. Тези инструменти са описани накратко в Таблица 15-3.

Таблица 15-3: Linux инструменти за управление на базата данни bindery

Име на командата	Описание на командата
nwfstime	Показва или настройва датата и часът в NetWare сървър
nwuserlist	Изброява потребителите, влезли в NetWare сървър
nwvolinfo	Показва информация за NetWare томове
nwbocreate	Създава обект в базата данни bindery на NetWare
nwbols	Изброява обектите в базата данни bindery на NetWare
nwboprops	Изброява свойствата на обекти в базата данни bindery на NetWare
nwborm	Премахва обект от базата данни bindery на NetWare
nwbpcreate	Създава свойство в базата данни bindery на NetWare
nwbpvalues	Отпечатва съдържанието на свойство в базата данни bindery на NetWare
nwbpadd	Задава стойността на свойство в базата данни bindery на NetWare
nwbprm	Премахва свойство от базата данни bindery на NetWare

Печат в NetWare опашка за печат

Пакетът *ncpfs* съдържа малка програма, наречена *nprint*, която изпраща задания за печат към NetWare опашка за печат през NCP връзка. Тази команда създава връзката, ако тя още не съществува и използва файла *~/.nwclient*, който описахме по-горе, за да скрие потребителското име и паролата от любопитни очи. Параметрите от командния ред, използвани за управление на процеса на влизане, са същите като използваните от *ncpmount*, затова тук няма да ги описваме отново. В примерите ще разгледаме само най-важните опции на командния ред; за повече подробности виж справочната страница на *nprint (1)*.

Единствената задължителна опция на *nprint* е името на файла, който ще се отпечатва. Ако името на файла е зададено като – или въобще не е дефинирано, *nprint* ще приеме задачата за печат от стандартния

входен поток `stdin`. Най-важните опции на *nprint* задават файловия сървър и опашката за печат, към която желаете да изпратите задачата. В Таблица 15-4 са изброени най-важните опции.

Таблица 15-4: Опции от командния ред на nprint

Опция	Описание
-S *име-на-сървъра*	Името на NetWare файловия сървър, който поддържа опашката на печат, чрез което желаете да печатате. Обикновено е удобно за сървъра да има запис във *~/.nwclient*. Тази опция е задължителна.
-q *име-на-опашка*	Опашката за печат, към която ще се изпраща задачата за печат. Тази опция е задължителна.
-d *описание-на-задачата*	Текст, който ще се появи в конзолата на обслужващата програма за печат при показване на списъка с чакащите задачи.
-l *редове*	Брой на редовете в отпечатана страница. По подразбиране са 66.
-r *колони*	Брой на колоните в отпечатана страница. По подразбиране са 80.
-c *копия*	Брой на копията на задачата, което ще се печата. По подразбиране е 1.

Един прост пример с използването на *nprint* изглежда така:

```
$ nprint -S REDS01 -q PSLASER -c 2 /home/matt/ethylene.ps
```

Тази команда ще отпечата две копия от файла */home/matt/ethylene.ps* на принтера `PSLASER` на файловия сървър `REDS01`, като използва потребителско име и парола, получени от файла *~/.nwclient*.

Използване на nprint заедно с демона lpd

Както си спомняте, по-горе споменахме, че опцията -c за *ncpmount* е полезна за печат. Най-после ще ви обясним защо и как.

Linux обикновено използва BSD-стил софтуер за управление на печата. Демонът за печат *lpd* (line printer daemon) проверява в локална буферна директория за поставени в опашка задачи за печат. *lpd* чете името на принтера и някои други параметри от специално форматиран буферен файл и изпраща данни на печатащото устройство, като по избор може да прекара данните през филтър, за да бъдат трансформирани или обработени по някакъв начин.

Демонът *lpd* използва проста база данни, наречена */etc/printcap*, за съхранение на информация за конфигурацията на принтера, включително кои филтри трябва да бъдат използвани. Обикновено *lpd* работи с правата на специален системен потребител, наречен **lp**.

Бихте могли да конфигурирате *nprint* като филтър, който да се използва от *lpd*, което дава възможност на потребителите на вашата Linux машина да извеждат информация директно към отдалечени печатащи устройства, управлявани от файлов сървър NetWare. За да направи това, потребителя **lp** трябва да може да дава NCP заявки към NCP връзката със сървъра.

Един лесен начин за постигането на това, без изискването потребителя **lp** да има собствена връзка и име за влизане е да зададете **lp** като собственик на връзка, установена от друг потребител. Пълен пример за настройката на системата за печат на Linux за обслужване на задачите за печат към NetWare е даден в следните три стъпки:

1. Напишете обвиващ скрипт:

   Файлът */etc/printcap* не позволява на филтриге да се задават опции. Затова е необходимо да създадете кратък скрипт, който извиква командата, която искате, заедно с нейните опции. Помощният скрипт би могъл да бъде просто:

   ```
   #!/bin/sh
   # p2pslaser – прост скрипт за пренасочване на stdin
   #   към опашката PSLASER на сървъра REDS01
   #
   /usr/bin/nprint -S REDS01 -U stuart -q PSLASER
   #
   ```

   Запишете скрипта във файла */usr/local/bin/p2pslaser*.

2. Създайте запис за */etc/printcap*

   Трябва да конфигурираме скрипта *p2pslaser*, който създадохме, като изходен филтър в */etc/printcap*. Това ще изглежда нещо такова:

```
pslaser|Postscript лазерен принтер, намиращ се на NetWare сървър:\
:lp=/dev/null:\
:sd=/var/spool/lpd/pslaser:\
:if=/usr/local/bin/p2pslaser:\
:af=/var/log/lp-acct:\
:lf=/var/log/lp-errs:\
:pl#66:\
:pw#80:\
:pc#150:\
:mx#0:\
:sh:
```

3. Добавете опцията –c към *ncpmount*.

```
ncpmount -S REDS01 .... -c lp ....
```

Нашият локален потребител **stuart** трябва да зададе потребителя **lp** като собственик на връзката, след като монтира отдалечения NetWare server.

Сега всеки Linux потребител може зададе `pslaser` като име на принтер, когато извиква **lp**. Задачата за печат ще бъде изпратена към декларирания NetWare сървър и поставена в опашка за печат.

Управление на опашките за печат

Командата *pqlist* изброява всички достъпни за вас опашки за печат в указания сървър. Ако не декларирате файлов сървър в командния ред с опцията *-S* или име и парола за влизане, те ще бъдат взети от записа по подразбиране във вашия файл *~/.nwclient*:

```
# pqlist -S vbrew_f1 -U guest -n
Server: ALES_F1
Print queue name                    Queue ID
----------------------------------------

TEST                                AA02009E
Q2                                  EF0200D9
NPI223761_P1                        DA03007C
Q1                                  F1060004
I-DATA                              0D0A003B
NPI223761_P3                        D80A0031
```

Нашият пример показва списък с опашките за печат, достъпни за потребителя `quest` на файловия сървър `ALES_F1`.[30]

За да разгледате задачите за печат в опашка за печат, използвайте командата *pqstat*. Тя приема като аргумент името на опашката за печат и показва списък с всички задачи в нея. Освен това, като опция можете да подадете още един аргумент, който указва колко от задачите в опашката бихте искали да бъдат изброени. Следните примерни изходни данни бяха малко сбити, за да се съберат на страницата на тази книга:

```
$ pqstat -S ALES_F1 NPI223761_P1
Server:  ALES_F1        Queue: NPI223761_P1     Queue ID: 6A0E000C

   SeqName          Description                 Status Form  Job ID
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      1  TOTRAN      LyX document - proposal.lyx Active 0  02660001
```

В опашката виждаме само една задача за печат, която е собственост на потребителя `TOTRAN`. Останалите опции включват описание на задачата, нейното състояние и идентификатор.

Командата *pqrm* се използва за изтриване на задачи от зададената опашка за печат. За да изтрием от опашката задачата, на която току що научихме състоянието, можем да използваме:

```
$ pqrm -S ALES_F1 NPI223761_P1 02660001
```

Синтаксисът на тази командата е очевиден, но е тромав, в случай че бързате. Струва си написването на базисен скрипт, който да опрости тази операция.

Емулация на NetWare сървър

Съществуват два независими програмни емулатора за файлови сървъри NetWare под Linux. Сървърът *lwared* бе разработен от Ales Dryak, а *mars_nwe* – от Martin Stover. И двата пакета предоставят проста

[30] Изглежда системните администратори са решили да изпробват някои от изделията на виртуалната пивоварна, преди да изберат имена на опашките за печат. Надяваме се, че имената на вашите опашки за печат ще са по-смислени!

емулация на файловите сървъри NetWare под Linux, като дават възможност на NetWare клиентите да монтират Linux директории, предоставяни като NetWare томове. Сървърът *lwared* е по-лесен за конфигуриране, но *mars_nwe* има повече възможности. Инсталирането и конфигурирането на тези пакети е извън обхвата на тази глава, но и двата са описани в документа IPX-HOWTO.

ГЛАВА 16

УПРАВЛЕНИЕ НА TAYLOR UUCP

UUCP беше проектиран в края на 70-те от Mike Lesk в AT&T Bell Laboratories за предоставяне на проста комутируема мрежа през публични телефонни линии. Въпреки популярността на PPP и SLIP връзките към Интернет през телефонна линия, много хора, които искат да имат електронна поща и Usenet новини на домашната си машина, все още използват UUCP, тъй като често пътите по-евтин, особено в страни, в които потребителите на Интернет трябва да плащат на минута за локални телефонни разговори или в страни, в които потребителите нямат локален ISP и трябва да плащат такси за разговори на далечни разстояния, за да се свържат. Въпреки че има много реализации на UUCP, които работят на различни хардуерни платформи и операционни системи, като цяло те са високо съвместими.

Освен това, повечето софтуер по някакъв начин през годините е станал "стандартен", не съществува UUCP, който би могъл да се нарече *истинското* UUCP. UUCP е претърпял е сериозно развитие, откакто беше реализирана първата версия през 1976. В момента съществуват две основни разновидности, които се различават главно в хардуерната поддръжка и конфигурация. И на двете разновидности съществуват разнообразни реализации, всяка от които се различава незначително от нейните родственици.

Едната разновидност е известна като UUCP Версия 2, която е реализирана през 1977 от Mike Lesk, David A. Novitz и Greg Chesson. Въпреки че е доста стара, тя все още се използва често. Последните реализации на Версия 2 предоставят голяма част от удобството, което се предлага от по-новите разновидности на UUCP.

Втората разновидност беше разработена през 1983 и обикновено е известна като BNU (Basic Networking Utilities) или HoneyDanBer UUCP. Второто име произлиза от имената на авторите (P. Honeyman, D. A. Novitz и B. E. Redman) и често се съкращава допълнително до HDB – терминът, който ще използваме в тази глава. HDB беше измислена за отстраняване на някои от недостатъците на UUCP Версия 2. Например, бяха добавени нови протоколи за прехвърляне, а spool-директорията беше разделена, така че сега за всеки сайт, с който реализирате UUCP трафик, съществува една директория.

Реализацията на UUCP, която в момента се разпространява с Linux, е Taylor UUCP 1.06. Това е версията, на която е базирана тази глава.[31] Taylor UUCP Версия 1.06 беше реализирана през август 1995. Освен традиционните конфигурационни файлове, Taylor UUCP може да бъде компилирана, така че да разпознава конфигурационните файлове с нов стил – - a.k.a. Taylor.

Обикновено, Taylor UUCP се компилира за HDB съвместимост, конфигурационната схема на Taylor или и за двете. Тъй като схемата на Taylor е много по-гъвкава и вероятно по-лесна за разбиране, отколкото често срещаните неясни конфигурационни файлове на HDB, ще опишем тази схема по-нататък в тази глава.

Замисълът на тази глава не е изчерпателно описание на опциите от командния ред за UUCP командите и тяхното действие. Главата представлява едно въведение в начина, по който се настройва работещ UUCP възел. Първият раздел е въведение в начина, по който UUCP реализира отдалечено изпълнение и прехвърляне на файлове. Ако имате някакъв опит с UUCP, може би ще искате да прескочите направо на раздела "Конфигурационни файлове на UUCP" по-нататък в тази глава, в който са разгледани различните файлове за настройка на UUCP.

---

31 Написана от Ian Taylor през 1995, който притежава и авторските права.

Ще приемем обаче, че сте запознати с потребителските програми от комплекта UUCP - *uucp* и *uux*.

За описание се обърнете към електронните справочните страници.

Освен публично достъпните програми *uucp* и *uux*, комплектът UUCP съдържа голям брой команди, използвани само за административни цели. Те са използвани за наблюдение на UUCP трафика през вашия възел, премахване на стари дневници или за събиране на статистики. Нито едно от тези неща обаче няма да бъде описано тук, тъй като те са в страни от главните задачи на UUCP. Освен това, те са добре документирани и доста лесни за разбиране; за повече информация погледнете справочните страници. Все пак, съществува и трета категория, която се включва истинските "работни коне" на UUCP. Наричат се *uucico* (където *cico* е съкращение от copy-in copy-out) и *uuxqt*, които изпълнява задачи, изпратени от отдалечени системи. В тази глава ще се концентрираме върху тези две важни програми.

Ако не сте удовлетворени от предложената от нас информация по тези теми, можете да прочетете документацията, която се разпространява с пакета UUCP. Това е набор от Texinfo файлове, които описват настройването чрез използване на конфигурационната схема на Taylor. Можете да преобразувате Texinfo файловете в *dvi* файлове, използвайки *texi2dvi* (от Texinfo пакета във вашата дистрибуция), и да визуализирате файла *dvi* чрез командата *xdvi*.

Документа UUCP-HOWTO на Guylhem Aznar е друг добър източник на информация за UUCP в Linux среда. Той е достъпен от във всяко огледално копие на проекта за документиране на Linux и периодично се публикува в *comp.os.linux.answers*.

Съществува и група по интереси за дискусия на UUCP, наречена *comp.mail.uucp*. Ако имате специфични за Taylor UUCP въпроси, може би ще е по-добре да ги зададете там, отколкото в групите *comp.os.linux.**.

UUCP прехвърляне и отдалечено изпълнение

Концепцията за задача е от голямо значение за разбирането на UUCP. Всяко прехвърляне, което даден потребител инициира с *uucp* или *uux*, се нарича задача. Задачата се състои от команда, която ще се изпълнява на отдалечената система, група от файлове, които ще се пренасят между сайтове или и от двете.

Например, следващата команда указва на UUCP да копира файла *netguide.ps* в отдалечен хост, наречен **pablo**, и да изпълни командата *lpr* на **pablo**, за да отпечата файла:

```
$ uux -r pablo!lpr !netguide.ps
```

UUCP обикновено не извиква веднага отдалечената система, за да изпълни задача (иначе бихте могли да се задоволите с *Kermit*). Вместо това, той временно съхранява някъде описанието на задачата. Това се нарича *спулинг* (*spooling*). Следователно, дървото на директорията, под която се съхраняват задачите, се нарича *буферна (спул) директория* и обикновено е разположена в */var/spool/uucp*. В нашия пример описанието на задача ще съдържа информация за отдалечената команда, която ще бъде изпълнена (*lpr*), потребителят, който заявява изпълнението, и два други елемента. Като допълнение към описанието на задача, UUCP трябва да съхрани входния файл *netguide.ps*.

Точното местоположение и именуване на файлове в буфера може да варира в зависимост от някои опции, зададени при компилация. UUCP пакетите, които са HDB-съвместими, обикновено съхраняват буферните файлове в поддиректория на */var/spool/uucp* с името на отдалечения сайт. След като се компилира за конфигурация на Taylor, UUCP създава поддиректории за различни типове буферни файлове под специфичната за сайта буферна директория.

През определени интервали от време, UUCP набира отдалечената система. Когато се създаде връзка с отдалечената машина, UUCP пренася файловете, които описват задачата, заедно с всички входни файлове. Входящите задачи няма да бъдат изпълнени веднага, а само след като връзката прекъсне. Изпълнението се управлява от *uuxqt*, която се грижи и за препредаването на всякакви задачи, предназначени за друг сайт.

За да различава по-важните и по-маловажните задачи, UUCP асоциира *степен* с всяка задача. Това е една-единствена цифра, варираща от 0 до 9, от A до Z и от a до z с намаляващо предпочитание. За поща обикновено се използва степен B или C, а за новини – степен N. Задачи с по-високи степени се прехвърлят по-рано. Степените могат да се задават чрез флага *–g* при извикване на *uucp* или *uux*.

Освен това, в определени моменти можете да забраните прехвърлянето на задачи под определена степен. За да направим това, задаваме *максимална степен за буфера*, която ще бъде забранена по време на диалог. По подразбиране, тя е z и означава, че всеки път всички степени ще бъдат пренасяни. Забележете семантичното двусмислие тук:

даден файл се пренася, само ако има степен *равна на* или *по-голяма от* прага на максималната степен на буфера.

Вътрешен начин на работа на uucico

За да се разбере защо е необходимо *uucico* да знае определена информация, едно кратко описание на начина, по който всъщност се свързва с отдалечена система, ще е от полза.

Когато изпълните *uucico –s система* от командния ред, *uucico* първо трябва да се свърже физически. Предприетите действия зависят от типа на връзката, която ще се създаде. По този начин, когато използваме телефонна линия, тя трябва да отвори модем и да го набере. През TCP програмата трябва да извика *gethostbyname*, за да преобразува името в мрежов адрес, да намери кой порт да отвори и да свърже адреса със съответния сокет (socket).

Една успешна връзка е последвана от упълномощаване. Тази процедура обикновено се състои от отдалечената система, питаща за потребителското име за влизане и вероятно за парола. Тази размяна обикновено се нарича *chat за влизане*. Процедурата за упълномощаване се изпълнява или от обичайния комплект *getty/login*, или през TCP sockets от самата програма *uucico*. Ако упълномощаването е успешно, отдалеченият край стартира *uucico*. Локалното копие на *uucico*, което инициира връзката, се разглежда като *master*, а отдалеченото копие – като *slave*.

След това следва *фазата на договаряне*: master изпраща името на неговия хост плюс няколко флага. Slave проверява това име на хост за право за достъп за влизане, изпраща и получава файлове и т.н. Флаговете описват (наред с други неща) максималната степен на файловете в буфера за прехвърляне. Ако е разрешена, тук се извършва проверка за брой на диалозите или за пореден номер на обаждането. Чрез тази възможност и двете страни поддържат общ брой на успешните връзки, които се сравняват. Ако те не съвпадат, договарянето е неуспешно. Това е полезно, за да се защитите от измамници.

Накрая, двете *uucico* правят опит да се договорят за общ *протокол за прехвърляне*. Този протокол управлява начина, по който се пренасят данните, които се проверяват за устойчивост и се връщат обратно в случай на грешка. Необходими са различни протоколи поради различните типове връзки, които се поддържат. Например, телефонните линии изискват "сигурен" протокол, който е песимистичен относно грешки, докато TCP трансмисията е надеждна и може да използва

по-ефиктивен протокол, който предшества повечето допълнителни проверки за грешки.

След като договарянето е завършило, започва фазата на действителното предаване. И двете страни включват избрания драйвер на протокола. В тази точка, драйверите вероятно изпълняват специфична за протокола последователност при инициализация.

След това master изпраща всички файлове от опашката за отдалечената система, чиито степени на буфера са достатъчно високи. След като приключи с това, той уведомява slave, който вече може да прекъсне връзката. Сега slave може или да се съгласи да прекъсне връзката, или да приеме диалога. Това е смяна на ролите: сега отдалечената система става master, а локалната – slave. Новият master изпраща своите файлове. След това двете *uucico* програми обменят съобщения за прекъсване и затварят връзката.

Ако се нуждаете от допълнителна информация за UUCP, можете да използвате изходния код. Съществува и една наистина антична статия, която се носи из Мрежата[32]. Тази статия е писана е от David A. Novitz и дава подробно описание на протокола UUCP. Taylor UUCP FAQ също разглежда някои подробности, свързани с реализацията на UUCP, и се публикува редовно в *comp.mail.uucp*.

Опции на uucico от командния ред

В този раздел част ще опишем най-важните опции от командния ред за *uucico*:

- - system, -s `system`

Извиква именуваната `система`, освен ако не е забранена от ограничения за времето на обаждане.

-S `system`

Извиква именуваната `система` безусловно.

- -master, -r1

Стартира *uucico* в master режим. Това е подразбираща се опция, когато е зададено *–s* или *-S*. Като цяло, опцията *-r1* указва на *uucico* да направи опит да извика всички системи във файла *sys*,

[32] Тя е включена и в ръководството на 4.4BSD *System Manager*.

описан в следващия раздел на тази глава, освен ако не са забранени от ограничения за обаждане или времето за нов опит.

- *-slave, -r0*

  Стартира *uucico* в slave режим. Това е подразбиращата се опция, когато не е зададено *–s* или *–S*. В slave режим или се приема стандартния вход/изход да бъде свързан към сериен порт, или се използва TCP порта, указан от опцията *–p*.

- *-ifwork, -C*

  Тази опция допълва *–s* или *–S* и указва на *uucico* да извика посочената система, само ако в буфера има процеси за нея.

- *-debug* `type, -x type, -X type`

  Включва отстраняване на грешки на зададения тип. Няколко типа могат да бъдат зададени като разделен със запетаи списък. Следващите типове са валидни: `abnormal, chat, handshake, uucp-proto, proto, port, config, spooldir, execute, incoming` и `outgoing`.

  Чрез `all` се включват всички опции. Вместо това, за съвместимост с други UUCP реализации може да се укаже число, което включва отстраняването на грешки за първите *n* елемента от горния списък.

  Съобщения за отстраняване на грешки ще бъдат записвани във файла *Debug*, който се намира под */var/spool/uucp*.

Конфигурационни файлове на UUCP

За разлика от по-простите програми за прехвърляне на файлове, UUCP беше проектиран да може да обработва всички прехвърляния автоматично. След като веднъж е настроен правилно, не би трабвало да е необходима ежедневната намеса на администратора. Информацията, която се изисква за автоматизирано прехвърляне, се съхранява в двойка конфигурационни файлове, които се намират в директорията */usr/lib/uucp*. Повечето от тези файлове се използват само при набиране.

Внимателно въведение в Taylor UUCP

Бихме омаловажили нещата, ако просто кажем, че конфигурирането на UUCP е трудно. Това е наистина твърде сложна тема и понякога сбитият формат на конфигурационните файлове не прави нещата полесни (въпреки че форматът на Taylor е почти лесен за четене в сравнение с по-старите формати в HDB или Версия 2).

За да разберете как си взаимодействат всички конфигурационни файлове, ще ви запознаем с най-важните от тях и ще разгледаме примерни записи от тези файлове. Сега няма да обясняваме всичко подробно; по-точно описание е дадено в отделните раздели на тази глава. Ако искате да настроите машината си за UUCP, най-добре е да започнете с няколко примерни файла и постепенно да ги адаптирате за вашата машина. Можете да използвате или показаните по-долу примерни файлове, или тези, които са включени в любимата ви дистрибуция на Linux.

Всички описани в този раздел файлове се съхраняват в */etc/uucp* или в нейна поддиректория. Някои дистрибуции на Linux съдържат двоични UUCP файлове, които имат поддръжка както за HDB, така и за разрешена конфигурация на Taylor, и използват различни поддиректории за всеки набор от конфигурационни файлове. Обикновено, в */usr/lib/uucp* ще има *README* файл.

За да работи правилно UUCP, тези файлове трябва да се притежават от **uucp** потребителя. Някои от тях съдържат пароли и телефонни номера, и следователно, трябва да имат права за достъп 600. Забележка: въпреки че на повечето UUCP команди трябва да се задава потребителски исентификатор за **uucp**, трябва да се увериге, че за програмата *uuchk* това *не* е така. В противен случай, потребителите ще могат да показват системните пароли, въпреки че файловете имат режим 600.

Централният конфигурационен файл на UUCP е */etc/uucp/config* и се използва за задаване на общи параметри. Най-важният от тях (и засега единствен) е UUCP името на вашия хост. Във Виртуалната пивоварна те използват **vstout** като UUCP шлюз:

```
# /etc/uucp/config - основен конфигурационен файл на UUCP
nodename        vstout
```

Следващият важен конфигурационен файл е *sys* файла. Той съдържа цялата специфична за системата информация за сайтовете, с които сте осъществили връзка. Това включва името на сайта и информация

за самата връзка, например, телефонен номер при използване на връзка чрез модем. Типичен запис за сайт, свързан чрез модем и наречен **pablo** ще изглежда по следния начин:

```
# /usr/lib/uucp/sys - име на UUCP съседите
# system:pablo
system    pablo
time      Any
phone     555-2112
port      serial1
speed     38400
chat      ogin: vstout ssword: lorca
```

`time` задава моментите, в които отдалечената система може да бъде извикана. `chat` описва chat-скриптовете за влизане – последователността от низове, които трябва да бъдат разменени, за да се позволи на *uucico* да влезе в **pablo**. По-късно отново ще се върнем към chat-скриптовете.

Ключовата дума *port* просто дава име на запис във файла *port*. (Вижте Фигура 16.1.) Можете да зададете каквото и да е име, стига да се отнася за валиден запис във файла.

Файлът *port* съдържа специфична информация за самата връзка. При връзки чрез модем тази информация описва специалния файл за устройството, което трябва да се използва, обхвата на поддържаните скорости и типа на устройството за набиране, свързано към порта. Следващият запис описва */dev/ttyS1* (a.k.a. COM 2), към който администраторът е свързал модем NakWell, който може да работи при скорости до 38400 bps. Името на порта е избрано така, че да съвпада с името на порта в sys файла:

```
# /etc/uucp/port - UUCP портове
# /dev/ttyS1 (COM2)
port        serial1
type        modem
device      /dev/ttyS1
speed       38400
dialer      nakwell
```

Информацията, която се отнася за устройствата за набиране се съхранява все още в друг файл, наречен – познахте – *dial*. За всеки тип устройство за набиране той съдържа основно последователност от команди, които се използват за установяване на връзка през телефонна линия с отдалечен сайт с даден телефонен номер. Отново, това се задава чрез chat-скрипт. Например, записът за NakWell може да изглежда по следния начин:

```
# /etc/uucp/dial - информация за устройството за набиране
# NakWell модеми
dialer      nakwell
chat        "" "AT&F OK ATDT\T CONNECT
```

Редът, който започва с `chat` задава chat-скрипт за модема, който е последователност от команди, изпратени към и получени от модема, който го инициализира и му указва да набере желания номер. *uucico* ще замести последователността `\T` с телефонния номер.

За да получите бегла представа за това как *uucico* работи с тези конфигурационни файлове, нека да допуснем, че задавате следната команда:

```
$ uucico -s pablo
```

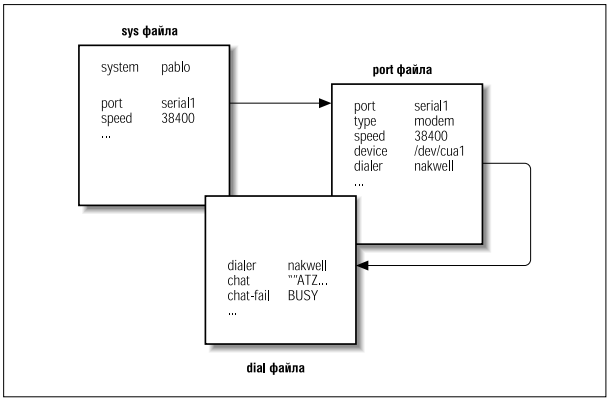
Първото нещо, което *uucico* прави, е да търси **pablo** в *sys* файла. От записа за **pablo** в *sys* файла се вижда, че за да създадете връзката, трябва да използва порта `serial1`. Файлът *port* указва на *uucico*, че това е порт на модем, и че към него е свързан NakWell модем.

Сега *uucico* търси *dial* за записа, който описва NakWell модема и когато го намери, отваря серийния порт */dev/cua1* и изпълнява chat-скрипт за набиране на номер. С други думи, той изпраща *AT&F*, изчаква за отговор *OK* и т.н. Когато попадне на низа `\T`, *uucico* го замества с телефонния номер (555-22112), извлечен от *sys* файла.

След като модемът върне *CONNECT*, връзката вече е създадена, а chat-скриптът за модема е завършен. Сега *uucico* се връща към *sys* файла и изпълнява chat-скрипт за влизане. В нашия пример, *uucico* изчаква за покана за влизане *login:*, след това изпраща потребителското си име (**vstout**), изчаква за показа за въвеждане на парола *password:* и изпраща паролата си (*lorca*).

След като упълномощаването приключи, се предполага, че отдалеченият край стартира собствена програма *uucico*. След това двете страни влизат във фазата на договаряне, която описахме в предишния раздел.

Фигура 16.1 илюстрира зависимостите между конфигурационните файлове.



Фигура 16.1: Взаимодействие между конфигурационните файлове на Taylor UUCP

Какво трябва да знае UUCP

Преди за започнете да пишете конфигурационни файлове на UUCP, трябва да съберете информация, която UUCP изисква.

Първо трябва да определите към какво серийно устройство е свързан вашия модем. Обикновено, (DOS) портовете от COM1: до COM4: насочват към специалните файлове на устройството */dev/ttyS0* през */dev/ttyS3*. Някои дистрибуции, като Slackware, създават връзка */dev/modem* към подходящия *ttyS** файл и конфигурират *kermit*, *seyon* и всякакви други програми за комуникация, за да използват този универсален файл. В този случай, във вашата UUCP конфигурация трябва да използвате и */dev/modem*.

Символна връзка се използва, тъй като всички програми за набиране използват т.нар. *дневници*, за да сигнализират, когато серийният порт се използва. Имената на тези дневници представляват конкатенация от низа *LCK..* и името на файла за устройството, например, *LCK..ttyS1*. Ако програмите използват различни имена за едно и също устройство, те няма да успеят да разпознаят една на друга дневниците си. Като следствие от това, при стартиране по едно и също време, те ще разрушат една на друга сесиите си. Това е доста вероятно, когато планирате вашите UUCP повиквания, използвайки запис *crontab*. За

подробности относно настройването на сериен порт, можете за последните Глава 4, *Конфигуриране на сериен хардуер*.

След това трябва да откриете при каква скорост ще комуникират вашият модем и Linux. Трябва да зададете тази скорост до максимално ефективната скорост за прехвърляне, която очаквате да постигнете. Ефективната скорост за прехвърляне може да бъде много по-висока от физическата скорост за прехвърляне на необработени данни, която вашият модем може да постигне. Например, много модеми изпращат и получават данни при 56 kbps. Като използвате протоколи за компресия, като V.42bis, действителната скорост за прехвърляне може да надвиши 100 kbps.

Разбира се, ако UUCP изобщо трябва да прави нещо, ще ви е необходим телефонния номер на системата, която ще набирате. Освен това, трябва да разполагате и с валиден идентификатор за влизане, а вероятно и с парола за отдалечената машина.[33]

Освен това, трябва да знаете как *точно* да влезете в системата. Трябва ли да натиснете клавиша Enter преди да се появи поканата за влизане в системата? Показва ли се `login`: или `user`:? Това е необходимо за съставянето на *chat-скрипт*. Ако не знаете или ако изпълнението на обикновения chat-скрипт е неуспешно, опитайте се да извикате системата с терминална програма като *kermit* или *minicom* и запишете какво точно трябва да направите.

Именуване на сайтове

Както TCP/IP-базирани мрежи, вашият хост трябва да има име за UUCP мрежи. Ако просто искате да използвате UUCP за прехвърляне на файлове към или от сайтове, които набирате директно, или в локални мрежи, това име не трябва да отговаря на никакви стандарти.[34]

---

33 Ако ще опитате UUCP, вземете номера на сайт за архиви близо до вас. Запишете потребителското име за влизане и паролата, които са публични, за да направят анонимното сваляне възможно. В повечето случаи, те са нещо подобно на **uucp/uucp** и **nuucp/uucp**.

34 Единственото ограничение е името да не бъде по-дълго от седем символа, така че да не обърква UUCP реализациите, които работят с операционни системи, които налагат тесни граници за файловите имена. Имена, които са с повече от седем символа често биват отрязани от UUCP. Някои версии даже ограничават името до шест символа.

Ако използвате, обаче, UUCP за *mail* или *news* връзки, ще трябва да помислите за имена, регистрирани с UUCP Mapping Project.*

UUCP Mapping Project регистрира всички имена на хостове по света и проверява дали са уникални.

UUCP Mapping Project е описан в глава 17, "*Електронна поща*".даже ако участвате в домен, може да помислите за получаването на официално UUCP име за вашия сайт.

Често хората избират тяхно UUCP име, което да съответства на първата компонента на техното напълно квалифицирано име на домен. Да предположим,че домен-адресът на сайта ви е **swim.twobirds.com.** Тогава UUCP името на хоста ви ще е **swim**. Мислете за UUCP сайтовете като за такива, които се разпознават едни други чрез първото си име. Разбира се, може да използвате и UUCP име, което няма никаква връзка с напълно квалифицираното ви име на домен.

Убедете се, обаче, че не използвате неквалифицираното име на сайт в mail адреси, освен ако не сте го регистрирали като официално UUCP име. В най-добрия случай, съобщение, изпратено към нерегистриран UUCP хост, ще изчезне в някое голямо черно запомнящо устройство за двоична информация. Ако използвате име, което вече принадлежи на друг сайт, това съобщение ще бъде изпратено към този сайт и ще причини големи главоболия на неговия postmaster.

По подразбиране, UUCP пакетът използва името, определено *от име на хост* като UUCP име на сайт. Това име обикновено се установява от команда по време на зареждане на *rc* скриптове и обикновено се съхранява в */etc/hostname*. Ако UUCP името ви е различно от това, за което сте настроили хост-името си, то трябва да използвате опцията за име на хост в *config* файла, за да съобщите на *uucico* за вашето UUCP име. Това е описано по-нататък.

Конфигурационни файлове на Taylor

Нека сега се върнем към конфигурационните файлове. Taylor UUCP получава информацията си от следните файлове:

config

Това е основния конфигурационен файл. Тук можете да дефинирате UUCP име за вашия сайт.

sys Този файл описва всички известни сайтове. За всеки сайт той указва име, в кои моменти ще се извика, кой номер да се набере (ако има такъв), какъв тип устройство да се използва и начина за влизане в този сайт.

port

Този файл съдържа записи, които описват всеки достъпен порт, заедно с поддържаната скорост на линията и устройството за набиране, което ще се използва.

dial

Този файл описва устройствата за набиране, използвани за създаване на телефонна връзка.

dialcode

Този файл съдържа разширения за символични dial кодове.

call

Този файл съдържа потребителското име за влизане и паролата, които ще се използват, когато извикваме система.

passwd

Този файл съдържа потребителски имена за влизане и пароли, които системите могат да използват при влизане. Използва се само когато *uucico* прави собствена проверка на паролата.

Конфигурационните файлове на Taylor обикновено се състоят от редове, съдържащи двойки ключова дума – стойност. Символът # въвежда коментар, който продължава до края на реда. За да използвате такъв символ, който да има значение, извършете за него escape като използвате обратно наклонена черта. Например: \#.

Съществуват голям брой опции, които можете да настроите с тези конфигурационни файлове. Тук не можем да разгледаме всички параметри, но ще се спрем на най-важните от тях. След това трябва да можете да конфигурирате UUCP връзка, базирана на модем. Допълнителни раздели описват необходимите промени, ако искате да използвате UUCP през TCP/IP или през директна серийна линия. Пълна справка е дадена в документите на Texinfo, които придружават изходните кодове на Taylor UUCP.

Когата мислите, че сте конфигурирали напълно вашата UUCP система, можете да проверите конфигурацията си като използвате инструмента *uuchk* (намира се в */usr/lib/uucp*). *uuchk* чете конфигурационните ви файлове и отпечатва подробен отчет за конфигурационните стойности, използвани за всяка система.

Основни конфигурационни опции, използващи файла config

Обикновено няма да използвате този файл, за да описвате нещо повече от вашето UUCP име на хост. По подразбиране, UUCP ще използва името, което сте задали с командата *hostname*, но обикновено е добра идея да зададете явно UUCP името. Следва примерен *config* файл:

```
# /usr/lib/uucp/config – основен конфигурационен файл на UUCP
hostname        vstout
```

Освен това, тук могат да се зададът и голям брой различни параметри, например, име на директорията на буфера или правата за достъп за анонимен UUCP. Последният ще бъде описан по-късно в тази глава в раздела "Анонимен UUCP".

Как да съобщим на UUCP за други системи, използващи sys файла

sys файлът описва системите, които са известни на вашата машина. Запис се въвежда чрез ключовата дума `system`; редовете след нея до следващата директива `system` изброяват подробно специфичните за този сайт параметри. Обикновено, един системен запис дефинира параметри като телефонен номер и chat-скрипт за влизане.

Параметрите преди първия `system` ред задават подразбиращите се стойности, използвани за всички системи. Обикновено, задавате протоколни параметри и подобните им в подразбиращите се секции.

В следващите раздели са разгледани отдробно най-важните полета.

Имена система

Командата *system* задава име на отдалечената система. Трябва да зададете правилното име на отдалечената система, а не псевдоним,

който сте измислили, тъй като *uucico* ще го свери с това, което съобщава отдалечената система, когато е извикана при влизане.[35]

Всяко име на система може да се появява само веднъж. Ако искате да използвате няколко набора от конфигурации за една и съща система (като различните телефонни номера, които *uucico* трябва да изпробва подред), можете да зададете *алтернативи*, които ще опишем след основните конфигурационни опции.

Телефонен номер

Ако до отдалечената система трябва да се достигне през телефонна линия, полето `phone` задава номера, който модемът трябва да набере. То може да съдържа няколко лексеми, интерпретирани от процедурата за набиране на *uucico*. Знакът равно (=) означава изчакване за втори тон за набиране, а тирето (-) генерира пауза от една секунда. Някои телефонни инсталации се задъхват, ако не правите пауза между набирането на специален код за достъп и телефонния номер.[36]

Често пъти е удобно да използвате имена вместо числа, за да опишите кодовете за набиране на дадена област. Файлът *dialcode* ви позволява да свържете име с код, който ще използвате в последствие при задаването на телефонни номера за отдалечени хостове. Да предположим, че имате следния *dialcode* файл:

```
# /usr/lib/uucp/dialcode - транслация на dialcode
Bogoham      024881
Coxton       035119
```

С тези translations в *sys* файла можете да използвате телефонен номер като `Bogoham7732`, което вероятно ще направи нещата малко почетливи и като цяло по-лесни за обновяване, ако кодът за набиране за Bogoham някога се промени.

port и speed

Опциите `port` и `speed` се използват за избор на устройство, използвано за извикване на отдалечената система и на максималната ско-

---

[35] По-стари Версия 2 UUCP не изпращат имената си при извикване; по-нови реализации обаче, го правят често. Същото се отнася и за Taylor UUCP.

[36] Например, повечето частни инсталации на компании изискват от вас да наберете 0 или 9, за да получите външна линия.

рост, за която трябва да се настрои устройството.[37] Запис `system` може да използва или самостоятелна опция, или две опции в конюнкция. Притърсене на подходящо устройство във файла *port*, се избират само портовете, които имат съвпадащи име на порта и/или обхват на скоростите.

Обикновено, използването на опцията `speed`е достатъчно. Ако имате само едно серийно устройство, дефинирано в *port*, *uucico* винаги избира правилното, така че трябва само да зададете желаната скорост. Ако имате няколко модема, прикрепени към системата ви, често няма да искате да именувате определен порт, тъй като ако *uucico* открие, че съществуват няколко съвпадения, тя изпробва подред всяко устройство, докато не открие неизползвано такова.

chat-скрипт за влизане

Вече се сблъскахме с chat-скрипт за влизане, който указва на *uucico* как да влиза в отдалечени системи. Той се състои от списък с лексеми, които задават очаквани и изпратени от локалния процес на *uucico* низове. *uucico* изчаква докато отдалечената машина не изпрати покана за влизане. След това връща потребителското име за влизане, изчаква отдалечената система да изпрати покана за парола и изпраща паролата. В скрипта се редуват низове, които се очакват и такива, които ще се изпращат. *uucico* автоматично добавя символ за връщане на каретката (`\r`) към всеки низ за изпращане. Така, един прост chat-скрипт би изглеждал по следния начин:

```
ogin: vstout ssword: catch22
```

Вероятно ще забележите, че очакваните полета не съдържат целите покани. Това гарантира, че влизането в системата е успешно, дори ако отдалечената система предава *Login*: вместо *login*:. Ако низът, който очаквате или изпращате, съдържа интервали или други символи за празни пространства, трябва да заградите текста в кавички.

uucico позволява някои условни изпълнения. Да приемем, че *getty* на отдалечената машина трябва да се установи в изходно положение преди изпращането на покана. За целта, можете да добавите chat-подскрипт към очаквания низ. Той ще се изпълнява, само ако основното очакване е неуспешно, т.е. ако възникне таймаут. Един начин за

[37] Скоростта за предаване на битове от *tty* трябва да бъде поне равна на максималната скорост за прехвърляне.

използване на тази възможност е като се изпрати BREAK, ако отдалечения сайт не показва покана за влизане. Следващият пример дава chat-скрипт с общо предназначение, който трябва да работи и ако сте натиснали Enter преди да се появи поканата за влизане. Първият празен аргумент, "", указва на UUCP да не чака нищо, а да продължи със следващия изпратен низ:

```
"" \n\r\d\r\n\c ogin:-BREAK-ogin: vstout ssword: catch22
```

В chat-скрипта могат да възникнат няколко специални низа и escape-символи. Следва част от списък със символи, валидни в очаквани низове:

"" Празен низ, който указва на *uucico* да не чака нищо, а да продължи веднага със следващия изпратен низ.

`\t` Символ за табулация.

`\r` Символ за връщане на каретката.

`\s` Символ за интервал. Този символ вие необходим, за да вмъквате интервали в chat-низове.

`\n` Символ за нов ред.

`\\` Символ за обратно наклонена черта.

При изпратени низове, следващите escape-символи и низове са валидни като допълнение към изброените догук:

`EOT`
: Символ за край на предаването (*^D*).

`BREAK`
: Символ за прекъсване.

`\c` Потиска изпращането на символ за връщане на каретката в края на низ.

`\d` Отлага изпращането с 1 секунда.

`\E` Разрешава echo проверка. Това изисква от *uucico* да чака устройството да прочете отново всичко, което записва, преди да може да продължи с chat-скрипта. Това е особено, когато се използва в chat-скриптове за модеми (на това ще се спрем по-късно). Echo проверката е изключена по подразбиране.

`\e` Дезактивира проверка с ехо.

`\K` Същата като BREAK.

`\p` Пауза за част от секундата.

Алтернативи

Понякога искате да имате множество входове за една единствена система, например, ако системата може да бъде достигната през различни модемни линии. С Taylor UUCP можете да направите това като дефинирате т.нар. *алтернатива*.

Един алтернативен запис запазва всички настройки от основния системен запис и задава само онези стойности, които трябва да бъдат отменени и/или добавени към подразбиращият се системен вход. Алтернатива се отмества системния запис посредством ред, съдържащ ключовата дума `alternate`.

За да използвате два телефонни номера за **pablo**, трябва да промените *sys* му запис по следния начин:

```
system      pablo
phone       123-456
… entries as above ...
alternate
phone       123-455
```

Когато извиквате **pablo**, *uucico* първо ще набере 123-456. Ако това не успее, тя ще опита алтернативата. Алтернативния запис запазва вички настройки от основния системен запис и отменя само телефонния номер.

Ограничаване на времето за обаждане

Taylor UUCP предоставя голям брой начини, чрез които можете да ограничите времето за извикване на отдалечена система. Бихте могли да направите това или поради ограничения, които отдалечения хост налага върху услугите си по време на натоварени часове, или просто, за да избегните моменти, в които тарифите за обаждане са високи. Забележете, че винаги е възможно да отмените ограниченията на времето за обаждане, като подадете на *uucico* опцията *–s* или опцията *-f*.

По подразбиране, Taylor UUCP забранява връзки по всяко време, така че *трябва* да използвате някакъв вид спецификация на времето в *sys* файла. Ако не се интересувате от ограниченията на времето за

обаждане, можете да зададете в *sys* файла опцията *time* със стойност `Any`.

Най-простия начин, по който можете да ограничите времето за обаждане, е като включите `time` запис, следван от низ, който се състои от подполе за ден и час. Денят може да бъде комбинация от `Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa` и `Su`. Освен това, за дните от седмицата можете да зададете `Any`, `Never` или `Wk`. Времето се състои от две 24-часови стойности, разделени от тире. Те определят времето, през което може да има обаждания. Комбинацията от тези лексеми се записва без празно пространство между тях. Всички спецификации за номер на ден и време могат да бъдат групирани със запетаи, както показва следващия ред:

```
time            MoWe0300-0730,Fr1805-2200
```

Този пример позволява обаждания в понеделник и сряда от 3:00 до 7:00 сутринта и в петък, между 18:05 и 22:00 часа. Когато полето за време обхваща полунощ, например, `Mo1830-0600`, всъщност означава понеделник между полунощ и 6:00 сутринта, и между 18:30 следобед и полунощ.

Специалните низове за време `Any` и `Never` означават това, че обаждания може да има съответно винаги или никога.

Освен това, Taylor UUCP има голям брой специални лексеми, които можете да използвате, като `NonPeak` и `Night`. Тези специални символи са съкращения съответно за `Any2300-0800`, `SaSu0800-1700` и `Any1800-0700,SaSu`.

Командата *time* приема втори незадължителен аргумент, който описва време за повторен опит в минути. Когато опит за създаване на връзка е неуспешен, *uucico* няма да позволи друг опит за набиране на отдалечен хост в рамките на определен интервал. Например, когато задавате време за повторен опит от 5 минути, *uucico* ще откаже да извика отдалечената система в рамките на 5 минути след последния неуспех. По подразбиране, *uucico* използва експоненциална *backoff* схема, където интервалът за повторен опит се увеличава с всеки нов неуспех.

Командата *timegrade* ви позволява да добавите максимална степен на буфера в таблица. Например, да предположим, че имате следните *timegrade* команди в `system` запис:

```
timegrade       N Wk1900-0700,SaSu
timegrade       C Any
```

Това ви позволява задачи със степен C или по-висока (обикновено пощата се подрежда в опашка със степен B или C) да бъдат предадени при всяко извикване, докато новини (обикновено се подреждат в опашка със степен N) се предават само нощем и в края на седмицата.

Точно както *time*, командата *timegrade* приема като незадължителен трети аргумент интервал за повторен опит в минути.

Тук обаче има едно възражение относно степените на буфера. Първо, опцията *timegrade* се прилага само за това, което *вашите* системи изпращат; отдалечената система все още може да предава всичко, което пожелае. Можете да използвате опцията *call-timegrade*, за да дадете явна заявка да изпраща само задачи със степен над някаква зададена степен на буфера; няма гаранция обаче, че тя ще се подчини на заявката.[38]

По подобен начин, полето `timegrade` не се проверява при извикване на отдалечена система, така че всяка задача в опашката за извикващата системата, ще бъде изпратена. Отдалечената система обаче може изрично да даде заявка *uucico* да се ограничи до определена степен на буфера.

Идентифициране на достъпни устройства през port файла

Файлът *port* съобщава на *uucico* за достъпни портове. Това обикновено са портове за модеми, но се поддържат и други типове, като директни серийни линии и TCP sockets.

Като *sys* файла, *port* се състои от отделни записи, започващи с ключовата дума `port`, следвана от името на порта. Това име може да бъде използвано в `port` конструкция на *sys* файла. Името не трябва да бъде уникално; ако има няколко порта с едно и също име, *uucico* ще изпробва подред всеки порт, докато не открие някой, който не се използва в момента.

Непосредствено след командата *port* трябва да има `type` конструкция, която указва типа на описания порт. Валидни типове са: `modem`, `direct` за директни връзки и `tcp` за TCP sockets. Ако командата *port* липсва, типът на порта по подразбиране е modem.

[38] Ако отдалечената система поддържа Taylor UUCP, тя ще се подчини на заявката.

В този раздел разглеждаме само портове за модеми. TCP портове и директни линии са разгледани един от следващите раздели.

За директни портове и портове за модеми трябва да укажете устройство за извикване чрез директивата `device`. Обикновено, това е името на специален файл за устройство в директорията */dev*, като */dev/ttyS1*.

Ако устройството е модем, записът за порта определя и типа на модема, който е свързан към порта. Различните типове модеми трябва да бъдат конфигурирани по различен начин. Дори модеми, които са Hayes-съвместими в действителност не винаги са съвместими помежду си. Следователно, трябва да укажете на *uucico* как да инициализира модема и да му укажете да набере желания номер. Taylor UUCP съхранява описанията на всички устройства за набиране във файл, наречен *dial*. За да използвате някое от тях, трябва да укажете името на устройството за набиране, използвайки командата *dialer*.

Понякога ще искате да използвате модем по различни начини, в зависимост от системата, която извиквате. Например, някои по-стари модеми не разбират, когато високоскоростен модем се опитва да се свърже при скорост 56 kbps; те просто прекъсват линията вместо да договорят връзка, например, при скорост 9 600 bps. Ако знаете, че сайта **drop** използва такъв модем, трябва да настроите вашия модем по различен начин. За тази цел ви е необходим допълнителен запис за порт във файла *port*, който задава различно устройство за набиране. Сега може да дадете на новия порт друго име, като `serial1-slow`, и да използвате директивата `port` в *sys* файла на записа за системата.

Един по-добър начин за различаване на портовете е сравняването на скоростите, които те поддържат. Например, двата записа за портове в горната ситуация, могат да изглеждат по следния начин:

```
# NakWell модем; свързва се при висока скорост
port            serial1         # име на порта
type            modem           # порт за модема
device          /dev/ttyS1      # това е COM2
speed           115200          # поддържана скорост
dialer          nakwell         # обикновено устройство за набиране
# NakWell модем; свързва се при ниска скорост
port            serial1         # име на порта
type            modem           # порт за модема
device          /dev/ttyS1      # това е COM2
speed           9600            # поддържана скорост
dialer          nakwell-slow    # не прави опит за бърза връзка
```

Сега системният запис за сайта **drop** би трябвало да е `serial1` като име на порт, но дайте заявка да го използва само при скорост 9 600 bps. След това *uucico* автоматично използва втория запис за порт. Всички останали сайтове, които поддържат скорост от 115 200 bps в системния запис ще бъдат извикани чрез използването на първия запис за порт. По подразбиране, той ще бъде използван със съответна скорост.

Как да набираме номер като използваме файла dial

Файлът *dial* описва начинът, по който се използват различните устройства за набиране. Обикновено, UCCP говори за устройства за набиране, отколкото за модеми, тъй като преди време беше обичайна практика да имаш едно (скъпо) автоматично устройство за набиране, което обслужва цяла група модеми. Днес, повечето модеми имат вградена поддръжка за набиране, така че това свойство постепенно остарява.

Въпреки това, различни устройства за набиране или модеми могат да изискват различна конфигурация. Във файла *dial* можете да опишете всяка от тях. Записите в *dial* започват с командата *dialer*, която дава името на устройството за набиране.

Най-важният запис, освен *dialer*, е chat-скрипта за модема, зададен от командата *chat*. Подобно на chat-скрипта за влизане, той се състои от последователност от низове, които *uucico* изпраща към устройството за набиране, и отговори, които очаква да бъдат върнати. Често установяваме модема в някакво познато състояние и набираме номера. Следващият примерен запис в `dialer` показва типичен chat-скрипт за модем за Hayes-съвместими модеми:

```
# NakWell модем; свързва се при висока скорост
dialer      nakwell     # име на устройството за набиране
chat        "" AT&F OK\r ATH1E0Q0 OK\r ATDT\T CONNECT
chat-fail   BUSY
chat-fail   ERROR
chat-fail   NO\sCARRIER
dtr-toggle  true
```

chat-скрипта за модема започва с “ ”, празен очакван низ. Следователно, *uucico* изпраща веднага първата команда *AT&F*. *AT&F* е Hayes команда за установяване на модема във подразбиращата се конфигурация. След това *uucico* изчаква, докато модемът не изпрати `OK` и из-

праща следващата команда, която изключва локално ехо и други подобни. След като модемът върне отново `OK`, *uucico* изпраща командата за набиране *ATDT*. Escape-последователността `\T` в този низ е заместена с телефония номер, взет от *sys* файла на записа за системата. След това *uucico* чака модемът да върне низа `CONNECT`, който сигнализира, че е връзката с отдалечения модем е създадена успешно.

Понякога модемът не може да се свърже с отдалечената система; например, ако другата система комуникира с някой друг и линията е заета). В този случай, модемът връща съобщение за грешка, указващо причината. Chat-скриптовете за модемите не могат да откриват такива съобщения; *uucico* продължава да чака очаквания низ, докато не изтече таймаута. Следователно, дневникът на UUCP показва само вежливото "*timed out in chat script*" вместо самата причина.

Все пак, Taylor UUCP ви позволява да съобщите на *uucico* за тези съобщения за грешки, като използвате командата *chat-fail*, както е показано по-горе. Ако *uucico* открие *chat-fail* низ докато изпълнява chat-скрипт за модема, това анулира обаждането и записва съобщението за грешка в дневника на UUCP.

Последната команда в показания по-горе пример указва на UUCP да превключи контролната линия DTR (Data Terminal Ready) преди да стартира chat-скрипта за модема. Обикновено, серийният драйвер активира DTR, когато даден процес отваря устройството, за да укаже на прикачения модем, че някой иска да говори с него. След това функцията `dtr-toggle` прекъсва DTR, изчаква за момент и отново го активира. Много модеми могат да бъдат конфигурирани така, че да реагират на прекъсване на DTR като се изключват, влизат в команден режим или се установяват в изходно състояние.[39]

UUCP през TCP

Може да звучи абсурдно, но използването на UUCP за предаване на данни през TCP не е толкова лоша идея, особено при предаване на голям обем от данни като Usenet новини. При TCP-базирани връзки, новините обикновено се обменят посредством протокола NNTP, чрез който статии се заявяват и изпращат индивидуално без компресия или някаква друга оптимизация. Въпреки че е напълно достатъчно за големи сайтове с няколко съвместно действащи генератори на новини, тази техника е много неблагоприятна за малки сайтове, които

[39] Някои модеми изглежда не харесват това и от време на време увисват.

получават новините си през относително бавна връзка като ISDN. Тези сайтове обикновено ще искат да комбинират качествата на TCP с предимствата при изпращането на новини в големи пакети, които могат да бъдат компресирани и по този начин предадени с много малко служебна информация. Един общ начин за предаване на тези пакети е използването на UUCP през TCP.

В *sys* файла трябва да укажете система, която ще бъде извикана през TCP, по следния начин:

```
system      gmu
address     news.groucho.edu
time        Any
port        tcp-conn
chat        ogin: vstout word: clouseau
```

Командата *address* дава IP адреса на хоста или неговото пълно квалифицирано име на домейн. Съответният *port* запис ще прочете:

```
port        tcp-conn
type        tcp
service     540
```

Входът констатира, че трябва да бъде използвана TCP връзка, когато sys запис се обръща `tcp-conn` и по този начин *uucico* трябва да се опита да се свърже към TCP мрежов порт 540 на отдалечения хост. Това е подразбиращият се номер на порта на UUCP услугата. Вместо номера на порта към командата *service* можете да подадете и символно име на порт. Номерът на порта, който съответства на това име, ще се търси в */etc/services*. Общото име за UUCP услуга е `uucpd`.

Използване на директна връзка

Да предположим, че използвате директна линия за свързване на вашата система **vstout** към **tiny**. Това е подобно на случая с модема, трябва да напишете системен запис в *sys* файла. Командата *port* идентифицира, че серийният порт `tiny` е свързан към:

```
system    tiny
time      Any
port      direct1
speed     38400
chat      ogin: cathcart word: catch22
```

Във файла *port* трябва да опишете серийния порт за директната връзка. Не е необходим `dialer` запис, тъй като не е необходимо набиране:

```
port      direct1
type      direct
speed     38400
device        /dev/ttyS1
```

Контролиране на достъпа до възможностите на UUCP

UUCP е доста гъвкава система. С тази гъвкавост се появява и необходимостта от внимателен контрол на достъпа до нейните възможности, за да се предотврати неправилното използване, независимо от това дали е умишлено или случайно. От първостепенно значение за UUCP администратора са изпълнение на отдалечена команда, прехвърляне на файлове и препращане на данни. Taylor UUCP предоставя средство за ограничаване на свободата, която отдалечени UUCP хостове имат при използване на всяка от тези възможности. Ако се изберат внимателно правата за достъп, UUCP администраторът може да гарантира, че сигурността на хоста е запазена.

Изпълнение на команди

Задачата на UUCP е да копира файлове от една система в друга и да заявява изпълнение на определени команди на отдалечени хостове. Разбира се, като администратор може да искате да контролирате правата, които предоставяте на други системи – позволявайки им да изпълняват всяка команда, която си изберат, във вашата система.

По подразбиране, единствените команди, които Taylor UUCP позволява да бъдат изпълнявани от други системи на вашата система, са *rmail* и *rnews*, които най-често се използват за обмен на електронна поща и Usenet новини през UUCP. За да промените набора от команди за определена система, в `sys` файла можете да използвате ключовата дума `commands`. Аналогично, можете да искате да ограничите пътя за търсене само за онези директории, които съдържат позволените команди. Чрез конструкцията `command-path` можете да промените пътя за търсене, разрешен за отдалечен хост. Например, може

да искате да позволите на системата **pablo** да изпълни командата *bsmtp* като допълнение към *rmail* и *rnews*:[40]

```
system      pablo
...
commands    rmail rnews bsmtp
```

Прехвърляне на файлове

Освен това, Taylor UUCP ви позволява да настройвате фино прехвърлянето на файлове до големи подробности. Като крайна мярка, можете да забраните прехвърлянията към и от определена система. Само установете `request` на no и отдалечената система няма да може както да извлича файлове от вашата система, така и да ѝ изпраща. Аналогично, можете да забраните на вашите потребители да прехвърлят файлове към и от система, като установите `transfer` на no. По подразбиране, на потребителите както в локална, така и в отдалечена система е разрешено да качват и свалят файлове.

Като допълнение, можете да конфигурирате директориите, в и от които могат да се копират файлове. Обикновено, ще искате да ограничите достъпа на отдалечени системи до една единствена йерархия от директории, но все пак да позволите на вашите потребителите да изпращат файлове от тяхната home директория. Най-често, на отдалечените потребители е разрешено да получават файлове само от публичната UUCP директория */var/spool/uucppublic*. Това е традиционното място, където се създават публично достъпни файлове много, подобно на FTP сървъри в Internet.[41]

Taylor UUCP предоставя четири различни команди за конфигуриране на директории за изпращане и получаване на файлове. Те са: *local-send*, която задава списък с директории, от който потребител може да поиска от UUCP да изпраща файлове; *local-receive*, която дава списък с директории, в които потребител може да поиска да получава файлове; *remote-send* и *remote-receive*, които правят аналогичното за заявки от чужди системи. Разгледайте следния пример:

```
system          pablo
...
local-send      /home ~
```

[40] *bsmtp* се използва за доставяне на поща с пакетиран SMTP.

[41] Можете да използвате символа тилда (~), за да се обърнете към публичната директория на UUCP, но само в конфигурационни файлове на UUCP; извън тях обикновено той се отнася за home директорията на потребителя.

```
local-receive   /home ~/receive
remote-send     ~ !~/incoming !~/receive
remote-receive  ~/incoming
```

Командата *local-send* позволява на потребители на вашия хост да изпращат всякакви файлове под */home* и от публичната директория на UUCP към **pablo**. Командата *local-receive* им позволява да получават файлове както в *receive* директорията в *uucppublic*, в която всеки може да записва, така и във всяка такава директория под */home*. Директивата *remote-send* позволява на **pablo** да заяви файлове от */var/spool/uucppublic*, с изключение на файлове от директориите *incoming* и *receive*. За това се сигнализира на *uucico* като пред имената на директориите се поставя удивителен знак. Накрая, последният ред позволява на **pablo** да качи файлове в **incoming**.

Основен проблем при прехвърлянето на файлове, използвайки UUCP, е че той получава файлове само в директории, в които всеки може да записва. Това може да изкуши някои потребители да поставят капани за други. Няма начин обаче да избегнете този проблем, освен ако не забраните като цяло прехвърлянето на файлове посредством UUCP.

Препращане

UUCP предоставя механизъм, чрез който други системи изпълняват прехвърляне на файлове от ваше име. Например, предположете, че вашата система има *uucp* достъп до система, наречена **seci**, но не и до друга система, наречена **uchile**. Това ви позволява да укажете на **seci** да извлече вместо вас файл от **uchile** и да го изпрати към системата ви. Това може да се направи чрез следващата команда:

```
$ uucp -r seci!uchile!~/find-ls.gz ~/uchile.files.gz
```

Тази техника на предаване на задача през няколко системи се нарича *препращане*. Във вашата собствена UUCP система може да искате да ограничите услугата за препращане до няколко хоста, за които знаете, че не натрупват потресаващи телефонни сметки като ви карат да сваляте изходния код на най-новата реализация на X11R6 за тях.

По подразбиране, Taylor UUCP забранява напълно препращането. За да го разрешите за определена система, можете да използвате командата *forward*. Тази команда задава списък със сайтове, към и от които системата може да поиска от вас да препратите задачата. Например, UUCP администраторът на **seci** ще трябва да добави следващите ре-

дове към *sys* файла, за да позволи на **pablo** да заяви файлове от **uchile**:

```
##################
# pablo
system          pablo
...
forward         uchile
##################
# uchile
system          uchile
...
forward-to      pablo
```

`forward-to` записът за **uchile** е необходим, така че всякакви файлове, върнати от нея, в действителност са предадени към **pablo**. В противен случай, UUCP ще ги пренебрегне. Този запис използва вариант на командата *forward*, която разрешава на **uchile** да изпраща файлове само към **pablo** през **seci**, но не и обратното.

За да разрешите услугата препращане към която и да е система, използвайте специалната ключова дума `ANY` (главните букви са задължителни).

Настройване на вашата система за набиране

Ако искате да настроите вашия сайт за набиране, трябва да разрешите влизане в серийния ви порт и да настроите някои системни файлове за предоставяне на UUCP акаунти, които ще разгледаме в този раздел.

Предоставяне на UUCP акаунти

Като начало, трябва да настроите потребителските акаунти, които дават възможност на отдалечените сайтове да влизат в системата ви и да стъдадете UUCP връзка. Обикновено, ще предоставяте отделно потребителско име за влизане за всяка система, която извиква. Когато настройвате акаунт за система **pablo**, можете да й дадете потребителското име **Upablo**. Няма наложена политика за потребителските имена за влизане; те могат да бъдат всякакви, но за вас ще е по-удобно, ако лесно се прави връзка между името за влизане и името на отдалечения хост.

За системи, които набират през серийния порт, обикновено трябва да добавяте тези акаунти към системния файл за парола */etc/passwd*. Добра практика е поставянето на всички UUCP имена за влизане в специална група като **uuguest**. Home директорията на акаунта трябва да бъде настроена като публичната директория на буфер */var/spool/uucppublic*; нейната обвивка за влизане трябва да бъде *uucico*.

За да облужите UUCP системи, които се свързват към вашия сайт през TCP, трябва да настроите *inetd* за обработка на постъпващи на *uucp* порта връзки, като добавите следния ред към: */etc/inetd.conf*:[42]

```
uucp stream tcp  nowait  root  /usr/sbin/tcpd  /usr/lib/uucp/uucico -l
```

Опцията *-l* указва на *uucico* да изпълнява собствено упълномощаване при влизане в системата. Тя пита за потребителско име за влизане и парола, точно както стандартната програма *login*, но разчита на личната си база данни с пароли вместо на */etc/passwd*. Този личен файл за пароли се нарича */etc/uucp/passwd* и съдържа двойки от потребителски имена за влизане и пароли:

```
Upablo    IslaNegra
Ulorca    co'rdoba
```

Този файл трябва да е собственост на **uucp** и да има права за досъп 600.

Звучи ли тази база данни като толкова добра идея, която бихте искали да използвате и при обикновени серийни влизания? Е, в някои случаи можете. Това, от което се нуждаете е програма *getty*, на която можете да укажете да извиква *uucico* вместо */bin/login* за вашите UUCP потребители.[43]

Извикването на *uucico* би изглеждало по следния начин:

```
/usr/lib/uucp/uucico -l -u потребител
```

Опцията *-u* по-скоро указва на програмата да използва зададеното потребителско име, отколкото да пита за него.[44]

---

[42] Забележете, че *tcpd* обикновено е в режим 700, така че трябва се извика като потребител с права **root**, а не като *uucp*. *tcpd* е разгледан по-подробно в Глава 12, *Важни мрежови възможности*.

[43] mgetty на Gert Doering е истински звяр. Тя работи с разнообразие от платформи, включително SCO Unix, AIX, SunOS, HP-UX и Linux.

[44] Тази опция липсва във Версия 1.04.

Как да се защитите от мошеници

Основният проблем с UUCP е, че извикващата система може да лъже за името си; тя съобщава името си на викания система, след като влезе в нея, но сървърът няма как да го провери. По този начин, атакуващият може да влезе в неговия или нейния собствен UUCP акаунт като някой друг и да получи пощата на друг сайт. Особено ненадеждно е, ако предложите влизане в системата през анонимен UUCP, където паролата е направена публична.

Трябва да за защитите от такъв вид мошеник. Лекарството за тази болест е да изискате от всяка система да използва отделно потребителско име за влизане, като в `sys` файла зададете `called-login`. Един примерен запис може да изглежда по следния начин:

```
system      pablo
... usual options ...
called-login Upablo
```

Крайният резултат е, че винаги когато система влиза в друга система и се представя като **pablo**, *uucico* проверява дали тя е влязла като **Upablo**. Ако това не е така, викащата система е отхвърлена и връзката се прекъсва. Трябва да ви стане навик да добавяте командата *called-login* към всеки системен запис, който добавяте във вашия *sys* файл. Важно е да правите това за *всички* системи в *sys* файла, независимо дали те някога се обръщат или не към сайта ви. За онези сайтове, които никога не ви се обаждат, вероятно трябва да зададете `called-login` с някакво абсолютно фалшиво потребителско име, например, **neverlogsin**.

Бъдете параноични: проверки на последователност от обаждания

Друг начин да откриете и отстраните мошеници е използването на проверки на последователност от обаждания. Това ви помага да се защитите от досадници, които по някакъв начин успяват да открият паролата, с която влизате във вашата UUCP система.

Когато използвате проверки на последователност от обаждания и двете машини следят броя на създадените до момента връзки. Броячът се увеличава с всяка връзка. След влезе викащата система изпраща поредния си номер на обаждане и получателят го сверява със своя собствен номер. Ако те не съвпадат, опитът за връзка е отхвърлен. Ако началният номер е избран произволно, атакуващите трудно ще отгатнат правилният пореден номер на обаждане.

Но проверките на последователност от обаждания правят и нещо повече за вас. Дори ако някой много умен човек открие поредния номер на вашето обаждане, както и вашата парола, вие ще можете да го откриете. Когато атакуващият се обади на вашия UUCP feed и откраднe пощата ви, това ще увеличи поредния номер на обаждането с единица. Следващият път, когато *вие* се обадите на вашия *feed* и се опитате да влезете в системата, отдалечената *uucico* ще ви откаже, тъй като номерата вече не съвпадат!

Ако сте разрешили проверките на последователност от обаждания, трябва често да проверявате вашите дневници за съобщения за грешки, които загатват за възможни атаки. Ако системата ви отхвърли поредния номер на обаждането, който викащата система предлага, *uucico* ще постави съобщение в дневника, подобно на: "Out of sequence call rejected". Ако системата ви е отхвърлена от нейния *feed* поради това, че поредните номера не са синхронизирани, тя ще постави в дневника съобщение, което гласи: "Handshake failed (RBADSEQ)".

За да разрешите проверките на последователност от обаждания, добавете следващата команда към записа за системата:

```
# разрешаване на проверки на последователност от обаждания
sequence        true
```

Като допълнение, трябва да създадете файл, съдържащ самия номер на последователността. Taylor UUCP съхранява поредния номер във файл *.Sequence* в директорията за буфера на отдалечения сайт. Той трябва да бъде притежаван от **uucp** и да е в режим 600 (т.е. достъпен за четене и запис само от **uucp**). Най-добре е да инициализирате този файл с произволна, предварително договорена начална стойност. Прост начин за създаване на този файл е:

```
# cd /var/spool/uucp/pablo
# echo 94316 > .Sequence
# chmod 600 .Sequence
# chown uucp.uucp .Sequence
```

Разбира се, отдалеченият сайт също трябва да разреши проверки на последователност от обаждания и да стартира, като използва точно същия пореден номер като вас.

Анонимен UUCP

Ако искате да предоставите анонимен UUCP достъп до вашата система, първо трябва да настроите специален акаунт за това, както вече описахме. Обичайна практика е да дадете на анонимния акаунт потребителско име за влизане и парола на **uucp**.

Като допълнение, трябва да зададете някои от опциите за сигурност за непознати системи. Например, може да искате да им забраните да изпълняват всякакви команди в системата ви. Не можете обаче да зададете тези параметри в запис от *sys* файла, тъй като командата *system* изисква името на системата, което вие нямате. Taylor UUCP решава тази дилема чрез командата *unknown*. *unknown* може се използва в *config* файла за задаване на команди, които обикновено могат да се появят в системен запис:

```
unknown   remote-receive ~/incoming
unknown   remote-send ~/pub
unknown   max-remote-debug none
unknown   command-path /usr/lib/uucp/anon-bin
unknown   commands rmail
```

Това ще ограничи непознати системи до сваляне на файлове под директорията *pub* и предаване на файлове към директорията *incoming*, която се намира под */var/spool/uucppublic*. Следващият ред ще укаже на *uucico* да игнорира всяка заявка от отдалечената система, за да включи локално дебъгване. Последните два реда разрешават на непознати системи да изпълняват *rmail*; но зададения път на командата указва на *uucico* да търси командата *rmail* в лична директория, наречена *anon-bin*. Това ограничение ви позволява да предоставяте специална *rmail* команда, която например, препраща цялата поща към супер-потребител за проверка. По този начин анонимни потребители могат да достигат до поддържащия системата, но в същото време им пречи да вмъкват всякаква поща в други сайтове.

За да разрешите анонимен UUCP, трябва да зададете поне една `unknown` конструкция в *config* файла. В противен случай, *uucico* ще отхвърли всички непознати системи.

UUCP протоколи на ниско ниво

За договаряне на контрола върху сесиите и прехвърлянето на файлове с отдалечения край, *uucico* използва набор от стандартизирани съобщения. Това често се разглежда като протокол от високо ниво. По време на фазата на инициализация и фазата на затваряне те просто се

изпращат като низове. По време на фазата на действителното прехвърляне обаче се използва допълнителен протокол от ниско ниво, който в повечето случаи е прозрачен за по-високите нива. Този протокол предлага някои добавени предимства като разрешаване на проверки за грешка в данни, изпратени през ненадеждни връзки.

Преглед на протоколите

UUCP се използва в различни типове връзки като серийни линии, TCP, а понякога дори и X.25; полезно е да прехвърляте UUCP в рамките на протоколи, проектирани специално за основния мрежови протокол. Като допълнение, няколко реализации на UUCP са въвели различни протоколи, които грубо казано правят същото.

Протоколите могат да бъдат разделени на две категории: *потокови* и *пакетни*. Потоковите протоколи прехвърлят целия файл като може да изчислят и контролната сума. Това се извършва почти без служебна информация, но изисква надеждна връзка, тъй като всяка грешка ще доведе до препредаване на целия файл. Обикновено, тези протоколи се използват при TCP връзки, но не са подходящи при телефонни линии. Въпреки че съвременните модеми вършат доста добра работа при корекция на грешки, те не са нито идеални, нито съществува някакъв начин за откриване на грешка между вашия компютър и модема.

От друга страна, пакетно-ориентираните протоколи разделят файла на няколко части с еднакъв размер. Всеки пакет се изпраща и получава поотделно, изчислява се контролната сума, а към изпращача се връща потвърждение. За да бъде това по-ефективно са измислени sliding-window протоколи, които позволяват ограничен брой (прозорец) чакащи потвърждения по всяко време. Това доста намалява времето, през което *uucico* трябва да чака при предаване. И все пак, относително голямата служебна информация в сравнение с потоковите протоколи прави пакетните протоколи неефективни за използване при TCP връзки, но идеални за телефонни линии.

Ширината на пътя на данните също има значение. Понякога изпращането на 8-битови символи през серийна връзка е невъзможно; например, връзката може да минава през "глупав" терминален сървър, който орязва осмия бит. Когато предавате 8-битови символи през 7-битова връзка, при предаването те трябва да бъдат заградени в кавички. В най-лошия случай, кавичките удвояват обема на данните, които ще се предават, въпреки че компресията, която хардуерът пра-

ви може да компенсира това. Линиите, които могат да предават произволни 8-битови символи, обикновено се наричат *8-bit clean*. Такъв е случая както за всички TCP връзки, така и за повечето модемни връзки.

Taylor UUCP 1.06 поддържа голямо разнообразие от UUCP протоколи. Най-често използваните от тях са:

g Това е най-често използвания протокол и трябва да се разпознава виртуално от всички *uucico*. Той прави цялостна проверка за грешки и следователно е доста подходящ за шумни телефонни връзки. g изисква 8-bit clean връзка. Той е пакетно-ориентиран протокол, който използва техниката sliding-window.

i Това е двупосочен пакетен протокол, който може да изпраща и получава файлове по едно и също време. Изисква връзка пълна дуплексна връзка и 8-bit clean път за данни. В момента се разпознава само от Taylor UUCP.

t Този протокол е проектиран за използване през TCP връзка или други наистина свободни от грешки мрежи. Използва пакети от 1 024 байта и изисква 8-bit clean връзка.

e Този протокол по същество трябва да прави същото като *t*. Основната разлика е, че *e* е потоков протокол и поради това е подходящ само за надеждни мрежови връзки.

f Този протокол е проектиран за използване в надеждни X.25 връзки. Той е потоков протокол и очаква 7-битов път за данни. 8-битовите символи се заграждат в кавички, което може да го направи много неефективен.

G Версия на g за System V Release 4. Този протокол се разпознава и от някои други версии на UUCP.

a Този протокол е подобен на ZMODEM. Той изисква 8-битова връзка, но заграждa в кавички определени контролни символи като XON и XOFF.

Настройване на протокола за предаване

Всички протоколи позволяват някакво изменение на размера на пакета, таймаута и т.н. Обикновено, стойностите по подразбиране работят добре при стандартни обстоятелства, но може да не бъдат оптимални във вашия случай. Например, протоколът g използва размери на прозореца от 1 до 7 и големина на пакета в степени на 2 от 64

до 4096. Ако вашата телефонна линия обикновено е толкова шумна, че отхвърля повече от 5% от всичките пакети, вероятно ще трябва да намалите размера на пакета и прозореца. От друга страна, при много добри телефонни линии служебната информация на протокола при изпращане на потвърждения на всеки 128 байта може да се окаже излишна, така че можете да увеличите размера на пакета до 512 или дори до 1 024. Повечето двоични данни включени в дистрибуциите на Linux по подразбиране са с размер на прозореца 7 и 128-байтови пакети.

Taylor UUCP ви дава възможност да настроите параметрите с командата *protocol-parameter* в *sys* файла. Например, за да зададете размера на пакета на g протокола на 512, когато комуникирате с **pablo**, трябва да добавите следното:

```
system   pablo
...
protocol-parameter g  packet-size  512
```

Настройваните параметри и техните имена при различните протоколи са различни. За пълен списък вижте документацията, приложена в изходния код на Taylor UUCP.

Избиране на специфични протоколи

Не всяка реализация на *uucico* комуникира и разбира всеки протокол, така че по време на началната фаза на договаряне и двата процеса трябва да се договорят за един общ протокол. Главната *uucico* предлага на slave списък с поддържани протоколи, като изпраща `Pprot-list`, от който slave може да си избере някой.

В зависимост от типа на използвания порт (модем, TCP или директен), *uucico* ще състави списък с протоколи по подразбиране. За модемни и директни връзки, този списък обикновено включва *i, a, g, G* и *j*. За TCP връзки списъкът е: *t, e, i, a, g, G, j* и *f*. Можете да отмените този списък по подразбиране с командата *protocols*, която може да бъде зададена в системния запис, а също и в запис за порта. Например, можете да редактирате записа от *port* файла за вашия модемен порт по следния начин:

```
port            serial1
...
protocols       igG
```

Това ще изисква всяка входяща или изходяща връзка през този порт да използва *i*, *g* или *G*. Ако отдалечената система не поддържа нито един от тези протоколи, разговорът ще бъде неуспешен.

Отстраняване на грешки

Този раздел описва какво може да се обърка с вашата UUCP връзка и на места дава предложения за поправяне на грешката. Въпреки че тези проблеми се откриват по обичаен начин, нещата, които могат да се объркат, са много повече.

Ако имате проблем, разрешете дебъгване с *-xall* и погледнете изходните данни в *Debug* в директорията на буфера. Файлът трябва да ви помогне бързо да разпознаете проблема. Често е полезно да включите говорителя на модема, когато той не осъществява връзка. При Hayes-съвместими модеми можете да включите говорителя като добавите `ATL1M1 OK` към chat-скрипта за модема във файла *dial*.

Първата проверка винаги трябва да бъде за това дали всички права за достъп до файлове са правилно зададени. *uucico* трябва да е setuid **uucp**, а всички файлове в */usr/lib/uucp*,

/var/spool/uucp и */var/spool/uucppublic* трябва да се притежават от **uucp**. Освен това, има и скрити файлове в директорията на буфера, които също трябва да се притежават от **uucp**.[45]

Когато сте сигурни, че правата за достъп за всички файлове са зададени правилно и все още имате проблеми, тогава можете да започнете да приемате съобщенията за грешка буквално. Сега ще разгледаме някои от често срещаните грешки и проблеми.

uucico продължава да казва "Wrong Time to Call"

Това вероятно означава, че в системния запис в *sys* файла не сте задали команда *time*, която описва подробно кога отдалечената система може да бъде извикана, или сте задали такава, която всъщност забранява извикане в момента. Ако не е дадена таблица на повикванията, *uucico* приема, че системата никога не може да бъде извикана.

[45] Т.е. файлове с имена, започващи с точка. Такова файлове обикновено не се показват чрез командата *ls*.

uucico се оплаква, че сайтът е вече заключен

Това означава, че *uucico* открива файл за заключване за отдалечена система в */var/spool/uucp*. Файлът за заключване може да е от поранно извикване на системата, което се е провалило или е било анулирано. Друго възможно обяснение е, че има друг *uucico* процес наоколо, който се опитва да набере отдалечената система и е "затънал" в chat-скрипт или е блокиран поради някаква друга причина.

За да поправите тази грешка, унищожете всички *uucico* процеси, отворени за сайта със сигнал за прекъсване и премахнете всички файлове за заключване, които те са оставили.

Можете да се свържете с отдалечения сайт, но chat-скриптът не се изпълнява

Погледнете текста, който получавате от отдалечения сайт. Ако е неразбираем, може би имате проблем със скоростта. В противен случай, потвърдете, че той действително се съгласува с това, което вашият chat-скрипт очаква. Запомнете, че chat-скриптът стартира с очакван низ. Ако получите покана за влизане и изпратите името си, но никога не получите покана за парола, вмъкнете някакво закъснение преди да го изпратите или дори между буквите. Може би сте твърде бързи за вашия модем.

Модемът ви не набира

Ако модемът не показва, че DTR линия е активирана, когато *uucico* се обажда, вероятно не сте задали правилното устройство към *uucico*. Ако модемът ви разпознае DTR, проверете с терминална програма, която може да запишете в модема. Ако това работи, включете ехото чрез `\E` при стартирането на chat-скрипта за модема. Ако модемът не прави ехо на командите ви по време на изпълнение на скрипта, проверете дали скоростта на линията ви не е твърде висока или твърде ниска. Ако има ехо, проверете дали не сте забранили отговорите на модема или сте ги настроили за числови кодове. Уверете се, че самият chat-скрипт е правилен. Запомнете, че трябва да напишете две обратно наклонени черти, за да изпратите една към модема.

Модемът ви се опитва да набира, но не може да излезе

Вмъкнете закъснение в телефонния номер, особено ако трябва да набирате специална последователност, за да получите външна линия от корпоративна телефонна мрежа. Уверете се, че използвате правилният тип набиране, тъй като някои телефонни мрежи поддържат само един тип. Като допълнение, проверете два пъти номера на телефона, за да сте сигурни, че е правилен.

Влизането в системата е успешно, но договарянето се проваля

Е, в такава ситуация може да има много проблеми. Изходът в дневника ще ви каже много неща. Вижте какви протоколи предлага отдалеченият сайт (по време на договарянето той изпраща низ `P protlist`). За да бъде успешно договарянето, и двата края трябва да поддържат поне един общ протокол, така че проверете дали това е така.

Ако отдалечената система изпраща *RLCK*, тогава съществува остарял дневник за вас в отдалечената система, която вече е свързана към отдалечената система на друга линия. В противен случай, помолете администратора на отдалечената система да премахне файла.

Ако отдалечената ситема изпраща *RBADSEQ*, тя има разрешена за вас проверка за броя на разговорите, но числата не съвпадат. Ако тя изпраща *RLOGIN*, не ви е разрешено да влизате под този идентификатор.

Дневници и отстраняване на грешки

Когато компилирате UUCP пакета, за да използвате влизане в системата в стил Taylor, имате само три глобални дневника, всеки от които постоянно се съхранява в директорията на буфера. Основният дневник се нарича *Log* и съдържа цялата информация за създадените връзки и прехвърлените файлове. Типична извадка изглежда по следния начин (след малко преобразуване, за да може да се вмести на страницата):

```
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 successful
               (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 17:16:02.50 539) Call complete (26 seconds)
uuxqt pablo postmaster (1994-05-28 17:16:11.41 546) Executing
               X.pabloX04ai
               (rmail okir)
uuxqt pablo postmaster (1994-05-28 17:16:13.30 546) Executing
X.pabloX04as
               (rmail okir)
uuxqt pablo postmaster (1994-05-28 17:16:13.51 546) Executing
X.pabloX04c1
               (rmail okir)
```

Следващият важен дневник е *Stats*, който изброява статистики за прехвърляне на файлове. Частта от *Stats*, която съответства на горното прехвърляне изглежда по следния начин (отново, редовете са разделени, за да се вместят в страницата):

```
postmaster pablo   (1994-05-28 17:15:44.78)
                   received 1714 bytes in 1.802 seconds (951 bytes/)
postmaster pablo   (1994-05-28 17:15:46.66)
                   received 57 bytes in 0.634 seconds (89 bytes/)
postmaster pablo   (1994-05-28 17:15:49.91)
                   received 1898 bytes in 1.599 seconds (1186 bytes/)
postmaster pablo   (1994-05-28 17:15:51.67)
                   received 65 bytes in 0.555 seconds (117 bytes/)
postmaster pablo   (1994-05-28 17:15:55.71)
                   received 3217 bytes in 2.254 seconds (1427 bytes/)
postmaster pablo   (1994-05-28 17:15:57.31)
                   received 65 bytes in 0.590 seconds (110 bytes/)
```

Третият файл е *Debug*. Тук е записана информация, свързана с отстраняването на грешки. Ако използвате отстраняване на грешки, трябва да се увериге, че този файл има защитен режим 600. В зависимост от режима за отстраняване на грешки, който избирате, този файл може да съдържа потребителското име за влизане и паролата, която използвате за връзка към отдалечената система.

Ако имате някакви инструменти, които очакват дневниците ви да са в традиционен формат, използван от HDB-съвместими реализации на UUCP, можете и да компилирате Taylor UUCP, за да получите дневници в HDB-стил. Това е просто въпрос на разрешаване на опцията compile-time в *config.h* файла.

ГЛАВА 17

ЕЛЕКТРОННА ПОЩА

Най-честото използване на мрежите от момента на тяхното създаване беше прехвърлянето на електронна поща. Електронната поща първоначално е създадена като проста услуга, предназначена за копиране на файл от една машина на друга и за добавяне на този файл към файла със съобщения *mailbox* на получателя. Концепцията остава същата, въпреки непрекъснатото разрастване на мрежите, техните сложни изисквания за маршрутизиране и непрекъснатото увеличаващия се брой на съобщенията.

Бяха създадени различни стандарти за обмен на електронна поща. Интернет сайтове се базират на стандарта RFC-822, допълнен от някои RFC стандарти, описващи независимото от конкретната машина прехвърляне на данни, включващи *всичко*, за което бихте могли да се сетите – графики, звукови файлове и специални множества от символи, и всичко това изпращано чрез електронната поща.[46] CCITT дефинира друг стандарт, известен като X.400, който все още се използва в някои големи корпоративни и правителствени среди, но постепенно започва да излиза от употреба.

Доста голям брой програми за електронна поща са реализирани за операционната система UNIX. Една от най-добре познатите е програмата *sendmail,* разработена от Eric Allman в Калифорнийския Университет в Бъркли, Калифорния, САЩ. В момента Eric Allman предлага *sendmail* като комерсиален продукт, но програмата все още е

[46] Ако не вярвате, прочетете RFC-1437!

безплатна. *sendmail* се прилага като стандартен пощенски агент в някой дистрибуции на Linux. В следващата Глава 18, *Sendmail,* описваме конфигурирането на *sendmail.*

Освен това, Linux използва програмата *Exim,* написана от Philip Hazel от Университета в Кеймбридж. В Глава 19, *Конфигуриране и използване на Exim*, ще разгледаме конфигурирането на *Exim*.

Сравнена със *sendmail, Exim* е доста по-млада. За повечето сайтове с изисквания за електронна поща, техните възможности са доста близки.

И двете програми поддържат множество конфигурационни файлове, които трябва да бъдат настроени за вашата система. Освен информацията, която се изисква за функционирането на подсистемата за електронна поща (като името на локалния хост), съществуват още много други параметри, които могат да бъдат настройвани. Основният конфигурационен файл на *sendmail* е доста труден за разбиране. Общо взето, той изглежда като че ли вашата котка се е разходила по клавиатурата, при това с натиснат клавиш Shift. За разлика от него, конфигурационният файл на *Exim* е доста по-структуриран и полесен за разбиране. За съжаление, *Exim* не предоставя директна поддържа на UUCP, а работи само с адреси на домейни. Днес това не е толкова голямо ограничение, както е било преди и почти всички сайтове могат да се обслужват от *Exim.* За повечето сайтове обаче, работата за настройка на която и да е от двете програми за електронна поща е една и съща.

В тази глава ще разгледаме какво представлява електронната поща и какви са функциите на системния администратор. Глава 18 и Глава 19 съдържат инструкции за настройване на *sendmail* и *Exim* за първи път. Включената там информация ще помогне на по-малките сайтове да започнат да работят, но съществуват и много други опции, така че ще прекарате още доста щастливи часове пред компютъра, докато успеете да конфигурирате най-елегантните възможности.

В края на тази глава ще разгледаме накратко настройването на *elm*, стандартен пощенски агент в повечето UNIX-подобни системи, включително и Linux.

За повече информация относно свързаните с електронната поща в Linux въпроси се обърнете към електронното ръководство Electronic mail HOWTO от Guylhem Aznar,[47] което периодично се публикува в

[47] С Guylhem можете да се свържете на адрес *guylhem@danmark.linux.eu.org*.

comp.os.linux.answers. Освен това, дистрибуциите с изходен код на *elm, sendmail* и *Exim* съдържат изчерпателна информация, която би могла да отговори на вашите въпроси относно тяхното настройване и конфигуриране. В съответните глави сме направили необходимите препратки към тази документация. За обща информация относно електронната поща и прилагането ѝ, вижте изброените в библиографията RFC стандарти в края на тази книга.

Какво е пощенско съобщение?

Пощенското съобщение се състои от тяло, което всъщност е текста на съобщението, и специални административни данни, определящи получателя, средата за прехвърляне и т.н. – неща, които можете да видите на плика на едно обикновено писмо.

Тези административни данни се разделят на две категории. В първата категория влизат всички данни, определящи средата за прехвърляне като адреса на изпращача и получателя. Затова, тази част се нарича *плик* (*envelope*). Тя може да се промени от използвания софтуер при прехвърляне на съобщението.

Втората категория са всички данни, необходими за обработка на съобщението, които не зависят от никакъв механизъм за прехвърляне – това са реда с темата на съобщението (*subject*), списъка с всички получатели и датата на изпращане. В много мрежи е прието тези данни да се добавят към съобщението, оформяйки по този начин т. нар. *заглавие на съобщението* (*mail header*). Това заглавие е разделено от тялото на съобщението с един празен ред.[48]

В света на UNIX по-голямата част от софтуера за прехвърляне на електронна поща използва формат за заглавието съгласно стандарта RFC-822. Неговата първоначална цел е да се зададе стандарт за използване в ARPANET, но тъй като е проектиран да бъде независим от средата, лесно е адаптиран за използване в други мрежи, включително много UUCP-базирани мрежи.

RFC-822 обаче е само най-малкия общ знаменател. Създадени са много нови стандарти, за да задоволят нарастващите нужди от шифриране на данни, поддръжка на международни кодови таблици и

[48] Нещо обичайно е да добавите *подпис* или *.sig* към съобщението, който обикновено съдържа информация за автора и някаква шега или мото. Подписът е разделен от съобщението с ред съдържащ "-", следвано от интервал

MIME (Multipurpose Internet Mail Extensions), описани в RFC-1341 и в други RFC.

Във всички тези стандарти заглавието се състои от няколко реда, разделени от последователност EOF (end-of-line). Всеки ред се състои от име на поле в първата колона, последвано от самото поле. Името на полето и самото поле са разделени от двоеточие и празно пространство. Форматът и значението на всяко поле варират в зависимост от името на полето. Поле от заглавието може да продължава и на нов ред, ако следващият ред започва със символ за празно пространство, например табулация. Полетата могат да бъдат в произволна последователност.

Типично заглавие на съобщение може да изглежда по следния начин:

```
Return-Path: <ph10@cus.cam.ac.uk>
Received: ursa.cus.cam.ac.uk (cusexim@ursa.cus.cam.ac.uk [131.111.8.6])
   by al.animats.net (8.9.3/8.9.3/Debian 8.9.3-6 ) with ESMTP
   id WAA04654
   for <terry@animats.net>; Sun, 30 Jan 2000 22:30:01 +1100
Received: from ph10 (helo=localhost) by ursa.cus.cam.ac.uk with
local-smtp
   (Exim 3.13 #1) id 12EsYC-0001eF-00: Sun, 30 Jan 2000 11:29:52 +0000
Date: Sun, 30 Jan 2000 11:29:52 +0000 (GMT)
From: Philip Hazel <ph10@cus.cam.ac.uk>
Reply-To: Philip Hazel <ph10@cus.cam.ac.uk>
To: Terry Dawson <terry@animats.net>, Andy Oram <andy@oreilly.com>
Subject: Electronic mail chapter
In-Reply-To: <38921283.A58948F2@animats.net>
Message-ID: <Pine.SOL.3.96.100013011515.5800A-
200000@ursa.cus.cam.ac.uk>
```

Обикновено, всички необходими полета от заглавието са генерирани от пощенския интерфейс, който използвате – *elm, pine, mush* или *mailx*. Някои полета обаче могат да се добавят от потребителя. elm, например, ви позволява да редактирате част от заглавието на съобщението. Други полета се добавят от софтуера за прехвърляне на съобщения. Ако разгледате локалния файл със съобщения, можете да видите всяко съобщение, предшествано от "From" (Забележка: без двоеточие след него). Това НЕ е RFC-822 заглавие; този ред е бил вмъкнат от вашия софтуер за поща за улеснение на програмите, четящи съобщенията. За да се избегнат потенциалните проблеми с редовете в тялото на съобщението, които също започват с "From", стандартна процедура е да се използва специално кодиране всеки път, когато се достигне до такава последователност, като пред нея се добави символа >.

Следва списък на общите полета в заглавието на съобщение и тяхното значение:

`From:`

Това поле съдържа e-mail адреса и вероятно "истинското име" на изпращача. Тук се използват цял зоопарк от различни формати.

`To:`

Това е списък с e-mail адресите на получателите. Множество адресисе разделят със запетайки.

`Cc:`

Това е списък с e-mail адресите, които ще получат копие на съобщението. Множество адресисе разделят със запетайки.

`Bcc:`

Това е списък с e-mail адресите, които ще получат копие на съобщението. Основната разлика между "Cc:" и "Bcc:" е, че изброените в "Bcc:" адреси няма да присъстват в заглавието на съобщенията, получени от който и да е получател. Това е начин да съобщите на получателите, че сте изпратили копия на съобщението и до други хора, без да ги уведомявате кои са те. Множество адресисе разделят със запетайки.

`Subject:`

Описва с няколко думи съдържанието на съобщението.

`Date:`

Прилага датата и времето на изпращане на съобщението.

`Reply-To:`

Указва адреса, на който изпращащият съобщението иска да му бъде върнат отговор. Това може да бъде полезно, ако имате няколко различни адреса, но искате да получавате по-голямата част от пощата си на един точно определен адрес. Това поле не е задължително.

`Organization:`

Организацията-собственик на машината, от която идва съобщението. Ако машината е ваша лична собственост или оставете това поле празно, или въведете "private" или някакъв произволен текст. Това поле не е описано в RFC и не е задължително. Някои програми за електронна поща го поддържат директно, а други не.

`Message-ID:`

Низ, генериран от програмата за електронна поща на изпращащата система. Това е уникалния идентификатор на съобщението.

`Received`:

Всеки сайт, обработващ вашата поща (включително машините на изпращача и получателя), вмъква такова поле в заглавието, задавайки името на сайта, идентификатора на съобщението, дата и време на получаване на съобщението, откъде е дошло и какъв софтуер за прехвърляне на съобщения е използван. Тези редове ви позволяват да проследите маршрута на съобщението и имате възможност да се оплачете на отговорното лице, ако нещо не е наред.

`X-`*`нещо`*:

Всички програми за поща приемат заглавие, започващо с `X-`. Това поле се използва за реализиране на допълнителни възможности, които още не са описани в RFC или никога няма да бъдат описани. Например, съществуваше доста голям пощенски сървър под Linux, който ви позволяваше да указвате кой канал искате да използвате за изпращане на поща чрез добавяне на низа **`X-Mn-Key:`**, следван от името на канала.

Как се доставя пощата?

Обикновено съставяте съобщение, използвайки интерфейса на програми за електронна поща като *mail* и *mailx* или по-усъвършенствани като *mutt, tkrat,* или *pine*. Тези програми се наричат *mail user agents (потребителски програми за поща)* или MUA. Когато изпращате e-mail съобщение, в повечето случаи програмата с потребителски интерфейс ще го предаде на друга програма, която ще го достави. Такава програма се нарича *mail transport agent (програма за прехвърляне на поща*) или MTA. В повечето системи една и съща програма за прехвърляне на поща се използва за доставяне и на локалната, и на отдалечената поща, и обикновено се извиква като */usr/sbin/sendmail* или на неотговарящи на FSSTND системи като */usr/lib/sendmail*. При UUCP системи не е необичайно доставянето на пощата да се извършва от две отделни програми: *rmail* за доставяне на отдалечена поща и *lmail* за доставяне на локална поща.

Разбира се, доставянето на локална поща е нещо повече от добавяне на входящите съобщения към пощенската кутия на получателя. Обикновено, локалният MTA разпознава псевдонимите (задаването на адреса на локален получател да сочи към друг адрес) и препращането (пренасочване на пощата на потребителя към друго местоназначение). Освен това, съобщенията, които не могат да бъдат доставени, обикновено се връщат на изпращача с някакво съобщение за грешка.

При доставяне на отдалечена поща, използваният софтуер за прехвърляне зависи от характера на връзката. За доставяне на поща през мрежи, използващи TCP/IP, обикновено се използва протокола SMTP (*Simple Mail Transport Protocol* – прост протокол за прехвърляне на поща), който е описан в стандарта RFC-821. SMTP е проектиран да доставя пощата директно до машината на получателя, съгласувайки прехвърлянето на съобщението с SMTP демона на отдалечената страна. Обичайна практика днес е организациите да конфигурират специални хостове, които приемат всички съобщения за получатели в организацията, и за този хост да настроят доставянето до съответния получател.

При UUCP мрежи доставянето на съобщения не е директно, а преминава през определен брой междинни системи. За да изпратите съобщение през UUCP връзка, изпращащият MTA агент обикновено изпълнява *rmail* на препращащата система, използвайки *uux*, и пуска съобщението на стандартния вход.

Тъй като *uux* се извиква поотделно за всяко съобщение, това би могло да причини значително натоварване на основния пощенски концентратор, а освен това бъркотията от стотици малки файлове в UUCP спулера заема непропорционален обем от дисковото пространство.[49] Затова, някои програми за прехвърляне на поща ви позволяват да събирате по няколко съобщения, предназначени за отдалечена система, в един единствен пакет в т.нар batch-файл. Този файл съдържа SMTP командите, които локалният хост би използвал при директна SMTP връзка. Това се нарича BSMTP или *batched* SMTP. След това batch-файлът се изпраща към програмите *rsmtp* или *bsmtp* на отдалечената система, която обработва входа както при обикновена SMTP връзка.

E-mail адреси

Един e-mail адрес се състои поне от две части. Едната част е името на *пощенския домейн,* който в крайна сметка се преобразува до хоста на получателя или друг хост, който приема поща от името на получателя. Другата част е уникален потребителски идентификатор в някаква форма, който би могло да бъде името за влизане на този потре-

49 Това е така, тъй като дисковото пространство обикновено е разделено на блокове от по 1,024 байта. Следователно, дори съобщение с размер няколко десетки байта ще "погълне" цял килобайт.

бител, истинското му име във формат "Собствено_име.Фамилия" или произволен псевдоним, който ще се преобразува в потребител или списък от потребители. Други схеми за адресиране на електронна поща, например X.400, използват по-общо множество от "атрибути", които се използват за търсене на хост на получателя в X.500 директориен сървър.

Начинът, по който се интерпретират e-mail адресите, до голяма степен зависи от това какъв тип мрежа използвате. Ще се съсредоточим върху начина, по който TCP/IP и UUCP мрежите обработват e-mail адресите.

RFC-822

Сайтовете в Интернет се придържат към стандарта RFC-822, който изисква записване на адреса във формат *потребител@хост.домейн*, където *"хост.домейн"* е пълното хост-име на домейна. Правилният начин да се чете разделящият символ е "commercial at", но може да го четете просто "at". Този начин на запис не определя маршрута на съобщението до хоста на получателя. Маршрутът на съобщението се определя от механизмите, които ще обясним след малко.

Ако поддържате сайт, свързан към Интернет, ще използвате голяма част от RFC-822. Освен поща, той описва и други услуги, например новини. В Глава 20, *Мрежови новини*, ще разгледаме начина, по който RFC-822 се използва за новини.

Остарели формати за поща

В оригиналната UUCP среда преобладаващата форма беше "*path!host!user*", където *path* описва последователността от хостове, през които трябва да премине съобщението преди да достигне местоназначението *host*. Тъй като разговорно удивителният знак се нарича "бум", тази конструкция е наречена *бум-път*. Днес, много UUCP-базирани мрежи са се приспособили към стандарта RFC-822 и разпознават базираните на домейн адреси.

Други мрежи все още използват различни средства за адресиране. DECnet-базираните мрежи, например, използват две двоеточия като разделител за адрес, при което се получават адреси от вида *host::user*

(хост::потребител).[50] Стандартът X.400 използва съвсем различна схема, описвайки получателя чрез множество от двойки атрибут-стойност, например страна и организация.

И накрая, при FidoNet всеки потребител се идентифицира от кода като **2:320/204.9**, състоящ се от четирицифрено обозначаване на зона (2 е за Европа), мрежа (320 за Париж и Бенлю), възел (локалният концентратор) и точка (PC на потребителя). FidoNet адресите могат да се съставят съгласно RFC-822; горният адрес, например, ще бъде записан по следния начин: *Thomas.Quinot@p9.f204.n320.z2.fidonet.org*. Споменахме ли вече, че имената на домейните са лесни за запомняне?

Смесване на различни формати за поща

Неизбежно е когато съберете на едно място определен брой различни системи и определен брой умни хора, те да пожелят начини да свържат различаващите се системи, за да могат да обменят данни. Следователно, съществуват определен брой различни пощенски шлюзове, които могат да свързват две различни пощенски системи, така че пощата да може да бъде препращана от една към друга система. Адресирането е критичен въпрос при свързването на две системи. Няма да разглеждаме подробно самите пощенски шлюзове, но ще се спрем на някои усложнения при адресирането, възникващи при използването им.

Нека да разгледаме смесването на начина на записване на бум-път в UUCP стил и според RFC-822. Тези два типа на адресиране не се смесват много добре. Да предположим, че имаме адрес от типа: *домейнА!потребител@домейнВ*. Не е ясно дали знакът @ е с по-голям приоритет от пътя и обратно: трябва ли да изпратим съобщението на *домейнВ*, който ще го изпрати на потребителя *домейнА!потребител*, или съобщението трябва да се изпрати на *домейнА*, който ще го пренасочи към *потребител@домейнВ*?

Адреси, смесващи различни типове адресни оператори, се наричат *хибридни адреси*. Най-общият тип, който току що илюстрирахме, обикновено се преобразува като приоритетът се дава на знака @. В *домейнА!потребител@домейнВ*, това означава, че съобщението се изпраща първо към домейнВ.

[50] Когато правите опит да достигнете до DECnet адрес от RFC-822 среда, трябва да използвате *"host::user@relay*, където relay е името на известен DECnet ретранслатор в Интернет.

Освен това, съществува начин за задаване на маршрута по съвместим с RFC-822 начин: *<@домейнA, домейнB:потребител@домейнC>* обозначава адреса на *потребител* от *домейнC*, където *домейнC* може да се достигне през *домейнA* и *домейнB* (в тази последователност). Такъв тип адрес често се нарича *source routed* адрес (адрес с предварително зададен маршрут). Не е добра идея да разчитате на такова поведение, тъй като новите преработки на RFC, описващи маршрутизирането на поща, препоръчват предварително зададения маршрут да се игнорира и вместо това да се направи опит за доставяне на пощата директно до отдалеченото местоназначение.

Съществува и адресен оператор %: *потребител%домейнB@домейнA* първо се изпраща към домейнA, който разширява най-десният (в този случай единствен) знак за процент % в знака @. Адресът става **потребител@домейнB**, а програмата за поща безпроблемно препраща вашето съобщение към *домейнB*, който го доставя на потребителя. Понякога този тип адресиране се нарича "Ye Olde ARPAnet Kludge", но използването му не е препоръчително.

По-нататък ще ви запознаем с последиците от използването на тези различни типове за адресиране. В RFC-822 среда, трябва да избягвате използването на какъвто и да е друг тип адресиране, освен абсолютно адресиране от типа *потребител@хост.домейн*.

Как работи маршрутизирането на пощата?

Процеса на насочване на съобщението към хоста на получателя се нарича *маршрутизиране*. Освен намирането на път от изпращащия сайт до местоназначението, този процес включва и проверка за грешки, а може да включва и оптимизация на скоростта и цената.

Съществува голяма разлика в начина, по който един UUCP сайт и един Интернет сайт осъществяват маршрутизирането. При Интернет основната работа по насочването на данните към хоста на получателя (вече известен със своя IP адрес) се извършва от IP мрежовия слой, докато в UUCP зоната маршрутът се посочва от потребителя или се генерира от потребителската програма за поща.

Маршрутизиране на поща през Интернет

В Интернет конфигурацията на хоста на получателя определя дали се извършва някакво специфично маршрутизиране на пощата. По под-

разбиране, съобщението се доставя като първо се определи хоста, на който ще се изпрати това съобщение, и след това то се доставя директно на този хост. Повечето Интернет сайтове се опитват да насочат цялата постъпваща поща към пощенски сървър, който може да управлява целия този трафик и да разпредели пощата локално. За да обяви тази услуга, сайтът публикува т. нар. MX запис за своя локален домейн в DNS базата данни. MX е съкращение от *Mail eXchanger* и означава, че хостът на сървъра е готов да работи като пощенски посредник за всички пощенски адреси в домейна. Освен това, MX записите могат да се използват и за управление на трафика за хостове, които не са свързани към Интернет, например, UUCP мрежи или FidoNet хостове, които трябва да изпращат своята поща през шлюз.

На MX записите винаги се задава *предпочитание*. Това е цяло положително число. Ако за един хост съществуват няколко MX записа, програмата за поща ще се опита да прехвърли съобщението към MX с най-ниската стойност на предпочитанието и само ако не успее, ще се опита да използва хост с по-висока стойност. Ако самият локален хост е MX за адреса на местоназначението, той може да препраща съобщения само към MX хостове с по-ниско предпочитание от неговото; това е сигурен начин да се избегнат безкрайни пощенски цикли. Ако не съществува MX запис за домейн или няма други MX записи, подходящи за използване, на програмата за поща се позволява да повери дали домейнът има свързан с нея IP адрес и да се опита да достави пощата директно на този хост.

Да предположим, че дадена организация, например Foobar, Inc., иска цялата ѝ поща да се управлява от нейната машина **mailhub**. Тогава в DNS базата данни ще има MX записи като този:

```
green.foobar.com   IN MX 5  mailhub.foobar.com
```

Този запис обявява **mailhub.foobar.com** като MX за **green.foobar.com** с предпочитание 5. Хост, който иска да достави съобщение за *joe@green.foobar.com*, проверява DNS и открива MX запис, сочещ към **mailhub**. Ако няма друг MX с предпочитание по-малко от 5, съобщението се доставя до машината **mailhub,** която след това го разпределя към **green**.

Това е доста просто описание на начина, по който работят MX записите. За повече информация относно маршрутизирането на поща в Интернет вижте стандартите RFC-821, RFC-974 и RFC-1123.

Маршрутизиране на поща в UUCP света

Маршрутизирането на поща в UUCP мрежи е много по-сложно от това в Интернет, тъй като софтуера за пренасяне сам по себе си не извършва никакво маршрутизиране. Преди време, цялата поща трябваше да се адресира чрез използване на бум-пътища. Бум-пътищата задават списък на хостовете, през които се насочва съобщението, разделени от удивителни знаци и следвани от името на потребителя. За да адресирате писмо до потребител с име Janet на машина **moria**, трябва да използвате пътя: *eek!swim!moria!janet*. Това ще изпрати пощата от вашия хост до **eek**, оттам до **swim**, и накрая до **moria**.

Очевидният недостатък на тази техника е, че изисква да помните доста повече неща за мрежовата топология, бързите връзки и т.н., отколкото изисква маршрутизирането в Интернет. И още по-лошо от това, промените в топологията на мрежите – като унищожени връзки или премахнати хостове – могат да провалят доставянето на съобщение, просто защото не сте знаели за тях. И накрая, ако се преместите на друго място, ще трябва да обновите всичките си маршрути.

Едно нещо обаче, което направи необходимо използването предварително зададен маршрут, беше наличието на двусмислени имена на хостове. Например, да предположим, че съществуват два сайта с име **moria**, един в САЩ и един във Франция. За кой сайт тогава се отнася изразът *moria!janet*? Това би могло да се уточни единствено чрез указването на пътя за достигане до **moria**.

Първата стъпка при отстраняване на двусмислените имена на хостове беше създаването на проекта UUCP Mapping Project. Той се намира в Университета Rutgers и регистрира всички официални UUCP имена на хостове, заедно с информация за техните UUCP съседи и географското им положение. По този начин се избягва дублирането на имена на хостове. Информацията, събрана от проекта, се публикува като *Usenet карти*, разпространявани периодично през Usenet. Типичен системен запис в карта (след премахване на коментарите) изглежда по следния начин:[51]

```
moria
    bert(DAILY/2),
    swim(WEEKLY)
```

[51] Карти за сайтове, регистрирани от проекта UUCP Mapping Project, се разпространяват чрез групата по интереси *comp.mail.maps*; други организации могат да публикуват отделни карти за техните мрежи.

Този запис показва, че **moria** има връзка към машината **bert**, която извиква два пъти дневно, и към **swim,** която извиква един път седмично. По-късно ще се върнем към формата на файлове с картис повече подробности.

Използвайки дадената в картите информация за връзките, можете автоматично да генерирате пълния път от вашия хост до който и да е сайт. Тази информация обикновено се записва във файла *path*, наричан още *pathalias database* (база данни с псевдоними на пътища). Да предположим, че картите определят пътя, чрез който можете да достигнете до **bert** през **ernie**; запис с псевдоним на път до **moria**, генериран от отрязък от предходната карта, ще изглежда по следния начин:

```
moria     ernie!bert!moria%s
```

Ако сега зададете адрес на получателя *janet@moria.uucp*, вашият MTA ще избере показания по-горе маршрут и ще изпрати съобщението към **ernie** с адрес на плика *bert!moria!janet*.

Изграждането на файла *path* от всички карти в Usenet не е много добра идея. Информацията в тях обикновено е остаряла и доста изкривена. Ето защо, само определен брой главни хостове използват пълните UUCP карти за изграждане на своите файлове с пътища. Повечето сайтове поддържат информация за маршрутите само за съседните им сайтове и изпращат поща към сайтове, които не откриват в своите бази данни, към по-интелигенти хостове с по-пълна информация за маршрутизирането. Тази схема се нарича *smart-host routing* (маршрутизиране чрез интелигентни хостове). Хостове, които имат само една UUCP пощенска връзка (наричани *сайтове-листа*) не извършват никакво маршрутизиране; те разчитат изцяло на техните интелигентни хостове.

Смесване на UUCP и RFC-822

Най-доброго решение на проблемите, свързани с маршрутизирането на поща в UUCP мрежи, е приемането на системата DNS в UUCP мрежите. Разбира се, не можете да отправите запитване към сървъра за имена през UUCP. Въпреки това, много UUCP сайтове са формирали малки домейни, координиращи вътрешно тяхното маршрутизиране. В картите тези домейни публикуват един или два хоста като техни пощенски шлюзове, така че не се налага да съществува запис в карта за всеки хост от домейна. Схемата за маршрутизиране вътре в домейна е напълно невидима за външния свят.

Това работи много добре със схемата за маршрутизиране чрез интелигентни хостове. Глобалната информация за маршрутите се поддържа само от шлюзовете; второстепенните хостове в рамките на домейна имат само малки, ръчно написани файлове с пътища, които съдържат списъци с маршрутите в техния домейн и маршрута до пощенския концентратор. Вече дори пощенските шлюзове нямат нужда от информация за маршрутите до всеки отделен UUCP хост в света. Освен пълната информация за маршрутите в обслужвания от тях домейн, те се нуждаят само от маршрутите до другите домейни, съдържащи се в техните бази данни. Например, следващият запис с псевдоним на път ще определи маршрута на пощата за всички сайтове в домейна **sub.org** до **smurf**:

```
.sub.org swim!smurf!%s
```

Поща, адресирана до *claire@jones.sub.org*, ще бъде изпратена към **swim** с адрес на плика *smurf!jones!claire*.

Йерархичната организация на пространството от имена на домейни позволява на пощенските сървъри да смесват по-специфични маршрути с други, не толкова специфични. Например, система във Франция може да има специфични маршрути за поддомейни на **fr**, но да насочва всякаква поща за хостове в домейна **us** към някоя система в САЩ. По този начин домейн-базираното маршрутизиране (така се нарича тази техника) намалява значително размера на базите данни с маршрутите, а също така и допълнителното натоварване, необходимо за административното им обслужване.

Основната полза обаче от използването на имена на домейни в UUCP средата е, че съвместимостта с RFC-822 позволява лесното използване на шлюз между UUCP мрежи и Интернет. В момента много UUCP домейни имат връзка с Интернет шлюз, който работи като техен интелигентен хост. Изпращането на съобщения през Интернет е по-бързо, а информацията за маршрутите е по-надеждна, тъй като Интернет хостове могат да използват DNS вместо Usenet Maps.

За да са достъпни от Интернет, UUCP-базираните домейни обикновено имат Интернет шлюз, който публикува MX запис за тях (MX записите бяха обяснени по-горе в раздела "Маршрутизиране на поща през Интернет"). Например, да предположим, че **moria** принадлежи на домейна **orcnet.org**. **gcc2.groucho.edu.** работи като негов шлюз към Интернет. Следователно, **moria** ще използва **gcc2** като свой интелигентен хост, така че цялата поща за чужди домейни се доставя през Интернет. От друга страна, **gcc2** ще публикува MX запис за ***.orcnet.org** и ще доставя цялата входяща поща за **orcnet** сайтовете

на **moria**. Звездата в ***.orcnet.org** е универсален символ, който съответства на всички хостове в този домейн, които нямат друг свързан с тях запис. Това е обичайната ситуация само при UUCP домейни.

Единственият оставащ проблем е, че UUCP програмите за прехвърляне не работят с пълните домейн имена. Повечето UUCP комплекти са проектирани да работят с имена до осем символа, някои дори и с по-малко, и използването на символи, различни от букви и цифри (например точка), е напълно извън възможностите им.

Затова трябва да използваме съответствие между RFC-822 имена и UUCP имена на хостове. Използването на такова съответствие е напълно зависимо от конкретната реализация. Общ начин за задаване на съответствие между FQDN и UUCP имена е използването на файл с псевдоним на пътя:

```
moria.orcnet.org      ernie!bert!moria!%s
```

Това ще генерира чист бум път в UUCP-стил от адрес, който задава пълното квалифицирано име на домейн. Някой програми за поща имат специален файл за това; *sendmail*, например, използва файла *uucpxtable*.

Понякога, когато изпращате поща от UUCP мрежа към Интернет, се изисква обратната трансформация (разговорно наричана *домейнизиране*). Тъй като изпращача на поща използва пълното квалифицирано име на домейн в адреса на получателя, проблемът се решава като не премахвате името на домейна от адреса на плика, когато препращате съобщението към интелигентния хост. Все още обаче има UUCP сайтове, които не са част от домейн. Те се домейнизират чрез добавяна на псевдо-домейна **uucp**.

Базата данни с псевдоними на пътища предоставя основната информация за маршрутите в UUCP-базираните мрежи. Типичен запис в нея ще изглежда по следния начин (името на сайта и пътя са разделени от табулации):

```
moria.orcnet.org        ernie!bert!moria!%s
moria                   ernie!bert!moria!%s
```

По този начин, всяко съобщение, предназначено за **moria**, се доставя през **ernie** и **bert**. И пълното квалифицирано име, и UUCP името на **moria** трябва да бъдат дадени, ако програмата за поща няма друг начин да определи съответствието между тях.

Ако искате да насочите всички съобщения към хостовете в домейн към неговия пощенски ретранслатор, можете да укажете пътя в базата данни с псевдоними на пътища, като дадете името на домейна, предхождано от точка като целта. Например, ако всички хостове в **sub.org** могат да бъдат достигнати през **swim!smurf**, записът за псевдонима на пътя ще изглежда по следния начин:

```
.sub.org swim!smurf!%s
```

Писането на файл с псевдоними на пътища е приемливо само, когато поддържате сайт, за който не трябва да генерирате много маршрути. Ако трябва да направите маршрутизиране за голям брой хостове, подобрият начин е да използвате командата *pathalias*, за да създадете файла от файловете с карти. Картите се поддържат по-лесно, тъй като можете просто да добавяте или премахвате система, редактирайки записа за картата на системата и създавайки отново файла с картата. Въпреки че картите, публикувани от проекта Usenet Mapping Project, вече не се използват много за маршрутизиране, има по-малки UUCP мрежи, които предлагат информация за маршрутизиране в техен собствен набор от карти.

Един файл с карта се състои от списък със сайтове, които всяка система избира или от които е избирана. Името на системата започва в първата колона и е следвано от разделен с запетая списък на връзките. Списъка може да се пренася и на нови редове, ако следващият ред започва с табулация. Всяка връзка се състои от името на сайта, следвано от стойност, поставена в скоби. Тази стойност е аригметичен израз, съставен от числа и символни изрази като DAILY (дневно) и WEEKLY (седмично). Редовете, започващи с диез (#) се игнорират.

Нека вземем за пример **moria**, който се свързва с **swim.twobirds.com** два пъти дневно, а **bert.sesame.com** – един път седмично. Връзката към **bert** използва бавен модем със скорост на предаване 2,400 bps. **moria** ще публикува следвания запис за карти:

```
moria.orcnet.org
    bert.sesame.com(DAILY/2)
    swim.twobirds.com(WEEKLY+LOW)
moria.orcnet.org = moria
```

Последния ред прави **moria** известен под неговото UUCP име. Обърнете вниманието, че неговата стойност трябва да бъде зададена като `DAILY/2`, тъй като обаждайки се два пъти на ден, вие всъщност делите стойността ден на две.

Използвайки информацията от такъв map файл, командата *pathfiles* може да изчисли оптималния маршрут до всеки сайт, изброен във файла с пътищата и да генерира база данни с псевдоними, която да се ползва за определяне на маршрутите до тези сайтове.

Pathalias предоставя двойка други възможности, например, скриване на сайт (т.е. сайтът е достъпен само през шлюз). За повече подробности вижте справочната страница на *pathalias* и пълен списък на стойностите за връзки.

Коментарите в map файла по принцип съдържат допълнителна информация за сайтовете, описани в него. Съществува строг формат, в който трябва да се задава тази информация, така че да може да бъде извличана от картите. Например, програмата *uuwho* използва база данни, създадена файлове с карти, за да покаже тази информация по приятно оформен начин. Когато регистрирате вашия сайт в организация, която разпределя файлове с карти на нейните членове, вие всъщност трябва да попълните такъв запис за карта. Следва примерен запис за карта (в действителност, това се отнася за сайта на Olaf):

```
#N     monad, monad.swb.de, monad.swb.sub.org
#S     AT 486 DX50; 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 1993
#
monad brwehq(DAILY/2)
# Domains
monad = monad.swb.de
monad = monad.swb.sub.org
```

Празното пространство след първите два символа е табулация. Значението на повечето полета е очевидно; подробно описание ще получите от всеки домейн, в който регистрирате. Полето `L` е доста забавно: то дава вашето географско положение в географска ширина/дължина и се използва за изчертаване на PostScript карти, показващи всички сайтове за всяка страна или в целия свят.[52]

Конфигуриране на elm

Името *elm* е съкращение от "electronic mail" (електронна поща) и е един от най-логично именуваните инструменти в UNIX. Elm предоставя пълноекранен интерфейс с добра възможност за помощна информация. Няма да дискутираме употребата на *elm*, а ще се спрем само на неговите конфигурационни опции.

Теоретично, можете да стартирате *elm* и без предварително конфигуриране, и всичко ще работи добре – ако имате късмет. Но има някои опции, които трябва да бъдат установени, въпреки че се изискват само при определени случаи.

При стартиране, *elm* чете набор от конфигурационни променливи от файла *elm.rc* в директория */etc/elm*. След това прави опит да прочете файла *.elm/elmrc*, намиращ се във вашата лична директория. Обикновено, този файл не го пишете вие. Той се създава, когато изберете "Save new options" от менюто с опции на *elm*.

Набора от опции за личния файл *elmrc* е достъпен и в глобалния файл *elm.rc*. Повечето настройки във вашия личен файл *elmrc* отменят тези от глобалния файл.

Глобални опции на elm

В глобалния файл *elm.rc* трябва да зададете опциите, които се отнасят за името на вашия хост. Например, в нашата Виртуална пивоварна файлът за **vlager** съдържа следното:

```
#
# Локалното име на хост
hostname = vlager
#
# Име на домейн
hostdomain = .vbrew.com
```

[52] Те се публикуват периодично в *news.lists.ps-maps*. Внимание! Те са ОГРОМНИ.

```
#
# Пълно квалифицирано име на домейн
hostfullname = vlager.vbrew.com
```

Тези опции задават представата на *elm* за локално име на хост. Въпреки че тази информация се използва рядко, трябва да зададете тези опции. Забележете, че тези отделни опции имат ефект, само ако са в глобалния конфигурационен файл; ако са в личния файл *elmrc*, ще бъдат игнорирани.

Национални кодови таблици

За поддръжка на различни типове съобщения като обикновен текст, двоични данни, PostScript файлове и други са разработени множество стандарти и RFCs, които променят стандарта RFC-822. Тези стандарти най-често са известни като MIME (Multipurpose Internet Mail Extensions). Освен другите функции, MIME показва на получателя дали е използван друг стандарт, различен от ASCII, при писането на съобщението. Например, използване на ударения във френския или на прегласи в немския език. *elm* поддържа тези символи до известна степен.

Кодовата таблица използвана от Linux за вътрешно представяне на символи обикновено е ISO-8859-1, което всъщност името на стандарта, към който се придържа. Той е известен също като Latin-1. Всяко съобщение, използващо символи от тази кодова таблица, ще има следния ред в заглавието си:

```
Content-Type: text/plain; charset=iso-8859-1
```

Получаващата система ще разпознае това поле и ще вземе съответните мерки при показването на съобщението. По подразбиране, за съобщения от типа `text/plain`, стойността на `charset` е `us-ascii`.

За да може да покаже съобщения с различна кодова таблица от ASCII, *elm* трябва да знае как да отпечата символите. По подразбиране, когато elm получава съобщение със стойност полето *charset*, различна от `us-ascii` (или друг тип на съдържанието, различен от `text/plain`), elm опитва да покаже съобщението, използвайки командата *metamail*. Съобщения, изискващи *metamail*, за да бъдат показни, се показват с буква M в първата колона на екрана.

Тъй като собствената кодова таблица на Linux е ISO-8859-1, не е необходимо извикването на *metamail* за показване на съобщения, използващи тази кодова таблица. Ако на *elm* е указано, че дисплея под-

държа ISO-8859-1, тогава няма да използва *metamail*, а ще показва директно съобщението. Това се прави посредством следната опция в глобалния файл *elm.rc*:

```
displaycharset = iso-8859-1
```

Обърнете внимание, че трябва да използвате тази опция дори никога да не изпращате или получавате съобщения, които в действителност съдържат символи, различни от ASCII символите. Това е така, тъй като хората, които изпращат такива съобщения, обикновено конфигурират своята програма за поща да зададат по подразбиране подходяща стойност на полето `Content-Type:` в заглавието на съобщението, независимо дали изпращат или не само ASCII съобщения.

Освен това, задаването на тази опция във файла *elm.rc* не е достатъчно. Когато показва съобщения с вградения си пейджър, *elm* извиква библиотечна функция за всеки символ, определяща дали символът може да се отпечата. По подразбиране, тази функция разпознава само ASCII символи, които могат да бъдат отпечатани, и показва всички останали като `^?`. Можете да игнорирате тази функция чрез задаване на променливата за средата `LC_CTYPE` да бъде равна на `ISO-8859-1`, която указва на библиотеката да приема символите от кодовата таблица Latin-1 като такива, които могат да бъдат отпечатвани. Поддръжката на тази и други възможности е валидна от версията на стандартната библиотека за Linux Version 4.5.8.

Когато изпращате съобщения, съдържащи специални символи от ISO-8859-1, във файла *elm.rc* трябва да зададете още две променливи:

```
charset = iso-8859-1
textxencoding = 8bit
```

Това указва на *elm* да докладва, че кодовата таблица в заглавието на съобщението е ISO-8859-1 и да я изпраща като 8-битова стойност (по подразбиране, всички символи се свиват до 7 бита).

Разбира се, всички разгледани опции за кодова таблица могат да бъдат установени в личния файл *elmrc* вместо в глобалния, така че отделните потребители могат да установят свои подразбиращи се настройки, ако глобалният файл не ги удовлетворява.

ГЛАВА 18

SENDMAIL

Въведение в sendmail

Казано е, че не сте *истински* UNIX администратор, докато не сте редактирали файла *senmail.cf*. Освен това е казано, че сте луд, ако сте се опитвали да го редактирате два пъти.

sendmail е невероятно мощна програма за поща. Освен това, тя е изключително трудна за научаване и разбиране. Всяка програма, чийто пълен справочник е 1,050 страници (*sendmail*, написан от Bryan Costales и Eric Allman, публикуван от O'Reilly), плаши повечето хората. Информация относно справочника за *sendmail* се съдържа в библиографията в края на книгата.

За щастие, новите версии на *sendmail* са различни. Вече не е необходимо да редактирате директно тайнствения файл *sendmail.cf*; новата версия предоставя помощна програма за конфигуриране, която ще създаде файла *sendmail.cf* от много по-прости макро файлове вместо вас. Не е необходимо да разбирате сложния синтаксис на *sendmail.cf*; макро файловете не го изискват. Вместо това, трябва само да изброите елементите като имената на възможностите, които искате да включите във вашата конфигурация, и да зададете някои от параметрите, определящи начина на действие на тези възможности. След това традиционна помощна програма на UNIX, наричана *m4*, взима вашата макро конфигурационна информация, смесва я с данните, които чете от файлове с шаблони, съдържащи истинския синтаксис на *sendmail.cf*, и по този начин генерира вашия *sendmail.cf* файл.

В тази глава ще ви запознаем със *sendmail* и ще опишем начина, по който можете да инсталирате, конфигурирате и тествате програмата, използвайки като пример Виртуалната пивоварна. Ако представената тук информация ви помогне да преодолеете страха от конфигурирането на *sendmail*, надяваме че ще се съберете смелост и за по-сложни конфигурации.

Инсталиране на sendmail

Програмата за прехвърляне на поща *sendmail* е включена в повечето дистрибуции на Linux. В този случай инсталирането е относително просто. Независимо от този факт, съществуват основателни причини да инсталирате *sendmail* от изходен код, особено ако сте подозрителни по отношение на сигурността. Програмата *sendmail* е много сложна и сие спечелила през годините репутация на програма, съдържаща грешки и пропуски в сигурността. Един от най-познатите примери е програмата RTM Internet червей, която използваше проблеми с препълването на буфера в ранните версии на *sendmail*. На този проблем ще се спрем накратко в Глава 9, *Защитна стена за TCP/IP*. Повечето програми, които използват пропуски в сигурността, включващи препълване на буфера, разчитат на това всички копия на *sendmail*, намиращи се на различни машини да бъдат еднакви, тъй като разчитат на данни, съхранявани на определени места. Това, разбира се, е точно това, което става с *sendmail*, инсталирана от дистрибуции на Linux. Когато компилирате *sendmail* от изходен код, можете да намалите този риск. Съвременните версии на *sendmail* са помалко уязвими, тъй като подробното разглеждане на проблемите, свързани със сигурността, стана много по-популярно в Интернет обществото.

Изходният код на *sendmail* е достъпен чрез анонимен FTP от **ftp.sendmail.org.**

Компилацията е много лесна, тъй като пакетът с изходния код на *sendmail* директно поддържа Linux. Стъпките при компилиране на *sendmail* са:

```
# cd /usr/local/src
# tar xvfz sendmail.8.9.3.tar.gz
# cd src
# ./Build
```

Необходими са ви права за достъп `root`, за да завършите инсталирането на резултатните двоични файлове, използвайки следното:

```
# cd obj.Linux.2.0.36.i586
# make install
```

Сега вече сте инсталирали двоичните файлове на *sendmail* в директорията */usr/sbin*. Няколко символни връзки към двоичните файлове на *sendmail* ще бъдат инсталирани в директорията */usr/bin*. За тези връзки ще говорим, когато разглеждаме общите задачи при изпълнението на *sendmail*.

Преглед на конфигурационните файлове

Обикновено, *sendmail* беше настроена чрез системен конфигурационен файл (обикновено наричан */etc/mail/sendmail.cf*, или */etc/sendmail.cf* в по-стари дистрибуции, или дори */usr/lib/sendmail.cf*), който няма нищо общо с езиците, които сте виждали преди. Редактирането на файла *sendmail.cf* за предоставяне на специализирано поведение, може да бъде прост опит.

Днес, *sendmail* прави всички конфигурационни опции управлявани от макрос с лесен за разбиране синтаксис. Макро методът генерира конфигурации, задоволяващи повечето инсталации, но винаги имате възможността да настройте ръчно файла *sendmail.cf* за работа в по-сложни среди.

Файловете sendmail.cf и sendmail.mc

Макро процесорната програма *m4* генерира файла *sendmail.df*, когато обработва макро конфигурационния файл, предоставен от администратора на локалната система. В останалата част от главата ще се наричаме този файл *sendmail.mc*.

Конфигурационният процес всъщност е въпрос на създаване на подходящ *sendmail.mc* файл, който включва макрос, описващ желаната от вас конфигурация. Макросите са изрази, които са разбираеми за макро процесора *m4* и се разширяват до сложния синтаксис на *sendmail.cf*. Макро изразите се състоят от име на макрос (текстът с главни букви в началото), което може да бъде оприличено на функция в езиците за програмиране, и някои параметри (текстът в скоби), използвани при разширяването. Параметрите могат да се предават

буквално към изхода на *sendmail.cf* или да бъдат използвани за управление на начина, по който се извършва обработката.

Дължината на файла *sendmail.mc* за минимална конфигурация (UUCP или SMTP и цялата не-локална поща, препредавана към директно свързан интелигентен хост) варира между 10 и 15 реда без коментарите.

Два примерни sendmail.mc файла

Ако сте администратор на няколко различни пощенски хоста, може да не искате да наречете вашия конфигурационен файл *sendmail.mc*. Общоприета практика в такива случаи е на файл да се даде име след хоста; в нашия случай-*vstout.m4*. В действителност, името няма значение, тъй като изходният файл се нарича *sendmail.cf*. Задаването на уникално име на конфигурационния файл за всеки хост, ви позволява да пазите всички конфигурационни файлове в една и съща директория, което е удобно за администратора. Нека да разгледаме два примерни макро конфигурационни файла, така че да знаем къде задаваме заглавието.

Днес, повечето конфигурации на *sendmail* използват само протокола SMTP. Конфигурирате на *sendmail* за SMTP е много просто. Пример 18.1 очаква DNS сървър за имена да бъде достъпен за преобразуване на имената на хостове и ще опита да приеме и достави цялата поща за хостовете, използвайки само SMTP.

Пример 18.1: Примерен конфигурационен файл vstout.smtp.m4

```
divert(-1)
#
# Примерен конфигурационен файл за vstout - използва само smtp
#
divert(0)
VERSIONID('@(#)sendmail.mc      8.7 (Linux) 3/5/96')
OSTYPE('linux')
#
# Включва поддръжка на локалния и smtp протоколи за прехвърляне на поща
MAILER('local')
MAILER('smtp')
#
FEATURE(rbl)
FEATURE(access_db)
# край
```

Файлът *sendmail.mc* за **vstout** от Виртуалната пивоварна е показан в Пример 18.2. **vstout** използва протокола SMTP за комуникация с всички хостове от мрежата на Пивоварната и ще видите приликата с общата конфигурация за използване само на SMTP, която преди малко представихме. В допълнение, конфигурацията на **vstout** изпраща цялата поща за други местоназначения към **moria**, който е неговият Интернет хост за препредаване, през UUCP.

Пример 18.2: Примерен конфигурационен файл vstout.uucpsmtp.m4

```
divert(-1)
#
# Примерен конфигурационен файл за vstout
#
divert(0)
VERSIONID('@ (#)sendmail.mc     8.7 (Linux) 3/5/96')
OSTYPE('linux')
dnl
# moria е нашия интелигентен хост, използващ "uucp-new" прехвърляне.
define('SMART_HOST', 'uucp-new:moria')
dnl
# Поддръжка за локалния, smtp и uucp протоколите за прехвърляне на
поща.
MAILER('local')
MAILER('smtp')
MAILER('uucp')
LOCAL_NET_CONFIG
# Това правило гарантира, че цялата локална поща се доставя чрез
# използване на smtp, всичко друго ще преминава през интелигентния
# хост.
R$* < @ $*  .$m.  > $*   $#smtp $@ $2.$m. $: $1 < @ $2.$m. > $3
dnl
#
FEATURE(rbl)
FEATURE(access_db)
# край
```

Ако сравните и съпоставите двете конфигурации, ще ви стане ясно каква е ролята на конфигурационните параметри. Ще обясним тези параметри подробно.

Обичайно използвани параметри sendmail.mc

Някои от елементите във файла *sendmail.mc* са задължителни; други могат да бъдат пропуснати, ако стойността им по подразбиране ви удовлетворява. Последователността на дефинициите във файла *sendmail.mc* е следната:

1. `VERSIONID`
2. `OSTYPE`
3. `DOMAIN`
4. `FEATURE`
5. Локални макро дефиниции
6. `MAILER`
7. `LOCAL_*` множества от правила

В следващите раздели ще разгледаме всяка от тези дефиниции и ще се позоваваме на нашите примери в Пример 18.1 и Пример 18.2, и ще ги обясняваме, когато е необходимо.

Коментари

Редовете във файла *sendmail.mc*, които започват със символа #, не се анализират от *m4*, а по подразбиране се извеждат директно във файла *sendmail.cf*. Това е полезно, ако искате да коментирате какво прави вашата конфигурация едновременно във входния и в изходния файл.

За да разрешите коментарите във вашия *sendmail.mc* файл, които не са поставени в *sendmail.cf*, можете да използвате лексемите `divert` и `dnl`. `divert(-1)` спира извеждането. `divert(0)` възстановява извеждането по подразбиране. Всеки изход, генериран от редовете между тези два реда се игнорира. В нашия пример използваме този механизъм, за да поставим коментари само във файла *sendmail.mc*. За да постигнете същия резултат само за един ред, можете да използвате лексемата `dnl`, която буквално означава "започвайки от началото на следващия ред, изтрий всички символи, включително символа за нов ред". Това също го използвахме в нашия пример.

Повече информация за тези стандартни възможности на *m4* можете да получите от справочната страница на програмата.

VERSIONID и OSTYPE

```
VERSIONID ('@(#)sendmail.mc 8.9 (Linux) 01/10/98')
```

Макросът `VERSIONID` не е задължителен, но е полезен за записване на версията на sendmail конфигурацията във файла *sendmail.cf*. Така че често ще го срещате, а ние ви го препоръчваме. Във всеки случай обаче се уверете, че сте включили:

```
OSTYPE('linux')
```

Това е може би най-важната дефиниция. Макросът `OSTYPE` указва, че трябва да бъде включен файл с дефиниции, които са подходящи подразбиращи се настройки за вашата операционна система. Повечето дефиниции в `OSTYPE` макро файла задават пътищата до различни конфигурационни файлове, потребителска програма за поща и аргументите, и местонахождението на директориите, използвани от sendmail за съхранение на съобщенията. Стандартната версия с изходен код на sendmail включва такъв файл за Linux, който ще бъде включен от предишния пример. Някои дистрибуции на Linux, най-вече дистрибуцията Debian, включват собствения си файл с дефиниции, който е напълно съвместим с Linux-FHS. Когато вашата дистрибуция прави това, вероятно ще използвате нейният файл с дефиниции вместо подразбиращият файл за Linux.

Дефиницията на `OSTYPE` е една от първите дефиниции, които се появяват във вашия файл *sendmail.mc*, тъй като много други дефиниции зависят от нея.

DOMAIN

Макросът `DOMAIN` е полезен, когато искате да конфигурирате голям брой машини в една и съща мрежа по стандартния начин. Ако конфигурирате малък брой хостове, вероятно не си струва да се занимавате с него. Обикновено, конфигурирате елементи като име на пощенски хостове за препредаване или концентратори, които ще се използват от всички хостове във вашата мрежа.

Стандартната инсталация съдържа директория с *m4* макрос шаблони, използвани за управление на конфигурационния процес. Името на тази директория обикновено е */usr/share/sendmail.cf* или нещо подобно. В нея ще намерите поддиректория *domain*, която съдържа специфичните за домейна конфигурационни шаблони. За да можете да използвате макроса `DOMAIN`, трябва да създадете свой собствен макро файл, съдържащ стандартните дефиниции необходими за вашия сайт, и да го запишете в поддиректорията *domain*. Обикновено, но не е задължително, в него трябва да включите само макро дефинициите, които са уникални за вашия домейн.

В дистрибуцията с изходен код на *sendmail* има определен брой примерни макро файлове за домейни, които ще ви помогнат да моделирате свой собствен.

Ако сте записали своя макрос файл за домейн като */usr/share/sendmail.cf/domain/vbrewm4*, трябва да включите дефинициите във вашия файл *sendmail.mc* по следния начин:

```
DOMAIN ('vbrew')
```

FEATURE

Макросът `FEATURE` ви разрешава да включите предварително дефинирани възможности на *sendmail* във вашата конфигурация. Тези възможности правят използването на поддържаните конфигурации много просто. Те са доста голям брой, но в тази глава ще разгледаме само най-важните и най-полезните от тях. В пакета с изходен код ще намерите пълните подробности за възможностите в *CF* файла, включен в пакета с изходния код.

За да използвате която и да е от тези възможности, ще трябва да включите подобен ред във вашия файл *sendmail.mc*:

```
FEATURE (име),
```

където замествате *име* с името на съответната възможност. Някои възможности приемат един незадължителен параметър. Ако искате да използвате нещо друго освен стойността подразбиращата се стойност на този параметър, ще трябва да запис, който изглежда по следния начин:

```
FEATURE (име, параметър),
```

където *параметър* е параметъра, който се подадете.

Локални макро дефиниции

Стандартният макро конфигурационен файл на *sendmail* предоставя много и променливи, с помощта на които имате възможност да зададете специализирани настройки на вашата конфигурация. Те се наричат *локални макро дефиниции*. Много от тях са изброени в *CF* файла, включен в пакета с изходния код на *sendmail*.

Локалните макро дефиниции обикновено се извикват чрез подаване на името на макроса заедно с аргумент, представляващ стойността, която искате да присвоите на променливата, управлявана от макроса. Отново ще изследваме някои от най-общите локални макро дефиниции в примерите, които ще ви представим по-нататък в тази глава.

Дефиниране на протоколи за прехвърляне на поща

Ако искате програмата *sendmail* да прехвърля поща по някакъв друг начин освен локално, трябва да ѝ укажете какви транспортни протоколи да използва. Чрез макроса `MAILER` това е много лесно. Текущата версия на *sendmail* поддържа разнообразни протоколи за прехвърляне на поща; някои от тях са експериментални, други вероятно са рядко използвани.

За да изпращаме и приемаме поща между хостовете в нашата локална мрежа, се нуждаем от протокола SMTP, а за да изпращаме и приемаме поща от нашия интелигентен хост, трябва да използваме протокола UUCP. За тази цел просто включваме `smtp` и `uucp` прехвърляне. `local` е включено по подразбиране, но ако искате, можете да го дефинирате за яснота. Ако сте включили във вашата конфигурация и двете потребителски програми за поща `smtp` и `uucp`, винаги трябва да сте сигурни, че сте дефинирали първо `smtp`.

Най-често използваните прехвърляния, достъпни при използването на макроса `MAILER`, са обяснени в следващия списък:

`local`

Това прехвърляне включва и агента за локално доставяне, използван за изпращане на пощата в пощенските кутии на потребителите на тази машина, и потребителската програма за поща `prog`, използвана за изпращане на съобщения до локалните програми. Това прехвърляне е включено по подразбиране.

`smtp`

Това прехвърляне реализира протокола SMTP (Simple Mail Transport Protocol), което е най-стандартния начин за прехвърляне на поща през Интернет. Когато включите това прехвърляне, се конфигурират четири потребителски програми за поща: `smtp` (базов SMTP), `esmtp` (Extended SMTP - разширен SMTP), `smtp8` (8 битов двоичен чист SMTP) и `relay` (специално проектиран за генериране на шлюз за съобщения между хостове).

`uucp`

`uucp` прехвърлянето предоставя поддръжка на две потребителски програми за поща: `uucp-old`, която е обикновен UUCP, и `uucp-new`, която позволява множество получатели да бъдат обработени при един трансфер.

`usenet`

Това прехвърляне ви позволява да изпращате пощенски съобщения директно към мрежи за новини в стил Usenet. Всяко локално съобщение, насочено към адрес *news.group.usenet*, ще бъде изпратено към групата по интереси *news.group* в мрежата за новини.

`fax`

Ако имате инсталиран софтуера HylaFAX, това прехвърляне ще ви позволи да насочвате e-mail съобщения към него, така че да можете да създадете email-fax шлюз. По време на писането на тази книга, тази възможност още се експериментираше, а повече информация за нея можете да намерите на адрес *http://www.vix.com/hylafax/*.

Съществуват и други полезни, но не толкова често използвани прехвърляния като `pop`, `procmail`, `mail11`, `phquery` и `cyrus`. Ако любопитство ви е раздразнено, можете да прочетете за тях в книгата за *sendmail* или в документацията, доставяна с пакета с изходния код.

Конфигуриране на маршрутизацията на поща за локални хостове

Конфигурирането на Виртуалната пивоварна вероятно е доста по-сложно от необходимото за повечето сайтове. Днес, повечето сайтове биха използвали само прехвърляне посредством SMTP и въобще няма да се занимават с UUCP. В нашата конфигурация конфигурирахме "ингелигенген хост", който се ползва за обработка на цялата изходяща поща. Тъй като в нашата локална мрежа използваме прехвърляне посредством SMTP, трябва да укажем на *sendmail* да не изпраща локална поща през ингелигентния хост. Макросът `LOCAL_NET_CONFIG` ви позволява да вмъквате *sendmail* правила директно в изходния файл *sendmail.cf*, за да промените начина, по който се обработва локалната поща. Ще разгледаме по-подробно правилата за преобразуване малко по-късно, но за момента трябва да приемете, че използваното в нашия пример правило указва, че всяка поща предназначена за хостове от домейна **vbrew.com** ще бъде доставена до тях директно, използвайки транспортния протокол SMTP.

Генериране на файла sendmail.cf

Когато завършите редактирането на вашия конфигурационен файл за *m4*, трябва да го обработите така, че да генерира файла */etc/mail/*

sendmail.cf, четен от *sendmail*. Това е илюстрирано от следващия пример:

```
# cd /etc/mail
# m4 /usr/share/sendmail.cf/m4/cf.m4 vstout.uucpsmtp.mc > sendmail.cf
```

Тази команда извиква макро процесора *m4*, предавайки й за обработка името на два файла с макро дефиниции. *m4* обработва тези файлове в посочения ред. Първият файл е стандартен *sendmail* макро шаблон, който се доставя заедно с пакета с изходния код на *sendmail*. Вторият, разбира се, съдържа нашите собствени макро дефиниции. Резултатът от командата се насочва към файла */etc/mail/sendmail.cf*, който всъщност е и нашата цел.

Вече можете да стартирате *sendmail* с новата конфигурация.

Интерпретиране и писане на правила за преобразуване

Може да се поспори, че най-мощната възможност на *sendmail* са правилата за преобразуване. Тези правила се използват от *sendmail* за определяне на начина, по който се обработва получено пощенско съобщение. *sendmail* предава адреса от заглавието на съобщението посредством набори от правила за преобразуване, наречени *rulesets*. Тези правила за преобразуване трансформират пощенския адрес от една в друга форма и можете да мислите за тях като подобни на команда във вашия редактор, който заменя целия текст, който съвпада с определен шаблон, с друг текст.

Всяко правило има лява и дясна страна, разделени от поне един символ за табулация. Когато *sendmail* обработва пощата, тя сканира правилата за преобразуване, търсейки съвпадение в лявата страна. Ако адрес съвпада с лявата страна на правилото за преобразуване, той се заменя със стойността на дясната страна и се обработва отново.

R и S команди на sendmail.cf

Във файла *sendmail.cf*, наборите от правила са дефинирани чрез използване на команди, кодирани като `Sn`, където `n` определя текущия набор от правила.

Самите правила се появяват в команди, кодирани като *R*. Когато всяка *R* команда се прочете, тя се добавя към текущия набор от правила.

Ако работите само с файла *sendmail.mc*, не е необходимо да се притеснявате за *S* командите въобще, тъй като макросите ще ги създадат вместо вас. Вашите *R* правила обаче ще трябва кодирате ръчно.

Следователно, набор от правила на *sendmail* ще изглежда по следния начин:

```
sn
Rlhs rhs
Rlhs2 rhs2
```

Някои полезни макро дефиниции

sendmail използва вътрешно определен брой стандартни макро дефиниции. Най-полезните от тях за писане на наборите от правила са:

$j Пълното квалифицирано име на този хост (FQDN).

$w Компонентът име на хост на FQDN.

$m Компонентът име на домейн на FQDN.

Можем да обединим тези макро дефиниции в нашите правила за преобразуване. Нашата конфигурация на Виртуалната пивоварна използва макроса `$m`.

Лявата страна

В лявата страна на правилото за преобразуване, задавате шаблон, който ще съвпадне с адрес, който искате да преобразувате. Повечето символи съвпадат буквално, но съществуват и определен брай специални символи, които имат специално значение; те са описани в следващия списък. Правилата за преобразуване за лявата страна са:

`$@` Съвпада точно с нула лексеми

`$*` Съвпада с нула или повече лексеми

`$+` Съвпада с една или повече лексеми

`$-` Съвпада с точно една лексема

`$=`*x*
: Съвпада с всяка фраза от клас *x*

`$~`*x*
: Съвпада с всяка дума, която не е от клас *x*

Лексемата е низ от символи, ограничени от празни пространства. Не съществува начин за включване на празни пространства в лексема, нито пък е необходимо, тъй като шаблоните в израз са достатъчно гъвкави. Когато правило съвпада с адрес, текстът съвпаднал с всеки от шаблоните в израза ще бъде присвоен на специални променливи, които ще използваме в дясната страна. Единственото изключение е правилото `$@`, което не съвпада с лексеми и следователно никога няма да генерира текст за използване от дясната страна.

Дясната страна

Когато лявата страна на правило за преобразуване съвпада с адрес, изходният текст се изтрива и заменя с дясната страна на правилото. Всички лексеми в дясната страна се копират буквално, освен ако не започват със знака за долар. Точно както за лявата страна, определен брой метасимволи могат да бъдат използвани и от дясната страна; те са описани в следващия списък. Правилата за преобразуване за дясната страна са:

`$n` Този метасимвол се замества с *`n`*-я израз от лявата страна.

`$[име$]`

Този метасимвол преобразува име на хост в канонично име. Замества се с каноничната форма на името на хоста.

`$(map key $@arguments $:default $)`

Това е по-обща форма за търсене. Изходът е резултата от търсенето на ключ *`key`* в картата *map,* предавайки *`arguments`* като аргументи. Картата може да бъде всяка от картите, поддържани от *sendmail*, например `virtusertable`, която ще опишем малко по-нататък. Ако търсенето е неуспешно, резултатът е *`default`*. Ако не е ориложен резултат по подразбиране и търсенето е неуспешно, входа не се променят, а резултатът е *`key`*.

`$>n`

При използване на това правило останалата част от този ред ще бъде анализирана и след това ще се предаде към набора от правила *`n`* за изчисление. Изходните данни от извикания набор ще бъдат записани като изходни данни на това правило. Това е механизмът, който позволява на правила да извикват други набори от правила.

$# потребителска програма за поща

Този метасимвол спира изчисляването на набора от правила и задава потребителската програма за поща, която ще се използва за прехвърляне на това съобщение в следващата стъпка от доставянето му. Този метасимвол трябва да бъде извикан само от набора от правила 0 или от някой от подпрограмите му. Това е последният етап от анализа на адреса и ще бъде придружено от следващите два метасимвола.

$@хост

Този метасимвол задава хоста, към който ще се препрати това съобщение. Ако хоста на местоназначението е локалния хост, може да се пропусне. Хостът *host* може да бъде разделен с двоеточия списък на хостове на местоназначението, който ще бъде изпълнен последователно за доставяне на съобщението.

$:потребител

Този метасимвол указва целевия потребителя *user* за пощенското съобщение.

Правило за преобразуване, за което е открито съвпадение, се повтаря докато има съвпадение, след това анализирането продължава със следващото правило. Това поведение може да се промени, ако дясната страна се предхожда от един или два от метасимволите за дясна страна, описани в следващият списък. Правилата за преобразуване за метасимволите за контрол на цикъла на дясната страна са:

`$@` Този метасимвол указва на набора от правила да върне като стойност останалата част дясната страна. Не се изчисляват никакви други правила от набора от правила

`$:` Този метасимвол указва на правилото да спре веднага, но всички останали правила от текущия набор се изпълняват.

Пример за просто правило за шаблон

За да видите по-добре как работят шаблоните за заместване чрез макрос, разгледайте следващото правило за лявата страна:

```
$* < $+ >
```

Това правило съвпада с "Нула или повече лексеми, следвани от символа <, следван от една или повече лексеми, следвани от символа >."

Ако това правило беше приложено върху `brewer@vbrew.corn` или `Head Brewer < >`, нямаше да открие съвпадение. Първият низ няма

да съвпадне, тъй като не включва символ <, а вторият няма съвпадне, тъй като $+ съвпада с *една или повече* лексеми, а между символите <> няма лексеми. Във всеки случай, при който няма съвпадение на правило, дясната страна на това правило не се използва.

Ако правилото беше приложено върху `Head Brewer < brewer@vbrew.com >`, то щеше да съвпадне, а в дясната страна `$1` щеше да се замести с `Head Brewer`, `$2` - с `brewer@vbrew.com`.

Ако правилото беше приложено върху `< brewer@vbrew.com >`, то щеше да съвпадне, тъй като `$*` съвпада с *нула* или повече лексеми, а в дясната страна `$1` щеше да се замести с празен низ.

Семантики на наборите от правила

Всеки набор от правила на *sendmail* се извиква, за да изпълни различна задача при обработката на пощата. Когато пишете правила, е важно да се разбере какво се очаква да направи от всеки набор от правила. Ще разгледаме всеки набор от правила, които конфигурационните скриптове на *m4* ни позволяват да променяме:

LOCAL_RULE_3

Наборът от правила 3 е отговорен за преобразуването на адреса от произволен формат в стандартен формат, който след това *sendmail* ще обработи. Очакваният изходен формат е нещо подобно на *`локална-част@хост-домейн-спецификация`*.

Наборът от правила 3 ще постави частта с името на хоста на преобразувания адрес между символите < и >, за да направи полесно анализирането от следващи набори от правила. Наборът от правила 3 се прилага преди *sendmail* да извърши друга обработка на e-mail адреса, така че ако искате *sendmail* да създаде пощенски шлюз от система, която използва някакъв необичаен формат за адреси, трябва да добавите правило, използващо макроса `LOCAL_RULE_3`, за да преобразувате адресите в стандартен формат.

LOCAL_RULE_0 и LOCAL_NET_CONFIG

Наборът от правила 0 се прилага върху адресите на получателите от *sendmail* след Набор от правила 3. Макросът `LOCAL_NET_CONFIG` вмъква правила в *долната половина* на Набора от правила 0.

От Набора от правила 0 се очаква да извърши доставянето на съобщението до получателя, така че то трябва да се преобразува до тройката стойности, задаваща потребителската програма за поща, хоста и потребителя. Правилата ще бъдат поставени преди всяка дефиниция на интелигентен хост, която можете да включите, така че ако добавите правила, преобразуващи адресите по подходящ начин, всеки адрес, който съвпада с правило, няма да се обработва от ингелигентния хост. Това всъщност е начина, по който управляваме непосредственото *smtp* прехвърляне за потребители в нашата локална мрежа от нашия пример LAN.

LOCAL_RULE_1 и LOCAL_RULE_2

Наборът от правила 1 се прилага към адресите на всички изпращачи на съобщения, а Наборът от правила 2 се прилага към адресите на всички получатели. И двата набора обикновено са празни.

Интерпретиране на правилото в нашия пример

В Пример 18.3 е използван макроса `LOCAL_NET_CONFIG` за деклариране на локално правило, което гарантира, че всяка поща в рамките на нашия домейн се доставя директно чрез използване на потребителската програма за поща *smtp*. Сега, след като видяхме как се създават правилата за преобразуване, ще можем да разберем и начина, по който работят.

Пример 18.3: Правило за преобразуване от vstout.uucpsmtp.m4

```
LOCAL_NET_CONFIG
# Това правило гарантира, че цялата локална поща се доставя чрез
# използване на smtp прехвърляне, всичко останало преминава през
# интелигентния хост.
R$* < @ $* .$m. > $*      $#smtp $@ $2.$m. $: $1 < @ $2.$m. > $3
```

Знаем, че макросът `LOCAL_NET_CONFIG` ще вмъкне правилото някъде близо до края на набор от правила 0, но преди дефиницията на интелигентния хост. Освен това, знаем, че наборът от правила 0 е последния набор от правила, който ще бъде изпълнен и че е разделен на три части, задаващи потребителската програма за поща, потребителя и хоста.

Можем да игнорираме двата реда коментар; те не правят нищо полезно. Самото правило е реда, започващ с `R`. Знаем, че `R` е *sendmail* команда, която добавя това правило към текущия набор от правила, в този случай към набор от правила `0`. Нека да погледнем лявата и дясната страна на това правило.

Лявата страна изглежда по следния начин: `$* < @ $* .$m. > $*`.

Наборът от правила 0 очаква символите < и >, тъй като те са подготвени от набор от правила 3. Наборът от правила 3 преобразува адреса в стандартен формат и поставя частта от адреса с хоста между символите < и >, за да направи анализирането по-лесно.

Това правило съвпада с всеки пощенски адрес, който изглежда по следния начин: '`получател < @ хост.нашия_домейн. > текст`', т.е. правилото съвпада с пощата за всеки потребител от който и да е хост в рамките на нашия домейн.

Ще запомните, че текстът, съвпаднал с метасимволите от лявата страна на правилото за преобразуване, се присвоява на макро дефиниции за използване от дясната страна. В нашия пример, първото `$*` съвпада с целия текст от началото на адреса до символа <. Целият този текст се присвоява на `$1`, за да се използва от дясната страна. По подобен начин, второто `$*` в нашето правило за преобразуване се присвоява на `$2`, а последното – на `$3`.

Сега вече имаме достатъчно знания, за да разберем лявата страна. Това правило съвпада с пощата за всеки потребител от всеки хост в рамките на нашия домейн. То присвоява името на потребителя на `$1`, името на хоста на `$2` и всеки следващ текст на `$3`. След това се извиква дясната страна, за да се обработи всичко това.

Сега нека сега да видим какво можем да очакваме за изходни данни. Дясната страна на нашия пример изглежда по следния начин: `$# smtp $@ $2.$m. $: $1 < @ $2.$m. > $3`.

Когато дясната страна на нашия набор от правила е обработена, всеки от метасимволите се интерпретира и се прави съответното заместване.

Метасимволът `$#` преобразува правилото към специфична потребителска програма за поща, в нашия случай това е *smtp*.

Метасимволът `$@` разпознава целевия хост. В нашия пример, този хост е зададен като `$2. $m.`, което е пълното квалифицирано име на домейна на хост от нашия домейн. FQDN се изгражда от компонента име на хост, присвоен на `$2` от лявата страна като се добави и името на нашия домейн (`.$m.`).

Метасимволът `$:` задава целевия потребител, който отново сме прехванали от лявата страна и сме съхранили в `$1`.

Запазваме съдържанието на секцията <> и всеки текст, следващ след нея, използвайки данните, които сме събрали от лявата страна на правилото.

Тъй като правилото се преобразува към потребителска програма за поща, съобщението се препраща към тази програма, за да бъде доставено. В нашия пример, съобщението ще бъде препратено към хоста на местоназначението чрез протокола SMTP.

Конфигуриране на опциите на sendmail

Програмата *sendmail* има голям брой опции, които ви позволяват да настройвате начина, по който тя изпълнява определени задачи. В следващия списък ще изброим само най-често използваните от тях.

За да конфигурирате всяка от тези опции, можете или да я дефинирате в конфигурационния файл на *m4*, което е за предпочитане, или да я вмъкнете директно във файла *sendmail.cf*. Например, ако искаме ново разклонение на *sendmail* за нова задача за доставяне на всяко пощенско съобщение, трябва да добавим следващия ред в нашия *m4* конфигурационен файл:

```
define ('confSEPARATE_PROC', 'true')
```

Съответният създаден запис във файла *sendmail*.cf ще бъде:

```
O   ForkEachJob=true
```

Следващият списък описва общите опции на *sendmail* за *m4* (и техните еквиваленти за *sendmail.cf*):

`confMIN_FREE_BLOCKS (MinFreeBlocks)`

Съществуват случаи, при които някакъв проблем възпрепятства незабавната доставка на пощенските съобщения. Тогава съобщенията се подреждат в опашка в пощенския спулер (spooler). Ако вашият пощенски хост обработва големи обеми от поща, възможно е спулерът да се увеличи до такъв размер, че да запълни файловата система, която го поддържа. За да предотврати това, чрез тази опция *sendmail* задава минималния брой на свободните блокове на диска, които трябва да съществуват преди да се приеме пощенско съобщение. Това ви позволява да гарантирате, че *sendmail* никога няма да позволи файловата система на спулера да се запълни (стойността по подразбиране е: 100).

`confME_TOO (MeToo)`

Когато целта на пощата, като e-mail псевдоними, е разширена, понякога е възможно изпращащият съобщението да се появи в списъка с получателите. Тази опция определя дали този, който генерира e-mail съобщение, ще получи копие от него, ако присъства в разширения списък с получатели. Валидни стойности са "true" и "false" (стойността по подразбиране е: false).

`confMAX_DAEMON_CHILDREN (MaxDaemonChildren)`

Винаги, когато *sendmail* свърза получава SMTP връзка от отдалечен хост, тя стартира свое ново копие за обработка на входящото пощенско съобщение. По този начин е възможно *sendmail* да обработва едновременно множество входящи пощенски съобщения. Въпреки че това е полезно, всяко ново копие на *sendmail* заема памет в хоста. Ако се получат твърде голям брой входящи връзки, е възможно *sendmail* демоните да заемат цялата системна памет. Тази опция ви позволява да ограничите максималния брой деца на демона, които ще бъдат стартирани. Когато се достигне зададения брой, новите връзки се отхвърлят, докато не се прекрати работата на някое от съществуващите деца (не е дефинирана стойност по подразбиране).

`confSEPARATE_PROC (ForkEachJob)`

При обработването на опашка от писма и изпращането на пощенски съобщения, *sendmail* обработва съобщенията едно след друго. Когато тази опция е разрешена, *sendmail* ще стартира свое копие паралелно за всяко съобщение, което ще бъде доставено (стойност по подразбиране е: false)

`confSMTP_LOGIN_MSG (SmtpGreetingMessage)`

Винаги, когато се осъществи връзка със *sendmail*, се изпраща приветстващо съобщение. По подразбиране, това съобщение съдържа името на хоста, името потребителската програма за прехвърляне на поща, номера на версията на *sendmail*, номера на локалната версия и текущата дата. RFC-821 указва, че първата дума от приветстващото съобщение трябва да бъде пълното квалифицирано име на домейна, в който се намира хоста, но останалата част от него може да бъде конфигурирана по ваше желание. Тук можете да зададете макроси на *sendmail*, които ще бъдат разширени при използването им. Единствените хора, които ще видят това съобщение, са нещастни системни администратори, диагностициращи проблемите, свързани с доставянето на поща,

или много любопитни потребители, които се интересуват от начина, по който е конфигурирана вашата машина. Можете да улесните някои от техните скучни задачи като внесете малко духовитост в приветстващото съобщение; бъдете добри. Думата "ESMTP" ще бъде вмъкната от *sendmail* между първата и втората дума като сигнал до отдалечените хостове, че поддържаме протокола ESMTP (стойността по подразбиране е: `$j Sendmail $v/$Z; $b`).

Някои полезни конфигурации на sendmail

Съществуват огромен брой конфигурации на *sendmail*. Тук ще илюстрираме само няколко важни типове конфигурации, които ще ви бъдат полезни при много от инсталациите на *sendmail*.

Доверяване на потребителите да задават полето From:

Понякога е полезно да пренаписвате полето `From:` на изходящо пощенско съобщение. Нека да приемем, че имате web-базирана програма, която генерира електронна поща. Обикновено, пощенското съобщението ще идва от потребителят, който притежава процеса на web-сървъра. Бихме могли да искаме да зададем друг адрес на източника, така че пощата да идва от някой друг потребител или адрес на тази машина. *sendmail* предоставя начин за указване на потребителите на системата, на които може да се разреши да правят това.

Възможността `use_ct_file` разрешава спецификацията и използването на файл, съдържащ списък с имената на доверените потребители (trusted users). По подразбиране, *sendmail* се доверява на малък брой потребители на системата (например `root`). Подразбиращото се име на файла за тази възможност е */etc/mail/trusted-users* в системи, използващи конфигурационна директория */etc/mail*, и */etc/sendmail.ct* в другите системи. Можете да определите името и местонахождението на този файл като отмените дефиницията `confCT_FILE`.

За да разрешите тази възможност, във вашия *sendmail.mc* добавете FEATURE(use_ct_file).

Управление на пощенски псевдоними

Пощенските псевдоними са мощна възможност, която разрешава пощата да бъде насочена към пощенски кутии, които са алтернативни имена на потребители или процеси на хоста на местоназначението. Например, обичайна практика е обратна връзка или коментари, отнасящи се за World Wide Web сървър, да се насочват към "webmaster". Често обаче не съществува потребител на целевата машина, известен като "webmaster", а това е псевдоним на друг потребител на системата. Друга честа употреба на пощенските псевдоними е използването им от програми на сървъра за пощенски списъци, в които псевдонимът насочва входящите съобщения към програмата на сървъра.

Псевдонимите се съхраняват във файла */etc/aliases*. Програмата *sendmail* се допитва до този файл, когато определя начина за обработка на входящо пощенско съобщение. Ако тя открие елемент в този файл, съвпадащ с целевия потребител в пощенското съобщение, програмата пренасочва съобщението там, където описва елемента.

Определено, псевдонимите позволяват следните три неща:

- Предоставят съкратено или добре познато име за адресиране на пощата към един или повече потребители.
- Те могат да извикват програма с пощенско съобщение като входни данни за програмата.
- Псевдонимите могат да изпращат поща към файл.

За да бъдат съвместими с RFC, всички системи изискват използването на псевдоними за **Postmaster** и **MAILER-DAEMON**.

Винаги бъдете изключително внимателни, когато дефинирате псевдоними, които извикват програми или пишат в програми, тъй като *sendmail* обикновено работи с **root** права за достъп.

Подробности, свързани с пощенските псевдоними, бихте могли да намерите в справочната страница *aliases(5)*. Примерен файл *aliases* е показан в Пример 18.4.

Пример 18.4: Примерен файл aliases

```
#
# За да получите съвместимост с RFC, използвайте следващите два
# псевдонима.
#
# Важно е да ги преобразувате към 'човек', който редовно чете пощата си.
#
postmaster:     root                    # задължителен елемент
MAILER-DAEMON:  postmaster              # задължителен елемент
#
#
# демонстрира обичайните типове псевдоними
#
usenet:         janet                   # псевдоним за един човек
admin:          joe, janet              # псевдоним за няколко човека
newspak-users:  :include: /usr/lib/
                lists/newspak           # чете получателите от файл
changefeed:     | /usr/local/lib/gup    # псевдоним, който извиква
                                        # програма
complaints:     /var/log/complaints     # псевдонимът пише пощата към
                                        # файл
#
```

Винаги, когато обновявате файла */et/aliases*, изпълнявайте командата:

```
# /usr/bin/newaliases
```

за да изградите отново базата данни, която *sendmail* използва вътрешно. Командата */usr/bin/newaliases* е символна връзка към *sendmail* и когато е извикана по този начин, прави точно същото както, ако я извикате като:

```
# /usr/lib/sendmail -bi
```

Командата *newaliases* е алтернативния и по-удобен начин, за да направите това.

Използване на интелигентен хост

Понякога хост открива поща, която не може да достави директно до желания отдалечен хост. Често пъти е по-удобно да имате един единствен хост в мрежата, който да управлява предаването на поща до отдалечените хостове, които трудно могат да бъдат достигнати, отколкото всеки локален хост да се опитва да се опитва да прави това независимо.

Съществуват няколко основателни причини да имате само един единствен хост, ангажиран с управлението на пощата. Можете да опростите управлението, като използвате само един хост с обширна

конфигурация на пощата, който знае как да обработва всечки различен тип прехвърляне на поща, например, UUCP, Usenet и т.н. Всички други хостове се нуждаят само от един единствен транспортен протокол за изпращане на пощата до този централен хост. Хостовете, изпълняващи тази централна роля за маршрутизиране и препращане на поща, се наричат *"интелигентни хостове"* (*smart hosts*). Ако имате интелигенген хост, който ще приема поща от вас, можете да му изпратите поща от какъвто и да е вид и той ще се справи с управлението и предаването ѝ до желаното от вас отдалечено местоназначение.

Друго добро приложение на конфигурациите на интелигенген хост е управлението на предаването на поща през частна защитна стена. Някоя организация може да избере да инсталира частна IP мрежа и да използва свои собствени нерегистрирани IP адреси. Тази мрежа може да бъде свързана към Интернет посредством защитна стена. Не можете обаче да използвате SMTP за изпращане на поща към и от хостовете, намиращи се в частната мрежа, към външния свят, тъй като хостовете не могат да приемат или да създават директни мрежови връзки към хостове в Интернет. Вместо това, организацията може да избере защитната стена да изпълнява функцията на интелигентен хост. Интелигентният хост, работещ на защитната стена може да създаде директни мрежови връзки с хостове от частната мрежа и от Интернет. Интелигентния хост ще приема поща и от хостовете от частната мрежа, и от Интернет хостове, ще я съхранява в локално хранилище и след това ще управлява препредаването на тази поща директно към правилния хост.

Интелигентните хостове обикновено се използват тогава, когато всички други методи за доставяне на поща са неуспешни. В случая с организацията с частната мрежа, би било логично всеки хост да се опитва първо да достави пощата директно и ако се не успее, да я изпраща към интелигентния хост. Това освобождава доста интелигентния хост от трафика, тъй като другите хостове в частната мрежа могат директо да изпращат пощата един на друг.

sendmail предоставя прост метод за конфигуриране на интелигенген хост, използвайки възможността `SMART_HOST`; при реализирането му в конфигурацията на Виртуалната пивоварна направихме точно това. Съответните части от нашата конфигурация, които дефинират интелигентния хост са:

```
define ('SMART_HOST', 'uucp-new:moria')
LOCAL_NET_CONFIG
# Това правило гарантира, че цялата локална поща се доставя чрез
# използването на smtp прехвърляне, всичко останало ще преминава през
# интелигентния хост.
R$* < @ $* .$m. > $* $#smtp $@ $2.$m. $: $1 < @ $2.$m. > $3
```

Макросът `SMART_HOST` ви позволява да укажете хоста, който ще препраща цялата изходяща поща, която не можете да доставите директно, и транспортен протокол, който ще комуникира с него.

В нашата конфигурация използваме `uucp-new` прехвърляне към UUCP хоста **moria**. Ако искахме да конфигурираме *sendmail* така, че да използва SMTP-базиран интелигентен хост, вместо това трябваше да използваме нещо подобно на:

```
define ('SMART_HOST', 'mail.isp.net')
```

Не е необходимо да зададете SMTP като транспортен протокол, тъй като той е зададен по подразбиране.

Можете ли да отгатнете какво могат да правят макросът `LOCAL_NET_CONFIG` и правилото за преобразуване?

Макросът `LOCAL_NET_CONFIG` ви позволява да добавяте правила за преобразуване на *sendmail* към вашата конфигурация, която определя коя поща трябва да остане в локалната пощенска система. В нашия пример, използвахме правило, което съвпада с всеки пощенския e-mail адрес, където хостът принадлежи към нашия домейн (`.$m.`) и го преобразува така, че го изпраща директно към `SMTP` потребителската програма за поща. Това гарантира, че всяко съобщение за хост от нашия локален домейн е насочено директно към `SMTP` потребителската програма за поща и оттам е препратено към този хост, а не минава през нашия интелигентен хост, което е определено по подразбиране.

Управление на нежелана или непоръчана комерсиална поща (SPAM)

Ако сте се абонирали за някакъв пощенски списък, публикували сте своя e-mail адрес в web-сайт или сте изпратили статия в UseNet, вероятно сте започнали да получавате непоръчана електронна поща. Вече е банално за хората да обхождат мрежата в търсене на e-mail адреси, за да ги добавят към пощенски списъци, които след това

продават на компании, желаещи да рекламират продуктите си. Това масово изпращане на поща се нарича спаминг (spamming).

Речникът Free ON-Line Dictionary of Computing предлага следната специфична за пощата дефиниция за "spam":

1. Можете да намерите Free ON-Line Dictionary of Computing пакетиран много дистрибуции на Linux или online в главната му страница на адрес *http://wombat.doc.ic.ac.uk/foldoc/*.

2. (стесняване на смисъла на 1) Безразборно изпращане на големи количества от нежелана електронна поща с цел промоция на продукт или услуга. В този смисъл "spam" е още подобно на електронния еквивалент на непотребната поща, изпращана на "собственика".

   През 90-те с нарастване на комерсиалното значение на мрежата, съществуват "бизнесмени", предлагащи спаминг като "услуга" на компании, желаещи да рекламират по мрежата. Те правят това като изпращат поща до колекции от e-mail адреси, Usenet новини или пощенски списъци. Много потребители на мрежата реагират срещу поведение и замесените "особи".

За щастие, *sendmail* включва поддръжка на механизми, които могат да ви помогнат да се справите с нежеланата поща.

Списък с черни дупки в реално време

Списъкът с черни дупки в релно време е публично средство, предоставено за намаляване на обема на нежеланата реклама, с която трябва да се борите. Известни източници на електронна поща и хостове са изброени в списък в база данни, към която можете да отправите запитване през Интернет. Те са въведени от хората, които са получили нежелана рекламна поща. Понякога в този списък попадат главни домейни, поради тъй като ако пропуснат да изключат свой потребител пращащ "спам". Независимо, че много хора се оплакват от отпределени предпочитания, направени от тези, които поддържат списъка, той остава много популярен, а несъгласията обикновено се изглаждат. Пълни подробности за начина, по който работи тази услуга, можете да намерите в главния сайт на Mail Abuse Protection System на адрес: *http://maps.vix.com/rbl/*.

Ако разрешите тази възможност на *sendmail*, тя ще тества изходния адрес на всяко входящо пощенско съобщение спрямо списъка с чени дупки в реално време (Real-time Blackhole List), за да определи дали да приеме съобщението. Ако използвате голям сайт с много потребители, тази възможност може да ви спести значително количество

дисково пространство. Тя приема параметър за задаване на името на сървъра, който ще се използва. По подразбиране, това е името на главния сървър в **rbl.maps.vix.com**.

За да конфигурирате възможността Real-time Blackhole List, добавете следната макродекларация във вашия файл *sendmail.mc*:

```
FEATURE (rbl)
```

За да укажете някакъв друг RBL сървър, ще трябва да използвате декларация, подобна на следната:

```
FEATURE (rbl, 'rbl.host.net')
```

База данни за достъп

Алтернативна система, която предлага по-голяма гъвкавост и контрол с цената на ръчна конфигурация е възможността `access_db` на *sendmail*. Базата данни за достъп ви позволява да конфигурирате кои хостове или потребители ще получат поща и на кои ще препредавате поща.

Управлението на това кой ще препраща поща е много важно, тъй като това е друга често използвана от спаминг хостове техника за надхитряване на системите, както Real-time Blackhole List описа. Вместо да ви изпращат поща директно, спамерите ще препредават пощата през някакъв друг нищо неподозиращ хост, който позволи това. Тогава входящата SMTP връзка няма да идва от познатия спаминг хост, а от хост, препредаващ пощата. За да се уверите, че вашите собствени хостове не се използват по този начин, трябва да препредавате поща само от познати хостове. По-новите версии на *sendmail* от 8.9.0 насам забраняват препредаването по подразбиране, така че ще трябва да използвате базата данни за достъп, за да разрешите на отделни хостове да препредават.

Основната идея е проста. Когато се получи нова входяща SMTP връзка, *sendmail* извлича информацията от заглавието на съобщението, а след това се допитва до базата данни за достъп дали да продължи да приема и тялото на самото съобщение.

Базата данни за достъп е съвкупност от правила, описващи какво действие трябва да се предприеме за съобщения, получени от определените хостове. По подразбиране, файлът за контрол на достъпа е наречен */etc/mai/access*. Таблицата има прост формат. Всеки ред съдържа правило за достъп. Лявата страна на всяко правило е шаблон, използван за съвпадение с изпращача на входящото пощенско съоб-

щение. Това може да бъде пълен e-mail адрес, име на хост или IP адрес. Дясната част е действието, което трябва да се предприеме. Съществуват пет типа действия, които можете да конфигурирате. Те са:

OK

Приема съобщението.

RELAY

Приема съобщения от този хост или потребител, дори ако те не са предназначени за нашия хост; т.е. приема и съобщения, които ще се препредават от този хост към други хостове.

REJECT

Отхвърля съобщението със стандартно съобщение.

DISCARD

Игнорира съобщението, използвайки потребителската програма за поща `$#discard`.

някакъв текст

Връща съобщение за грешка, използвайки `###` като код на грешката (трябва да има съвместимост с RFC-821), и "някакъв текст" като съобщение.

Примерен файл */etc/mail/access* би изглеждал по следния начин:

```
friends@cybermail.com REJECT
aol.com               REJECT
207.46.131.30         REJECT
postmaster@aol.com    OK
linux.org.au          RELAY
```

Този пример би отхвърлил всяка поща, получена от *friends@cybermail.com*, всеки хост от домейна **aol.com** и хоста **207.46.131.30**. Следващото правило ще приеме поща от *postmaster@aol.com*, независимо от факта, че самият домейн има правило за отхвърляне. Последното правило позволява препредаването на поща от всеки хост, намиращ се в домейна **linux.org.au**.

За да разрешите възможността access_db, използвайте следващата декларация във вашия файл *sendmail.mc*:

```
FEATURE (access_db)
```

Дефиницията по подразбиране изгражда базата данни, използвайки `hash -o /etc/mai/access`, което генерира проста хеширана база данни от обикновения текстов файл. Това е напълно достатъчно за повечето инсталации. Съществуват и други опции, които трябва да

имате предвид, ако възнамерявате да използвате на голяма база данни. Подробности можете да използвате книгата за *sendmail* или друга *sendmail* документация.

Ограничаване на потребители от приемане на поща

Ако имате потребители или автоматизирани процеси, които изпращат поща, но никога няма да има нужда да приемат такава, понякога е полезно да откажете да получавате поща, предназначена за тях. Това спестява дисково пространство, съхранявайки поща, която никога няма да се прочете. Възможността `blacklist_recipients`, използвана в комбинация с `access_db`, ви позволява да забраните получаването на поща за локални потребители.

За да разрешите тази възможност, добавете следващите редове във вашия *sendmail.mc* файл, ако вече не са добавени:

```
FEATURE (access_db)
FEATURE (blacklist_recipients)
```

За да забраните получаването на поща за локален потребител, просто добавете неговите данни към базата данни за достъп. За това обикновено се използва елемент от типа ###, което ще върне на изпращача на съобщението някакво смислено обяснение за грешка, така че да знае защо пощата не е доставена. Тази възможност се прилага еднакво добре за потребители във виртуални e-mail домейни, просто трябва да включите домейна в спецификацията на базата данни за достъп. Някои примерни елементи във файла */etc/mail/access* може да изглеждат по следния начин:

```
daemon          550 Демона не приема или чете поща.
flacco          550 Пощата за този потребител е забранена
                    административно.
grump@dairy.org 550 Пощата за този получател е забранена.
```

Конфигуриране на виртуален e-mail хостинг

Виртуалният e-mail хостинг предоставя хост с възможност да прима и доставя поща от името на голям брой различни домейни, все едно, че съществуват много отделни пощенски хостове. Най-често, виртуалният хостинг се използва от доставчиците на Интернет услуги в комбинация с виртуален web-хостинг, но тъй като конфигурирането е просто, никога няма да знаете кога ще ви се наложи да предоставите виртуален хостинг за пощенски списък за любимия ви Linux проект, така че тук ще го опишем.

Приемане на поща за други домейни

Когато *sendmail* получи съобщение, тя сравнява хоста на местоназначението от заглавието с името на локалния хост. Ако те съвпаднат, *sendmail* приема съобщението за локално доставяне; ако се различават, *sendmail* може да реши да приеме съобщението и да се опита да го препрати към крайното му местоназначение (за подробности за начина, по който *sendmail* се конфигурира за приемане на поща за препращане вижте "База данни за достъп" по-горе в тази глава).

Ако искаме да конфигурираме виртуален e-mail хостинг, първо трябва да убедим *sendmail*, че трябва да приема поща и за домейните, за които ще направим това. За щастие, да направим това е много просто.

Възможността на *sendmail use_cw_file* ни позволява да зададем името на файл, в който ще съхраняваме имена на домейни, за които *sendmail* приема поща. За да конфигурирате това свойство, добавете декларацията на тази възможност във вашия *sendmail.mc* файл:

```
FEATURE (use_cw_file)
```

Подразбиращото се име на този файл ще бъде */etc/mail/local-host-names* за дистрибуции, използващи конфигурационна директория */etc/mail* или */etc/sendmail.cw* за останалите. Освен това, можете да зададете името и местонахождението на този файл, отменяйки макроса `confCW_FILE` използвайки различни варианти като:

```
define ('confCW_FILE', '/etc/virtualnames')
```

За да се придържаме към подразбиращото се име, ако искахме да предложим виртуален хостинг на домейните **bovine.net**, **dairy.org** и **artist.org**, би трябвало да създадем файл */etc/mail/local-host-names*, който изглежда по следния начин:

```
bovine.net
dairy.org
artist.org
```

Когато това е направено, и допуснем, че съществуват подходящите DNS записи, насочващи имената на тези домейни към нашия хост, *sendmail* ще приема пощенските съобщения за тези домейни все едно, че са предназначени за реални имена на домейни.

Препредаване на поща, за която е предоставен виртуален хостинг, към други местоназначения

Възможността на *sendmail* `virtusertable` конфигурира поддръжка за таблицата с виртуални потребители, където конфигурираме виртуалния e-mail хостинг. Тази таблица всъщност насочва входящата поща, предназначена за някакъв потребител *потребител@хост*, към *друг_потребител@друг_хост*. Можете да мислите за нея като за разширена възможност за пощенски псевдоним, който работи, използвайки не само крайния потребител, но и домейна на местоназначението.

За да конфигурирате възможността `virtusertable`, я добавете във вашата *sendmail.mc* конфигурация, както е показано:

```
FEATURE (virtusertable)
```

По подразбиране, файлът, съдържащ правилата за преобразуване, ще бъде */etc/mail/virtusertable*. Можете да огмените това, като подадете аргумент към макро дефиницията; за да разберете какви опции съществуват, разгледайте подробния справочник за *sendmail*.

Форматът на таблицата с виртуални потребители е много прост. Лявата страна на всеки ред съдържа шаблон, представляващ изходното местоназначение на пощенския адрес; дясната страна съдържа шаблон, представляващ адреса, за който е предоставен виртуален хостинг, към който ще бъде насочена пощата.

Следващия пример показва три възможни типа елемента:

```
samiam@bovine.net       colin
sunny@bovine.net        darkhorse@mystery.net
@diary.org              mail@jhm.org
@artist.org             $1@red.firefly.com
```

В този пример предоставяме виртуален хостинг на три домейна: **bovine.net**, **dairy.org** и **artist.org**.

Първият елемент в пренасочва поща, изпратена към потребител от виртуалния домейн **bovine.net** към локалния потребител на машината. Вторият елемент пренасочва поща към потребител от същия виртуален домейн към потребител от друг домейн. Третият пример пренасочва поща, адресирана до който и да е потребител от виртуалния домейн **dairy.org** към един единствен отдалечен пощенски адрес. Накрая, последният елемент пренасочва всякаква поща към потребител от виртуалния домейн artist.org към същия потребител от друг домейн; например, *julie@artists.org* ще бъде пренасочена към *julie@red.firefly.com*.

Тестване на вашата конфигурация

Командата *m4* обработва файловете с макро дефиниции в зависимост от нейните собствени правила за синтаксис, без да има представа за правилния синтаксис на *sendmail*; следователно, ако поставите нещо грешно във вашия файл с макро дефиниции, няма да има никакво съобщение за грешка. Поради тази причина, е много важно внимателно да тествате конфигурацията, която сте направили. За щастие, *sendmail* предоставя относително лесен начин за тази цел.

sendmail поддържа режим за тестване на адресите (address test mode). Този режим ни позволява да тестваме нашата конфигурация и да идентифицираме всякакви грешките в нея. В този режим на работа, извикваме програмата *sendmail* от командния ред. Тя изисква от нас спецификация на набора от правила и пощенския адрес на местоназначението. След това *sendmail* обработва този адрес, използвайки зададените правила, като показва изходните данни от всяко правило за преобразуване, докато то продължава да се прилага. За да стартираме *sendmail* в такъв режим на работа, трябва да я извикаме с аргумента `-bt`:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>
```

Използваният по подразбиране конфигурационен файл е */etc/mail/sendmail.cf*; като използвате аргумента `-C`, можете да зададете алтернативен конфигурационен файл. За да тестваме нашата конфигурация, трябва да изберем адреси за обработка, за да разберем дали всички наши изисквания за обработка на пощата са изпълнени. За да илюстрираме това, ще направим тест на нашата по-сложна UUCP конфигурация, показана в Пример 18.2.

Първо ще тестваме дали *sendmail* може да доставя поща на локални потребители на системата. Очакваме всички адреси да бъдат преобразувани към потребителската програма за поща **local** на тази машина:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 isaac
rewrite: ruleset   3       input: isaac
rewrite: ruleset  96       input: isaac
rewrite: ruleset  96     returns: isaac
rewrite: ruleset   3     returns: isaac
rewrite: ruleset   0       input: isaac
rewrite: ruleset 199       input: isaac
rewrite: ruleset 199     returns: isaac
rewrite: ruleset  98       input: isaac
rewrite: ruleset  98     returns: isaac
rewrite: ruleset 198       input: isaac
rewrite: ruleset 198     returns: $# local $: isaac
rewrite: ruleset   0     returns: $# local $: isaac
```

Тези изходни данни ни показват как *sendmail* обработва поща, адресирана до **isaac** на тази система. Всеки ред ни показва каква информацията е била приложена към набора от правила или резултата, получен от обработката, извършена от този набор. Указахме на *sendmail*, че искаме да използваме наборите от правила 3 и 0 за обработка на адреса. Обикновено се извиква набора от правила 0, а ние извикваме набора от правила 3, тъй като той не се тества по подразбиране. Последният ред ни показва, че резултатът от набора от правила 0 в действителност насочва поща към **isaac** чрез потребителската програма за поща **local**.

Сега ще тестваме поща, адресирана до нашия SMTP адрес: **isaac@vstout.vbrew.com**. Би следвало да можем да генерираме същия краен резултат, както в последния ни пример:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 isaac@vstout.vbrew.com
rewrite: ruleset   3      input: isaac @ vstout . vbrew . com
rewrite: ruleset  96      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset  96    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset   3    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset   0      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 199      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 199    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset  98      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset  98    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset 198      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 198    returns: $# local $: isaac
rewrite: ruleset   0    returns: $# local $: isaac
```

И отново, тестът е направен. Сега ще тестваме поща към нашия адрес от тип UUCP: **vstout!isaac**.

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>3,0 vstout!isaac
rewrite: ruleset   3      input: vstout ! isaac
rewrite: ruleset  96      input: isaac < @ vstout . UUCP >
rewrite: ruleset  96    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset   3    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset   0      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 199      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 199    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset  98      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset  98    returns: isaac < @ vstout . vbrew . com >
rewrite: ruleset 198      input: isaac < @ vstout . vbrew . com >
rewrite: ruleset 198    returns: $# local $: isaac
rewrite: ruleset   0    returns: $# local $: isaac
```

И този тест е направен. Тези тестове потвърждават, че всяка поща, получена за локалните потребители на тази машина, ще бъде правилно доставена, независимо от това как е форматиран адреса. Ако сте дефинирали някакви псевдоними за вашата машина, като виртуални хостове, трябва да повторите тези тестове за всяко от алтернативните имена, под което е известен този хост, за да гарантирате, че те също работят правилно.

Сега ще тестваме дали поща, адресирана до други хостове в домейна **vbrew.com**, се доставя директно до този хост посредством потребителската програма за поща, използваща SMTP:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 isaac@vale.vbrew.com
rewrite: ruleset   3      input: isaac @ vale . vbrew . com
rewrite: ruleset  96      input: isaac < @ vale . vbrew . com >
rewrite: ruleset  96    returns: isaac < @ vale . vbrew . com >
rewrite: ruleset   3    returns: isaac < @ vale . vbrew . com >
rewrite: ruleset   0      input: isaac < @ vale . vbrew . com >
rewrite: ruleset 199      input: isaac < @ vale . vbrew . com >
rewrite: ruleset 199    returns: isaac < @ vale . vbrew . com >
rewrite: ruleset  98      input: isaac < @ vale . vbrew . com >
rewrite: ruleset  98    returns: isaac < @ vale . vbrew . com >
rewrite: ruleset 198      input: isaac < @ vale . vbrew . com >
rewrite: ruleset 198    returns: $# smtp $@ vale . vbrew . com . /
   $: isaac < @ vale . vbrew . com . >
rewrite: ruleset   0    returns: $# smtp $@ vale . vbrew . com . /
   $: isaac < @ vale . vbrew . com . >
```

Можем да видим, че този тест е насочил съобщението към потребителската програма за поща, използваща SMTP, за да бъде препредадено директно към хоста **vale.vbrew.com**, и указва потребителя **isaac**. Тестът потвърждава, че нашата дефиниция на LOCAL_NET_CONFIG работи правилно. За да бъде успешен този тест, името на хоста на местоназначението трябва да бъде разпознато правилно, следователно, трябва да има елемент за него или в нашия файл */etc/hosts*, или в нашата локална DNS. Можем да видим какво се получава, ако името на хоста на местоназначението не може да бъде разпознато чрез умишлено указване на непознат хост:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 isaac@vXXXX.vbrew.com
rewrite:ruleset   3    input: isaac @ vXXXX . vbrew . com
rewrite: ruleset 96    input: isaac < @ vXXXX . vbrew . com >
vXXXX.vbrew.com:Name server timeout
rewrite: ruleset  96 returns: isaac < @ vXXXX . vbrew . com >
rewrite: ruleset   3 returns: isaac < @ vXXXX . vbrew . com >
== Ruleset 3,0 (3) status 75
rewrite:ruleset   0    input: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset 199    input: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset 199 returns: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset  98    input: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset  98 returns: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset 198    input: isaac < @ vXXXX . vbrew . com >
rewrite:ruleset  95    input: < uucp-new : moria > isaac </
   @ vXXXX . vbrew . com >
rewrite: ruleset  95 returns: $# uucp-new $@ moria $: isaac </
   @ vXXXX . vbrew . com >
rewrite:ruleset 198 returns: $# uucp-new $@ moria $: isaac </
   @ vXXXX . vbrew . com >
rewrite: ruleset   0 returns: $# uucp-new $@ moria $: isaac </
   @ vXXXX . vbrew . com >
```

Резултатът е много различен. Първо, набор от правила 3 връща съобщение за грешка, указващо, че името на хоста не може да бъде разпознато. Второ, справяме се с тази ситуация, като разчитаме на другата ключова възможност на нашата конфигурация – интелигентния хост. Той ще обработва всяка поща, която не може да се достави по друг начин. Името на хоста, което зададохме в този тест, не можеше да бъде разпознато, а наборите от правила определиха, че пощата трябва да бъде препредадена към нашия интелигентен хост **moria** посредством **uucp-new**. Нашият интелигентен хост може да бъде подобре свързан и да знае какво да направи с този адрес.

Последният ни тест гарантира, че всяка поща, адресирана до хост, извън нашия домейн, е доставена до нашия интелигентен хост. Резултатът би трябвало да е подобен на този от предишния пример:

```
# /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> 3,0 isaac@linux.org.au
rewrite: ruleset   3       input: isaac @ linux . org . au
rewrite: ruleset  96       input: isaac < @ linux . org . au . >
rewrite: ruleset  96     returns: isaac < @ linux . org . au  .>
rewrite: ruleset   3     returns: isaac < @ linux . org . au  .>
rewrite: ruleset   0       input: isaac < @ linux . org . au . >
rewrite: ruleset 199       input: isaac < @ linux . org . au . >
rewrite: ruleset 199     returns: isaac < @ linux . org . au . >
rewrite: ruleset  98       input: isaac < @ linux . org . au . >
rewrite: ruleset  98     returns: isaac < @ linux . org . au . >
rewrite: ruleset 198       input: isaac < @ linux . org . au . >
rewrite: ruleset  95       input:  < uucp-new : moria > isaac < /
   @ linux . org . au . >
rewrite: ruleset  95     returns: $# uucp-new $@ moria $: isaac < /
   @ linux . org . au . >
rewrite: ruleset 1 98    returns: $# uucp-new $@ moria $: isaac < /
   @ linux . org . au . >
rewrite: ruleset   0     returns: $# uucp-new $@ moria $: isaac < /
   @ linux . org . au . >
```

Резултатът от този тест показва, че името на хоста е разпознато, и че съобщението все още ще бъде маршрутизирано към нашия интелигентен хост. Това доказва, че нашето дефиниция на `LOCAL_NET_CONFIG` работи правилно и обработва и двата случая правилно. Този тест също беше успешен, така че доволно можем да предположим, че нашата конфигурация е правилна и да я използваме.

Стартиране на sendmail

Демонът *sendmail* може да бъде стартиран по два начина. Единият е да го стартирате от демона *inetd*; другият начин, който е по-често използван, е да стартирате *sendmail* като отделен демон. Освен това, за потребителските програми за поща е обичайно да извикват *sendmail* като потребителска команда за приемане на локално генерирана поща за доставяне.

Когато стартирате *sendmail* в самостоятелен режим, поставете командата в *rc* файл, така че се стартира по време на зареждането. Обикновено, използваният синтаксис:

```
/usr/sbin/sendmail -bd -q10m
```

Аргументът `-bd` указва на *sendmail* да се стартира като демон. Тя ще се разклони и ще се стартира във фонов режим. Аргументът `-q10m` указва на *sendmail* да проверява опашката на всеки десет минути. Можете да изберете да използвате различна опашка за проверка на времето.

За да стартирате *sendmail* от мрежовия демон *inetd*, трябва да използвате запис като следния:

```
smtp    stream    tcp    nowait    nobody    /usr/sbin/sendmail -bs
```

Тук аргументът `-bs` указва на *sendmail* да използва протокола SMTP на стандартния вход/изход, който се използва задължително с *inetd*.

Командата *runq* обикновение символна връзка към *sendmail* binary и е по-удобната форма на:

```
# sendmail -q
```

Когато извикате *sendmail* по този начин, тя обработва всяка поща, чакаща в опашката за предаване. Освен това, когато стартирате *sendmail* от *inetd*, трябва да създадете и процеса *cron*, който периодично стартира командата *runq*, за да гарантира, че пощенския буфер се обслужва периодично.

Подходящ запис в таблица за *cron* ще бъде подобен на:

```
# Стартира пощенския буферна всеки петнайсет минути
0, 15, 30, 45  *  *  *  *  /usr/bin/runq
```

При повечето инсталации *sendmail* обработва опашката на всеки 15 минути, както е показано в нашия пример за *crontab*, правейки опит да предаде всяко съобщение от нея.

Съвети и трикове

Съществуват много неща, чрез които можете да направите управлението на *sendmail* сайт ефективно. Много от инструменти за управление са предоставени в *sendmail* пакета; нека разгледаме най-важните от тях.

Управление на пощенския буфер

Преди да бъде предадена, пощата се подрежда в опашка в директорията */var/spool/mqueue*. Тази директория се нарича пощенски буфер. Програмата *sendmail* предоставя средство за показване на форматиран списък с всички съобщения, намиращи се в пощенския буфер и техния статус.

Командата */usr/bin/mailq* е символна връзка към *sendmail* и има идентично поведение като:

```
# sendmail -bp
```

В резултат на това се показват идентификатора на съобщението, неговия размер, времето на постъпването му в опашката, кой го е изпратил и съобщение, указващо текущия му статус. Следващият пример показва пощенско съобщение, което е заседнало в опашката поради някакъв проблем:

```
$ mailq
                Mail queue (1 request)
--Q-ID--  --Size--  -----Q-Time-----  ---------sender/recipient---------
RAA00275      124    Wed Dec 9 17:47     root
                 (host map: lookup (tao.linux.org.au) :   deferred)
                                  terry@tao.linux.org.au
```

Това съобщение все още е в опашката, тъй като IP адресът на хоста на местоназначението не може да бъде разпознат.

Можем да накараме *sendmail* да обработи опашката сега, като зададем командата */usr/bin/runq*.

Командата *runq* не извежда нищо на изхода. *sendmail* ще започне обработката на пощенската опашка във фонов режим.

Налагане на отдалечен хост да обработва своята пощенска опашка

Ако използвате временна връзка към Интернет през телефонна линия с фиксиран IP адрес и разчитате на MX хост да събира вашата поща, докато не сте свързани, ще откриете, че е полезно да принудите MX хоста да обработва пощенската опашка скоро, след като сте установили връзка

Малка програма на *perl* е включена в дистрибуцията на *sendmail*, която прави това просто за пощенските хостове, които я поддържат. Скриптът *etrn* има същия ефект върху отдалечения хост, както командата *runq* върху вашия собствен хост. Ако извикаме командата както е показано в този пример:

```
# etrn vstout.vbrew.com
```

ще принудим хоста **vstout.vbrew.com** да обработи всяко съобщение от опашката за нашата локална машина.

Обикновено, трябва да добавяте тази команда към вашия PPP *ip-up* скрипт, така че да се изпълнява скоро след като вашата мрежова връзка е установена.

Анализиране на пощенската статистика

sendmail събира данни за обема на пощенския трафик и информация за хостовете, до които е доставила поща. Съществуват две команди за показване на тази информация: *mailstats* и *hoststat*.

mailstats

Командата *mailstats* показва статистика за обема на пощата, обработена от *sendmail*. Първо се отпечатва времето, когато е започнало събирането на данни, следвано от таблица, съдържаща по един ред за всяка конфигурирана потребителска програма за поща и един ред за общо резюме на цялата поща. Във всеки ред има осем полета с информация:

Поле	Значение
M	Номерът на потребителската програма за поща (транспортния протокол)
msgsfr	Броят на съобщенията, получени потребителската програма за поща
bytes_from	Килобайтите на пощата от потребителската програма за поща
msgsto	Броят на съобщенията, изпратени към потребителската програма за поща
bytes_tp	Килобайтите на пощата, изпратена към потребителската програма за поща
msgsreg	Броят на отхвърлените съобщения
msgsdis	Броят на игнорираните съобщения
Mailer	Името на потребителската програма за поща

Примерни изходни резултати от командата mailstats са показани в Пример 18.5.

Пример 18.5: Примерни изходни данни от командата mailstats

```
# /usr/sbin/mailstats
Statistics from Sun Dec 20 22:47:02 1998

 M  msgsfr   bytes_from  msgsto  bytes_to  msgsrej  msgsdis  Mailer
 0  0        0K          19      515K      0        0        prog
 3  33       545K        0       0K        0        0        local
 5  88       972K        139     1018K     0        0        esmtp
=====================================================================
T  121      1517K       158     1533K     0        0
```

Тези данни се събират, ако опцията *StatusFile* е разрешена във файла *sendmail.cf* и съществува файл за статуса. Обикновено, към вашия файл *sendmail.cf* трябва да добавите следното:

```
# status file
O StatusFile=/var/log/sendmail.st
```

За да рестартирате събирането на статиста, трябва да направите дължината на файла със статистиката равна на нула:

```
> /var/log/sendamil.st
```

и да рестартирате *sendmail*.

hoststat

Командата *hoststat* показва информация за статуса на хостовете, до които *sendmail* е опитала да достави поща. Тя е еквивалентна на извикването на *sendmail* като:

```
sendmail -bh
```

Изходните данни представят всеки хост на един ред, времето, когато е направен опита за доставяне на поща и полученото по това време съобщение за статуса.

Пример 18.6 показва типа на изходните резултати, които можете да очаквате като резултат от изпълнението на командата hoststat. Забележете, че по-голямата част от резултатите показват успешно доставяне на поща. От друга страна, резултатът за **eartlink.net** показва, че доставянето е било неуспешно. Съобщението за статуса понякога може да ви помогне да определите причината за неуспеха. В този случай, таймаутът на връзката е изтекъл, вероятно защото хостът е спрял или не може да бъде достигнат по времето, когато е правен опит за доставяне на поща.

Пример 18.6: Примерни изходни данни от командата hoststat

```
# hoststat
----------Hostname--------How long ago----------results----------
mail.telstra.com.au                 04:05:41 250 Message accepted for
scooter.eye-net.com.au            81+08:32:42 250 OK id=0zTGai-0008S9-0
yarrina.connect.com.au               53+10:46:03 250 LAA09613 Message
accepted
happy.optus.com.au                     55+03:34:40 250 Mail accepted
mail.zip.com.au                  04:05:33 250 RAA23904 Message accepted
kwanon.research.canon.com.au    44+04:39:10 250 ok 911542267 qp 21186
linux.org.au                  83+10:04:11 250 IAA31139 Message accepted
albert.aapra.org.au                 00:00:12 250 VAA21968 Message
accepted
field.medicine.adelaide.edu.au    53+10:46:03 250 ok 910742814 qp 721
copper.fuller.net             65+12:38:00 250 OAA14470 Message accepted
amsat.org                       5+06:49:21 250 UAA07526 Message accepted
mail.acm.org                  53+10:46:17 250 TAA25012 Meassage accepted
extmail.bigpond.com              11+04:06:20 250 ok
earthlink.net                 45+05:41:09 Deferred: Connection time out
```

Командата *purgestat* изчиства събраните сатитстически данни за хоста и е еквивалентна на извикването на sendmail като:

```
# sendmail -bH
```

Ако не изчистите статистиките, те ще продължат да нарастват. Може да искате периодично да стартирате командата *purgestat*, за да улесните търсенето и откриването на последните данни, особено ако сайтът ви е натоварен. Можете да поставите командата във файла *crontab*, така че да се стартира автоматично, или просто да я стартирате от време на време.

ГЛАВА 19

НАСТРОЙКА И СТАРТИРАНЕ НА EXIM

Тази глава представлява кратко въведение в настройването на Exim и преглед на функционалността на програмата. Въпреки че Exim е до голяма степен съвместима със *sendmail* по отношение на поведението, нейните конфигурационни файлове са напълно различни.

Основният конфигурационен файл обикновено се нарича */etc/exim.conf* или */etc/exim/config* в повечето дистрибуции на Linux, или */usr/lib/exim/config* в по-стари конфигурации. Чрез следващата команда можете да намерите къде се намира конфигурационния файл:

```
$ exim -bP configure_file
```

Може да се наложи да редактирате конфигурационния файл, за да отразите специфичните за вашия сайт стойности. При повечето конфигурации няма какво толкова да се променя и работещата ще се модифицира рядко.

По подразбиране, Exim веднага обработва и доставя цялата входяща поща. Ако трафикът е относително голям, можете да настроите Exim да събира всички съобщения в така наречената *опашка* и да ги обработва само през определени интервали от време.

При обработка на поща в TCP/IP мрежа, Exim често се стартира в режим демон: по време на зареждане на системата тя се извиква от */etc/init.d/exim* и се стартира сама във фонов режим, където чака за входящи TCP връзки на SMTP порта (обикновено, това е порт 25). Когато очаквате значително количество трафик, това е полезно, тъй като Exim не трябва да се стартира за всяка входяща връзка. От друга

страна, *inetd* може да управлява SMTP порта и да стартира Exim винаги, когато съществува връзка на този порт. Тази конфигурация може да ви е от полза, когато разполагате с ограничена памет и трафик с малък обем.

Exim има сложен набор от опции на командния ред, много от които съвпадат с тези на *sendmail*. Вместо да правите опити да съберете точно подходящите опция, които са ви необходими, можете да реализирате най-общите типове операции чрез извикване на традиционни команди като *rmail* или *rsmtp*. Това са символни връзки, сочещи към Exim (или ако не са, лесно можете да ги насочите към нея). Когато стартирате някоя от тези команди, Exim проверява името, което сте използвали, за да извикате програмата и сама задава правилните опции.

Съществуват две връзки към Exim, които трябва да имате при каквито и да е обстоятелства: */usr/bin/rmail* и */usr/sbin/sendmail*. Когато съставяте и изпращате пощенско съобщение с потребителски агент като *elm*, съобщението се предава през канал към *sendmail* или *rmail* за доставяне, което е и причината */usr/sbin/sendmail* и */usr/bin/rmail* да сочат към Exim. Списъкът с получателите на съобщението се предава към Exim чрез командния ред. Същото става и с пощата, пристигаща през UUCP. Можете да настроите съответните имена на пътища, сочещи към Exim, въвеждайки следното в поканата за въвеждане на команди на обвивката:

```
$ ln -s /usr/sbin/exim /usr/bin/rmail
$ ln -s /usr/sbin/exim /usr/sbin/sendmail
```

Ако искате да се разровите по-подробно в детайлите, свързани с конфигурирането на Exim, ще трябва да се използвате пълната спецификация на Exim. Ако тя не е включена във вашата любима дистрибуция на Linux, можете да я получите от пакета с изходния код или да я прочетете online от web-сайта на Exim на адрес: *http://www/exim.org*

Стартиране на Exim

За да стартирате Exim, първо трябва да решите дали искате тя обработва входящите SMTP съобщения, като я стартирате като отделен демон, или демона *inetd* да управлява SMTP порта и да извиква Exim, само когато клиент прави заявка за SMTP връзка. Обикновено, предпочитания вариант е демон, работещ на пощенския ви сървър, тъй като това натоварва много по-малко машината, отколкото стартира-

нето на Exim отново и отново за всяка нова връзка. Тъй като пощенският сървър доставя по-голямата част от входящата поща директно до потребителите, за повечето от другите хостове ще трябва да изберете *inetd*.

Какъвто и режим на работа да изберете за всеки отделен хост, трябва да сте сигурни, че притежавате следния запис в вашия файл */etc/services*:

```
smtp      25/tcp    # Прост протокол за прехвърляне на поща
```

Това определя номера на TCP порта, използван за диалог посредством SMTP. 25-ти порт е стандартния порт, определен от "Assigned Numbers" RFC-1700.

Когато работи в режим демон, Exim се стартира във фонов режим и чака за връзки на SMTP порта. При възникване на връзка програмата се разклонява, а дъщерният процес осъществява SMTP диалог с процеса на другата страна на връзката на викащия хост. Демонът Exim обикновено се стартира като се извиква по време на зареждане от скрипта *rc*, използвайки следната команда:

```
/usr/sbin/exim -bd -q15m
```

Флагът *-bd* включва режима демон, а *-q15m* го кара да обработва всякакви съобщения, които се натрупват в пощенската опашка на всеки петнайсет минути.

Ако вместо това искате да използвате *inetd*, вашият файл */etc/inetd.conf* файл трябва да съдържа ред като следния:

```
smtp  streamtcp  nowait   root   /usr/sbin/exim   in.exim -bs
```

Запомнете, че трябва да укажете на *inetd* да прочете отново файла *inetd.conf*, като изпратите сигнал HUP, след като направите каквито и да било промени.

Двата режима *inetd* и демон са взаимно изключващи се. Ако стартирате exim в режим демон, трябва да сте сигурни, че сте задали като коментар всички редове във файла *inetd.conf* за услугата `smtp`. По същия начин, когато *inetd* управлява Exim, уверете се, че *rc* скриптът няма да стартира Exim.

Можете да провериге дали Exim е правилно настроена за получаване на входящи SMTP съобщения чрез осъществяване на telnet връзка към SMTP порта на вашата машина. Ето какво се получава при успешна връзка към SMTP сървъра:

```
$ telnet localhost smtp
Trying 127.0.0.1…
Connected to localhost.
Escape character is '^]'.
220 richard.vbrew.com ESMTP Exim 3.13 #1 Sun, 30 Jan 2000 16:23:55
+0600
quit
221 richard.brew.com closing connection
Connection closed by foreign host.
```

Ако този тест не генерира SMTP банер (реда с код, започващ с 220), проверете дали сте стартирали процес на демона Exim или правилно сте конфигурирали *inetd*. Ако това не реши проблема, погледнете в дневниците на Exim (описани по-нататък) в случай, че грешката е в конфигурационния файл на Exim.

Ако вашата поща не отива където трябва

Съществуват много възможности за отстраняване проблемите, свързани с инсталацията. Първо трябва да проверите дневниците на Exim. В Linux системите те обикновено се съхраняват в */var/log/exim/log*, а имената им са *exim_mainlog, exim_rejectlog* и *exim_paniclog*. В други операционни системи тези файлове често се съхраняват в */var/spool/exim/log*. Можете да откриете къде се намират дневниците, като използвате следната команда:

```
exim -bP log_file_path
```

Дневникът main съдържа списък с всички транзакции, дневникът reject съдържа подробности за всички отхвърлени от указаната политика съобщения, а дневникът panic е за съобщения, свързани с грешки в конфигурацията и други подобни.

Следват типични записи в дневника main. Всеки запис в дневника е един единствен ред с текст, започващ с дата и време. За да се съберат на страницата, сме ги разделили на няколко реда:

```
2000-01-30  15:46:37  12EwYe-0004WO-00  <= mailto:jack@vstout.vbrew.com
    H=vstout.vbrew.com [192.168.131.111] U=exim P=esmtp S=32100
    mailto:id=38690D72.286F@vstout.vbrew.com
2000--30    15:46:37  12EwYe-0004WO-00  => jill <mailto:jill@vbrew.com>
    D=localuser T=local_delivery
2000--31    15:46:37  12EwYe-0004WO-00  Completed
```

Тези записи показват, че съобщение от *mailto:jack@ystout.vbrew.com* към *mailto:jill@ybrew.com* е било успешно доставено до пощенската кутия на локалния хост. Пристигането на съобщение се отбелязва с флаг <=, а предаването му – с флаг му =>.

Съществуват два вида грешки при доставяне на съобщение: постоянни и временни. Постоянна грешка при доставяне се записва в дневника по следния начин с флаг "**":

```
2000-01-30 14:48:28 12EvcH-0003rC-00 ** bill@lager.vbrew.com
  R=lookuphost T=smtp: SMTP error from remote mailer after RCPT TO:
  <bill@lager.vbrew.com>: host lager.vbrew.com [192.168.157.2]:
  550 <bill@lager.vbrew.com>...User unknown
```

След неуспешно доставяне на съобщение като в случая, Exim изпраща съобщение за неуспех на изпращача, което често се нарича съобщение за отхвърляне (*bounce message*).

Временните грешки се маркират с флаг "==":

```
2000-01-30 12:50:50 12E9Un-0004Wq-00 == jim@bitter.vbrew.com
  T=smtp defer (145): Connection timed out
```

Тази грешка е типична за ситуация, в която Exim правилно разпознава, че съобщението трябва да се достави до отдалечен хост, но не може да осъществи връзка с услугата SMTP на този хост. Отдалеченият хост може временно да е спрял или има някакъв проблем с мрежата. Винаги когато едно съобщение е *отложено* по този начин, то остава в опашката на Exim и през определени интервали се прави опит да бъде изпратено. Ако обаче съобщението не може да бъде доставено достатъчно дълъг период от време (обикновено няколко дни), възниква постоянна грешка и съобщението се отхвърля.

Ако не можете да определите възникналия проблем от генерираното от Exim съобщение за грешка, можете да включите възможността за дебъгване на съобщенията. Можете да направите това като използвате флага *-d*, следван от число, определящо нивото многословие на съобщението (стойност 9 дава максимално изчерпателна информация). След това Exim показва на екрана отчет за своята работа, който може да ви даде повече съвети за това, което не работи правилно.

Компилиране на Exim

Exim все още се развива активно; версията на Exim, включена в дистрибуциите на Linux, вероятно не е последната реализация. Ако имате нужда от възможност или от фиксиране на грешка в последната

версия, трябва да получите копие от изходния код и да го компилирате сами. Най-новата реализация може да се намери от web-страницата на Exim на адрес: *http://www.exim.org*.

Linux е една от многото операционни системи, поддържани от изходния код на Exim. За да компилирате Exim за Linux, трябва да редактирате файла *src/EDITME* и да поставите получения резултат във файла *Local/Makefile*. Коментарите във файла *src/EDITME* ви указват за какво се използват различните настройки. След това стартирайте *make*. Подробна информация относно компилирането на Exim от изходен код можете да намерите в ръководството за програмата.

Режими за доставяне на поща

Както вече отбелязахме, Exim може да доставя съобщения веднага или да ги поставя в опашка, за да бъдат обработени по-късно. Цялата входяща поща се съхранява в директорията *input*, намираща под */var/spool/exim*. Когато съобщенията не могат да бъдат поставяни в опашка, процесът на доставяне се стартира за всяко съобщение веднага щом пристигне. В противен случай, съобщението се оставя в опашката, докато процесът *queue-runner* го извлече за обработка. Поставянето на съобщения в опашка може да бъде е безусловно чрез задаване на *queue_only* в конфигурационния файл или да зависи от 1-минутното зареждане на системата, използвайки настройка като следната:

```
queue_only_load = 4
```

която поставя съобщенията в опашка, ако зареждането на системата превиши 4.

Ако вашият хост не е постоянно свързан към Интернет, може да искате да включите поставянето на съобщения в опашка за отдалечени адреси, позволявайки по този начин на Exim веднага да извърши локално доставяне на поща. Можете да направите това като конфигурационния файл зададете:

```
queue_remote_domains = *
```

Включвайки под каквато и да било форма поставянето на съобщения в опашка, трябва да се уверите, че опашката се проверява редовно на всеки 10 или 15 минути. Дори и без някаква явно зададена опция за опашка, опашката трябва да се проверява за съобщения, които са били отложени поради неуспехи при временното доставяне. Ако стартирате Exim в режим демон, в командния ред трябва да добавите оп-

цията *-q15m*, за да проверявате опашката на всеки 15 минути. Същото нещо можете да направите като извиквате *exim -q* от *cron* в тези интервали.

Ако извикате Exim с опцията *-bp*, можете да покажете текущата пощенска опашка. Аналогично, можете да направите *mailq* да бъде връзка към Exim и да я:

```
$ mailq
  2h    52K  12 EwGE-0005jD-00 <mailto:sam@vbrew.com >
         D mailto:bob@vbrew.com
           mailto:harry@example.net
```

Това показва, че в опашката има едно единствено съобщение от *mailto:sam@vbrew.com* до двама получатели. Съобщението е било успешно доставено до *mailto:bob@vbrew.com*, но все още не е доставено до *mailto:harry@example.net*, въпреки че е в опашката от два часа. Размерът на съобщението е 52K, а идентификаторът, чрез който Exim го идентифицира, е `12EwGE-0005jD-00`. Можете да разберете защо доставянето на съобщението още не е завършило, като разгледате дневника за всяко съобщение, който се съхранява в директорията *mslog*, намираща се в директорията за опашката на Exim. Лесен начин да направите това е да използвате опцията *-Mvl*:

```
$ exim  -Mvl 12EwGE-0005jD-00
2000-01-30 17:28:13 example.net [192.168.8.2]: Connection timed out
2000-01-30 17:28:13 harry@example.net: remote_smtp transport deferred:
  Connection timed out
```

Отделните дневници съхраняват копия на записите в дневника за всяко съобщение, така че лесно можете да ги разглеждате. Същата информация би могла да бъде извлечена и от дневника main като се използва помощната програма *exigrep*:

```
$ exigrep 12 EwGE-0005jD-00  /var/log/exim/exim_mainlog
```

Това би ви отнело повече време, особено при натоварена система, при която дневниците са доста големи. Помощната програма *exigrep* се използва при търсене на информация за повече от едно съобщение. Първият ѝ аргумент е регулярен израз и exigrep намира всички редове дневник, които са свързани със съобщения с поне един ред в дневника, който съвпада с израза. Това би могло да се използва за намиране на всички съобщения за определен адрес или всички съобщения до/от определен хост.

Можете да следите какво прави Exim, като стартирате *tail* за нейния дневник main. Друг начин, по който може да се направи това, е да

стартирате помощната програма *eximon*, която се разпространява с Exim. Това е едно X11 приложение, което показва превъртащ се екран на дневника main, а също и списък със съобщенията, чакащи за доставяне, както и няколко графики отчитащи дейността по доставянето на съобщения.

Други конфигурационни опции

Ето и някои от най-използваните опции, които можете да зададете в конфигурационния файл:

message_size_limit
: Тази опция ограничава размера на приеманите от Exim съобщения.

return_size_limit
: Тази опция ограничава обема на входящо съобщение, което Exim ще върне като част от съобщение за отхвърляне.

deliver_load_max
: Ако зареждането на системата превиши стойността, зададена чрез тази опция, тогава доставянето на пощата се преустановява, въпреки че съобщенията продължават да се приемат.

smtp_accept_max
: Това е максималния брой входящи SMTP извиквания, които Exim ще приема едновременно.

log_level
: Тази опция управлява обема на информацията, записвана в дневника. Освен това, съществуват и някои опции, чиито имена започват с *log_*, които управляват записването на специфична информация в дневника.

Маршрутизиране и доставяне на съобщения

Exim разделя доставянето на поща на три различни задачи: маршрутизиране, насочване и прехвърляне. Съществуват определен брой модули с код от всеки тип и всеки от тях може да бъде конфигуриран поотделно. Обикновено, в конфигурационния файл се настройват голям брой различни маршрутизатори, модули за насочване и транспорти.

Маршрутизаторите разпознават отдалечения адрес, определяйки на кой хост трябва да бъде изпратено съобщението и какъв транспортен ще се използва. При свързаните към Интернет хостове често има само един маршрутизатор, който разпознава адреса чрез търсене на домейна в DNS. Аналогично, може да има един маршрутизатор, който обработва адресите, предназначени за хостовете в локална мрежа, и втори маршрутизатор, изпращащ всички други адреси към един единствен *интелигентния хост*; например, пощенски сървър на доставчик на Интернет услуги (ISP).

Локалните адреси се дават на модулите за насочване, някои от които обикновено обработват псевдонимите, препредаването на поща, както и идентифицирането на локални пощенски кутии. Пощенските списъци могат да бъдат обработени от модули за насочване за псевдоними или препредаване. Ако един адрес е псевдоним или е за препредаване, всички генерирани адреси се обработват независимо от маршрутизаторите или модулите за насочване, ако е необходимо. Най-често използвания случай е доставянето на съобщение до пощенска кутия, но съобщенията могат да бъдат изпращани през канал към команда или да се добавят към файл, различен от пощенската кутия по подразбиране.

Транспортът е отговорен за реализирането на метод за доставяне; например, изпращането на съобщение през SMTP връзка или добавянето му към определена пощенска кутия. Маршрутизаторите и модулите за насочване определят кой транспорт ще се използва за всеки адрес на получател. Ако транспортът се провали, Exim или генерира съобщение за отхвърляне, или отлага адреса за по-късна обработка.

С Exim имате голяма свобода при конфигурирането на тези задачи. За всяка от тях има голям брой драйвери, от които можете да избирате тези, които са ви необходими. Описвате ги в различни части на конфигурационния файл на Exim. Първо се дефинират транспортите, след това модулите за насочване, а след тях маршрутизаторите. Не съществуват вградени подразбиращи се настройки, въпреки че Exim се разпространява с подразбиращ се конфигурационен файл, който покрива простите случаи. Ако искате да промените политиката на маршрутизиране на Exim или да модифицирате транспорт, най-лесно е да започнете от подразбиращият се конфигурационен файл и да направите промени, отколкото да се опитвате да направите пълен конфигурационен файл от нулата.

Маршрутизиране на съобщения

Когато получи адреса за доставяне, Exim първо проверява дали домейнът е еднакъв с този, който се обработва на локалния хост, като го сравнява със списък, намиращ се в конфигурационната променлива `local_domains`. Ако тази опция не е зададена, името на локалния хост се използва като единствен локален домейн. Ако домейнът е локален, адресът се подава към модулите за насочване. В противен случай, адресът се подава към маршрутизаторите, за да се открие към кой хост трябва да се препрати съобщението.[53]

Доставяне на съобщения до локални адреси

Най-общо, локален адрес е просто името на потребителя за влизане в системата, при което съобщението се доставя до пощенската кутия на потребителя, */var/spool/mail/*`име_на_потребителя`. Другите случаи включват псевдоними, имена на пощенски списъци и препращането на поща. При тях локалният адрес се разширява до нов списък с адреси, които могат да са локални или отдалечени.

Освен тези "обикновени" адреси, Exim може да обработва и други типове локални местоназначения на съобщения, като имена на файлове и команди за изпращане по канал. Когато доставя пощата до файл, Exim добавя към съобщението и ако е необходимо, създава файла. Местоназначенията за файл или канал сами по себе си не са адреси, така че не можете да изпратите поща да кажем на */etc/passwd@ybrew.com* и да очаквате да презапишете по този начин файла с паролите; доставките до определен файл са валидни, само ако идват от файлове с псевдоними или препращане. Забележете обаче, че */etc/passwd@ybrew.com* е синтактично правилен пощенски адрес, но когато Exim го получи, ще потърси (което е нормално) потребител, чието име за влизане е */etc/passwd*, няма да намери такъв и ще отхвърли съобщението.

В списъка с псевдоними или във файла за препращане, името на файла е всичко, което започва с наклонена черта (/) и не се анализира като пълен квалифициран пощенски адрес. Например, */tmp/junk* във

[53] Това е опростяване. Възможно е модулите за насочване да предават адреси към транспорти, които доставят поща до отдалечени хостове, и подобно, възможно е маршрутизатори да предават адреси към локални транспорти, които записват съобщението във файл или канал. Освен това, при определени обстоятелства е възможно маршрутизатори да предават адреси към модулите за насочване

файл за псевдоними или препращане се интерпретира като име на файл, но */tmp/junk@ybrew.com* е пощенски адрес, въпреки че не изглежда да е особено полезен. Въпреки това, при изпращане на поща през X.400 шлюзове минават валидни адреси от този тип, тъй като X.400 адресите започват с наклонена черта.

По подобен начин, *команда за предаване през канал* може да бъде всяка Unix команда, предхождана от символа за канал (|), освен ако низът не се анализира като валиден пощенски адрес. Освен ако не сте променили конфигурацията, Exim не използва обвивката, за да стартира командата; вместо това, тя разделя командата на име на команда, самите аргументи и я стартира директно. Съобщението се подава на стандартния вход на командата.

Например, за да насочите пощенски списък към локална група по интереси, можете да използвате shell-скрипт с име *gateit* и да настроите локален псевдоним, който доставя всички съобщения от този пощенски списък до скрипта, използвайки `|gateit`. Ако командният ред съдържа запетайка, трябва да я поставите в двойни кавички заедно с предхождащия я символ за изпращане по канал (|).

Локални потребители

Локален пощенски адрес най-общо обозначава пощенската кутия на потребител. Обикновено се намира в */var/spool/mail*, а името е името на самия потребител, който притежава файла. Ако файлът не съществува, той се създава от Exim.

В някои конфигурации, групата е зададена като групата на потребителя, а режимът е 0600. В такива случаи, процесите за доставяне се стартират с правата на потребителя, а той може да изтрие изцяло съдържанието на пощенската кутия. В други конфигурации обаче, групата на пощенската кутия е **mail**, а режимът е 660; процесите на доставяне се стартират чрез идентификатора на потребителя uid на система и групата **mail**, а потребителите не могат да изтриват файловете на пощенските кутии, въпреки че могат да ги изпразват.

Забележете, че въпреки че текущото стандартно място за файловете на пощенските кутии е */var/spool/mail*, част от софтуера за поща може да бъде компилиран да използва различни пътища, например, */usr/spool/mail*. Ако доставянето на поща до потребителите на вашата машина непрекъснато се проваля, трябва да видите дали, ако направите символна връзка към */var/spoll/mail* няма да разрешите проблема.

Адресите **MAILER-DAEMON** и **postmaster** обикновено се появяват във вашия файл за псевдоними и да се разширяват до пощенския адрес на системния администратор. Адресът **MAILER-DAEMON** се използва от Exim като адрес на изпращача в съобщения за отхвърляне. Освен това, се препоръчва **root** да се настрои като псевдоним на администратора, особено когато доставянето се стартира с правата за достъп на потребителите получатели, за да се избегне стартирането на доставянето като **root**.

Препращане

Потребителите могат да пренасочват пощата си към алтернативни адреси, като създадат в своите лични директории файл за препращане *.forward*. Той съдържа разделен със запетаи и/или нови редове списък с получатели. Всички редове от файла за препращане се четат и интерпретират. Може да се използва адрес от всякакъв тип. Един практичен пример за *.forward* файл за ваканции е:

```
janet, "|vacation"
```

В други описания на *.forward* файлове можете да видите потребителското име в началото, предхождано от обратно наклонена черта. Това беше необходимо за някои по-стари MTAs, за да спрат да търсят нов *.forward* файл за новото име, което може да доведе до непрекъснат цикъл. Обратно наклонената черта не е необходима в Exim, която автоматично избягва цикли от този род.[1] Въпреки това, използването на обратно наклонена черта е разрешено и на практика води до разлика в конфигурациите, където няколко домейна се обработват едновременно. Без обратно наклонена черта, неквалифицирано име на потребител се квалифицира с подразбиращ се домейн; с обратно наклонена черта се запазва домейна, от който идва адреса.

Първият адрес във файла за препращане доставя входящото съобщение до пощенската кутия на **janet**, а командата *vacation* връща кратко съобщение на изпращача.

Като допълнение към поддържането на "традиционни" файлове за препращане, Exim може да бъде конфигурирана да позволява по-сложни файлове, наречени *филтри*. Те не са просто списък с адреси за препращане, а могат да съдържат тестове за съдържанието на входящото съобщения, така че, например, съобщенията да се препращат, само ако в полето за тема на съобщението subject се съдържа съобщението "urgent" (спешно). Системният администратор трябва да реши дали да позволи на потребителите да използват тази гъвкавост.

Файлове за псевдоними

Exim може да обработва файлове за псевдоними, които са съвместими с файловете *sendmail* alias на Berkeley. Записите във файла за псевдоними могат да имат следната форма:

```
псевдоним: получатели
```

получатели е разделен със запетаи списък с адреси, които ще бъдат заместени с псевдонима. Списъкът с получателите може да продължава на нови редове, ако следващият ред започва с празно пространство.

Специална възможност позволява на Exim да обработва пощенски списъци, които се държат отделно от файла за псевдоними: ако зададете `:include:`*име_на_файл* като получател, Exim чете зададения файл и замества неговото съдържание като списък с получатели. Алтернативен начин за обработка на пощенски списъци е показан по-натам в тази глава в раздела "Пощенски списъци".

Главния файл за псевдонимите е */etc/aliases*. Ако направите този файл с права за писане от всички или от група потребители, Exim ще откаже да го използва и ще отхвърли локалните доставки. Можете да управлявате теста, който програмата прилага към правата за достъп до този файл, като зададете *modemask* в модула за насочване *system_aliases*.

Това е примерен файл за псевдоними *aliases*:

```
# vbrew.com /etc/aliases файл
hostmaster: janet
postmaster: janet
usenet: phil
# Пощенски списък development.
development: joe, sue, mark, biff,
         /var/mail/log/development
owner-development: joe
# Пощенски за общ интерес се изпращат по пощата до всички
# от персонала
announce: :include: /etc/Exim/staff,
         /var/mail/log/announce
owner-announce: root
# насочва пощенския списък ppp към локална група по интереси
ppp-list: "|/usr/local/bin/gateit local.lists.ppp"
```

Когато съществуват имена на файлове и команда за изпращане по канал в един файл за псевдоними, както в този пример, на Exim трябва да се укаже под какъв потребителски идентификатор да стартира доставянето. Опцията *user* (вероятно и *group*) трябва да бъде зададена в конфигурационния файл на Exim или в модула за насочване, който обработва псевдонимите, или в транспортите, към които насочва тези елементи.

Ако възникне грешка при доставяне до адрес, генериран от файл за псевдоними *aliases*, Exim ще изпрати съобщение за отхвърляне до изпращача на съобщението, както обикновено, но това може да се окаже неподходящо. Опцията *errors_to* може да бъде използвана, за да укажете, че съобщенията за отхвърляне трябва да бъдат изпратени на друго място; например, на postmaster.

Пощенски списъци

Пощенските списъци могат да бъдат управлявани не само от файла за псевдоними, но и от модула за насочване *forwardfile*. Тези списъци се съхраняват в една единствена директория като */etc/exim/lists/*, а пощенски списък с име `naq-bugs` е описан от файла *lists/naq-bugs*. Той трябва да съдържа адресите на членовете, разделени от запетаи или нови редове. Редовете, които започват със знака #, се третират като коментари. Прост модул за насочване, който използва такива данни е следния:

```
lists:
  driver = forwardfile
  file = /etc/exim/lists/${local_part}
  no_check_local_user
  errors_to = ${local_part}-request
```

Когато се стартира този модул за насочване, стойностите на *file* и *error_to* се *разширяват*. Това е причина определени части от низове, започващи със символа долар, да се заместват всеки път, когато се използва този низ. Най-простия начин за разширяване е вмъкването на стойността на една от променливите на Exim, а точно това става тук. Поднизът `${local_part}` замества стойността на `$local_part`, което всъщност е локалната част от адреса, който се обработва.

За всеки пощенски списък трябва да съществува потребител (или псевдоним, или пощенски списък), наречен *`listname-request`*; всички грешки, които възникват при преобразуване на адрес или доставяне до член от списък, се изпращат на този адрес.

Защита срещу непоръчана рекламна поща

Mail spam или непоръчаната рекламна електронна поща е досаден проблем за много потребители. Беше създаден проект, свързан с този проблем. Проектът се нарича MAPS (Mail Abuse Protection System – Система за защита от злоупотреба с поща). В този проект беше реализиран механизъм за намаляване на проблема, наречен RBL (Real Time Blackhole List – Списък с черни дупки в реално време). Информация за начина, по който работи MAPS RBL, можете да получите от неговата online документация на адрес *http://maps.vix.com/rbl/*. Идеята е проста. Сайтовете, генериращи непоръчана рекламна поща се добавят в база данни. Потребителски програми за поща като Exim могат да отправят запитване към базата данни, за да потвърдят, че източник на съобщението не е спамър преди да приеме поща от него.

Преди появата на RBL бяха създадени няколко подобни списъка. Един от най-полезните е списъка Dial_up List (DUL), който съдържа IP адресите на хостове за достъп през телефонна линия. Тези хостове обикновено изпращат изходящата поща само към пощенските сървъри на своите доставчици на Интернет услуги. Много сайтове блокират доставянето на поща от външни Dial-Up връзки, тъй като, когато такъв хост избягва своя собствен ISP сървър, то нещата не са много чисти.

Exim предоставя поддръжка на черни списъци в реално време и на други такива. Това може да се конфигурира много лесно. За да разрешите такава възможност, добавете следващите редове към вашия файл */etc/exim.conf*:

```
# Vixie / MAPS RBL (http://maps.vix.com/rbl)
rbl_domains = rbl.maps.vix.com : dul.maps.vix.com
```

Този пример проверява и RBL, и DUL като отхвърля всички съобщения от хостове, които са в някой от тези списъци. Опцията *rbl_hosts* ви позволява да зададете групи от хостове, към които се прилага (или не се) RBL проверка. Подразбиращата се настройка е:

```
rbl_hosts = *
```

което означава, че за всички хостове се прави RBL проверка. Ако искахте да отмените черните списъци и да приемате поща от определен хост без да правите RBL проверка, трябваше да използвате, например, следното:

```
rbl_hosts = ! nocheck.example.com : *
```

Удивителният знак преди първия елемент в този списък указва отрицателен елемент: ако викащият хост е *nocheck.example.com*, той ще съвпадне с този елемент. Но поради отрицанието, RBL проверка няма да бъде извършена. Всеки друг хост съвпада с втория елемент в списъка.

Настройване на UUCP

Exim няма никакъв специфичен код за пренасяне на поща през UUCP, нито пък поддръжка UUCP адреси от бум път. Въпреки това, ако използвате адресиране на домейн, Exim може да осъществи връзката към UUCP много просто. Следва конфигурационен фрагмент за изпращане на определени домейни към UUCP, взет от реална инсталация:

```
# Транспорт
uucp:
  driver = pipe
  user = nobody
  command = "/usr/local/bin/uux -r - \
     ${substr_-5:$host}!rmail ${local_part}
  return_fail_output = true

# Маршрутизатор
uucphost:
  transport = uucp
  driver = domainlist
  route_file = /usr/exim/uucphosts
  search_type = lsearch
```

В един пълен конфигурационен файл, транспортът ще бъде вмъкнат между другите транспорти, а маршрутизаторът вероятно ще бъде дефиниран като първия маршрутизатор. Файлът */usr/exim/uucphosts* съдържа записи като този:

```
darksite.example.com:    darksite.UUCP
```

което се интерпретира до следното значение: "Изпрати пощата, адресирана до домейна **darksite.example.com**, към UUCP хоста **darksite**". Тази конфигурация може да бъде настроена по много по-прост начин без маршрутизатор, като се добави наставката .UUCP към **darksite** и по този начин кажете на транспорта какво да прави. Но показаният начин е полезен, тъй като показва ясно разликата между името на домейна **darksite.example.com** и UUCP името на хоста **darksite**.

Винаги, когато маршрутизаторът минава през домейн от файла за маршрутизиране, той ще изпрати адреса към UUCP транспорта, който го изпраща по канал към командата *uux* (описана в Глава 16, *Управление на Taylor UUCP*). Ако съществува проблем, uux ще генерира някакви изходни данни и ще завърши с код на грешка, различен от нула. Задаването на `return_fail_output` гарантира, че изходните данни се връщат към изпращача на съобщението.

Ако входящите UUCP съобщения са групирани във файлове в batched SMTP format, те могат да се предадат директно към Exim, използвайки команда, подобна на следната:

```
exim -bS </var/uucp/incoming/001
```

Тук обаче има една уловка. Когато Exim получава съобщение локално, програмата изисква изпращачът да бъде влязъл в системата потребител, а за UUCP batched

искаме изпращачите да се взимат от входящите съобщения. Exim ще направи това, ако процесът който я извиква е *доверен потребител*. Ако направите така, че вашите входящи UUCP съобщения се обработват от потребител, наречен **uucp**, например, трябва да го зададете по следния начин:

```
trusted_users = uucp
```

в конфигурационния файл на Exim, за да сте сигурни, че адресите на изпращачите се обработват правилно.

ГЛАВА 20

НОВИНИ ПО МРЕЖАТА

Днес мрежовите новини или новините по мрежата Usenet си остават едни от най-важните и стойностни услуги, осигурявани чрез компютърните мрежи. Независимо, че някои ги отхвърлят като прояви на нежелана комерсиална реклама и порнография, мрежовите новини все още си остават място, където могат да се намерят някои висококачествени дискусионни групи, което ги направи важен информационен ресурс във времената преди Web. Даже и в настоящите времена на милиарди web-страници, мрежовите новини все още са ценен източник за онлайн помощ и обединяват различни теми за обсъждане.

История на Usenet

Идеята за мрежови новини се появи през 1979 г., когато двама дипломирани студенти – Tom Truscott и Jim Ellis, предлагат използването на UUCP за обмен на информация между машини на UNIX-потребители. За тази цел в Северна Каролина те правят малка мрежа от три машини.

Първоначално трафикът се обработва от няколко shell-скрипта (покъсно пренаписани на C), които обаче никога не са публикувани официално. Скриптовете скоро са заменени от A News (“А Новините”), първото публично издание на софтуера за новини.

A News не бяха проектирани да обработват повече от няколко статии на ден за една група. Тъй като обемът на новините продължаваше да нараства, софтуерът беше пренаписан от Mark Horton и Matt Glickman, които го нарекоха издание “B” (B News). Първото официално издание на B News беше версия 2.1, разпространено през 1982 г. Сис-

темата непрекъснато се развиваше, като бяха добавени някои нови възможности. Сегашната версия е B News 2.11. Тя бавно излиза от употреба; последният ѝ официален поддръжник вече се прехвърли на INN.

Geoff Collyer и Henry Spencer пренаписаха B News и новото издание излезе през 1987 г. Това е изданието "C" или C News (C новини). От момента на публикуването им са направени голям брой поправки, най-важната от които е C News Performance Release. Претоварването на сайтове, поддържащи значително количество групи, предизвикано от честото стартиране на *relaynews*, която отговаря за разпределянето на пристигащите статии към други хостове, е значително. Performance Release добавя към *relaynews* опция, която ѝ дава възможността да се стартира като демон (*daemon mode*), благодарение на който програмата преминава във фонов режим. Тази версия на C News в момента е включена в почти всички дистрибуции на Linux. Системата C News ще бъде разгледана подробно в Глава 21, "*C Новини*".

Всички издания на софтуера за новини до издание C бяха насочени основно към мрежи, базирани на UUCP, въпреки че съществуваше възможност те да бъдат използвани и в други обкръжения. Ефикасното прехвърляне на новини през мрежи, базирани на TCP/IP или DECNet изискваше съвсем нова схема за предаването им. Затова през 1986 г. беше представен новия протокол NNTP (*Network News Transfer Protocol* – мрежов протокол за прехвърляне на новини). Той е базиран на мрежови връзки и дефинира определен брой команди за интерактивно прехвърляне и изтегляне на статии.

Съществуват множество NNTP-базирани приложения, достъпни от Мрежата. Едно от тях е пакетът *nntpd*, написан от Brian Barber и Phil Lapsley, който можете да използвате за осигуряването на услуга за четене на новини за хостове, свързани в локална мрежа. *nntpd* беше проектиран да допълни пакетите за новини като B News и C News с новите NNTP възможности. Ако искате да използвате NNTP със сървъра C News, трябва да прочетете Глава 22, *NNTP и демона nntpd*, в която е описано как да конфигурирате демона *nntpd* и да го използвате със C News.

Алтернативен пакет, поддържащ NNTP, е пакетът INN или *Интернет новини*. Той е не само един потребителски интерфейс, а напълно самостоятелна система за новини. Тя включва усъвършенстван демон за препращане на новини, който ефикасно може да поддържа няколко едновременни NNTP връзки. Поради това, тази система е естественият избор за много Интернет сайтове. Ние ще я разгледаме по подробно в глава 23, *Интернет новини*.

Какво всъщност представлява Usenet?

Един от най-поразителните факти за Usenet е, че не е нито част от някаква организация, нито пък има някакво отговорно за нея централизирано мрежово управление. Всъщност, като изключим техническото описание на Usenet, никой не може да определи какво представлява тази система. С риск да изглежда глупав, човек би могъл да опише Usenet като съвместна работа на отдалечени един от друг сайтове, обменящи помежду си Usenet новини. За да бъдете Usenet сайт, всичко което трябва да направите, е да намерите друг Usenet сайт и да се договорите с неговите притежатели или тези, които го поддържат, да обменят новини с вас. Доставянето на новини към друг сайт се нарича захранване (*feeding*), затова една от аксиомите на Usenet философията гласи: "Намери захранване и си готов".

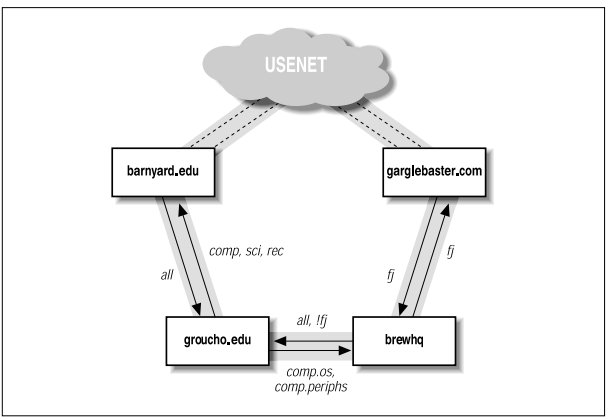
Основната градивна част на Usenet се нарича статия (*article*). Това е съобщение, което е написано и публикувано по мрежата от потребител. За да може системата за новини да го обработи, към него се добавя административна информация – така наречения заглавие. То доста прилича на заглавието на пощенско съобщение, описано в стандарта за Интернет поща RFC-822, което се състои от няколко реда текст, всеки от които започва с име на поле, завършващо с двоеточие, следвано от стойността на полето.[54]

Статиите се предават на една или повече групи по интереси (*newsgroups*). Можете да мислите за тях като за форуми, на които се изпращат статии, свързани с определена тема. Всички групи по интереси са организирани в йерархия, като всяко име на група показва мястото ѝ в йерархията. Така е по-лесно да се разбере какво се обсъжда в групата. Например, от името на групата всеки може да разбере, че *comp.os. linux. announce* е група за съобщения, свързани с операционната система, наречена Linux.

След това, тези статии се обменят между всички Usenet-сайтове, които желаят да пренасят новини, свързани с тази група. Когато два сайта се договорят да обменят новини, те са напълно свободни да обменят каквито искат групи по интереси и дори биха могли да добавят свои собствени локални йерархии. Например, сайтът **groucho.edu** би могъл да има главна захранваща връзка от сайта **barnyard.edu** и няколко връзки към по-малки сайтове, които самият той захранва с но-

[54] Форматът на Usenet новините е определен в RFC-1036,"Стандарт за обмен на Usenet съобщения"

вини. Сайтът на Barnyard College от своя страна би могъл да приема всички групи по интереси от Usenet, докато GMU иска да приема само някои от основните йерархии, например *sci*, *comp* и *rec*. Някой от подчинените му в йерархията сайтове, например UUCP-сайта, наречен **brewhq**, би могъл да иска да поддържа още по-малко групи, защото няма необходимите хардуерни или мрежови ресурси. От друга страна, **brewhq** би могъл да желае да приема групите по интереси от йерархията *fj*, която не се поддържа от сайта на GMU. Затова brewhq използва своя собствена връзка със сайта **gargleblaster.com**, който обменя всички *fj* групи и захранва с тях сайта **brewhq**. Потокът с движението на новините е показан на Фигура 20-1.



Фигура 20-1: Usenet потокът новини през Groucho Marx University

Все пак, етикетите на стрелките, излизащи от сайта **brewhq** изискват известно обяснение. По подразбиране всички локални новини от **brewhq** се изпращат към **groucho.edu**. Обаче, тъй като **groucho.edu** не поддържа групите по интереси от йерархията *fj*, няма никакъв смисъл да му изпращате новини от тези групи. Поради това, захранващата линия от **brewhq** към GMU има етикет `all, !fj`, който означава, че всички новини, с изключение на тези от групата *fj*, се изпращат към него.

Как Usenet обработва новините?

Днес Usenet има огромни размери. Сайтовете, поддържащи целия обем мрежови новини, обикновено прехвърлят нищожните 60MB[55] на ден. Разбира се, този процес изисква нещо повече от простото прехвърляне на файлове насам-натам. Нека погледнем начина, по който повечето Unix системи обработват Usenet-новините.

Началото на новините се слага тогава, когато потребители създадат и публикуват статии. Всеки потребител въвежда съобщението в специално приложение, наречено четец на новини (*newsreader*), който го форматира по подходящия начин за прехвърляне към локалния сървър за новини. В Unix среда четецът обикновено използва командата *inews*, за да предаде статии до сървъра за новини като използва протокола TCP/IP. Възможно е обаче да запишете статията направо във файл в една специална директория, наречена буфер (спул) за новини. След като съобщението бъде доставено до локалния сървър за новини, той поема отговорността за изпращането му до другите потребители на новини.

Новините се разпространяват по мрежата по най-различен начин. Преди се използваше UUCP, но днес основният трафик се поддържа от Интернет сайтове. Използваният алгоритъм за маршрутизация се нарича *flooding* (наводняване). Всеки сайт поддържа определен брой връзки (*news feeds* – захранващи линии с новини) към други сайтове. Всяка статия, генерирана или приета от локалната система за новини се препраща към тях, освен ако вече не е била в този сайт, като в този случай се игнорира. Един сайт може да разбере през кои други сайтове е преминала статията, като провери полето Path: в заглавието ѝ. Заглавието съдържа списък с всички системи, през които е преминала статията, като се използва запис в стила “бум” път (bang path).

За да се различават отделните статии и да се разпознават дубликатите, Usenet статиите трябва да съдържат идентификационен номер на съобщението (задава се в полето `Message-Id:` на заглавието), който обединява името на сайта, от който идва съобщението и един сериен номер във формата `<сериен_номер@име_на_сайта>`. Системата за новини записва този идентификационен номер за всяка обработена статия във файла *history* и сравнява записите в този файл с номерата на всички новополучени статии.

[55] Чакайте малко: 60 MB при скорост на трафика от 9600 bps, това е 60 милиона умножено по 1024, това е … да видим … Ей! Ама това са 34 часа!

Потокът от новини между кои да е два сайта може да се ограничи по два критерия. Първо, на статията се задава разпространение (в полето `Distribution`: на заглавието), което може да се използва за ограничаване само до определена група от сайтове. От друга страна, броят на групите по интереси, които се обменят, може да бъде ограничен както от изпращащата, така и от приемащата система. Наборът от групите по интереси и разпространенията, които е разрешено да бъдат предавани към даден сайт, обикновено се съхранява във файла *sys*.

Големият брой статии изисква да се направят подобрения в горната схема. При UUCP мрежи, системите събират статиите през определен период от време и ги обединяват в общ файл, който се компресира и изпраща на отдалечения сайт. Това се нарича пакетиране (*batching*).

Алтернативен метод представлява протоколът *ihave/sendme*, който предотвратява повторното прехвърляне на една и съща статия и по този начин запазва пропускателната способност на мрежата. Вместо да обединява всички съобщения в пакетни файлове и да ги изпраща по мрежата, този протокол събира само идентификационните номера на статиите в едно голямо съобщение "ihave" (аз-имам) и го изпраща до отдалечения сайт. Отдалеченият сайт го прочита, сравнява го със своя файл history и връща съобщение "sendme" (изпрати-ми), съдържащо всички статии, които той желае да му бъдат изпратени. Изпращат се само заявените статии.

Разбира се, *ihave/sendme* има смисъл да се използва само между големи сайтове, всеки от които приема новини от няколко независими захранващи линии, които достатъчно често се използват взаимно, за да е ефективно движението на потока с новини.

В общия случай, Интернет сайтовете разчитат на TCP/IP-базиран софтуер, който използва протокола NNTP (Network News Transfer Protocol – мрежов протокол за прехвърляне на новини). NNTP е описан в RFC-977; той е отговорен за прехвърлянето на новини между сървърите за новини и осигурява достъп до Usenet на единични потребители от отдалечени хостове.

NNTP познава три различни начина за прехвърляне на новини. Първият е версията на *ihave/sendme* за реално време, позната още като разпространение на новини (*pushing news*). Втората техника се нарича изтегляне на новини (*pulling news*), в която клиентът изисква списък със статиите в определена група по интереси или йерархия, пристигнали в сървъра след определена дата и избира онези от тях,

които не може да намери в своя файл *history*. Третата техника е за интерактивно четене на новини и позволява на вас или вашия четец за новини да преглеждате статии от изрично указани групи, а също и да публикувате статии с непълна информация в заглавието.

На всеки сайт новините се съхраняват в директорията */var/spoll/news*, като всяка статия е в отделен файл, а всяка група по интереси – в отделна директория. Името на директорията се получава от името на групата, като отделните ѝ компоненти се получават от компонентите на пътя. Например, статиите от групата *comp.os.linux.misc* се пазят в директорията */var/spool/news/comp/os/linux/misc*. На всяка статия от групата по интереси се задава номер по реда на пристигането ѝ. Този номер служи за име на файла ѝ. Диапазонът от номера на текущите активни (*online*) статии се пази във файл, наречен *active*, който в същото време служи като списък на групите по интереси, за които знае вашият сайт.

Тъй като дисковото пространство е ограничен ресурс, след известно време ще трябва да започнете да изтривате статии.[56] Това е така наречените изтичане на срока (*expiring*). Обикновено срокът на статиите от определени групи и йерархии изтича за фиксиран брой дни след пристигането им. Това може да бъде отменено от този, който ги публикува чрез определянето на срок на изтичане в полето `Expires:` в заглавието на статията.

Сега вече притежавате достатъчно информация, за да изберете какво да четете по-нататък. Потребителите на UUCP би трябвало да четат за C News в Глава 21. Ако използвате TCP/IP мрежа, прочетете повече за NNTP в Глава 22. Ако трябва да прехвърляте средно по обем количество новини през TCP/IP, сървърът описан в тази глава, може да ви е достатъчен. За да инсталирате високопроизводителен сървър за новини, който да обработва огромно количество материал, прочете за Интернет Новините в Глава 23.

[56] Някои хора твърдят, че Usenet всъщност е таен заговор на производителите на модеми и твърди дискове.

ГЛАВА 21

С НОВИНИ

С новините (C News) са един от най-популярните софтуерни пакети за мрежови новини. Той беше проектиран за сайтове, пренасящи новини през UUCP връзки. В тази глава ще разгледаме основните концепции на C News, инсталирането и задачите, свързани с поддръжката.

C News съхранява конфигурационните си файлове в директорията */etc /news*, а по-голямата част от изпълнимия код се намира в директорията */usr/bin/news*. Статиите се пазят във */var/spool/news*. Трябва да се уверите, че практически всички файлове в тези директории се притежават от потребителя **news** или групата **news**. Повечето проблеми възникват от невъзможността за достъп на C News до някои файлове. Използвайте командата *su*, за да влезете като потребителя **news**, преди да промените каквото и да било в тази директория. Единственото изключение е командата *setnewsids*, която се използва, за да укажете идентификатор на истински потребител за някои програми за новини. Тя трябва да бъде собственост на root със зададен бит *setuid*.

В тази глава ще опишем подробно всички конфигурационни файлове на C News и ще ви покажем какво точно трябва да направите, за да поддържате нормалната работа на вашия сайт.

Доставяне на новини

Статиите могат да се подадат на C News по няколко начина. Когато локалният потребител публикува статия, четецът на новини (*newsreader*) обикновено я подава на командата *inews*, която от своя страна попълва информацията в заглавието. Новините, идващи от отдалечени сайтове, независимо дали представляват една статия или цял пакет, се предават на командата *rnews*, която ги съхранява в директорията */var/spool/news/in.coming*, откъдето те по-късно се взимат за обработка от *newsrun*. Независимо коя от тези две техники се използва, в края на краищата статиите ще бъдат подадени на командата *relaynews*.

За всяка статия *relaynews* първо проверява дали тя вече не е била в локалния сайт, като търси идентификационния номер на съобщението във файла *history*. Дублираните статии се отстраняват. След това *relaynews* проверява в полето `Newsgroups:` на заглавието, за да разбере дали локалният сайт е заявил статии от някои от тези групи. Ако е така и групата по интереси е включена във файла *active*, *relaynews* се опитва да запише статията в съответната директория в областта на буфера за новини. Ако такава директория не съществува, тя се създава. След това идентификаторът на статията се добавя във файла *history*. В противен случай *relaynews* отстранява статията.

Понякога *relaynews* не може да запише входяща статия, защото групата, в която тя е публикувана, не е в включена във вашия файл *active*. В такъв случай статията се премества в групата *junk*[1]. *relaynews* проверява също и за статии, които са стари или с грешна дата и ги отхвърля. Входящи пакети, отхвърлени поради каквато и да било причина, се преместват във */var/spool/news/in.coming/bad* и в системния дневник се записва съобщение за грешка.

След това статията се препредава към всички други сайтове, които са направили заявка за новини от тези групи, като се използва зададения за всеки отделен сайт начин за транспортиране. За да е сигурно, че статията няма да се изпрати на сайтове, на които тя вече е известна, за всеки сайт-получател се прави проверка на полето `Path:` от

[1] Може да има разлика между групите, които съществуват на вашия сайт и тези, които той иска да получава. Например, абонаментния списък може да задава *comp.all*, който трябва да изпраща всички групи по интереси от йерархията *comp*, но във вашия сайт да не сте включили във файла *active* някои от групите на *comp*. Статии, изпратени към тези групи, ще бъдат преместени в групата *junk*.

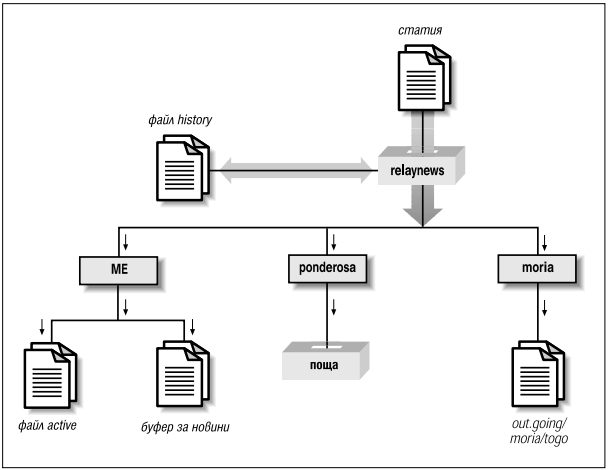
заглавието на статията. Това поле съдържа списък на сайтовете, през които статията вече е преминала, като се използва UUCP стила "бум-път", описващ маршрута ѝ. Този начин на запис е разгледан в Глава 17, *Електронна поща*. Ако името на сайта-получател не е в този списък, статията му се изпраща.

Обикновено C News се използва за препредаване на новини между UUCP сайтове, въпреки че е възможно да се използва и в NNTP среда. За доставка на новини до отдалечен UUCP сайт, независимо дали като отделна статия или цели пакети, се използва *uux* за изпълнение на командата rnews на отдалечения сайт и захранването му с тази отделна статия или целия пакет от стандартния му вход. За повече информация относно UUCP вижте Глава 16, "Управление на Taylor UUCP".

Пакетирането (*batching*) е термин, с който се означава изпращането с едно предаване на големи пакети от обединени в едно цяло отделни статии. Когато за определен сайт е разрешено пакетирането, C News не изпраща входящата статия веднага; вместо това системата добавя името на пътя ѝ към файл, обикновено наречен *out.going/site/togo*. Периодично се изпълнява програма, указана със запис във файла *crontab* на програмата *cron*, която прочита този файл и пакетира всички включени в списъка статии в един или повече файлове, като евентуално ги компресира и ги изпраща към *rnews* на отдалечения сайт[2].

На Фигура 21-1 е показан потока с новини през *relaynews*. Статиите могат да се препредават към локалния сайт (отбелязан като **me**), към сайта **ponderosa** чрез електронната поща и към сайта **moria**, за който е разрешено пакетирането.

[2] Обърнете внимание, че това трябва да бъде *crontab*-файла за потребителя **news**; правата за достъп до него няма да се променят.



Фигура 21-1: Потока от новини през relaynews

Инсталиране

Системата C News предварително е пакетирана за всяка модерна дистрибуция на Linux, така че инсталирането ѝ е лесно. Разбира се, винаги можете да използвате за инсталиране оригиналната дистрибуция с изходен код[1]. Независимо как сте я инсталирали, ще трябва да редактирате конфигурационните ѝ файлове. Техните формати са описани в следния списък:

sys файлът *sys* задава кои групи по интереси приема и препредава вашият сайт. Ще разгледаме подробно този файл в следващия раздел.

[1] Можете да изтеглите дистрибуцията с изходен код на C News от сайта на пакета **ftp.cs.toronto.edu** в директорията */pub/c-news/c-news.tar.Z*.

active

обикновено не се редактира за администриране; съдържа указания за обработването на статиите във всяка група по интереси, поддържана от сайта.

organization

Този файл трябва да съдържа името на вашата организация, например "Виртуална пивоварна фабрика" ООД. На вашият домашен компютър въведете "private site" (частен сайт) или каквото и да било друго, което ви допада. Повечето хора няма да считат вашия сайт за правилно конфигуриран, ако не сте въвели информация тук.

newsgroups

Този файл представлява списък на всички групи по интереси, с кратко описание за предназначението на всяка една от тях. Тези описания често се използват от вашия четец на новини, когато показва списъка от всички групи, за които сте абониран.

mailname

Пощенското име на вашия сайт, например **vbrew.com**

whoami

Името на сайта ви, използвано от системите за новини. Много често се използва името на UUCP сайт, например **vbrew**.

explist

Вероятно ще трябва да редактирате този файл, за да изберете желаното от вас време за изтичане на срока на статиите в определени групи по интереси. Обемът на дисковото ви пространство играе важна роля при този избор.

За да създадете начална йерархия от групи по интереси, изтеглете файловете *active* и *newsgroups* от сайта, който ви захранва. Инсталирайте ги в */etc/news*, като се уверите, че те са собственост на **news** и промените правата за достъп на 644 като използвате командата *chmod*. Премахнете всички групи *to*.* от файла *active*, и добавете *to.моя-сайт*, *to.захранващия-сайт*, *junk* и *control*. Групите *to*.* обикновено се използват за обмен на *ihave/sendme* съобщения, но трябва да ги изброите, независимо дали ще използвате *ihave/sendme* или не.

След това заместете всички номера на статии, намиращи се във второто и третото поле на файла *active*. За целта използвайте следните команди:

```
# cp active active.old

# sed 's/ [0-9]* [0-9]* / 0000000000 00001 /' active.old > active
# rm active.old
```

Втората команда стартира редактора на потоци *sed*. С нея заместваме двата низа от цифри съответно с низа от десет нули и с низа 00001.

Накрая създайте директория-буфер за новини и поддиректориите за входящите и изходящите новини:

```
# cd /var/spool
# mkdir news news/in.coming news/out.going news/out.master
# chown -R news.news news
# chmod -R 755 news
```

Ако използвате предварително компилиран четец на новини от различна дистрибуция от тази на C News сървъра, с който работите, възможно е той да очаква буфера за новините да е в */usr/spool/news*, а не във */var/ spool/news*. Ако вашият четец не може да намери никакви статии, направете символна връзка от */usr/spool/news* към */var/ spool/ news* по следния начин:

```
# ln -sf /var/spool/news /usr/spool/news
```

Сега вече сте готови да получавате новини. Обърнете внимание, че не е нужно да създавате буферна директория за всяка група по интереси. C News автоматично създава буферна директория за всяка група, за която приема статия, ако такава директория не съществува.

В частност това става и за *всички* групи, за които е препредадена статия. Така че, след известно време вашият буфер за новини ще се напълни с директории за групи по интереси, за които никога не сте се абонирали, например *alt.lang.teco*. Това може да се предотврати или като премахнете всички нежелани групи от файла *active*, или като редовно стартирате shell-скрипт, който премахва всички празни директории, намиращи се във */var/spool/news* (с изключение на *out.going* и *in.coming*, разбира се).

C News се нуждае от потребител, на който да изпраща съобщения за грешки и доклади за състоянието си. По подразбиране това е **usenet**. Ако използвате подразбиращата се стойност, трябва да направите псевдоним, който да препраща цялата негова поща към един или няколко човека, грижещи се за функционирането на системата. Можете да промените този начин на поведение, като запишете в променлива-

та от обкръжението NEWSMASTER съответното име. Тези промени трябва да се извършват във файла *crontab* на **news**, а също така и всеки път, когато стартирате инструмента за администриране ръчно, така че инсталирането на псевдоним би било доста по-лесно. Пощенските псевдоними са описани в Глава 18, *Sendmail* и в Глава 19, *Конфигуриране и използване на exim*.

Докато редактирате файла */etc/passwd* се уверете, че всеки потребител е записал истинското си име в полето pw_gecos на файла с пароли (това е четвъртото поле от файла). Въпрос на мрежов етикет за Usenet е истинското име на всеки изпращач да присъства в полето From: на статията. Разбира се, бихте искали същото и когато изпращате поща.

Файлът sys

Файлът sys, намиращ се в */etc/news*, управлява кои йерархии приемате и препредавате към други сайтове. Въпреки че съществуват инструменти за поддръжката му, наречени *addfeed* и *delfeed*, смятаме, че е по добре да поддържате този файл ръчно.

Файлът *sys* съдържа записи за всеки сайт, към който препредавате новини, а също и описание на групите, които приемате. Първият ред е запис ME, който описва вашата система. Най-сигурно е да използвате следния запис:

```
ME:all/all::
```

Освен това, трябва да добавите по един ред за всеки сайт, който захранвате с новини. Всеки ред изглежда по следния начин:

сайт[*/изключения*]*:списък-с-групи*[*/списък-за-разпространение*]
[*:флагове*[*:команди*]]

Записите могат да преминават и на нов ред. За целта трябва да използвате обратна наклонена черта (\) в края на всеки ред, който желаете да продължите на нов ред. С помощта на знака диез (#) се обозначават коментари.

сайт

Това е името на сайта, за който се отнася записа. Обикновено се избира UUCP името на сайта. Във файла *sys* трябва да присъства и запис с името на вашия сайт, в противен случай няма да можете да приемате никакви статии.

Специалното име `ME` указва вашия сайт. Записът `ME` дефинира всички групи, които желаете да съхранявате локално. Статиите, несъответстващи на реда `ME`, се изпращат в групата *junk*.

C News отхвърля всяка статия, която вече е преминала през този сайт, с цел да се избегнат безкрайни цикли. C News прави това, като проверява дали името на локалния сайт се намира в полето `Path:` на статията. Някои сайтове имат множество валидни имена. Например, някои сайтове използват в това поле своите пълни домейн-имена (FQDN) или псевдоними като `news.site.domain`. За да сте напълно сигурни, че механизмът за предотвратяване на безкрайните цикли работи, много важно е да добавите всички псевдоними на даден сайт в списъка с изключенията, като ги разделите със запетайки.

Например, записът за сайта **moria** ще съдържа в полето `site` следното: `moria/moria.orcnet.org`. Ако сайтът **moria** има и псевдоним **news.orcnet.org**, тогава полето *site* трябва да съдържа `moria/moria.orcnet.org,news.orcnet.org`.

списък-с-групи

Това е разделен със запетайки списък, включващ групите и йерархиите, за които е абониран този сайт. Йерархиите се определят, като се задава префикса на йерархията (например, *comp.os* за всички групи, чиито имена започват с този префикс), по желание следван от ключовата дума *all* (например *comp.os.all*).

Можете да изключите определена йерархия или група от предаване, като сложите удивителен знак пред името ѝ. Когато групата по интереси се сравнява с този списък, предимство се дава на най-дългите записи. Например, ако списъкът с групи съдържа следното:

```
!comp,comp.os.linux,comp.folklore.computers
```

този сайт няма да се захранва с никакви групи от йерархията *comp*, с изключение на групите от йерархиите, намиращи се в *comp.os. linux* и *comp.folklore.computers*.

Ако сайтът желае да му препредавате всички новини, които приемате, запишете `all` в *списъка с групи*.

списък-за-разпространение

Тази стойност е допълнение към *списъка-с-групи*, което се отделя от него с наклонена черта и съдържа списък за разпространение, който се препредава. Отново можете да изключите опре-

делени разпространения от списъка, като поставите пред тях удивителен знак. Всички списъци за разпространение се обозначават с `all`. Ако пропуснете списъка за разпространение, по подразбиране се прилага `all`.

Например, бихте могли да ползвате списък за разпространение `all`, `!local` за да предотвратите изпращането на локалните новини до отдалечени сайтове.

Обикновено има най-малко две списъка за разпространение: `world`, който често се използва по подразбиране, когато няма списък за разпространение, определен от потребителя, и `local`. Възможно е да съществуват и други списъци за разпространение, прилагани към определен регион, щат, страна и т.н. И накрая, има два списъка за разпространение, които се използват само от C News; това са `sendme` и `ihave`. Те се използват при протокола ihave/sendme.

Използването на списъци за разпространение е обект на дискусии. Полето за разпространение в статията може да съдържа произволни данни, но за да бъде разпространението ефективно, сървърът за новини в мрежата трябва да ги познава. Някои неправилно работещи четци на новини създават фалшиви списъци за разпространение, като просто предполагат, че йерархията от групи по интереси, която е от най-високо ниво представлява един допустим списък за разпространение. Например, един такъв четец би могъл да приеме *comp* за списък за разпространение, който да използва, когато публикува новини до групата *comp.os.linux.networking*. Списъците за разпространение, прилагани за определени региони често също са съмнителни, защото е напълно възможно при пътуването си през Интернет новините да напуснат вашия регион[1]. Списъците за разпространение, използвани за организации, обаче имат голямо значение; например, за да предотвратят напускането на поверителна информация извън пределите на мрежата на компанията. За тази цел обаче е по-добре да създадете отделна група по интереси или йерархия.

[1] Не е необичайно за статия, публикувана да речем в Хамбург, да отиде до Франкфурт през **reston.ans.net**, който се намира в Холандия или дори през някой сайт в САЩ.

флагове

Тази опция описва определени параметри, отнасящи се до захранването с новини. Тя може да е празна или да е комбинация от следните флагове:

`F` Този флаг разрешава пакетирането.

`f` Този флаг е почти идентичен с флага F, но позволява на C News да изчислява размера на изходящите пакети по-точно и затова е за предпочитане да се използва.

`I` Този флаг указва на C News да създаде списък със статии, подходящ за използване от *ihave/sendme*. За да разрешите *ihave/sendme* се изискват допълнителни модификации на файловете *sys* и *batchparms*.

`n` Този флаг създава пакетни файлове за активни NNTP клиенти за прехвърляне, например *nntpxmit* (виж Глава 22, *NNTP и демона nntpd*). Пакетните файлове съдържат името на статията и нейния идентификационен номер.

`L` Указва на C News да предава само статии, публикувани от вашия сайт. Този флаг може да бъде следван от десетично число *n*, указващо на C News да предава статии, публикувани в рамките на *n* ретранслации от вашия сайт. C News определя броя на стъпките от полето `Path`.

`u` Указва на C News да пакетира само статии, взети от неарбитрирани групи.

`m` Указва на C News да пакетира само статии, взети от арбитрирани групи.

Можете да използвате най-много един от флаговете `F`, `f`, `I` или `n`.

команди

Това поле съдържа команда, която ще се изпълни за всяка статия, освен ако не сте разрешили пакетирането. Статията ще се предаде на командата чрез стандартния ѝ вход. Това трябва да се използва само при осъществяването на много малки захранвания, в противен случай натоварването на системите ще бъде доста голямо.

Командата по подразбиране е:

```
uux - -r -z отдалечена-система!rnews
```

Този ред ще стартира командата *rnews* на отдалечената система и ще я захрани със статията от стандартния вход.

Пътят за търсене по подразбиране за командите, зададени в това поле е */bin:/usr/bin:/usr/lib/news/batch*. Последната директория съдържа много shell-скриптове, чиито имена започват с *via*; те се описват накратко по-долу в тази глава.

Ако с използването на един от флаговете `F`, `f`, `I` или `n` пакетирането е разрешено, CNews очаква да намери в това поле име на файл, а не команда. Ако това име на файл не започва с наклонена черта (/), се предполага, че то е относително спрямо */var/spool/news/out.going*. Ако полето е празно, по подразбиране се използва *remote-system/togo*. Очаква се файлът да е в същия формат както файла *remote-system/togo* и да съдържа списък със статиите за предаване.

Когато настройвате C News, най-вероятно ще се наложи да напишете свой собствен sys файл. Ето един примерен файл за **vbrew.com**, от който бихте могли да копирате всичко, от което се нуждаете:

```
# Взимаме всичко, което получаваме.
ME:all/all::
# Изпращаме към moria всичко, което приемаме, с изключение
# на локалните статии и тези, които се отнасят до пивоварството.
# Използваме пакетиране.
moria/moria.orcnet.org:all,!to,to.moria/all,!local,!brewery:f:
# Изпращаме по пощата comp.risks до jack@ponderosa.uucp
ponderosa:comp.risks/all::rmail jack@ponderosa.uucp
# swim получава минимално захранване
swim/swim.towbirds.com:comp.os.linux,rec.humor.oracle/all,!local:f:
# Записваме статиите за по-нататъшна обработка
# в системния дневник mail map
usenet-maps:comp.mail.maps/all:F:/var/spool/uumaps/work/batch
```

Файлът active

Файлът *active* се намира в директорията */etc/* и описва всички групи, известни на вашия сайт и текущо активните статии. Рядко ще се налага да го променяте, но ние ще го разгледаме за пълнота на изложението. Записите имат следния вид:

```
newsgroup high low perm
```

`newsgroup` е името на групата. *low* и *high* са най-малкият и най-големият брой статии, достъпни в момента. Ако няма такива *low* е равно на *high*+1.

Поне първоначално това е било предназначението на полето `low`. За по-голяма ефективност, обаче, C News не го актуализира. Това не би било толкова голяма загуба, ако не съществуваха четци на новини, които разчитат на данните в него. Например *trn* проверява това поле, за да види дали може да изчисти някои статии от своята база данни с нишки. За да актуализирате полето `low` се налага да изпълнявате редовно командата *updatemin* (или в някои по-ранни версии на C News скрипта *upact*).

`perm` е параметър, показващ че на потребителите с достъп им е разрешено да участват в групата. Той може да приеме една от следните стойности:

`y` Потребителите имат право да публикуват статии в тази група.

`n` Потребителите нямат право да публикуват статии в тази група. Все пак, те имат право да четат от нея.

`x` Тази група е блокирана локално. Това се случва понякога, когато администраторите на новини (или техните началници) се чувстват обидени от статии, публикувани в определени групи.

Статии, приемани от тази група не се съхраняват локално, въпреки че се препредават до сайтове, които ги изискват.

`m` Указва арбитрирана група. Когато потребител се опитва да публикува статия в тази група, интелигентният четец го уведомява, че групата е арбитрирана и изпраща статията до арбитъра, вместо до групата. Адресът на арбитъра се взима от файла *moderator*, намиращ се във */var/lib/news*.

`newsgroup=real-group`

Маркира `newsgroup` като локален псевдоним на друга група с име `real-group`. Всички статии, публикувани в `newsgroup`, ще бъдат пренасочени към real-group.

В C News в повечето случаи няма да ви се налага да редактирате този файл директно. Групите могат да бъдат добавени или изтрити локално като използвате *addgroup* и *delgroup* (вижте раздела "Задачи и инструменти за поддръжка" по-долу в тази глава). Управляващото съобщение `newgroup` добавя група за цялата мрежа Usenet, а съобщението `rmgroup` изтрива група. *Никога не изпращайте сами такова съобщение!* Указания за начина, по който се създава нова група, може да намерите в ежемесечните публикации в *news.announce.newusers*.

Файлът *active.times* е тясно свързан с файла *active*. Всеки път, когато се създава група, C News записва в този файл съобщение с името на създадената група, датата на създаването ѝ, дали е създадена с контролното съобщение `newgroup` или локално и кой я е създал. Това е много удобно за четците на новини, които имат възможност да уведомяват потребителя за всички наскоро създадени групи и се използва също и от командата *NEWGROUPS* на NNTP .

Пакетиране на статии

Пакетите с новини спазват специален формат, който е еднакъв за B News, C News и INN. Всяка статия се предхожда от следния ред:

```
#! rnews count
```

count е броят байтове в статията. Когато използвате компресиране на пакети, полученият файл се компресира като едно цяло и се предхожда от още един ред, посочващ какъв инструмент трябва да се използва за разпакетирането му. Стандартния инструмент за компресиране е *compress* и се обозначава последния начин:

```
#! cunbatch
```

Понякога, когато сървърът за новини изпраща пакети чрез пощенски софтуер, който премахва осмия бит от всички данни, компресираният пакет може да бъде защитен с използването на т.нар. *c7-кодиране*. Такива пакети се маркират със *c7unbatch*.

Когато пакет се подава на *mews* на отдалечен сайт, той проверява тези означения и обработва пакета по подходящ начин. Някои сайтове използват и други компресиращи инструменти като *gzip* и имената на техните *gzip*-файлове се предхождат от думата *zunbatch*. C News не разпознават подобни нестандартни заглавия; вие трябва да модифицирате изходния код, за да ги поддържате.

В C News пакетирането на статии се извършва от командата */usr/lib/news/batch/sendbatches*, която взима списък със статиите от файла *site/togo* и ги разполага в няколко пакета с новини. Тази команда трябва да се изпълнява на всеки час или дори по-често, в зависимост от обема на трафика. Работата ѝ се контролира от файла *batchpams*, който се намира в */var/lib/news*. Този файл описва максимално допустимия размер на пакета за всеки сайт, програмите за пакетиране и евентуално компресирането, което трябва да се използва, както и транспорта за доставяне до отдалечения сайт. Бихте могли да определите параметриге на пакетирането както за всеки сайт поотделно,

така и като набор параметри по подразбиране за сайтове, които не са споменати изрично.

Когато инсталирате C News, най-вероятно ще намерите във вашата дистрибуция файл *batchparms* с настроени по подразбиране параметри, така че шансът да не се налага да редактирате този файл е доста голям. За всеки случай ние ще опишем неговия формат. Всеки ред съдържа шест полета, разделени от интервали или символи за табулация:

```
site size max batcher muncher transport
```

site

това е името на сайта, за който важи записа. Файлът *togo* за този сайт трябва да се намира в *out.going/togo* в буфера за новини. Името `/default/` указва подразбиращ се запис и съответства на всяко име, което не е изрично указано със съответен запис.

size

това е максималният размер на създадените пакети от статии (преди компресирането). За статии, по-големи от този размер, C News прави изключение и пакетира всяка една от тях поотделно.

max

това е максималният брой създадени и планирани за предаване пакети, преди да бъде преустановено пакетирането за съответния сайт. Това е доста полезно, в случай че отдалеченият сайт трябва да бъде спрян за дълго време, защото предотвратява препълването на вашите буферни UUCP директории с безкраен брой пакети с новини.

C News определя броя на пакетите в опашката чрез скрипта *queuelen*, намиращ се в */usr/lib/news*. Ако сте инсталирали C News в предварително пакетиран формат, скриптът не би трябвало да се нуждае от промяна, но ако изберете да използвате друга разновидност на буферните директории, например Taylor UUCP, може да ви се наложи да напишете свой скрипт. Ако броят на буферните файлове не ви притеснява (например, защото сте единственият, който използва компютъра и не пишете статии от порядъка на мегабайти), можете да замените съдържанието на скрипта с оператора *exit 0*.

`batcher`

това поле съдържа командата, използвана за създаване на пакет от списъка със статии, намиращ се във файла *togo*. За стандартни захранвания това обикновено е *batcher*. За други цели могат да бъдат предоставени алтернативни команди. Например, протоколът *ihave/sendme* изисква списъкът със статии да се преобразува в управляващи *ihave* или *sendme* съобщения, които се публикуват до групата по интереси *to.site*. Това се извършва от *batchih* и *batchsm*.

`muncher`

това поле задава командата за компресиране. Обикновено това е *compcun* - скрипт, който създава компресирани пакети[1]. Можете като алтернатива да използвате програмата за компресиране *gzip* и в това поле да въведете **gzipcun** (обърнете внимание, че ще трябва да го напишете сами). Убедете се, че програмата *uncompress* на отдалечения сайт е версия, разпознаваща файлове, компресирани с *gzip*.

Ако отдалеченият сайт не притежава команда *uncompress*, можете да укажете опцията *nocomp*, която не извършва никаква компресия.

`transport`

последното поле описва вида *транспорт*, който трябва да се използва. Съществуват множество стандартни команди за различните видове транспорт, като имената им започват с думата *via*. Чрез командния ред *sendbatches* им предава името на сайта-получател. Ако записът *batchparms* не е `/default/`, *sendbatches* извлича името на сайта от полето `site`, като премахва всичко след първата точка или наклонена черта включително. Ако записът в `batchparms` е `/default/`, използват се имената на директориите в *out.going*.

За да извършите пакетиране за точно определен сайт, използвайте следната команда:

```
# su news -c "/usr/lib/news/batch/sendbatches име_на_сайта"
```

[1] Както се разпространява със C News, *compcun* използва *compress* с 12-битова опция, тъй като това е минималният, поддържан от всички сайтове стандарт. Можете да създадете копие на скрипта, например *compcun16*, за което се използва 16-битова компресия. Все пак подобрението не е особено впечатляващо.

Когато се извиква без аргументи, *sendbatches* обработва всички опашки от пакети. Интерпретацията на "all"("всички") зависи от наличието на запис по подразбиране в *batchparms*. Ако се намери такъв запис, всички директории във *var/spool/news/out.going* се проверяват; в противен случай *sendbatches* циклично преминава през всички записи в *batchpams*, като обработва само сайтовете, намерени там. Обърнете внимание, че сканирайки директорията *out.going*, *sendbatches* обработва само тези директории, които не съдържат точки или знака "at" (@) в имената на сайтове.

Има две команди, които използват *uux*, за да изпълнят *rnews* на отдалечена система: *viauux* и *viauuxz*. Втората команда задава флага -z на *uux*, за да предотврати връщането на съобщение за успешно изпълнение за всяка доставена статия при по-старите версии. Друга команда, *viamail*, изпраща пакетите със статии към потребителя **rnews** на отдалечената система чрез електронна поща. Разбира се, това изисква отдалечената система по някакъв начин да може да захрани своята собствена система за новини с пощата, предназначена за **rnews**. За пълния списък с видовете транспорт се обърнете към справочната страница на *newsbatch*.

Всички команди от последните три полета трябва да се намират или в */out.going/site*, или в */usr/lib/news/batch*. Повечето от тях са скриптове; можете лесно да приспособите нови инструменти за вашите лични нужди. Те се извикват чрез канали (pipes). Списъкът със статиите се подава на batcher през стандартния му вход, което води до създаването на пакет на стандартния изход. Този пакет се подава на стандартния вход на muncher през канала и т.н.

Ето един примерен файл:

```
# файл batchparms за пивоварната фабрика
# site     | size    |max | batcher | muncher | transport
#----------+---------+----+---------+---------+----------
/default     100000  22    batcher   compcun   viauux
swim          10000  10    batcher   nocomp    viauux
```

Изтичане на срока на новините

В B News изтичането на срока на новините се изпълнява от програма, наречена *expire*, която получава като аргументи списък от групи по интереси и спецификация за времето, след което срокът на статиите изтича. За да може срокът на валидност на различни йерархии да е различен, трябва да напишете скрипт, стартиращ *expire* за всяка йерархия поотделно. C News предлага по-удобно решение. Във файл с

име *explist* може да зададете групи по интереси и времето за изтичане на срока им. Команда, наречена *doexpire* обикновено се стартира един път на ден от *cron* и обработва всички групи съгласно този списък.

От време на време, вие бихте могли да поискате да запазвате статии от определени групи, даже и след изтичане на срока им. Например, бихте могли да искате да запазите програми, публикувани в групата *comp.sources.unix*. Това се нарича архивиране (*archiving*). *explist* ви позволява да маркирате определени групи за архивиране.

Един запис във файла explist изглежда по следния начин:

```
grouplist perm times archive
```

`grouplist` е разделен със запетайки списък от групи по интереси, за които се прилага записа. Йерархии могат да се определят, като се зададе префикс на името на групата, който по желание може да бъде следван от `all`. Например, за запис който се прилага за всички групи, намиращи се в **`comp.os`**, въведете **`comp.os`** или **`comp.os.all`**.

При проверката за валидност на статии от група, името се сравнява с всички записи в *explist* по реда, в който те са дадени. Прилага се първия съвпадащ запис. Например, за да изгриете повечето статии от *comp* след период от четири дни, с изключение на статиите от *comp.os.linux*. *announce*, които бихте искали за запазите за срок от седем дни, просто правите запис за последните, който определя седем дневен срок за валидност, последван от запис за *comp*, определящ срок за валидност четири дни.

Полето `perm` определя дали записът се отнася за арбитрирана, неарбитрирана или произволна група. Стойността му може да бъде `m`, `u` или `x`, които обозначават съответно арбитрирана, неарбитрирана или произволна група.

Третого поле (`times`) обикновено съдържа само едно число. Това е броят на дните, след които срокът на статиите изтича, в случай, че не им е определен изкуствен срок на валидност в полето `Expires`: от заглавието на статията. Обърнете внимание, че броят на дните се пресмята от датата на пристигането на съобщението във вашия сайт, а не от датата на публикуването му.

Все пак, полето `times` би могло да бъде доста по-сложно. То може да съдържа комбинация от най-много три числа, разделени едно от друго с тирета. Първого число определя минималния брой дни, след изминаването на които може да изтече срокът на статията, дори ако в

полето `Expire`: срокът вече е изтекъл. Рядко има полза това число да е различно от нула. Второго число е преди това споменатият брой дни по подразбиране, след които срокът на статията изтича. Третото число е броят дни, след които срокът на статията изтича безусловно, независимо от това дали е записано нещо в полето `Expires`: или не. Ако е дадено само второто число, останалите две приемат стойности по подразбиране. Тези стойности могат да бъдат определени като използвате специалния запис `/bounds/`, който е описан малко по-долу.

Четвъртото поле *archive* указва дали групата по интереси ще бъде архивирана и къде точно. Ако не възнамерявате да извършвате архивиране, трябва да използвате тире. В противен случай използвате пълния път (сочещ до директория) или знака "at" (@). Знакът @ указва директория за архивиране по подразбиране, която трябва да се подаде на *doexpire*, като в командния ред се добавя флага *-a*. Архивната директория трябва да бъде собственост на **news**. Когато doexpire архивира статия например от *comp.sources.unix*, тя я записва в директорията *comp/souces/unix*, намираща се в архивната директория, като я създава, ако това е необходимо. Самата архивна директория, обаче, не се създава.

Съществуват два специални записа във вашия файл *explist*, на които разчита *doexpire*. Вместо списък с групи, те съдържат ключовите думи `/bounds/` и `/expired/`. Записът `/bounds/` съдържа стойностите по подразбиране за трите числа от полето *times*, описано преди малко. Полето `/expired/` определя колко дълго C News ще запазва редове във файла *history*. C News няма да премахне редове от файла *history*, след изтичане на срока на съответните им статии, а ще ги запази в случай, че след тази дата получи дубликат. Ако се захранвате с новини само от един сайт, то стойността на полето може да е малка. В противен случай препоръчителният срок за UUCP мрежи е няколко седмици, в зависимост от забавянето, с което получавате статии от тези сайтовете.

Ето един пример на файл *explist* с доста кратки срокове на валидност:

```
# пази записите във файла history две седмици.
# нито една статия не се пази повече от три месеца.
/expired/                   x     14      -
/bounds/                    x     0-1-90  -
# групи, които бихме искали да пазим по-дълго от останалите
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        -
# архивираме всички групи *.sources
comp.sources,alt.sources    x     5         @
# срокове по подразбиране за техническите групи
comp,sci                    x     7         -
# достатъчно за двата дълги почивни дни в седмицата
misc,talk                   x     4         -
# бързо изхвърляне на боклука
junk                        x     1     -
# управляващите съобщения също са от ограничен интерес
control                     x     1     -
# запис, прехващащ всичко останало
all                         x     2     -
```

Изтичането на срока може да доведе до възникването на някои проблеми. Първият е, че вашият четец на новини може да разчита на третото поле от файла *active*, описано по-горе, което съдържа най-малкия брой на активните статии. Когато срокът на статиите изтича, C News не обновява това поле. Ако имате нужда (или искате) това поле да представя реалната ситуация, ще трябва да стартирате програма, наречена *updatemin* след всяко изпълнение на *doexpire*. (В по-старите версии на C News това се извършваше от скрипт, наречен *upact*.)

C News не проверява за изтичане на срока чрез сканиране на директорията на групата по интереси, а просто проверява във файла *history*, дали срокът на статията подлежи на изтичане[1]. Ако по някаква причина файлът *history* не е синхронизиран и не съдържа верни данни, някои статии могат да останат на диска ви вечно, защото C News буквално ще ги е забравила[2]. Можете да поправите това като използвате или скрипта *addmissing*, намиращ се в */usr/lib/news/maint*, който ще добави липсващите статии във файла *history*, или *mkhistory*, който ще построи отново целия файл *history*. Не забравяйте да влезете като потребителя **news** преди да стартирате скрипта, в противен случай ще получите файл *history*, който няма да може да се чете от C News.

[1] Времето на пристигане на статията се съхранява в средното поле на записа и е зададено като изминалите секунди от 1 януари 1970 г.

[2] Не знам *защо* се случва това, но то просто става от време на време.

Други файлове

Има множество файлове, контролиращи поведението на C News, но те не са толкова важни. Всички те се намират в директорията */etc/news*. Ще ги опишем накратко по-долу:

newsgroups

> това е файл, придружаващ файла *active*, който съдържа списък с името на всяка група по интереси и кратко описание на главната ѝ тема. Този файл се обновява автоматично, когато C News получава контролното съобщение `checknews`.

localgroups

> Ако пригежавате много локални групи, можете да сложите техните имена и описания в този файл, точно както ще изглеждат в *newsgroups*, за да не получавате оплаквания от C News винаги, когато получавате контролното съобщение `checkgroups`.

mailpaths

> Този файл съдържа адреса на арбитъра за всяка арбитрирана група. Всеки ред съдържа името на групата, следвано от адреса на електронната поща на арбитъра (отделени със знак за табулация).
>
> По подразбиране се предоставят два специални записа: `backbone` и `internet`. И двата задават пътя до най-близкия сайт на опорната мрежа и до сайта, разпознаващ адреси в стил RFC-822 (*потребител@хост*), записани в стил "бум"-път (bang path). Записите по подразбиране са:
>
> ```
> internet    backbone
> ```
>
> Не е необходимо да променяте записа internet, ако имате инсталирани *exim* или *sendmail*; те разпознават адресирането във формат RFC-822.
>
> Записът `backbone` се използва винаги, когато потребител публикува статия до арбитрирана група, чийто арбитър не е указан изрично. Ако името на групата по интереси е *alt.sewer* и записът `backbone` съдържа *path!%s*, C News ще изпрати статията до *path!alt-sewer*, с надеждата че машина от опорната мрежа е способна да препрати статията. За да разберете какъв път да ползвате, попитайте администратора на новини на сайта, който ви захранва. Като последна възможност можете да използвате и *uunet.uu.net!%s*.

distributions

Това не е истински C News файл, но се използва от някой четци на новини и от демона *nntpd*. Той съдържа списък с разпространенията, които вашият сайт разпознава и описание на (предвиждания) ефект от тях. Например Виртуална пивоварна има следния файл:

```
world   навсякъде по света
local   само локално за този сайт
nl      само за Холандия
mugnet  само за MUGNET
fr      само за Франция
de      само за Германия
brewery само за Виртуалната пивоварна
```

log Този файл съдържа дневник на всички действия на C News. Той се обработва редовно чрез стартирането на *newsdaily*. Копия на старите дневници се пазят в *log.o*, *log.oo* и т.н.

errlog

Това е дневник с всички съобщения за грешка, генерирани от C News. Тук не се включва информация за статии, които са от изхвърлени като ненужни заради това, че са били изпратени до невалидна група или в резултат на други потребителски грешки. Този файл автоматично се изпраща по пощата от *newsdaily* на администратора на новините (по подразбиране **usenet**), освен ако не е празен.

errlog се изчиства от *newsdaily*, *errlog.o* пази старите копия.

batchlog

Този дневник съдържа информация за всички стартирания на програмата *sendbatches*. Обикновено интересът към него е малък. Поддържа се също от *newsdaily*.

watchtime

Това е празен файл, който се създава всеки път, когато се стартира *newsdaily*.

Управляващи съобщения

Протоколът за Usenet новини използва специална категория статии, които предизвикват определена реакция от страна на системата за новини. Те се наричат *управляващи* съобщения. Разпознават се по наличието на полето `Control`: в заглавието на статията, което съдържа името на управляващата операция, която трябва да се изпълни. Съществуват няколко типа такива операции и всички те се обработват от скриптове, намиращи се в */usr/lib/news/ctl*.

Повечето от тези съобщения извършват своята дейност автоматично в момента, в който системата CNews обработва статията, без да уведомява за това администратора на новините. По подразбиране към него се подават само управляващите съобщения `checkgroups`, но вие можете да промените това, като редактирате скриптовете.

Съобщението cancel

Най-добре познатото управляващо съобщение е `cancel`, с което потребителят може да анулира статия, изпратена преди това. Резултатът е, че статията се премахва от буферните директории, ако съществува. Съобщението `cancel` се препредава към всички сайтове, които получават новини от засегнатите групи, независимо от това дали съобщението вече е прочетено. Тук възниква възможността оригиналното съобщение да бъде забавено от съобщението за премахване. Някой системи за новини позволяват на потребителите си да анулират съобщения на други хора. Разбира се, това определено не е желателна практика.

newgroup и rmgroup

Двете съобщения, които се използват за създаването или премахването на групи по интереси, са `newgroup` и `rmgroup`. Групи по интереси могат да се създадат в "обичайните" йерархии само след провеждането на дискусия и гласуване между читателите в Usenet. Правилата, прилагани към йерархията *alt* могат да доведат до пълен безпорядък. За повече информация вижте редовните публикации в *news.announce.newusers* и *news.announce.newgroups*. Никога не изпращайте сами контролни съобщения `newgroup` или `rmgroup`, освен ако наистина не знаете какво правите.

Съобщението checkgroups

Съобщенията `checkgroups` се изпращат от администраторите на новини за синхронизация на файловете *active* на всички сайтове в рамките на една мрежа с реалното състояние на Usenet. Например, комерсиалните доставчици на Интернет могат да изпращат такива съобщения към сайтовете на своите клиенти. Веднъж месечно арбитърът на *comp.announce.newsgroups* публикува "официалното" съобщение `checkgroups` за всички главни йерархии. То обаче се публикува като обикновена статия, а не като управляващо съобщение. За да извършите операцията `checkgroups`, запишете тази статия във файл, например */tmp/check*, махнете всичко до началото на самото управляващо съобщение и го подайте на скрипта `checkgroups`, като използвате следната команда:

```
# su news -c "/usr/lib/news/ctl/chechgroups" < /tmp/check
```

Това ще обнови вашия файл *newsgroups* с новия списък от групи, добавяйки групите, изброени в *localgroups*. Старият файл *newsgroups* ще бъде преместен в *newsgroups.bac*. Забележете, че публикуването на съобщението локално работи много рядко, тъй като *inews* - командата приемаща и публикува статии от потребители, отказва да приеме толкова голяма по обем статия.

Ако системата C News намери несъответствия между списъка `checkgroups` и файла *active*, тя създава списък с команди, които ще обновят вашия сайт и го изпраща на администратора на новините.

Обикновено резултатът изглежда по следния начин:

```
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/maint/delgroup alt.ascii-art
         /usr/lib/news/maint/delgroup bionet.molbio.gene-org
         /usr/lib/news/maint/delgroup comp.windows.x.intrisics
         /usr/lib/news/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
```

Когато получите подобно съобщение от вашата система за новини, не му се доверявайте автоматично. В зависимост от това, кой е изпратил съобщението `checkgroups`, в него могат да липсват няколко групи или дори цели йерархии. Трябва да сте внимателни при премахването на групи. Ако разберете, че групи, които искате да поддържате на вашия сайт, са обявени за липсващи, ще трябва да ги добавите с помощта на скрипта *addgroups*. Съхранете списъка с липсващите групи във файл и го подайте на следния малък скрипт:

```
#!/bin/sh
#
WHOIAM=`whoami`
if [ "$WHOIAM" != "news" ]
then
        echo "You must run $0 as user 'news'" >&2
        exit 1
fi
#
cd /usr/lib/news
while read group; do
    if grep -si "^$group[[:space:]].*moderated" newsgroup; then
        mod=m
    else
        mod=y
    fi
    /usr/lib/news/maint/addgroup $group $mod
done
```

sendsys, version и senduuname

Съществуват още три съобщения, които могат да ви помогнат да разберете мрежовата топология. Това са `sendsys`, `version` и `senduuname`. Те указват на CNews да върне на изпращача съответно файла sys, низ, съдържащ информация за версията на софтуера и резултата от *uuname*. CNews е доста лаконична относно съобщенията `version`. Тя връща просто едно неукрасено C.

Още веднъж ще ви предупредим, че *никога* не трябва да генерирате такива съобщения, освен ако не сте абсолютно сигурни, че те няма да напуснат вашата (регионална) мрежа. Отговорите до съобщенията `sendsys` могат много бързо да претоварят една UUCP мрежа[1].

C News в NFS среда

Прост начин да разпространявате новини в рамките на локална мрежа е да ги съхранявате на централен хост и да предоставите съответните директории чрез NFS, така че четците на новини да могат да сканират статиите директно. Натоварването, свързано с изтеглянето и обработката на статии е значително по-ниско, отколкото при NNTP. От друга страна, NNTP е за предпочитане при хетерогенна мрежа, където оборудването е различно при различните хостове или където потребителите нямат еквивалентни акаунти на сървъра.

Когато използвате NFS, статиите, публикувани на локалния хост, трябва да се препредадат към централната машина, защото достъпа до административните файлове може да изложи системата на т.нар. състезателни условия (racing conditions) и като резултат между файловете ще останат вътрешно несъгласувани. Освен това, можете да защитите областта на буфера си за новини, като го предоставите във вариант само за четене, което също изисква препредаване към централната машина.

C News обработва конфигурацията на тази централна машина по начин, напълно прозрачен за потребителя. Когато публикувате статия, вашият четец на новини обикновено стартира *inews*, за да постави статията в системата за новини. Тази команда предизвиква определен брой проверки на статията, оформя заглавието и проверява файла *server* в */etc/news*. Ако такъв файл съществува и той съдържа име на хост, различно от това на локалния хост, на този хост чрез *rsh* се стартира *inews*. Тъй като скриптът *inews* използва множество двоични команди и поддържащи файлове от C News, ще трябва или да инсталирате C News локално или да монтирате софтуера за новини от сървъра.

[1] Аз не бих опитал това и в Интернет.

За да може *rsh* да работи правилно, всеки потребител, който публикува новини, трябва да има еквивалентен акаунт в системата на сървъра, т.е такъв, с който може да влиза в системата без да се налага да въвежда парола.

Уверете се, че името на хоста, което сте дали в *server*, съвпада точно с резултата от командата *hostname*, изпълнена на сървъра. В противен случай C News ще зацикли безкрайно в опит да достави статията. Разгледахме подробно NFS в Глава 14, "*Мрежова файлова система*".

Задачи и инструменти за поддръжка

Независимо от сложността на C News, животът на администратора на новините може да е сравнително лесен. C News ви предоставя голямо разнообразие от инструменти за поддръжка. Някои от тях са предназначени да се стартират редовно от *cron*, например *newsdaily*. Използването на тези скриптове значително намалява ежедневните грижи и изискванията за захранване на вашата инсталация на C News.

Ако не е изрично спомената друго, тези команди се намират в */usr/lib/news/maint*. (Забележка: трябва да влезете като потребителя **news** преди да стартирате тези команди. Ако ги стартирате като суперпотребител можете да направите критични файлове за новини недостъпни за C News):

newsdaily

> Името ѝ го подсказва: изпълнявайте тази програма веднъж дневно. Това е важен скрипт, който ви помага да поддържате вашите системни дневници малки, като съхранява копия на всеки един от тях от последните три стартирания. Освен това, програмата се опитва да разпознае аномалии като стари пакети в директориите за входящи и изходящи статии, публикуване на статии до несъществуващи групи или арбитрирани групи и т.н. Получените съобщения за грешки се изпращат на администратора на новините.

newswatch

> Този скрипт трябва да се стартира редовно, за да търси аномалии в системата за новини, например на всеки час. Той е проектиран да открива проблеми, които директно се отразяват на работоспособността на системата ви за новини и ако случаят е такъв, изпраща отчет с откритите грешки на администратора на новините. Проверките включват стари заключващи файлове, които не са

били изтрити, ненаблюдавани входни пакети и недостиг на дисково пространство.

addgroup

Този скрипт добавя група локално към вашия сайт. Правилното му извикване е:

```
addgroup име_на_групата y|n|m|=истинска_група
```

Вторият аргумент има същото значение както флагът във файла *active*, а именно, че всеки може да публикува статии до групата (`y`), никой не може да публикува (`n`), групата е арбитрирана (`m`), или че е псевдоним за друга група (`=realgroup`). Можете също да използвате *addgroup*, когато първите статии в новосъздадена група пристигнат *преди* управляващото съобщение `newgroup`, предназначено да създаде групата.

delgroup

Този скрипт ви позволява локално да изтриете група. Стартирайте го така:

```
delgroup име_на_групата
```

След това трябва да изтриете статиите, които остават в буферната директорията на групата. Другата възможност е да ги оставите на естествения ход на събитията (т.е. изтичане на срока им), за да ги премахнете.

addmissing

Този скрипт добавя липсващи статии във файла *history*. Стартирайте го, когато едни и същи статии висят непрекъснато в системата.

newsboot

Този скрипт се стартира по време на начално зареждане на системата. Той премахва всякакви заключващи файлове, останали след спирането на процесите на новините при изключването на системата и затваря и изпълнява всички пакети, останали от NNTP връзки, които са били прекъснати при спирането на системата.

newsrunning

Този скрипт се намира в */usr/lib/news/input* и може да бъде използван, за да забрани разпакетирането на входящите новини, например през работно време. Можете да спрете разпакетирането по следния начин:

```
/usr/lib/news/input/newsrunning off
```

Разпакетирането се разрешава отново, като използвате *on* вместо `off`.

ГЛАВА 22

NNTP И ДЕМОНА NNTPD

Протоколът NNTP (Network News Transfer Protocol – мрежов протокол за прехвърляне на новини) предоставя коренно различен подход за обмен на новини от C News и други сървъри за новини, които нямат директна поддръжка на NNTP. Вместо като UUCP да разчита на пакетирането за пренасяне на статии между машините, този протокол позволява статиите да се обменят през интерактивна мрежова връзка. NNTP е не отделен софтуерен пакет, а Интернет стандарт, описан в документа RFC-977. Той се основава на поточно-ориентирана връзка, обикновено през TCP, между клиент, който се намира някъде в мрежата и сървър на хост, който пази мрежовите новини на своя диск. Поточната връзка позволява на клиента и сървъра интерактивно да се договарят за преноса на статиите, почти без закъснение за обратно предаване, като по този начин запазва броя на дублираните статии нисък. Заедно с високата скорост на пренасяне по Интернет, пренасянето на новини чрез NNTP многократно надвишава по възможности оригиналните UUCP мрежи. Докато преди години не беше необичайно да минат две седмици, преди статията да достигне до най-отдалечените кътчета на Usenet, днес това често става за по-малко от два дни. През самият Интернет статията се пренася дори за минути.

Разнообразни команди позволяват на клиента да извлича, изпраща и публикува статии. Разликата между изпращане и публикуване е, че при публикуване може статията да е с непълна информация в заглавието. По принцип това означава, че потребителят просто е написал

статията.[57] Извличането на статии може да се използва както от клиенти за прехвърляне на новини, така и от четците на новини. Това прави NNTP отличен инструмент за осигуряване на достъп до новини за много клиенти на локалната мрежа като се избягват усложненията при използване на NFS.

Освен това, NNTP предвижда активен и пасивен начин на пренасяне на новини, съответно наричани "разпространение" (pushing) и "изтегляне" (pulling). Разпространението в основни линии е същото като в протокола ihave/sendme, използван от *C*News (описан в Глава 21, *C News*). Клиентът предлага статия на сървъра чрез командата *IHAVE msgid*, а сървърът като отговор връща код, който показва дали вече има тази статия и дали я иска. Ако сървърът иска статията, клиентът я изпраща, като обозначава края ѝ с една точка на отделен ред.

Разпространението на новини има един единствен недостатък и това е голямото натоварване на сървъра, тъй като системата трябва да претърси базата данни с историята си за всяка отделна статия.

Противоположната техника се нарича изтегляне на новини, като при нея клиентът прави заявка за списъка на всички (достъпни) статии от определена група, които са пристигнали след определена дата. Това запитване се извършва от командата *NEWNEWS*. От получения списък с идентификатори на съобщенията клиентът избира само тези статии, които все още не притежава, като използва последователно за всяка една от тях командата *ARTICLE*.

Изтеглянето на новини изисква строг контрол от страна на сървъра, за това кои групи и списъци за разпространение са достъпни за клиентите. Например, трябва да сте абсолютно сигурни, че не се изпраща никаква поверителна информация от локалните групи по интереси към непривилегировани клиенти.

Съществуват и множество удобни команди, които позволяват на четците на новини да извличат поотделно заглавието и тялото на статията, или дори отделни редове на заглавието за избрани статии. Това ви дава възможност да пазите всички новини на централен хост, като всички клиенти на (вероятно локалната) мрежа използват NNTP-базирани клиентски програми за четене и изпращане на статии. Това е алтернатива на предоставянето на директориите с новини през NFS, което е описано в Глава 21.

---

[57] Когато публикувате статия през NNTP, сървърът винаги добавя поне едно поле в заглавието – `NNTP-Posting-Host`: Това поле съдържа името на хоста на клиента.

Главния проблем на NNTP е, че позволява на достатъчно умели хора да вмъкват в потока с новини статии с фалшиво зададен подател. Това се нарича *фалшифициране на новини* или мамене (*spoofing*).[58] Едно разширение към NNTP ви позволява да изисквате потребителя да се идентифицира за определени команди, като ви осигурява защита от хора, които по този начин злоупотребяват с вашия сървър за новини.

Съществуват множество NNTP пакети. Един от най-широко разпространените е демонът NNTP, известен още като *справочна реализация*. Той е написан от Stan Barber и Phil Lapsley, за да илюстрира подробностите на стандарта RFC-977. Както всеки добър софтуер, достъпен днес, можете да го намерите в предварително пакетиран вид за вашата дистрибуция на Linux, или ако желаете, да се сдобиете с изходния му код, който да компилирате сами. Ако решите да го компилирате лично, ще трябва да сте напълно запознати с вашата дистрибуция, за да е съвсем сигурно, че ще настроите пътищата до всички файлове правилно.

Пакетът с демона *nntpd* съдържа сървър, два клиента за изтегляне и разпространение на новини и заместител на *inews*. Естественото им обкръжение е B News, но с малко настройки няма да имате проблем и в среда C News. Ако обаче планирате да използвате NNTP за нещо повече от това да предлагате на вашите четци достъп до сървъра с новини, справочната реализация не е добро решение. Затова ще разгледаме само демонът NNTP, включен в пакета *nntpd*, без да се спираме на клиентските програми.

Ако възнамерявате да управлявате голям сайт за новини, би трябвало да се запознаете с пакета INN (*InterNet News* – Интернет Новини), написан от Rich Salz. Той осигурява както NNTP, така и UUCP-базиран транспорт за новини. Неговият транспорт за новини определено е по-добър от *nntpd*. Ще разгледаме подробно INN в Глава 23, *Интернет Новини*.

Протоколът NNTP

Както споменахме по-рано, има две NNTP команди, които са основни при разпространението и изтеглянето на статии между сървърите.

[58] Същият проблем съществува и при протокола SMTP (Simple Mail Transfer Protocol), въпреки че повечето агенти за прехвърляне на поща имат механизъм, който предотвратява маменето.

Сега ще ги разгледаме в контекста на реална NNTP сесия, за да ви покажем простотата на този протокол. За целите на нашия пример ще използваме прост *telnet* клиент, за да се свържем към INN-базиран сървър за новини в нашата Виртуална пивоварна, наречен **news.vbrew.com.** Сървърът работи с минимална конфигурация, за да бъдат примерите по-кратки. Ще разгледаме как се извършва пълната конфигурация на такъв сървър в Глава 23. При нашия тест ще бъдем много внимателни и ще генерираме статии само в групата *junk*, за да не тревожим останалите потребители.

Свързване към сървъра за новини

Свързването към сървъра за новини представлява просто създаване на TCP връзка към неговия NNTP порт. Когато се свържете, ще получите начално поздравително съобщение. Една от първите команди, които бихте могли да опитате е `help` (помощ). Отговорът, който получавате, зависи основно от това, дали сървърът ви смята за отдалечен NNTP сървър или за четец на новини, тъй като в двата случая се използват различни набори команди. Можете да смените своя режим на работа с командата *mode*; ще разгледаме тази възможност след малко.

```
$ telnet news.vbrew.com nntp
Trying 172.16.1.1...
Connected to localhost.
Escape character is '^]'.
200 news.vbrew.com InterNetNews server INN 1.7.2 08-Dec-1997 ready
help
100 Legal commands
   authinfo
      help
      ihave
      check
      takethis
      list
      mode
      xmode
      quit
      head
      stat
      xbatch
      xpath
      xreplic
For more information, contact "usenet" at this machine.
.
```

Отговорите на NNTP команди винаги завършват с точка (.) на отделен ред. Числата, които виждате в генерирания листинг, са *кодове за отговор*, които посочват дали командата е завършила с успех или неуспех. Тези кодове са описани в RFC-977; ще поговорим за по-важните от тях след малко.

Разпространение на статия към сървъра

Споменахме командата *IHAVE*, когато говорихме за разпространението на статии към сървъра за новини. Сега нека погледнем как работи тази команда в действителност:

```
ihave <123456@gw.vk2ktj.ampr.org>
335
From: terry@gw.vk2ktj.ampr.org
Subject: test message sent with ihave
Newsgroups: junk
Distribution: world
Path: gw.vk2ktj.ampr.org
Date: 26 April 1999
Message-ID: <123456@gw.vk2ktj.ampr.org>
Body:

Това е тестово съобщение, изпратено с използването на NNTP командата
IHAVE.
.
235
```

Във всички NNTP команди главните и малките букви не се различават, така че няма значение дали ги въвеждате с малки или с главни букви. Командата *IHAVE* има един задължителен аргумент и това е идентификационният номер на статията, която се разпространява. При създаването ѝ, на всяка статия се задава уникален идентификационен номер. Командата *IHAVE* осигурява начин NNTP сървъра да обяви кои статии притежава, когато желае да разпространи статии към друг сървър. Изпращащият сървър ще издаде команда *IHAVE* за всяка статия, която желае да разпространи. Ако кодът за отговор, генериран от приемащия NNTP сървър е от вида “3xx”, изпращащият NNTP сървър ще предаде цялата статия, включително цялото ѝ заглавие и ще завърши статията с една точка на отделен ред. Ако кодът на отговор е от вида “4xx”, приемащият сървър е избрал да не приема тази статия или защото вече я има, или поради недостиг на дисково пространство.

Когато статията е предадена, приемащият сървър връща друг код за отговор, който показва дали предаването е било успешно.

Преминаване към режим за четене NNRP

Четците на новини използват свой собствен набор от команди, когато комуникират със сървъра. За да се активират тези команди, сървърът трябва да работи в режим *за четец* (*reader* mode). Повечето сървъри за новини по подразбиране са в режим за четец, освен ако IP адресът на свързващия се хост е указан като възел за препредаване на новини. При всички случаи NNTP предоставя команда за изрично превключване в режим за четец:

```
mode reader
200 news.vbrew.com InterNetNews NNRP server INN 1.7.2 08-Dec-1997
ready/
    (posting ok).
help
100 Legal commands
  authinfo user Name|pass Password|generic <prog> <args>
  article [MessageID|Number]
  body [MessageID|Number]
  date
  group newsgroup
  head [MessageID|Number]
  help
  ihave
  last
  list [active|active.times|newsgroups|distributions|distrib.pats|/
      overview.fmt|subscriptions]
  listgroup newsgroup
  mode reader
  newgroups yymmdd hhmmss ["GMT"] [<distributions>]
  newnews newsgroups yymmddhhmmss ["GMT"] [<distributions>]
  next
  post
  slave
  stat [MessageID|Number]
  xgtitle [group_pattern]
  xhdr header [range|MessageID]
  xover [range]
  xpat header range|MessageID pat [morepat...]
  xpath MessageID
Report problems to <usenet@vlager.vbrew.com>
.
```

Режима за четец на NNTP предоставя голям брой команди. Много от тях са проектирани за улеснение на четеца. По-горе споменахме за съществуването на команди, които дават указания на сървъра да изпраща заглавието и тялото на статията поотделно. Съществуват и команди, които дават списък с достъпните групи и статии, както и други, които позволяват публикуването на статии – алтернатива на изпращането на статии с новини до сървъра.

Получаване на списък с достъпните групи

Командата *list* показва списък с различни типове информация; особено групиге, поддържани от сървъра:

```
list newsgroups
215 Descriptions in form "group description".
control                 News server internal group
junk                    News server internal group
local.general           General local stuff
local.test              Local test group
.
```

Получаване на списък с активните групи

Командата *list active* показва всяка поддържана група и дава информация за нея. Двете числа на всеки ред на получения списък са маркери за най-високата и най-ниската точка, т.е. статията с най-голям и най-малък маркер във всяка група. От тях четецът може да оцени броя на статиите в групата. Ще поговорим по-подробно за тези числа след малко. Последното поле в резултата показва флагове, които указват дали е разрешено публикуването на статии в тази група, дали групата е арбитрирана и дали публикуваните статии се съхраняват или просто преминават през нея. Тези флагове са описани подробно в Глава 23. Ето един пример:

```
list active
215 Newsgroups in form "group high low flags".
control 0000000000 0000000001 y
junk 0000000003 0000000001 y
alt.test 0000000000 0000000001 y
.
```

Публикуване на статия

Както вече споменахме, има разлика между разпространението на статия и публикуването ѝ. Когато разпространявате статия се допуска, че тя вече съществува и притежава уникален идентификатор, зададен ѝ от сървъра, на който тя е публикувана след създаването си, и че притежава пълен комплект от полета в заглавието. Когато публикувате статия, вие я създавате за пръв път и единствените полета в заглавието, които задавате, са онези, които имат смисъл за вас. Такива са например полетата тема (Subject) и групи по интереси (Newsgroups), до които публикувате статията. Сървърът за новини, на който публикувате статията, ще добави всички останали полета вместо вас и ще създаде уникалния идентификатор, който ще използва при разпространението на статията през други сървъри.

Всичко това означава, че публикуването на статия е дори по-лесно от разпространението ѝ. Ето един пример за публикуване на статия:

```
post
340 Ok
From: terry@richard.geek.org.au
Subject: тестово съобщение номер 1
Newsgroups: junk
Body:

Това е тестово съобщение, можете спокойно да го
игнорирате.

.
240 Article posted
```

Ние създадохме още две подобни на това съобщения, за да придадем реализъм на примерите си.

Получаване на списък с новите статии

Когато четецът се свърже за пръв път с нов сървър за новини и потребителят избере коя група по интереси ще разглежда, четецът ще поиска да извлече списък с новите статии, които са публикувани или получени след последното влизане на потребителя в системата. За тази цел се използва командата *newnews*. Трябва да ѝ се подадат три задължителни аргумента: името на групата или групите, към които ще се извърши запитване, началната дата и началното време, от кои-

то да започне генерирането на списъка. Датата и времето представляват шестцифрени числа, като в началото са разположени най-големите единици: съответно *ггммдд* (година-месец-ден) и *ччммсс* (час-минута-секунда):

```
newnews junk 990101 000000
230 New news follows
<7g2o5r$aa$6@news.vbrew.com>
<7g5bhm$8f$2@news.vbrew.com>
<7g5bk5$8f$3@news.vbrew.com>
.
```

Избиране на група за работа

Когато потребителят избира група за преглеждане, четецът може да уведоми сървъра за новини, че групата вече е избрана. Това опростява комуникацията между сървъра и четеца. Вече не е нужно с всяка команда да се изпраща името на групата. Командата *group* просто приема като аргумент името на групата. Много от следващите команди използват това име по подразбиране, ако изрично не се укаже друга група:

```
group junk
211 3 1 3 junk
```

Командата *group* връща съобщение, което показва съответно броя на активните статии, статията с най-голям и статията с най-малък маркер и името на групата. Обърнете внимание, че макар че в нашия пример броят на активните съобщения съвпада с най-големия маркер за статия, в общия случай това не винаги е така. При сървър за новини с голяма активност, някои от статиите могат да бъдат с изтекъл срок или да бъдат изтрити, намалявайки по този начин броя на активните статии, без да променят най-големия маркер на статия.

Получаване на списък със статиите в група

За да адресирате статия към определена група по интереси, четецът ви трябва да знае кои номера на статии обозначават активни статии. Командата *listgroup* предлага списък с номера на активните статии в текущата група, или в конкретна група, ако изрично е зададено име на група:

```
listgroup junk
211 Article list follows
1
2
```

```
3
.
```

Извличане само на заглавието на статия

Потребителят трябва да притежава някаква информация за статията преди да реши дали да я прочете. Вече споменахме, че някои команди позволяват поотделно да се предават заглавието и тялото на статията. Командата *head* се използва за изпращане на заявка към сървъра за предаване само на заглавието на определена статия към четеца на новини. Ако потребителят не желае да чете тази статия, то не са изхабени време и мрежови ресурси за ненужното предаване на потенциално голямото тяло на статията.

Статиите могат да се указват или с техния номер (получен чрез командата *listgroup*), или с техния идентификатор:

```
head 2
221 2 <7g5bhm$8f$2@news.vbrew.com> head
Path: news.vbrew.com!not-for-mail
From: terry@richard.geek.org.au
Newsgroups: junk
Subject: тестово съобщение номер 2
Date: 27 Apr 1999 21:51:50 GMT
Organization: Виртуалната пивоварна
Lines: 2
Message-ID: <7g5bhm$8f$2@news.vbrew.com>
NNTP-Posting-Host: localhost
X-Server-Date: 27 Apr 1999 21:51:50 GMT
Body:
Xref: news.vbrew.com junk:2
.
```

Извличане само на тялото на статия

Ако, от друга страна, потребителят реши, че иска да прочете статията, неговият четец трябва да разполага с начин, по който да заяви предаването само на тялото на съобщението. Командата *body* се използва точно за тази цел. Тя работи почти по същия начин като командата *head*, с изключение на това, че се връща само тялото на статията:

```
body 2
222 2 <7g5bhm$8f$2@news.vbrew.com> body
Това е още едно тестово съобщение,
можете спокойно да игнорирате и него.

.
```

Прочитане на статия от група

Въпреки че обикновено е по-ефективно да предаваме отделно заглавието и тялото на избраните статии, съществуват ситуации, в които е по-добре да се предава цялата статия. Един добър пример за това са приложенията, чрез които искаме да предадем всички статии в определена група, без да се извършва някаква предварителна селекция, например когато използваме на NNTP програма за кеширане като *leafnode*.[59]

Естествено NNTP дава възможност за това и не е изненадващо, че тя работи почти по същия начин, по който и командата *head*. Командата *article* също приема като аргумент или номера на статията, или нейния идентификатор, но връща цялата статия, включително и заглавието ѝ:

```
article 1
220 1 <7g2o5r$aa$6@news.vbrew.com> article
Path: news.vbrew.com!not-for-mail
From: terry@richard.geek.org.au
Newsgroups: junk
Subject: тестово съобщение номер 1
Date: 26 Apr 1999 22:08:59 GMT
Organization: Виртуалната пивоварна
Lines: 2
Message-ID: <7g2o5r$aa$6@news.vbrew.com>
NNTP-Posting-Host: localhost
X-Server-Date: 26 Apr 1999 22:08:59 GMT
Body:
Xref: news.vbrew.com junk:1

Това е тестово съобщение, можете спокойно да го игнорирате.
.
```

Ако се опитате да извлечете непозната статия, сървърът ще върне съобщение с кодиран по подходящия начин код за отговор и вероятно пояснително текстово съобщение:

```
article 4
423 Bad article number
```

[59] Програмата *leafnode* е достъпна през анонимен FTP достъп от **wpxx02.toxi.uni-wuerzburg.de** в директорията */pub/*

В този раздел описахме как се използват най-важните NNTP команди. Ако се интересувате от проектирането на софтуер, който реализира протокола NNTP, трябва да прочетете съответните RFC документи. Те съдържат голямо количество подробна информация, която не можем да включим тук.

Нека сега погледнем как работи NNTP през сървъра **nntpd**.

Инсталиране на сървъра NNTP

Сървърът NNTP (**nntpd**) може да бъде компилиран по два начина в зависимост от очакваното от вас натоварване на системата за новини. Няма се предлагат готови компилирани версии, тъй като някой от специфичните за вашия сайт подразбиращи се стойности се кодират директно в изпълнимия код. Всички необходими конфигурации се извършват чрез макрос, дефиниран в *common/conf.h*.

Сървърът **nntpd** може да бъде конфигуриран или като самостоятелен сървър, стартиран по време на начално зареждане на системата от *rc*-файл, или като демон, управляван от *inetd*. Във втория случай трябва да разполагате със следния запис във файла */etc/inetd.conf*:

```
nntp    stream  tcp nowait      news    /usr/etc/in.nntpd    ntpd
```

Синтаксисът на *inetd.conf* е описан подробно в Глава 12, *Важни мрежови възможности*. Ако конфигурирате *nntpd* като самостоятелен сървър, трябва да се уверите, че сте горният ред във файла *inetd.conf* е коментиран. И в двата случая ще трябва да разполагате със следния ред във файла */etc/services*:

```
nntp    119/tcp    readnews untp    # Network News Transfer Protocol
```

За да съхранява временно входящите статии, *nntpd* се нуждае и от директория *.tmp* във вашия буфер за новини. Ще трябва да я създадете със следните команди:

```
# mkdir /var/spool/news/.tmp
# chown news.news /var/spool/news/.tmp
```

Ограничаване на достъпа до NNTP

Достъпът до ресурсите на NNTP се управлява от файла *nntp_access* в директорията */etc/news*. Редовете в този файл описват правата за достъп, дадени на външни хостове. Всеки запис има следния формат:

```
сайт read|xfer|both|no post|no      [!exceptgroups]
```

Когато клиент се свърже към NNTP порта, **nntpd** се опитва да получи пълното домейн име на хоста от неговия IP адрес, като използва обратно търсене. Името на хоста на клиента и неговия IP адрес се сравняват с полето *сайт* на всеки запис по реда, в който записите се появяват във файла. Съвпаденията могат да бъдат частични или пълни. Ако съвпадението е пълно, записа се прилага, а ако е частично, записа се прилага само, ако няма друго по-добро съвпадение след него. Полето *сайт* може да се дефинира по един от следните начини:

Име на хост

Това е пълното домейн име на хоста. Ако то съвпадне изцяло с каноничното име хоста на клиента, записа се прилага и всички следващи записи се игнорират.

IP адрес

Това е IP адреса, записан в десетично-точков формат. Ако IP адресът на клиента съвпада с него, записа се прилага и всички останали записи се игнорират.

Име на домейн

Това е името на домейн, зададено като *.*домейн*. Ако домейнът на клиента съвпадне с указания домейн, записът съвпада и се прилага.

Име на мрежа

Това е името на мрежата, както е дефинирано в */etc/networks*. Ако номера на мрежа от IP адреса на клиента съвпадне с номера на мрежа, асоциирана с даденото тук име на мрежа, записа се прилага.

default

Низът `default` съответства на всеки клиент.

Записите, които съдържат по-обобщено описание на сайта, трябва да се разполагат в началото, защото ще бъдат игнорирани от следващите по-точни съвпадения.

Второто и третото поле описват правата за достъп, предоставени на клиента. Второто поле определя разрешенията за извличане на новини чрез изтегляне (`read`) и предаване на новини чрез разпространение (`xfer`). Стойността `both` разрешава и двете, а `no` забранява достъпа изцяло. Третото поле разрешава на клиента да публикува статии, т.е да доставя статии с непълна информация в заглавието, която

след това се допълва от софтуера за новини. Ако второто поле съдържа стойността `no`, третото поле се игнорира.

Четвъртото поле не е задължително и съдържа разделен със запетайки списък от групи, до които клиентът няма право на достъп.

Ето един примерен файл *nntp_access*:

```
#
# по подразбиране, всеки може да предава новини, но не и да чете или
# публикува
default          xfer        no
#
# public.vbrew.com предлага публичен достъп през модем. Позволяваме
# да се чете и публикува към всички групи, с изключение на local.*
public.vbrew.com   read        post     !local
#
# всички други хостове в пивоварната могат да четат и публикуват
*.vbrew.com        read        post
```

NNTP удостоверяване на самоличността

Демонът *nntpd* предоставя проста схема за удостоверяване на личността. Ако напишете с главни букви кой да е от маркерите за достъп във файла *nntp_access*, *nntpd* изисква клиента да има разрешение за извършване на съответната операция. Например, ако зададете разрешение за достъп `Xfer` или `XFER` (противоположно на `xfer`) *nntpd* няма да позволи на клиента да предава статии към вашия сайт, докато не му бъдат предоставени права за това.

Процедурата за удостоверяване на личността се реализира посредством новата NNTP команда *AUTHINFO*. Чрез нея клиентът подава на NNTP сървъра своето потребителско име и парола. *nntpd* потвърждава достоверността им като ги сравнява с тези от файла */etc/passwd* и проверява, дали потребителят принадлежи към групата **nntp**.

Настоящата реализация на NNTP схемата за удостоверяване на личността е само експериментална и поради това не е особено преносима. Резултатът от това е, че тя работи само с обикновената текстова база данни с пароли; скрити пароли не се разпознават. Ако компилирате от изходния код и имате инсталиран пакета PAM, сравнително лесно е да промените проверката за валидността на паролата.

Взаимодействие на nntpd със C NEWS

Когато *nntpd* получи статия, той трябва да я достави до подсистемата за новини. В зависимост от това, дали статията е била приета в резултат на *IHAVE* или *POST* команда, тя се подава съответно на *rnews* или *inews*. Вместо да извиквате *rnews*, можете да го конфигурирате (по време на компилация) да пакетира идващите статии и да премества получените пакети в */etc/spool/news/in.coming*, където те се оставят на *relaynews* за по-нататъшна обработка.

nntpd трябва да има достъп до файла *history*, за да може правилно да изпълнява протокола ihave/sendme. По време на компилация трябва да се уверете, че пътя до този файл е зададен правилно. Ако използвате CNews се убедете, че C News и *nntpd* са единодушни по отношение на формата на файла *history*. C News използва *dbm* хеш-функции за достъп до файла. Съществуват обаче много взаимно несъвместими реализации на библиотеката *dbm*. Ако C News е свързан с друга *dbm* библиотека, а не тази във вашата стандартна библиотека *libc*, ще трябва да свържете и *nntpd* със същата библиотека.

Несъгласията между *nntpd* и CNews понякога водят до съобщения за грешки в системния дневник, указващи например, че *nntpd* не може да го отвори правилно или че са получени дублирани статии през NNTP. Добър начин за проверка на неправилното функциониране при предаване на новини е да вземете една статия от буфера, да се свържете чрез telnet до **nntp** порта и да я предложите на *nntpd*, както е показано в следващия пример. Разбира се, трябва да замените *msg@id* с идентификационния номер на конкретната статия, която искате да подадете на *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@id435 Got it.
QUIT
```

Този разговор демонстрира правилна реакция от *nntpd*. Съобщението `Got it` (имам я) ви показва, че демона вече има тази статия. Ако вместо това получите като отговор съобщението *335 Ok*, това означава, че проверката във файла history се е провалила поради някаква причина. Прекратете telnet сесията като натиснете Ctrl+D. Можете да проверите какво не е наред като проверите системния дневник. *nntpd* записва всички свои съобщения чрез услугата `daemon` на *syslog*. Една несъвместима *dbm* библиотека обикновено се открива по извеждането на съобщение, че *dbminit* се е провалил.

ГЛАВА 23

ИНТЕРНЕТ НОВИНИ

Демонът INN (InterNet News – Интернет новини) е безспорно най-използвания сървър за новини по мрежата днес. INN е изключително гъвкав и е подходящ за всички сайтове за новини с изключение на най-малките.[60] INN има добра мащабируемост и е удобен за конфигурации на големи сървъри за новини.

INN сървърът включва много компоненти, всеки от които със свои собствени конфигурационни файлове, които ще разгледаме по ред. Конфигурирането на INN може да бъде малко сложно, но в тази глава ще опишем всеки един етап от процеса на конфигуриране и ще ви въоръжим с достатъчно информация, за да можете да разберете справочните страници и документацията на INN, и да създадете своя конфигурация за всяко приложение.

Някой вътрешни подробности за INN

Основната програма на INN е демона *innd*. Задачата на *innd* е да обработва всички входящи статии, да ги съхранява локално и да ги

[60] Много малките сайтове за новини трябва да използват програма за NNTP сървър като leafnode, която е достъпна от *http://wpxx02.toxi.uni-wuerzburg.de/~krasel/leafnode.html*

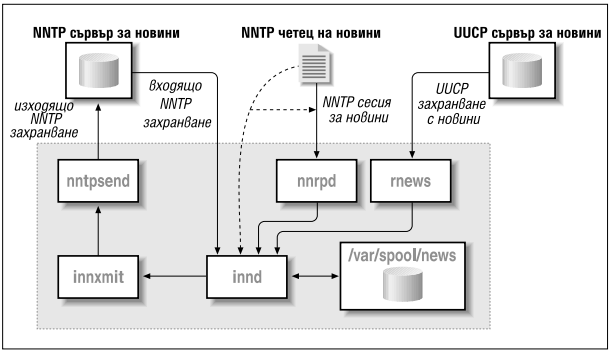
предава, ако е необходимо, на изходящите захранвания с новини. Той се стартира по време на зареждане и работи непрекъснато като фонов процес. Стартирането му като демон подобрява производителността, тъй като той трябва да чете файловете си за статуса става само веднъж при стартиране. В зависимост от обема на вашето захранване с новини, определени файлове, като *history* (съдържащ списък с последните обработени статии), могат да достигнат от няколко до десетки мегабайта.

Друга важна възможност на INN е, че винаги има едно копие на *innd*, работещо по всяко време. Това също е полезно за производителността, тъй като демонът може да обработва всички статии без да се интересува за синхронизацията между неговото вътрешно състояние и друго копия на *innd*, работещи по същото време с буфера за новини. Все пак, това се отразява на цялостното проектиране на системата за новини. Тъй като е много важно входящите новини да се обработват колкото е възможно по-бързо, неприемливо е сървърът да бъде ограничен до такива обикновени задачи като обслужването на четци, които осъществяват достъп до буфера с новини през NNTP, или декомпресирането на пакети с новини, пристигащи през UUCP. Следователно, тези задачи са отделени от главния сървър и са реализирани от отделни поддържащи програми. Фигура 23.1 илюстрира отношенията между *innd*, другите локални задачи, отдалечените сървъри за новини и четците.

Днес, NNTP е най-често използваното средство за прехвърляне на статии с новини, а *innd* не поддържа директно други протоколи. Това означава, че *innd* прослушва TCP socket (порт 119) за връзки и приема статиите с новини, използвайки протокола "ihave".

Статиите, пристигащи чрез други различни от NNTP протоколи, се поддържат непряко чрез други процеси, които приемат статиите и ги препращат към *innd* през NNTP. Пакетите с новини, идващи през UUCP връзка, например, обикновено се обработват от програмата *rnews*. Тази програма на INN декомпресира пакета, ако е необходимо, и го разбива на отделни статии; след това ги предава една по една към *innd*.

Четците могат да доставят новини, когато потребител публикува статия. Тъй като обработката на четците заслужава специално внимание, малко по-късно ще се върнем на това.



Фигура 23.1: Архитектура на INN (опростена е за по-голяма яснота)

Когато получава статия *innd* първо търси нейния идентификатор на съобщение във файла *history*. Дублиращите се статии се отхвърлят, а случайните се записват в дневник, което не е задължително. Същото важи и за статии, който са твърде стари или им липсва някое от задължителните полета в заглавието, например, полето `Subject:`.[61] Ако *innd* открие, че статията е подходяща за приемане, той преглежда полето `Newsgroups:` от заглавието, за да открие до кои групи е изпратена тази статия. Ако една или повече от тези групи се намира във файла *active*, статията се запазва на диска. В противен случай, тя се записва в специалната група *junk*.

Отделните статии се съхраняват под директорията */var/spool/news*, наричана още буфер за новини. Всяка група по интереси има отделна директория, в която всяка статия се съхранява в отделен файл. Имената на файловете са последователни номера, така че статия в групата *comp.risk* може да се съхрани като */var/spool/news/comp/risk/217*, например. Когато *innd* не открие директорията, в която иска да съхранява статията, той я създава автоматично.

Отделно от локалното съхраняване на статии, може да искате и да ги предавате към изходящи захранвания. Това се управлява от файла

[61] Това се указва от полето `Date`; ограничението обикновено е две седмици.

newsfeeds, който изброява всички downstream сайтове заедно с групите по интереси, които трябва да им бъдат подадени.

Точно както получаващият край на *innd*, обработката на изходящи новини също се извършва от един единствен интерфейс. Вместо да прави сам цялата специфична за прехвърлянето обработка, *innd* разчита на различни софтуерни пакети за управлението на предаването на статии до други сървъри за новини. Тези изходящи средства се наричат със събирателното име *канали*. В зависимост от предназначението един канал може да има различни атрибути, които определят каква точно информация му предава *innd*.

Например, за изходящо NNTP захранване *innd* може да разклони програмата *innxmit* в началото и за всяка статия, която трябва да се изпрати през това захранване, да предаде към стандартния вход на *innxmit* нейния идентификатор на съобщение, размера и името на файла. От друга страна, за изходящо UUCP захранване *innd* може да записва размера и името на файла на статията в специален дневник, който се управлява от друг процес през определени интервали, за да създаде пакети и да ги подреди в опашка към UUCP подсистемата.

Освен тези два примера, съществуват и други типове канали, които не са строго изходящи захранвания. Те се използват, например, при архивирането на определени групи по интереси или при генериране на обща информация. Тази обща информация се използва, за да помогне на четците да създават по-ефективно нишки за статиите. За да получат необходимата им за тази цел информация от заглавието, стариге четци трябва да сканират всички статии поотделно. Това може да подложи на голямо натоварване сървъра, особено когато се използва NNTP; нещо повече, това е много бавно.[62] Общият механизъм облекчава този проблем като презаписва всички съответни заглавия в отделен файл (наречен *.overview*) за всяка група по интереси. След това тази информация може да се вземе от четците като се прочете директно от директорията на буфера или като се използва командата XOVER, когато връзката е през NNTP. Демонът *innd* захранва всички статии до командата *overchan*, която е добавена към демона посредством канал. По-късно, когато разглеждаме конфигурирането на захранванията с новините, ще видим как става това.

[62] При натоварен сървър създаването на нишки за 1 000 статии отнема около 5 минути, което ще бъде приемливо само за най-пристрастените към Usenet.

Четци и INN

Четците работещи на същата машина, на която работи и сървърът (или присъединени към буфера за новини на сървъра през NFS), могат да четат статиите директно от директорията на буфера. За да изпратят статия, създадена от потребителя, те извикват програмата *inews*, която добавя всички липсващи полета от заглавието и ги препраща към демона *innd* през NNTP.

По аналогичен начин, четците могат да осъществяват достъп до сървъра през NNTP. Този тип връзка се обработва по начин различен от NNTP-базираните захранвания с новини, за да се избегне задържането на демона *innd*. Когато четецът се свързва към NNTP сървъра, *innd* разклонява отделна програма, наречена *nnrpd*, която обработва сесията, а *innd* се връща към по-важните неща (например, приемане на входящите новини).[63] Може би се чудите как *innd* процес различава входящите захранвания с новини и четеца. Отговорът е много прост: протоколът NNTP изисква от NNTP-базирания четец да използва командата *mode reader*, след като се свърже със сървъра; когато тази команда е получена, сървърът стартира *nnrdp* процеса, прехвърля връзката към него и продължава да прослушва за връзки, идващи от друг сървър за новини. Съществува поне един DOS-базиран четец, който не е конфигуриран да прави това, и комуникацията с INN няма да бъде успешна, тъй като самият *innd* не разпознава командите, използвани за четене на новини, ако не знае, че връзката е от четец.

По-късно в тази глава в раздела "Управление на достъпа на четците" ще разгледаме малко по-подробно достъпа на четците до INN.

Инсталиране на INN

Преди да се потопим в конфигурирането на INN, нека да поговорим за неговото инсталиране. Прочетете този раздел, дори ако вече сте инсталирали INN от една от различните дистрибуции на Linux; тя съдържа някои важни съвета относно сигурността и съвместимостта.

В продължение на доста време дистрибуциите на Linux включваха версията INN-1.4sec. За нещастие, тази версия имаше два сериозни проблема, свързани със сигурността. Съвременните версии нямат те-

[63] Очевидно, името nnrpd е съкращение на NetNews Read & Post Daemon.

зи проблеми и повечето дистрибуции включват прекомпилирания двоичен код за Linux на INN версия 2 или по-нова.

Ако решите, можете сами да компилирате INN. Можете да получите изходния код от **ftp.isc.org** в директорията */isc/inn*. Компилирането на INN изисква да редактиране конфигурационен файл, указващ на INN някои подробности относно вашата операционна система, а някои възможности могат да изискват малки промени в самия код.

Самото компилиране на пакета е доста просто; съществува скрипт *BUILD*, който ще ви ръководи през целия процес. Изходният код съдържа и подробна документация за начина, по който да инсталирате и конфигурирате INN.

След инсталирането на двоичния код, може да е необходимо допълнително ръчно настройване, за да съгласувате INN с други приложения, които може да искат да осъществят достъп до неговите програми rnews и inews. Например, UUCP очаква да намери програмата rnews в */usr/bin* или в */bin*, а INN я инсталира по подразбиране в директорията */usr/lib/bin*. Направете така, че */usr/lib/bin* да бъде в пътя за търсене по подразбиране или да има символни връзки, сочещи към действителното местоположение на командите rnews и inews.

Конфигуриране на INN: базовата настройка

Една от най-големите пречки за начинаещи е, че INN изисква настройката на работеща мрежа да функционира правилно, дори ако работи на отделен хост. Затова е много важно, когато стартирате INN вашето ядро да поддържа TCP/IP, а също и да сте настроили loopback интерфейса, както е обяснено в Глава 5, *Конфигуриране на TCP/IP мрежа*.

След това трябва да се уверите, че *innd* е стартиран при зареждане. Подразбиращата се инсталация на INN използва скрипт-файла *boot* от директорията */etc/news*. Ако вашата дистрибуция използва пакет init в стил System-V, всичко което трябва да направите е да създадете символна връзка от вашия */etc/int.d/inn* файл, сочеща към */etc/news/boot*. За други разновидности на *init* трябва да направите така, че скриптът */etc/news/boot* да се изпълнява от един от вашите rc скриптове. Тъй като INN изисква мрежова поддръжка, началният скрипт трябва да се изпълнява *след* конфигурирането на мрежовите интерфейси.

Конфигурационни файлове на INN

След като приключите с тези общи задачи, вече можете да се преминете към наистина интересната част на INN: неговите конфигурационни файлове. Всички конфигурационни файлове се намират в директорията */etc/news*. Някой промени в конфигурационните файлове бяха въведени във Версия 2 и тук ще опишем точно тази версия. Ако работите с по-стара версия, тази глава ще ви е полезна при обновяването на вашата конфигурация. В следващите няколко раздела ще разгледаме файловете един по един и като пример ще създадем конфигурация на Виртуална пивоварна.

Ако искате да научите повече подробности за възможностите на всеки отделен конфигурационен файл, можете да се обърнете към справочните страници; дистрибуцията на INN съдържа отделни справочни страници за всеки един от тях.

Глобални параметри

Съществуват голям брой глобални параметри на INN; те са свързани с всички пренасяни групи по интереси.

Файлът inn.conf

Основният конфигурационен файл на INN е *inn.conf*. Освен другите неща, той определя името, под което е известна вашата машина в Usenet. Версия 2 на INN позволява baffling брой параметри в този файл да бъдат конфигурирани. За щастие, повечето от тях имат подразбиращи се стойности, които са приемливи за повечето сайтове. Файлът *inn.conf(5)* описва подробно всичките параметри и ако имате някакви проблеми, трябва да го прочетете внимателно.

Прост примерен *inn.conf* файл може да изглежда по следния начин:

```
# Примерен inn.conf файл за Виртуалната пивоварна
server:           vlager.vbrew.com
domain:           vbrew.com
fromhost:         vbrew.com
pathhost:         news.vbrew.com
organization:     The Virtual Brewrey
mta:              /usr/sbin/sendmail -oi  %s
moderatormailer:  %s@uunet.uu.net
#
# Пътища до компонентите и файловете на INN.
#
```

```
pathnews:          /usr/lib/news
pathbin:           /usr/lib/news/bin
pathfilter:        /usr/lib/news/bin/filter
pathcontrol:       /usr/lib/news/bin/control
pathdb:            /var/lib/news
pathetc:           /etc/news
pathrun:           /var/run/news
pathlog:           /var/log/news
pathhttp:          /var/log/news
pathtmp:           /var/tmp
pathspool:         /var/spool/news
patharticles:      /var/spool/news/articles
pathoverview:      /var/spool/news/overview
pathoutgoing:      /var/spool/news/outgoing
pathincoming:      /var/spool/news/incoming
patharchive:       /var/spool/news/archive
pathuniover:       /var/spool/news/uniover
overviewname:      .overview
```

Първия ред указва на програмите *rnews* и *inews* с кой хост да контактуват, когато доставят статии. Този запис е абсолютно критичен; за да предават статии към *innd*, те трябва да създадат NNTP връзка със сървъра.

Ключовата дума `domain` задава частта за домейна в пълното квалифицирано името на домейн на хоста. Две програми трябва да използват пълното квалифицирано име на домейн; ако вашата библиотека на резолвера връща само неквалифицираното име на хост, зададеното в атрибута `domain` име се добавя към него. Не е проблем да го конфигурирате, така че по-добре го направете.

Следващият ред определя какво име на хост ще използва програмата *inews*, когато добавя ред `From:` за статии, изпратени от локални потребители. Повечето четци използват полето *`From:`*, когато съставят отговор на пощенско съобщение до автора на статия. Ако пропуснете това поле, по подразбиране се взима пълното квалифицирано име на домейн на вашия хост за новини. Това не винаги е най-добрия избор. Възможно е, например, обработката на новините и пощата да се извършва от два различни хоста. В такъв случай, трябва да добавите пълното квалифицирано име на домейн на вашия хост за поща след конструкцията `fromhost`.

Редът `pathhost` определя името на хоста, което INN ще добавя в полето `Path:` от заглавието винаги, когато получава статия. В повечето случаи ще искате да използвате пълното квалифицирано име на домейн на вашия сървър за новини; тогава можете да пропуснете това

поле, тъй като то е зададено по подразбиране. Понякога, когато обслужвате голям домейн, може да искате да използвате едно общо име като **news.vbrew.com**. Това ще ви позволи лесно да премествате новините от един хост на друг, ако решите.

Следващият ред съдържа ключовата дума `organization`. Тази конструкция ви позволява да конфигурирате какъв текст *inews* ще поставя в реда `Organization`: за статии, изпратени от вашите локални потребители. Формално, тук трябва да включите описание на вашата организация или нейното име. Ако не искате да сте толкова формални, модерно е организациите с чувство за хумор да го проявят точно в това поле.

Ключовата дума `organization` е задължителна и определя името на пътя на потребителската програма за поща, което ще бъде използвано за изпращане на съобщения от посредник. `%s` се заменя с пощенския адрес на посредника.

Записът `moderatormailer` определя подразбиращия се адрес, който се използва когато потребител се опита да изпрати статия до група по интереси, която се явява посредник. Списъкът с адреси на посредници за всяка група по интереси обикновено се пази в отделен файл; ще ви е доста трудно обаче да следите всички промени. Затова `moderatormailer` се приема за последна инстанция; ако той е дефиниран, *inews* ще замести низа `%s` с името на група по интереси, което е преобразувано, и ще изпрати цялата статия на този адрес. Например, когато изпращате статия до *soc.feminism*, тя се изпраща по пощата към *soc-feminsim@uunet.uu.net*, зададен в горната конфигурация. В UUNET трябва да има пощенски псевдоним, инсталиран за всеки от тези подчинени адреси, който автоматично препраща всички съобщения до съответния посредник.

Накрая, всеки от останалите записи определя местоположението на файл с компоненти или изпълним файл, принадлежащ на INN. Ако сте инсталирали INN от пакет, тези пътища трябва да са конфигурирани. Ако сте инсталирали INN от изходен код, ще трябва да се уверите, че те показват къде сте инсталирали INN.

Конфигуриране на групи по интереси

Администраторът на системата за новини може да контролира до кои групи по интереси потребителите имат достъп. INN предоставя два конфигурационни файла, позволяващи на администратора да решава кои групи ще поддържа и за кои ще предоставя описание.

Файловете active и newsgroups

Файловете *active* и *newsgroups* се използват за съхраняване и описание на групите по интереси, поддържани от вашия сървър за новини. Те съдържат списък с групите, от които се интересуваме при получаване и обслужване на статии и административна информация за тях. Тези файлове се намират в директорията */var/lib/news*.

Файлът *active* определя кои групи по интереси поддържа този сървър. Синтаксисът му е ясен. Всеки ред във файла *active* има четири полета, разделени с празно пространство.

```
name  himark lomark flags
```

Полето `name` съдържа името на групата по интереси. Полето `himark` е най-големия номер, използван за статия в тази група, а полето `lomark` е най-малкия използван активен номер. За да илюстрираме как работи всичко това, разгледайте следния сценарий. Представете си, че имаме току-що създадена група по интереси: полетата `himark` и `lomark` са 0, тъй като няма статии. Ако публикуваме 5 статии, те ще бъдат номерирани съответно от 1 до 5. Тогава `himark` ще е равно на 5, най-големия номер на статия в групата, а `lomark` ще бъде 1 – най-малкия номер. Ако статия 5 се отхвърли, няма да настъпят промени; `himark` ще остане 5, а `lomark` ще остане 1. Ако сега отхвърлим статия 1, `himark` няма да се промени, но `lomark` вече ще бъде 2, защото 1 вече не е активен номер. Ако сега публикуваме нова статия, тя ще получи номер 6 и `himark` ще бъде 6. Статия 5 вече е използвана, така че няма да и задаваме номер отново. `lomark` остава 2. Този механизъм ни позволява лесно да разпределяме уникални номера за новите статии и да изчисляваме приблизително колко активни статии има в групата: `himark-lomark`.

Полето може да съдържа следното:

y Позволено е директното изпращане до този сървър за новини

n Не е позволено директно изпращането до този сървър за новини. Това предотвратява директното пускане на статии от четците към този сървър. Могат да се получават нови статии само от други сървъри за новини

m Групата е посредник. Всяка статия, изпратена до тази група, се препраща за утвърждаване към посредника на групата преди да влязат в нея. Повечето групи не са посредници.

j Статиите в тази група не се пазят, а само преминават. По този начин сървърът приема статията, но всичко, което ще направи с нея е да я предаде към "up-stream" сървърите за новини. Статията няма да е достъпна за четците на сървъра.

x Към тази група не могат да бъдат изпращани статии. Единственият начин за доставяне на статии до този сървър е чрез захранването му с новини от друг сървър. Четците не могат директно да записват статии в този сървър.

=foo.bar

Статиите се подреждат локално в групата "foo.bar".

В нашата примерна конфигурация на сървър ще пренесем малък брой групи по интереси, така че нашият файл */var/lib/news/active* ще изглежда по следния начин:

```
control 0000000000 0000000001 y
junk 0000000000 0000000001 y
rec.crafts.brewing 0000000000 0000000001 y
rec.crafts.brewing.ales 0000000000 0000000001 y
rec.crafts.brewing.badtaste 0000000000 0000000001 y
rec.crafts.brewing.brandy 0000000000 0000000001 y
rec.crafts.brewing.champagne 0000000000 0000000001 y
rec.crafts.brewing.private 0000000000 0000000001 y
```

В този пример номерата *himark* и *lomark* са тези, които ще използвате при създаването на нови групи по интереси. Тези номера ще изглеждат по доста различен начин, ако група по интереси е активна от известно време.

Файла *newsgroups* е дори по-прост. Той предоставя едноредови описания за всяка група по интереси. Някои четци могат да четат и предоставят тази информация на потребителя, за да му помогнат да реши дали иска да се абонира.

Форматът на файла *newsgroups* е прост:

```
име описание
```

Полето *име* е името на групата по интереси, а *<описание* е едноредово описание на тази група.

Искаме да опишем групите по интереси, поддържани от нашия сървър, затова създаваме нашия файл *newsgroups* по следния начин:

```
rec.crafts.brewing.ales          Home brewing Ales and Lagers
rec.crafts.brewing.badtaste      Home brewing foul tasting brews
rec.crafts.brewing.brandy        Home brewing your own Brandy
rec.crafts.brewing.champagne     Home brew your own Champagne
rec.crafts.brewing.private       The Virtual Brewery home brewers group
```

Конфигуриране на захранвания с новини

INN предоставя на администратора на новини възможността да контролира кои групи по интереси се препращат до други сървъри за новини и начина, по който ще става това. Най-често използваният метод използва протокола NNTP, който вече описахме, но освен това, INN позволява захранвания с новини и през други протоколи като UUCP.

Файлът newsfeeds

Файлът *newsfeeds* определя къде ще бъдат изпратени статиите с новини. Обикновено, той се намира в директорията */etc/news*.

Форматът на файла *newsfeeds* е малко усложнен. Тук ще ви опишем общата схема на файла, а повече подробности ще намерите в справочната страница *newsfeeds(5)*. Форматът е следния:

```
# формат на файла newsfeeds
site:pattern:flags:param
site2:pattern2\
   :flags2:param2
```

Всяко захранване с новини на сайт се описва на един или множество редове, като се използва символа за продължение \. Двоеточието разделя полетата във всеки ред. Символът # в началото на ред показва, че този ред е коментар.

Полето *site* дава името на сайта, за който се отнася захранването. Името на сайта може да бъде кодирано по какъвто начин искате и не е задължително да бъде име на домейн на сайта. Това име ще се използва по-късно и ще сочи към запис от таблица, доставяща името на хоста на програмата *innxmit*, която предава статиите с новини до отдалечения сървър чрез протокола NNTP. За всеки сайт можете да имате множество записи; всеки запис ще бъде обработен поотделно.

Полето *pattern* определя кои групи по интереси да бъдат изпратени на този сайт. По подразбиране се изпращат всички групи, така че ако

искате точно това, просто оставете полето празно. Това поле съдържа разделени със запетаи изрази за съвпадение с шаблон. Символът * съвпада с нула или повече символа, символът . няма специално значение, ! означава логическо НЕ (ако е използван в началото на израз), а символът @ означава "Не препращай никакви статии, изпратени ? в тази група", ако се намира в началото на име на група. Списъкът се чете и анализира отляво надясно, така че трябва да сте сигурни, че първо сте поставили по-специфичните правила. Шаблонът:

```
rec.crafts.brewing*,!rec.crafts.brewing.poison,
@rec.crafts.brewing.private
```

ще изпрати всички новини от йерархията *rec.crafts.brewing* с изключение на *rec.crafts.brewing.poison*. Това няма да захранва никакви статии, изпратени към групата по интереси *rec.crafts.brewing.private*; тези статии ще бъдат прехващани и достъпни само за хората, които използват този сървър. Ако размените първите два шаблона, първият шаблон ще бъде отменен от втория, а вие ще захранвате със статии групата *rec.crafts.brewing.poison*. Същото е вярно и за първия и последния шаблон; затова винаги трябва да поставяте по-специфичните шаблони преди по-малко специфичните, за да имат ефект.

Полето `flags` контролира и поставя задължителни параметри за захранването на статии с новини към този сайт. Това поле е разделен със запетаи списък, който може да съдържа всеки от елементите от следващия списък, разделени от команди:

<размер

Статията трябва да бъде по-малка от зададения размер.

A*елементи*

Проверки на статия. *елементи* може да бъде едно или повече `d` (трябва да има заглавие на дистрибуция) или `p` (не проверява за сайт в заглавието Path).

B*най-голяма стойност/най-малка стойност*

Размер на вътрешния буфер преди записване на изхода.

H*[брой]*

Статията трябва да има по-малко от указания брой препращания; броят по подразбиране е 1.

I*размер*

Размер на вътрешния буфер (за захранване на файл).

M*шаблон*

Само групи, съвпадащи с шаблона, които са посредници.

`N`*шаблон*

Само групи, съвпадащи с шаблона, които не са посредници.

`S`*размер*

Стартира буфера, ако статиите в опашката превишават зададения размер.

`T`*тип*

Типове захранване: `f` (файл), `m` (фуния; полето *param* дава име на записа, указващ къде ще бъде насочени статиите), `p` (канал към програма), `c` (изпраща към канала на стандартния вход на подпроцеса на полето *param*), и `x` (като c, но обработва команди на стандартния вход)

`W`*елементи*

Какво да бъде записано: `b` (размера на статия в байтове), `f` (пълния път), `g` (първата група по интереси), `m` (идентификатор на съобщението), `n` (относителен път), `s` (сайт, който захранва статия), `t` (време на получаване), `*` (имена на захранвания на фунии или всички сайтове, които са получили статията), `N` (заглавие на групата по интереси), `D` (заглавие на дистрибуция), `H` (всички заглавия), `O` (общи данни), `R` (данни за репликация).

Полето `param` има специално кодиране, което зависи от типа на захранването. В повечето общо конфигурации в това поле задавате името на изходния файл, в който ще запишете изходящото захранване. В други конфигурации можете да го оставите празно. Съществуват и такива конфигурации, в които това поле може да има различни значения. Ако искате да направите нещо необичайно, справочната страница *newsfeed(5)* ще ви даде по-подробна информация за начина на използване на полето `param`.

Съществува едно специално име на сайт, което трябва да бъде кодирано като `ME` и трябва да бъде първия запис във файла. Записът се използва за управление на подразбиращите се настройки за вашите захранвания с новини. Ако записът `ME` има свързан с него списък с дистрибуции, този списък ще бъде добавен към всеки от другите записи за сайта преди да бъдат изпратени. Това ви позволява, например, да декларирате някои групи по интереси за автоматично захранване или автоматично блокиране на захранването, без да повтаряте шаблона във всеки запис за сайта.

Вече споменахме за възможността да използвате специални захранвания, за да генерирате данни за нишки, които улесняват работата на четците. Ще направим това като използваме командата *overchan*, ко-

ятo е част от дистрибуцията на INN. За тази цел, създадохме специално локално захранване, наречено `overview`, което предава статиите с новини към командата *overchan* за обработка в данни подходящи за резюме на съдържанието в групите.

Нашият сървър за новини ще предоставя само едно външно захранване с новини, което води до университета Groucho Marx, където се получават статиите за всички групи по интереси с изключение на групите *control* и *junk*, групата *rec.crafts.brewing.private*, която ще се пази локално и групата *rec.crafts.brewing.poison*, за която не искаме хората от нашата пивоварна да знаят, че изпращаме новини.

Ще използваме командата *nntpsend* за прехвърляне на новините през NNTP до сървъра **news.groucho.edu**. Тази команда изисква от нас да използваме метода за доставяне на файлове и да записваме името на пътя и идентификатора на всяка статия. Забележете, че в полето *param* сме задали името на изходния файл. Малко повече за командата *nntpsend* ще поговорим след малко. Ето как изглежда нашата конфигурация на файла newsfeeds:

```
# файлът /etc/news/newsfeeds за Виртуалната пивоварна
#
# По подразбиране изпраща всички групи с изключение на control и junk
ME::!control,!junk::
#
# Генерира overview данни за използване от четците.
overview::Tc,WO:/usr/lib/news/bin/overchan
#
# Захранва с всичко университета Groucho Marx с изключение на нашата
# частна група по интереси
# и статиите, изпратени към групата rec.crafts.brewing.poison.
gmarxu:! rec.crafts.brewing.poison.@ rec.crafts.brewing.private:\
   Tf,Wnm:news.groucho.edu
#
```

Файлът nntpsend.ctl

Програмата *nntpsend* управлява предаването на статии с новини, използвайки протокола NNTP като извиква командата *innxmit*. Преди малко видяхме проста употреба на командата *nntpsend*, която освен това има конфигурационен файл, който ни предоставя известна гъвкавост при конфигурирането на нашите захранвания с новини.

Командата *nntpsend* очаква да намери batch-файлове за сайтовете, които ще захранва. Тя очаква тези файлове да бъдат с имена */var/spool/news/out.going/име_на_сайт*. Демонът *innd* създава тези batch-

файлове, когато обработва запис във файла *newsfeeds*, който разгледахме в предишните раздели. Зададохме името на сайта като името на файл в полето *param*, което задоволява входните изисквания на командата *nntpsend*.

Конфигурационния файл на командата *nntpsend* се нарича *nntpsend.ctl* и обикновено се съхранява в директорията */etc/news*.

Файлът *nntpsend.ctl* ни позволява да свързваме пълното квалифицирано име на домейн, някои задължителни параметри на захранването с новини и някои параметри на предаването с името на захранвания сайт. Името на файла е средство у никално идентифициране на логическо захранване на статии. Общият формат на файла е:

```
sitename:fqdn:max_size:[args]
```

Следващият списък описва елементите на този формат:

sitename
: Името на файла както е зададено във файла *newsfeeds*.

fqdn
: Пълното квалифицирано име на домейн на сървъра за новини, който ще захранваме със статии с новини.

max_size
: Максималния обем на новините за захранване при всяко отделно прехвърляне.

args
: Допълнителни аргументи за предаване към командата *innxmit*.

Нашата примерна конфигурация изисква много прост *nntpsend.ctl* файл. Имаме само едно захранване с новини. Ограничили сме захранването максимум до 2 мегабайта трафик и ще предадем аргумент на командата *innxmit*, задаващ таймаут от 3 минути (180 секунди). Ако имахме по-голям сайт и много захранвания с новини, просто щяхме да създадем нови записи за всяко ново захранване на сайт, които са много подобни на следващия:

```
#  /etc/news/nntpsend.ctl
#
gmarxu:news.groucho.edu:2m:-t 180
#
```

Управление на достъпа на четци

Не беше много отдавна времето, когато обичайната практика на организациите беше да предоставят публичен достъп до техните сървъри за новини. Днес е трудно да намерите публични сървъри за новини; повечето организации внимателно контролират кой има право на достъп до техните сървъри, като обикновено се ограничават само до потребителите, поддържани от тяхната мрежа. INN предоставя конфигурационни файлове за управление на този достъп.

Файлът incoming.conf

Във въведението в INN споменахме, че INN постига своята ефективност и размер като разделя механизмите за захранване с новини от механизмите за четене. Файлът */etc/news/incoming.conf* е мястото, където указвате кои хостове ще ви захранват с новини, използвайки протокола NNTP. Освен това, в този файл задавате и някои параметри, които контролират начина, по който статиите се подават от тези хостове. Всеки хост, който не е в този файл, който се свързва към socket за новини, няма да се обработи от демона *innd*; вместо това, той ще бъде обработен от демона *nnrpd*.

Синтаксисът на файла */etc/news/incoming.conf* е много прост, но изисква малко време, за да се запознаете с елементите му. Позволени са три вида валидни записи: двойки ключ/стойност, определящи атрибутите и техните стойности; peers, указващ името на хоста, на който е позволено да ни изпраща статии, използвайки протокола NNTP; и групи – средство за прилагане на двойките ключ/стойност към групи от peers. Двойките ключ/стойност могат да имат три различни обхвата на прилагане. Глобални двойки се прилагат към всеки peer, дефиниран във файла. Двойки от групи се прилагат към всички peers, дефинирани в рамките на тази група. Двойки от peers се прилагат само към този peer. Специфичните дефиниции отменят по-малко специфичните: затова дефинициите на peer отменят дефинициите на групи, а те от своя страна отменят глобалните двойки.

Фигурните скоби (`{}`) се използват за ограничаване на началото и края на спецификациите `group` и `peer`. Символът # маркира останалата част от реда като коментар. Двойките ключ/стойност са разделени от двоеточие и на ред един ред има само една двойка.

Могат да се задават голям брой различни ключове. Най-често използваните са:

hostname

Този ключ задава разделен със запетаи списък с пълните квалифицирани имена или IP адреси на peers, на които ще позволим да ни изпращат статии. Ако няма такъв ключ, по подразбиране се взима етикета на peer.

streaming

Този ключ определя дали са разрешени команди за поток от новини от този хост. Стойността му е логическа и по подразбиране е `true`.

max-connections

Този ключ задава максималния брой връзки разрешени от тази група или peer. Стойност нула означава неограничен брой връзки (може да се зададе и чрез `none`).

password

Този ключ ви позволява да зададете паролата, която трябва да бъде използване от peer, за да му се позволи да прехвърля новини. По подразбиране, парола не се изисква.

patterns

Този ключ определя кои групи по интереси ще приемаме от свързания peer. Това поле е кодирано според точно същите правила, които използвахме в нашия файл *newsfeeds*.

В нашия пример имаме само един хост, който очакваме да ни захранва с новини – това е доставчика ни на upstream новини от университета Groucho Marx. Няма да имаме парола, но ще се уверим, че няма да приемаме статии за нашата частна група по интереси отвън. Нашият файл *host.nntp* изглежда по следния начин:

```
# Файлът incoming.conf на Виртуалната пивоварна.

# Глобални настройки
streaming:      true
max-connections:      5
# Разрешаване на изпращане посредством NNTP от нашия локален хост.
peer ME {
   hostname: "localhost, 127.0.0.1"
}
# Разрешаване на groucho да ни изпраща всички групи по интереси с
# изключение на нашите локални групи.
```

```
peer groucho {
    hostname: news.groucho.edu
    patterns: !rec.crafts.brewing.private
}
```

Файлът nnrp.access

Вече споменахме, че четците, а на практика и всеки хост, който не е в изброен в *host.nntp* и се свързва с INN сървъра за новини, се обработват от програмата *nnrpd*. *nnrpd* използва файла */etc/news/nnrp.access*, за да определи кой има право да използва сървъра за новини и какви права за достъп трябва да има.

Файлът *nnrp.access* има структура, подобна на вече разгледаните конфигурационни файлове. Той включва набор от шаблони за сравнение с името на домейн или IP адреса на свързващия хост и полетата, които определят какъв достъп и какви права трябва да му бъдат дадени. Всеки запис трябва да се намира на отделен ред, а полетата се разделят с двоеточие. Последният запис в този файл, който съвпада със свързващия се хост, ще бъде този, който използваме. Така че отново трябва да поставите първо общите шаблони, а след тях по-специфичните. Петте полета на всеки запис по реда в който следват са:

Име на хост или IP адрес

Това поле съответства на правилата за съвпадение с шаблон в *wildmat(3)*. Това е шаблон, който описва името или IP адреса на свързващия се хост.

Права за достъп

Това поле определя какви права за достъп трябва да бъдат предоставени на съвпадащия хоста. Съществуват два вида права за достъп, които можете да конфигурирате: `R` дава права за четене, а `P` дава права за изпращане.

Потребителско име

Това поле е не е задължително и ви позволява да зададете потребителското име, с което NNTP клиент трябва да влезе в сървъра преди да му се позволи да изпраща статии с новини. Това поле може да се остави празно. За четене на статии не се изисква удостоверяване на самоличността на потребителя.

Парола

Това поле не е задължително и определя паролата, съпровождаща полето за *потребителското име*. Ако полето е празно, не се изисква парола за изпращане на статии.

Групи по интереси

Това поле е шаблон, указващ кои групи по интереси са разрешени за достъп от клиента. Шаблонът следва същите правила като използваните във файла *newsfeeds*. По подразбиране, няма никакви групи, така че обикновено тук трябва да се конфигурира някакъв шаблон.

В примера за Виртуалната пивоварна ще позволим на всеки NNTP клиент от домейна както да чете, така и да изпраща статии към всички групи. Освен това, на всеки NNTP клиент ще позволим достъп до всички групи само за четене с изключение на нашата частна вътрешна група по интереси. Нашият файл *nnrp.access* ще изглежда по следния начин:

```
# Файлът nnrp.access на Виртуалната пивоварна
# Ще позволим публичен достъп за четене до всички групи с изключение на
# нашата частна група.
* :R:::,!rec.crafts.brewing.private*


# Всеки хост във домейна на  Виртуалната пивоварна може да чете и
изпраща до всички групи по
# интереси
*.vbrew.com:RP::*
```

Изтичане на срока на статии с новини

Когато статиите с новини се получават от сървър за новини, те се съхраняват в диска. Статиите с новини трябва да са достъпни за използване за известен период от време, така че голям сървър за новини може да заеме голяма част от дисковото пространство. За да сте сигурни, че дисковото пространство се използва ефективно, след време можете да изтривате статии автоматично. Този процес се нарича *изтичане на срока на статии*. Естествено, INN предоставя средство за автоматично изтичане на срока на статии с новини.

Файлът expire.ctl

INN сървърът използва програма, наречена *expire*, за изтриване на статии с изтекъл срок. Програмата *expire* използва файл */etc/news/expire.ctl*, за да конфигурира правилата, които управляват изтичането на срока на статия.

Синтаксисът на файла */etc/news/expire.ctl* е доста прост. Както при повечето конфигурационни файлове, празни редове или редове, започващи със символа #, се игнорират. Основната идея е да зададете всяко правило на отделен ред. Всяко правило определя начина, по който ще се изпълнява изтичането на статия при групи по интереси, съвпадащи с приложения шаблон. Синтаксисът на правило е следния:

```
pattern:modflag:keep:default:purge
```

Следващият списък описва тези полета:

pattern

Това поле е разделен със запетаи списък с шаблони, съвпадащи с имена на групи по интереси. Програмата *wildmat(3)* се използва за съвпадение с тези шаблони. Прилага се последното правило, съвпадащо с името на група по интереси, така че ако зададете правила за универсален символ (*), те трябва да са изброени в този файл.

modflag

Този флаг описва как това правило се прилага към групи по интереси, които са посредници. Възможно да го кодирате с `M`, което означава, че правилото се прилага само към посредници, с `U`, което означава, че правилото се прилага само към групи, които не са посредници, или с `A`- правилото игнорира статуса на посредника и се прилага към всички групи.

keep

Това поле ви позволява да задавате минималното време, през което статия с заглавие "Expires" ще се пази преди да изтече срокът ѝ. Единиците, в които се измерва това време са дни и числа с плаваща запетая, така че можете да зададете `7.5` за седем дни и половина. Освен това, можете да зададете стойност `never`, ако искате статиите да останат в групата по интереси завинаги.

default

Това поле е най-важното и ви позволява да зададете времето, през което се пази статия, която няма заглавие `Expires`. Повечето статии нямат такова заглавие. Това поле се кодира по същия начин както полето `keep`, като "`never`" означава, че срокът на статии без заглавия `Expires` никога няма да изтече.

purge

Това поле ви позволява да задавате максималното време, през което статия със заглавие Expires ще се пази преди да изтече срокът ѝ. Кодирането на това поле е същото както при полето "keep".

Нашите изисквания са прости. Ще пазим всички статии във всички групи по интереси 14 дни по подразбиране, и между 7 и 21 дни за статии, които имат заглавие `Expires`. Групата по интереси *rec.crafts.brewing.private* е нашата вътрешна група, така че ще трябва да се уверим, че срокът на статиите от нея няма да изтече:

```
# файл expire.ctl за Виртуалната пивоварна
# Срокът на всички статии изтича след по подразбиране след 14 дни, а
# тези със заглавия Expires след 7-21 дни
*:A:7:14:21
# Това е нашата специална вътрешна група по интереси, чийто срок
# никога няма да изтече
rec.crafts.brewing.private:A:never:never:never
```

Ще спомена един специален тип запис, който може да имате във вашия файл */etc/news/expires.ctl*. Можете да имате точно един ред, който изглежда по следния начин:

```
/remember/:days
```

Този запис ви позволява да задавате минималния брой дни, през които статията ще се пази във файла history, независимо дали срокът ѝ е изтекъл или не. Това може да ви бъде от полза, ако един от сайтовете, който ви захранва със статии, е рядък и има навика да ви изпраща отново и отново стари статии. Настройката на полето */remember/* предотвратява повторното изпращане на статии, дори ако вече срокът им е изтекъл. Ако вашият сървър помни, че вече е получил статията, той ще отхвърли опитите за повторно изпращане. Важно е да запомните, че тази настройка няма никакъв ефект върху изтичането на срока на статиите; оказва влияние само върху времето, през което подробностите за дадена статия се пазят в базата данни с историята.

Обработка на контролни съобщения

Както при C новините, INN може автоматично да обработва контролни съобщения. INN предоставя мощен конфигурационен механизъм за управление на действията, които ще възникнат за всеки вид контролни съобщения, и механизъм за контрол на достъпа, който се използва за управление на това кой може да инициира действия и за кои групи по интереси.

Файлът control.ctl

Структурата на файла *control.ctl* е доста проста. Синтаксисът на правилата за този файл е почти същия както при другите конфигурационни файлове на INN. Редове, започващи с # се игнорират; редовете могат да бъдат продължени като се използва символа /, а полетата се разделят с двоеточие (:).

Когато се получи контролно съобщение, то се тества спрямо всяко правило. Последното правило във файла, което съвпада със съобщението, е правилото, което ще се използва, така че трябва да поставите всички общи правила в началото на файла, а по-специфичните – в края. Общият синтаксис на файла е:

```
message:from:newsgroups:action
```

Полетата имат следното значение

message

Това е името на контролното съобщение. Типичните контролни съобщения са описани по-натам.

from

Това е шаблон в shell-стил, съвпадащ с пощенския адрес на лицето, което изпраща съобщението. Пощенският адрес се преобразува до малки букви преди сравнението.

newsgroups

Ако контролното съобщение е `newgroup` или `rmgroup`, това поле е шаблон в shell-стил, който съвпада със създадената или премахнатата група по интереси.

action

Това поле задава действието, което ще се предприеме за всяко съобщение, съвпадащо с правилото. Съществуват голям брой действия, които можем да предприемем; те са описани в следващия списък.

Полето `message` може от всеки ред може да има следните стойности:

checkgroups

Това съобщение изисква от администраторите на новини да синхронизират отново активната база данни с групи по интереси със списъка от групи доставен от контролното съобщение.

newgroup

Това съобщение заявява създаването на нова група по интереси. Тялото на контролното съобщение ще съдържа кратко описание на целта на групата, която ще бъде създадена.

rmgroup

Това съобщение заявява премахването на група по интереси.

sendsys

Това съобщение заявява, че файлът sys на този сървър за новини трябва да бъде предаден по пощата към създателя на съобщението. Стандартът RFC-1036 указва, че изискването за членство в Usenet е тази информация да бъде публично достъпна, тъй като се използва за съхраняване на актуалната карта на Usenet.

version

Това съобщение заявява, че името на хоста и версията на софтуера на сървъра за новини трябва да бъдат върнати на създателя на контролното съобщение.

all Това е специално кодиране, което ще съвпадне с всяко контролно съобщение.

Полето `action` може да включва следните действия:

doit

Изпълнява се заявената команда. В много случаи, пощенско съобщение ще бъде изпратено до администратора, за да го уведоми за предприетото действие.

doit=файл

Това е същото действие като `doit`, с изключение на това, че съобщение от дневника ще бъде записано в дневника *файл*. Ако зададеният файл е *mail*, записът в дневника се изпраща по пощата. Ако зададеният файл е празен низ, съобщението от дневника се изпраща към */dev/null* и е еквивалентно на използването на неквалифицираното действие `doit`. Ако името на *файла* започва със символа /, то съдържа абсолютния път до дневника; в противен случай, зададеното име се предава към */var/log/news/file.log*.

doifarg

Заявената команда се изпълнява, ако има аргумент. Ако командата няма аргумент, контролното съобщение се игнорира.

drop

Заявената команда се игнорира.

log

Съобщение от дневника се изпраща към изхода `stderr` на *innd* процеса. Съобщението обикновено се насочва към файла */var/log/news/errlog*.

log=файл

Това действие е същото като `log`, с изключение на това, че дневникът е зададен както при правилата, дадени за действието `doit=`*файл*.

mail

На администратора на новини се изпраща пощенско съобщение, съдържащо подробностите за заявената команда. Не се прави нищо друго.

*verify-**

Ако действие започва с низа "`verify-*`", тогава се изпълнява удостоверяване на самоличността за контролното съобщение посредством PGP (или GPG).[64]

И така, за да разберете какво представлява един *control.ctl* файл на практика, за илюстрация вижте следващия кратък пример:

```
## Примерен файл /etc/news/control.ctl
##
## Внимание: не използвайте този файл, той е само за илюстрация.

## Обработка на контролни съобщения
all:*:*:mail
checkgroups:*:*:mail
ihave:*:*:drop
sendme:*:*:drop
sendsys:*:*:log=sendsys
senduuname:*:*:log=senduuname
version:*:*:log=version
newgroup:*:*:mail
rmgroup:*:*:mail
```

[64] PGP и GPG са инструменти за удостоверяване на самоличността или криптиране на съобщения, използвайки техники за криптиране с публичен ключ. GPG е безплатната GNU версия на PGP. Можете да намерите GPG на адрес *http://www.gnupg.org/*, а PGP на адрес *http://www.pgp.com/*.

```
## Обработва контролни съобщения за осем от най-важните йерархии от
## новини.
## COMP, HUMANITIES, MISC, NEWS, REC, SCI, SOC, TALK
checkgroups:*:comp.*|humanities.*|misc.*|news.*|rec.*|sci.*|soc.*|
talk.*:drop
newgroup:*:comp.*|humanities.*|misc.*|news.*|rec.*|sci.*|soc.*|
talk.*:drop
rmgroup:*:comp.*|humanities.*|misc.*|news.*|rec.*|sci.*|soc.*|
talk.*:drop
checkgroups:group-admin@isc.org:*:verify-news.announce.newgroups
newgroup:group-admin@isc.org:comp.*|misc.*|
news.*:verify- news.announce.newgroups
newgroup:group-admin@isc.org:rec.*|sci.*|
soc.*:verify-news.announce.newgroups
newgroup:group-admin@isc.org:talk.*|
humanities.*:verify-news.announce.newgroups
rmgroup:group-admin@isc.org:comp.*|misc.*|
news.*:verify-news.announce.newgroups
rmgroup:group-admin@isc.org:rec.*|sci.*|
soc.*:verify-news.announce.newgroups
rmgroup:group-admin@isc.org:talk.*|
humanities.*:verify-news.announce.newgroups

## GNU (Фондация за безплатен софтуер)
newgroup:gnu@prep.ai.mit.edu:gnu.*:doit
newgroup:news@*ai.mit.edu:gnu.*:doit
rmgroup:gnu@prep.ai.mit.edu:gnu.*:doit
rmgroup:news@*ai.mit.edu:gnu.*:doit

## LINUX (Захранване с новини от news.lameter.com)
checkgroups:christoph@lameter.com:linux.*:doit
newgroup: christoph@lameter.com:linux.*:doit
rmgroup: christoph@lameter.com:linux.*:doit
```

Стартиране на INN

Пакетът с изходния код INN съдържа скрипт, подходящ за стартиране на *inn* при зареждане. Скриптът обикновено се нарича */usr/ lib/ news/rc.news*. Този скрипт чете аргументи от друг скрипт, обикновено наричан */usr/lib/news/innshellvars*, който съдържа дефиниции на имена и пътищата на файлове, които inn ще използва, за да намери необходимите му компоненти. По принцип е добра идея да изпълнявате *inn* с права за достъп на не-root потребител като `news`.

За да сте напълно сигурни, че *inn* се стартира по време на зареждане, трябва да проверите дали файлът */usr/lib/news/innshellvars* е правилно конфигуриран и след това да извикате скрипта */usr/lib/news/rc.news* от скрипт, който се изпълнява при зареждане.

Като допълнение, съществуват административни задачи, които трябва да бъдат изпълнявани периодично. Тези задачи обикновено се конфигурират за изпълнение от командата *cron*. Най-добрият начин да се направи това е да добавите подходящите команди във вашия файл */etc/crontab* или още по-добре – да създадете подходящия файл в директорията */etc/cron.d*, ако вашата дистрибуция позволява това. Следва пример за такъв файл:

```
# Примерен файл /etc/cron.d/inn, както се използва в дистрибуция на
# Debian.
#
SHELL=/bin/sh
PATH=/usr/lib/news/bin:/sbin:/bin:/usr/sbin:/usr/bin


# Изтичане на срока на новини и overview записи се извършва нощем,
# генериране на отчети.

15 0 * * *  news  news.daily expireover lowmark delayrm
# На всеки час се стартира rnews -U. Това е не само за UUCP сайтове,
# но и
# за обработка на статиите поставени в опашката от in.nnrpd,
# в случай че
# innd не е приел никакви статии.

10 * * * *  news  rnews -U
```

Тези команди ви гарантират, че срокът на старите новини автоматичното изтича всеки ден, и че всички чакащи в опашката статии се обработват на всеки час. Забележете, че те се изпълняват с правата за достъп на потребителя news.

Управление на INN: командата ctlinnd

INN сървърът за новини има с команда за управление на ежедневните си операции. Командата *ctlinn* може да бъде използвана за управление на групи по интереси и захранването им, за получаване на статуса на сървъра и за презареждане, спиране и стартиране на сървъра.

Синтаксисът на командата можете да получите като използвате следното:

```
# ctlinnd -h
```

Тук ще разгледаме някои от по-важните приложения на командата *ctlinnd*; за повече подробности вижте справочната страница на *ctlinnd*.

Добавяне на нова група

За добавяне на нова група използвайте следния синтаксис:

```
ctlinnd newgroup group rest creator
```

Аргументите се дефинират по следния начин:

group
: Име на групата, която ще бъде създадена.

rest
: Този аргумент трябва да бъде кодиран по същия начин както полето *flags* на файла *active*. Стойността по подразбиране е y.

creator
: Името на лицето, създало групата. Ако в името има интервали, поставете го в кавички.

Промяна на група

За промяна на група използвайте следния синтаксис:

```
ctlinnd changegroup group rest
```

Тези аргументи се дефинират по следния начин:

group
: Името на променяната група

rest
: Този аргумент трябва да бъде кодиран по същия начин както полето *flags* на файла *active*.

Тази команда е полезна за промяна на статуса на moderation на група.

Премахване на група

За премахване на група използвайте следния синтаксис:

```
ctlinnd rmgroup group
```

Аргументът се дефинира по следния начин:

group
: Името на групата, която се премахва.

Тази команда премахва определената група по интереси от файла *active*. Тя не оказва влияние върху буфера за новини. Срокът на

всички статии в буфера за определената група ще изтече по обичания начин, но нови статии няма да се приемат.

Промяна на номера на група

За промяна на номера на група използвайте следния синтаксис:

```
ctlinnd renumber group
```

Аргументът се дефинира по следния начин:

group
: Името на групата, чийто номер се променя. Ако *group* е празен низ, се променят номерата на всички групи.

Тази команда обновява lomark за определената група.

Разрешаване/забраняване на четци

За да разрешите или забраните четци използвайте следния синтаксис:

```
ctlinnd readers flag text
```

Аргументите се дефинират по следния начин:

flag
: Ако зададете `n`, всички връзки на четци ще бъдат забранени; ако зададете `y`, всички връзки на четци ще бъдат разрешени.

text
: Този текст ще бъде предоставян на четци, които се опитват да се свържат и обикновено описва причината за забраняването на достъпа на четец. Когато отново разрешите достъпа, това поле трябва да бъде празен низ или копие на текста, използван при забраната.

  Тази команда не оказва влияние върху входящите захранвания с новини. Тя само контролира достъпа на четците.

Отхвърляне на връзки за захранване с новини

За отхвърляне на връзки за захранвания с новини използвайте следния синтаксис:

```
ctlinnd reject reason
```

Аргументът се дефинира по следния начин:

reason

Приложеният текст трябва да обяснява защо се отхвърлят входящите връзки към *innd*,

Тази команда не оказва влияние върху връзки, които се предават на *nnrpd* (т.е. четците); тя влияе само върху връзките, обработвани директно от *innd*, например, отдалечени захранвания с новини.

Позволяване на връзки за захранване с новини

За да позволите връзки за захранване с новини, използвайте следния синтаксис:

```
ctlinnd allow reason
```

Аргументът се дефинира по следния начин:

reason

Приложеният текст трябва да бъде същия както при предходната команда *reject* или празен низ.

Тази команда отменя ефекта от командата *reject*.

Забраняване на сървър за новини

За да забраните сървър за новини, използвайте следния синтаксис:

```
ctlinnd throttle reason
```

Аргументът се дефинира по следния начин:

reason

Причината за изключването на сървъра.

Тази команда е еквивалентна едновременно и на `newsreader no`, и на `reject`, и е полезна при извършване на аварийна работа по базата данни с новини. Тя гарантира,че никой няма да се опита да обнови базата данни докато работите по нея.

Рестартиране на сървър за новини

За да рестаргирате сървър за новини, използвайте следния синтаксис:

```
ctlinnd go reason
```

Аргументът се дефинира по следния начин:

reason
: Указва причината при спиране на сървъра. Ако полето е празен низ, сървърът се рестартира безусловно. Ако причината е указана, се рестартират само тези функции, които са били забранени с причина, съвпадаща с приложения текст.

Тази команда се използва за възстановяване на функциите на сървъра след *throttle*, *pause*, или *reject* командите.

Показване на статуса на захранване с новини

За да покажете статуса на захранване с новини, използвайте следния синтаксис:

```
ctlinnd feedinfo site
```

Аргументът се дефинира по следния начин:

site
: Името на сайта, взето от файла *newsfeeds*, за който искате да се покаже статуса на захранването с новини.

Премахване на захранване с новини

За да премахнете захранване с новини, използвайте следния синтаксис:

```
ctlinnd drop site
```

Аргументът се дефинира по следния начин:

site
: Името на сайта, взето от файла *newsfeeds*, за който искате да се премахнете захрантванията с новини. Ако полето е празен низ, се премахват всички захранвания.

Премахването на захранване към сайт спира всички активни захранвания към сайта. Това не е постоянна промяна. Тази команда ще ви бъде полезна, ако модифицирате детайлите на захранването за сайта и то е активно.

Започване на захранване

За да започнете захранване към сайт, използвайте следния синтаксис:

```
ctlinnd begin site
```

Аргументът се дефинира по следния начин:

site
: Името на сайта, взето от файла *newsfeeds*, за който се стартират захранвания. Ако захранването е вече активно, първо се изпълнява автоматично командата *drop*.

Тази команда указва на сървъра да прочете отново файла *newsfeeds*, да намери съответния запис и да започне захранване към определения сайт, използвайки откритите детайли. Можете да използвате тази команда за проверка на ново захранване с новини към сайт, след като сте добавили или редактирали неговия запис във файла *newsfeeds*.

Отмяна на статия

За да отмените статия, използвайте следния синтаксис:

```
ctlinn cancel Message-ID
```

Аргументът се дефинира по следния начин:

Message-ID
: Идентификаторът на статията, която ще бъде отменена.

Тази команда изтрива указаната статия от сървъра. Тя не генерира съобщение за отмяна.

ГЛАВА 24

КОНФИГУРИРАНЕ НА ЧЕТЦИ НА НОВИНИ

Четецът на новини (*newsreader*) е програма, която потребителите използват, за да преглеждат, съхраняват и създават статии. За Linux са адаптирани няколко различни четеца. Ние ще опишем основните настройки за три от най-популярните четци: *tin*, *trn* и *nn*.

Един от най-ефективните четци на новините е командата:

```
$ find /var/spool/news -name '[0-9]*' -exec cat {} \; | more
```

Това е начинът, по който UNIX-маниаците четат своите новини.

Повечето четци обаче са много по усъвършенствани. Те обикновено предлагат пълноекранен интерфейс с различни нива за показване на всички групи, за които е абониран потребителя, резюме на всички статии във всяка от групите и отделни статии. Много web-браузъри работят и като четци на новини, но ако искате да използвате отделни програми четци, тази глава ще ви покаже как да настроите два класически четеца: *trn* и *nn*.

На ниво група по интереси повечето четци показват списък със статиите, техните редове subject (тема) и авторите им. В големите групи за потребителя е много трудно да следи движението на статиите, имащи отношение една към друга, макар че е напълно възможно да разпознаете отговорите на по-ранните статии.

Отговорът обикновено повтаря темата на оригиналната статия, добавяйки пред темата `Re:` (съкращение от Response, отговор – б.р.). Освен това, редът `References:` от заглавието съдържа идентификатора на съобщението, на което текущото съобщение е директен отговор.

Подреждането на статиите по тези два критерия генерира малки групи (всъщност дървета) от статии, които се наричат *нишки* (*threads*). Една от най-важните задачи на четеца е да предложи ефикасна схема за показване на нишките, защото времето, необходимо за това е пропорционално на квадрата на броя статии.

Няма да навлизаме в подробности за начина, по който се създават потребителските интерфейси. Всички четци за новини за Linux имат добри помощни менюта; обърнете се към тях за повече подробности.

В следващите раздели ще разглеждаме само задачите за администриране на четците. Повечето от тези задачи са пряко свързани със създаването на бази данни от нишки и създаването на потребителски акаунти.

Конфигуриране на tin

Най-добрият четец на новини, ако използваме като критерий поддръжката на нишки, е *tin*. Той беше написан от Iain Lea и е свободно моделиран на базата на по-стар четец с име *tass* (написан от Ritch Skrenta). Той извършва управлението на нишките, когато потребителя влиза в групата по интереси и е доста бърз, ако не получавате статиите през NNTP.

На машина 486DX50 са нужни около 30 секунди за създаване на нишките в 1000 статии, когато данните се четат директно от диска. Това би отнело повече от 5 минути през NNTP, за да се достигне до натоварен сървър за новини.[65] Можете да оптимизирате това време, ако периодично обновявате вашия индексен файл като стартирате *tin* с опцията *–u*, така че следващия път, когато използвате *tin*, за да прочетете новините, нишките вече ще са направени. Като алтернатива, можете да стартирате *tin* с опцията *–U*, за да четете новините. Когато е пуснат по този начин, *tin* се разклонява във фонов процес, който прави индексния файл, докато вие четете новините.

Обикновено *tin* поставя базата данни с нишки в личната директория на потребителя в поддиректорията *.tin/index*. Това обаче може да бъде скъпо решение поради загубата на ресурси, така че трябва да пазите само едно копие от базата данни на централно място. Бихте могли да постигнете това като направите изпълнимия файл *tin* setuid

[65] Нещата се променят драстично, ако NNTP сървъра създава сам нишките и остави на клиента само да прегледа базата данни с нишки; INN например прави това.

например **news**. По този начин tin ще пази базата данни с нишки в директорията */var/spool/news/.index*. За всеки достъп до файлове или до обвивката, *tin* ще промени своя потребителски идентификатор на реалния uid на потребителя, който го е стартирал.[66]

Версията на *tin*, включена в някои дистрибуции на Linux, е компилирана без поддръжката на NNTP, но в версията в повечето дистрибуции я притежава. Когато се стартира като *rtin* или с опция *–r*, tin се опитва да се свърже с NNTP сървъра, зададен във файла */etc/nntpserver* или в променливата от обкръжението `NNTPSERVER`. Файлът *nntpserver* съдържа просто името на сървъра на един-единствен ред.

Конфигуриране на trn

Четецът *trn* също е наследник на стар четец, наречен *rn* (съкращение от *read news* – чети новини). Буквата t в името идва от threaded (с нишки). Написан е от Wayne Davidson.

За разлика от *tin*, *trn* няма възможност за генериране на база данни с нишки. Вместо това той използва данните, подготвени от програма, наречена *mthreads*, която трябва да се стартира периодично от *cron*, за да обновява индексните файлове.

Можете да преглеждате новите статии и без да е използвате *mthreads*, но статиите от една тема ще бъдат разпръснати хаотично и няма да имате възможност лесно да следите нишката от съобщения върху определена тема.

За да включите подреждането във нишка за определени групи, стартирайте *mthreads* със списък на желаните групи в командния ред. Формата на този списък е като при файла *sys* при C новините:

```
$ mthreads 'comp,rec,!rec.games.go'
```

Тази примерна команда разрешава създаването на нишки за всички групи от йерархията *comp* и *rec*, с изключение на групите от *rec.games.go* (хората играещи Go, не се нуждаят от такива неща). След това просто стартирайте *mthreads* без опции, за да разпределите в нишки всички новопристигнали статии. Създаването на нишки за

[66] Това е причината, поради която получавате грозни съобщения за грешки, когато стартирате tin като суперпотребител. Но така или иначе, не трябва да вършите нормалната си работа като root.

всички групи във вашия файл *active* става, като стартирате програмата *mthreads* със списък на групи *all*.

Ако приемате новини през нощта, можете да стартирате *mthreads* всяка сутрин или по-често, ако имате нужда от това. Сайтовете с много тежък трафик могат да стартират *mthreads* в режим демон. Когато програмата е стартирана с опцията *–d* по време на началното зареждане на системата, тя работи във фонов режим и проверява на всеки десет минути дали има новопристигнали статии и ги разпределя в нишки. За да стартирате *mthreads* в режим демон, поставете следния ред във вашия скрипт *rc.news*:

```
/usr/local/bin/rn/mthreads -deav
```

Опцията *–a* указва на програмата автоматично да добавя в базата данни с нишки всички новосъздадени групи по интереси; опцията *–v* ви записва изчерпателни съобщения в дневника на *mthreads*, наречен *mt.log*, който се намира в директорията, в която сте инсталирали *trn*.

Старите статии, които вече са недостъпни, трябва да бъдат изгривани периодично от индексния файл. По подразбиране само статиите с номер по малък от най-ниския маркер ще бъдат премахнати от индексния файл.[67] Статии с по-голям от този номер и изтекъл срок (защото в заглавното поле `Expires`: е зададена по-ранна дата на изтичане) могат никога да не бъдат премахнати, ако използвате опцията *–e*, която указва "подобрен" режим на изтичане. Когато *mthreads* работи в режим демон, опцията *–e* указва на програмата да влиза в такъв режим веднъж дневно, малко след полунощ.

Конфигуриране на nn

Четецът *nn* е написан от Kim F. Storm, който твърди, че основната цел на този четец е да не чете новини. Името е съкращение на "No News" (няма новини) и могото му е "добрата новина е, че няма новини. *nn* е още по-добра."

За да постигне тази амбициозна цел, *nn* се разпространява с голям асортимент от поддържащи инструменти, които не само позволяват създаването на нишки, но и детайлни проверки на целостността на базата данни, акаунти, събиране на статистика на използването и ограничаване на достъпа. Предлага се и административна програма,

[67] Забележка: С новините (описани в глава 21, С новини) не обновяват автоматично най-ниския маркер; трябва да стартирате демона *updatemin*, за да направите това.

наречена *nnadmin*, която ви позволява да изпълнявате тези задачи интерактивно. Тя е много интуитивна, затова няма да се спираме на тези аспекти, а ще разгледаме само генерирането на индексни файлове.

Програмата за управление на базата данни с нишки на *nn* се нарича *nnmaster*. Тя обикновено работи като демон и се стартира от *rc*-файл по време на началното зареждане на системата. Това става с ред:

```
/usr/local/lib/nn/nnmaster -l -r -C
```

Това позволява създаването на нишки за всички групи по интереси, намиращи се във вашия файл *active*.

По същия начин можете да стартирате *nnmaster* периодично от *cron*, като подавате списък с групите, които искате да се обработят. Този списък е много подобен на списъка с абонаменти във файла sys, с изключение на това, че за разделители се използват празни интервали вместо запетаи. За да обозначите всички групи, използвайте празния аргумент " " вместо фиктивното име на група **all**, използвано във файла sys. Едно примерно извикване на *nnmaster* изглежда по следния начин:

```
# /usr/local/lib/nn/nnmaster !rec.games.go rec comp
```

Обърнете внимание, че реда на подреждане е важен. Най-ляво поставената група винаги печели. Затова, ако бяхме поставили групата `!rec.games.go` след `rec`, всички статии от тази група щяха да бъдат включени в създаването на нишки, независимо от следващото изключване на нейна подгрупа.

nn предлага различни методи за премахване на остарелите статии от своята база данни. Първият метод е да обновите базата данни, като сканирате директориите на групите по интереси и премахнете всички записи на статии с изтекъл срок. Това е подразбиращата се операция при стартиране на *nnmaster* с опцията *-E*. Този метод е достатъчно бърз, освен ако не го използвате с протокола NNTP.

Вторият метод действа точно както при стартирането на *mthreads*. Той премахва само записите, съответстващи на статии с номер по малък от най-ниския маркер във файла *active*. Този метод се разрешава, ако използвате опцията *-e*.

И накрая, третият метод изтрива цялата база данни и отново събира всички статии. Той може да се разреши с опцията *-E3*.

Списък на групите, на които трябва да се направи проверка за изтичане, се получава с опцията *–F* по подобен начин на горния. Все пак, ако вашия *nnmaster* работи като демон, трябва първо да го убиете (с опцията *-k*), преди да проверите изтичането на срока и след това отново да го стартирате с оригиналните опции. Затова, правилната команда за проверка на изтичането за всички групи с използване на първия метод е:

```
# nnmaster -kF ""
# nnmaster -lrC
```

Съществуват още много допълнителни флагове, предназначени за фина настройка на работата на *nn*. Ако се интересувате от повече подробности, например премахването на лоши статии или конструирането на профили на статии, прочете справочната страница на *nnmaster*.

За своята правилна работа *nnmaster* разчита на файл, наречен GROUPS, който се намира в директорията */var/bin/nn*. Ако този файл не съществува при първото стартиране на *nnmaster*, то той се създава автоматично. За всяка група по интереси този файл съдържа ред, започващ с името на групата, евентуално следвано от маркер за време и флагове. Можете да редактирате тези флагове, за да разрешите определено поведение за дадената група, но не можете да промените реда, по който са подредени групите.[68] Позволените флагове и тяхното действие са описани също в справочната страница на *nnmaster*.

[68] Техният ред трябва да е съгласуван с реда на записите в (двоичния) файл *MASTER*.

ПРИМЕРНА МРЕЖА: ВИРТУАЛНАТА ПИВОВАРНА

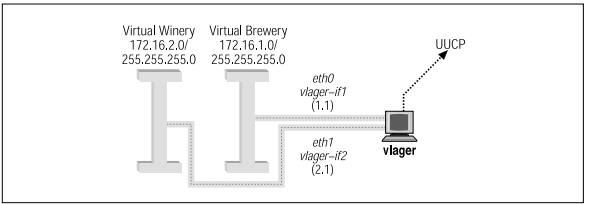
В тази книга използвахме следния пример, който е малко по-малко сложен от Университета Groucho Marx и може би по-близък до задачите, които ще трябва да решавате.

Виртуалната пивоварна е малка компания, която, както предполага името, се занимава с варенето на виртуална бира. За да управляват бизнеса си по-ефективно, виртуалните пивовари искат да свържат компютрите си в мрежа, като всеки компютър е PC, работещ с най-новата и най-добра стабилна версия на ядрото на Linux. На Фигура А-1 е показана конфигурацията на мрежата.

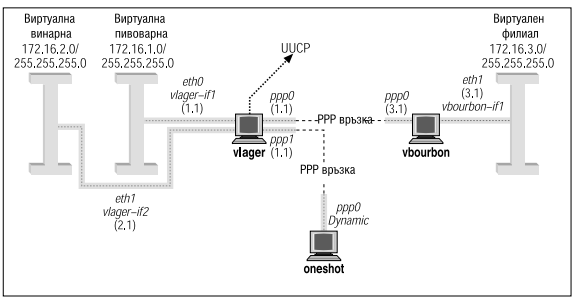
На същия етаж, точно на отсрещната страна на залата, се намира Виртуалната винарна, която работи в тясно сътрудничество с пивоварната. Винарите използват свой собствен Ethernet. Напълно естествено е двете компании да искат да свържат мрежите си в момента, в който те влязат в експлоатация. Като начало, те искат да конфигурират хост-шлюз, който препраща дейтаграми между две подмрежи. По-късно, те биха искали да имат и UUCP връзка с външния свят, през която да обменят поща и новини. В бъдеще, те искат да конфигурират PPP канали за връзка с места извън техния сайт и Интернет.

Виртуалната пивоварна и виртуалната винарна използват по една подмрежа от клас C на мрежата от клас B на пивоварните, които са свързани една с друга през хоста-шлюз **vlager***, който освен това поддържа UUCP връзка. Конфигурацията е показана на Фигура А-1.

* **vlager** е съкращение от виртуално лекo пиво – б.р.



Фигура А-1: Подмрежите на виртуалната пивоварна и виртуалната винарна



Фигура А-2: Мрежата на виртуалната пивоварна

Свързване към мрежата на виртуалния филиал

Виртуалната пивоварна се разраства и открива клон в друг град. Филиалът използва свой собствен Ethernet с номер на IP мрежа **172.16.3.0**, който представлява трета подмрежа на мрежата от клас B на пивоварната. Хостът **vlager** работи като шлюз за мрежата на пивоварната и ще поддържа PPP връзката. Еквивалентният му компонент от новия филиал се нарича **vbourbon**[+] и има IP адрес **172.16.3.1**. Тази мрежа е показана на фигура А-2.

[+] съкращение от виртуален бърбън – б.р.

ПОЛЕЗНИ КАБЕЛНИ КОНФИГУРАЦИИ

Ако искате да свържете два компютъра, но нямате Ethernet мрежа, ще ви е необходим или сериен null-модем кабел, или паралелен кабел за PLIP.

Тези кабели могат да се закупят наготово, но е доста лесно, а и излиза много по-евтино, ако си ги направите сами.

Паралелен кабел за PLIP

За да направите паралелен кабел, който да използвате за PLIP, ще са ви необходими два 25-пинови конектора (наричат се DB-25) и кабел с поне единадесет проводника. Кабелът трябва да е не по-дълъг от 15 метра (50 фута). Не е задължително проводниците да бъдат екранирани, но ако правите дълъг кабел, най-добре е да използвате екраниран кабел.

Като погледнете конектора, трябва да можете да видите малки номера в основата на всяко краче (пин) – от 1 за крачето в горния ляв край (ако държите конектора с широката страна нагоре) до 25 за крачето долу вдясно. Ако искате да направите нулев паралелен кабел, трябва да свържете крачетата на двата конектора един към друг, както е показано на фигура Б-1.

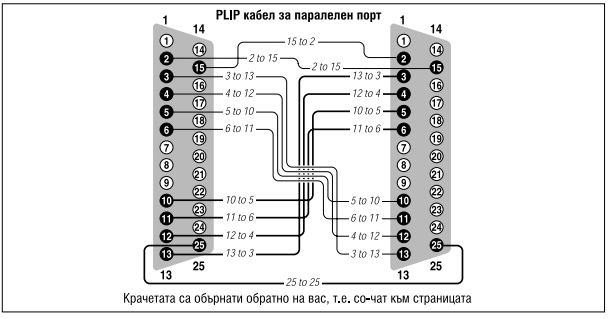
Всички останали крачета не се свързват. Ако кабелът е екраниран, екранът трябва да се свърже към металната обвивка на DB-25 само на *единия* край.

Сериен null-модем кабел

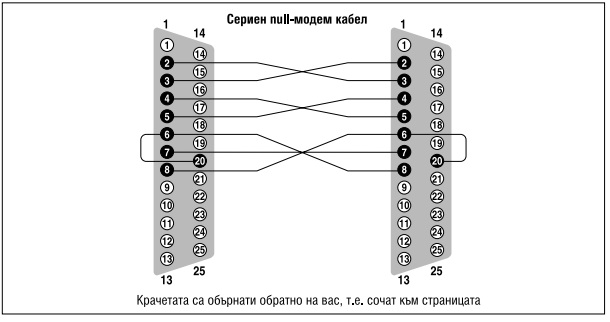
Всеки сериен null-модем кабел може да се използва както за SLIP, така и за PPP. И тук ще са ви необходими два DB-25 конектори, но този път за кабела ви са необходими само осем проводника.

Може да сте виждали и други схеми за null-модем кабел, но тази ви предоставя възможност да използвате сигнала Hardware Flow Control (хардуерно управление на потока данни), което е много по-добър вариант от XON/XOFF за контрол на потока, да не говорим, ако въобще да не използвате контрол. Конфигурацията на проводниците е показана на фигура Б-2:

И отново, ако искате да имате екранировка, трябва да я свържете към първото краче само в единия край.



Фигура Б-1: Паралелен кабел за PLIP.



Фигура Б-2: Сериен null-модем кабел

LINUX – РЪКОВОДСТВО НА МРЕЖОВИЯ АДМИНИСТРАТОР, ВТОРО ИЗДАНИЕ. ИНФОРМАЦИЯ ЗА АВТОРСКИТЕ ПРАВА

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft," which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals;

it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms

of this License. The "Document," below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you."

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque."

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats that do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete

Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History," and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications," preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements." Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements," provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements," and any sections entitled "Dedications." You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included

in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate," and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automati-

cally terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See *http://www.gnu.org/copyleft/*.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ГИЛДИЯТА НА СИСТЕМНИТЕ АДМИНИСТРАТОРИ

Ако не намирате всичко, което ви е необходимо, в съобщенията в групите *comp.os.linux.** и като четете документацията, може би е време да се замислите за постъпване в SAGE – гилдията на системните администратори, спонсорирана от USENIX. Главната цел на SAGE е да направи системното администриране професия. SAGE обединява системни и мрежови администратори, за да благоприятства професионалното и техническото им развитие, за да бъдат споделяни проблеми и техните решения и за комуникация с потребители, управа и търговци по въпросите на системната администрация.

Текущите инициативи на SAGE включват:

- Съвместно с USENIX спонсориране на изключително успешните годишни Конференции на системните администратори (LISA).
- Издаване на *Job Descriptions for System Administrators* с редактор Tina Darmohray – първата от серия много практични книжки и ресурсни ръководства по въпросите и техниките в системната администрация.
- Създаване на архивен сайт с адрес **ftp.sage.usenix.org**, съдържащ материалите от конференциите на системните администратори и друга документация, свързана със системното администриране.

Основаване на работни групи в области, които са важни за системните администратори като работа, публикации, политики, разпространение на електронна информация, обучение, производители и стандарти.

За да научите повече за асоциацията USENIX и нейната специална техническа група SAGE, свържете се с офиса на USENIX на телефон (510) 528-8649 в САЩ или чрез e-mail *office@usenix.org*. Ако искате да получите информация по електронен път, пишете на адрес *info@usenix.org*. Годишната такса за членство в SAGE е $25 (трябва

да сте член и на USENIX). Членовете се радват на безплатни абонаменти за *login:* и *Computing Systems* – издавано на три месеца рецензирано техническо списание, на отстъпки при регистрацията за конференции и симпозиуми и при закупуването на публикации на SAGE, както и на други услуги.