The NetBSD & pkgsrc Guide

(2005/09/29)

The NetBSD Developers

The pkgsrc Developers

The NetBSD & pkgsrc Guide

by The NetBSD Developers , The pkgsrc Developers

Published 2005/09/29 06:54:56

Copyright © 1999, 2000, 2001, 2002 Federico Lupi
Copyright © 2003, 2004, 2005 The NetBSD Foundation

All brand and product names used in this guide are or may be trademarks or registered trademarks of their respective owners.

NetBSD® and pkgsrc® are registered trademarks of The NetBSD Foundation, Inc.

Table of Contents

Purpose of this guide ..... **xix**

I. About NetBSD and pkgsrc ..... **xx**

1 What is NetBSD? 1
- 1.1 The story of NetBSD 1
- 1.2 NetBSD features 1
- 1.3 Supported platforms 2
- 1.4 NetBSD's target users 2
- 1.5 Applications for NetBSD 2
- 1.6 The philosophy of NetBSD 3
- 1.7 How to get NetBSD 3

2 What is pkgsrc? 4
- 2.1 Introduction 4
- 2.2 Overview 4
- 2.3 Terminology 5
- 2.4 Typography 6

II. System installation and related issues **7**

3 Installation 8
- 3.1 Documentation 8
- 3.2 The layout of a NetBSD installation 9
- 3.3 Installation 9
  - 3.3.1 Keyboard 9
  - 3.3.2 Geometries 10
  - 3.3.3 Partitions 10
  - 3.3.4 Hard disk space requirements 11
  - 3.3.5 Retry 11

4 Example Installation 13
- 4.1 Introduction 13
- 4.2 Preparing the installation 13
- 4.3 Creating the installation floppy 13
- 4.4 Last preparatory steps 14
- 4.5 Beginning the installation 15
- 4.6 Partitions 20
- 4.7 Disklabel 25
- 4.8 Creating a disklabel 27
- 4.9 The disk preparation process 31
- 4.10 Choosing the installation media 33
- 4.11 Installing from CD-ROM and DVD 35
- 4.12 Installing via FTP 36
- 4.13 Extracting sets 46
- 4.14 System configuration 49

5 The first boot 57
- 5.1 If something went wrong 57
- 5.2 Login 57
- 5.3 Changing the keyboard layout 58
- 5.4 The man command 58

5.5 Changing the `root` password 59
5.6 Changing the shell 59
5.7 System time 60
5.8 Basic configuration in `/etc/rc.conf` 60
5.9 Enabling FFS soft-dependencies 61
5.10 Rebooting the system 61
6 The second boot 63
6.1 dmesg 63
6.2 Mounting the CD-ROM 63
6.3 Mounting the floppy 64
6.4 Accessing a DOS/Windows partition 64
6.5 Adding users 66
6.6 Shadow passwords 67
6.7 Stopping and rebooting the system 67
7 rc.d System 68
7.1 The rc.d Configuration 68
7.2 The rc.d Scripts 69
7.3 The Role of rcorder and rc Scripts 70
7.4 Additional Reading 70
III. System configuration, administration and tuning 72
8 Editing 73
8.1 Introducing vi 73
8.1.1 The vi interface 73
8.1.2 Switching to Edit Mode 73
8.1.3 Switching Modes & Saving Buffers to Files 74
8.1.4 Yanking and Putting 74
8.1.4.1 Oops I Did Not Mean to do that! 74
8.1.5 Navigation in the Buffer 74
8.1.6 Searching a File, the Alternate Navigational Aid 75
8.1.6.1 Additional Navigation Commands 75
8.1.7 A Sample Session 75
8.2 Configuring vi 76
8.2.1 Extensions to `.exrc` 77
8.2.2 Documentation 77
8.3 Using tags with vi 78
9 X 79
9.1 What is X? 79
9.2 Configuration 80
9.3 The mouse 81
9.4 The keyboard 82
9.5 The monitor 82
9.6 The video card 82
9.6.1 XFree 3.x 82
9.6.2 XFree86 4.x 83
9.7 Starting X 83
9.8 Customizing X 83
9.9 Other window managers 84

9.10 Graphical login with xdm 85
10 Linux emulation 87
10.1 Emulation setup 87
10.1.1 Configuring the kernel 87
10.1.2 Installing the Linux libraries 87
10.1.3 Installing Acrobat Reader 88
10.2 Directory structure 88
10.3 Emulating /proc 89
11 Audio 91
11.1 Basic hardware elements 91
11.2 BIOS settings 91
11.3 Configuring the audio device 92
11.4 Configuring the kernel audio devices 92
11.5 Advanced commands 93
11.5.1 audioctl(1) 93
11.5.2 mixerctl(1) 93
11.5.3 audioplay(1) 93
11.5.4 audiorecord(1) 94
12 Printing 95
12.1 Enabling the printer daemon 95
12.2 Configuring `/etc/printcap` 96
12.3 Configuring Ghostscript 97
12.4 Printer management commands 99
12.5 Remote printing 99
13 Using removable media 101
13.1 Initializing and using floppy disks 101
13.2 How to use a ZIP disk 101
13.3 Reading data CDs with NetBSD 102
13.4 Reading multi-session CDs with NetBSD 104
13.5 Allowing normal users to access CDs 104
13.6 Mounting an ISO image 105
13.7 Using video CDs with NetBSD 105
13.8 Using audio CDs with NetBSD 106
13.9 Creating an MP3 (MPEG layer 3) file from an audio CD 106
13.10 Using a CD-R writer with data CDs 107
13.11 Using a CD-R writer to create audio CDs 108
13.12 Creating an audio CD from mp3s 108
13.13 Copying an audio CD 109
13.14 Copying a data CD with two drives 109
13.15 Using CD-RW rewritables 109
13.16 DVD support 109
13.17 Creating ISO images from a CD 110
13.18 Getting volume information from CDs and ISO images 110
14 The cryptographic device driver (CGD) 111
14.1 Overview 111
14.1.1 Why use disk encryption? 111
14.1.2 Logical Disk Drivers 111
14.1.3 Availability 112

14.2 Components of the Crypto-Graphic Disk system 112
14.2.1 Kernel driver pseudo-device 112
14.2.2 Ciphers 112
14.2.3 Verification Methods 112
14.3 Example: encrypting your disk 113
14.3.1 Preparing the disk 113
14.3.2 Scrubbing the disk 114
14.3.3 Creating the `cgd` 114
14.3.4 Modifying configuration files 115
14.3.5 Restoring data 116
14.4 Example: encrypted CDs/DVDs 116
14.4.1 Introduction 116
14.4.2 Creating an encrypted CD/DVD 116
14.4.3 Using an encrypted CD/DVD 119
14.5 Suggestions and Warnings 120
14.5.1 Using a random-key cgd for swap 120
14.5.2 Warnings 121
14.6 Further Reading 121
Bibliography 121
15 Concatenated Disk Device (CCD) configuration 123
15.1 Install physical media 123
15.2 Configure Kernel Support 124
15.3 Disklabel each volume member of the CCD 124
15.4 Configure the CCD 126
15.5 Initialize the CCD device 126
15.6 Create a 4.2BSD/UFS filesystem on the new CCD device 127
15.7 Mount the filesystem 128
16 NetBSD RAIDframe 129
16.1 RAIDframe Introduction 129
16.1.1 About RAIDframe 129
16.1.2 A warning about Data Integrity, Backups, and High Availability 129
16.1.3 Getting Help 129
16.2 Setup RAIDframe Support 130
16.2.1 Kernel Support 130
16.2.2 Power Redundancy and Disk Caching 130
16.3 Example: RAID-1 Root Disk 131
16.3.1 Pseudo-Process Outline 132
16.3.2 Hardware Review 133
16.3.3 Initial Install on Disk0/wd0 134
16.3.4 Preparing Disk1/wd1 136
16.3.5 Initializing the RAID Device 139
16.3.6 Setting up Filesystems 141
16.3.7 Setting up kernel dumps 143
16.3.8 Migrating System to RAID 144
16.3.9 The first boot with RAID 146
16.3.10 Adding Disk0/wd0 to RAID 147
16.3.11 Testing Boot Blocks 149
16.4 Testing kernel dumps 151

17 Pluggable Authentication Modules (PAM) 153
17.1 About 153
17.2 Introduction 153
17.3 Terms and conventions 153
17.3.1 Definitions 153
17.3.2 Usage examples 155
17.3.2.1 Client and server are one 155
17.3.2.2 Client and server are separate 156
17.3.2.3 Sample policy 156
17.4 PAM Essentials 157
17.4.1 Facilities and primitives 157
17.4.2 Modules 158
17.4.2.1 Module Naming 158
17.4.2.2 Module Versioning 158
17.4.2.3 Module Path 158
17.4.3 Chains and policies 159
17.4.4 Transactions 160
17.5 PAM Configuration 160
17.5.1 PAM policy files 160
17.5.1.1 The `/etc/pam.conf` file 160
17.5.1.2 The `/etc/pam.d` directory 161
17.5.1.3 The policy search order 161
17.5.2 Breakdown of a configuration line 161
17.5.3 Policies 162
17.6 PAM modules 163
17.6.1 Common Modules 163
17.6.1.1 pam_deny(8) 163
17.6.1.2 pam_echo(8) 163
17.6.1.3 pam_exec(8) 163
17.6.1.4 pam_ftpusers(8) 163
17.6.1.5 pam_group(8) 163
17.6.1.6 pam_guest(8) 163
17.6.1.7 pam_krb5(8) 164
17.6.1.8 pam_ksu(8) 164
17.6.1.9 pam_lastlog(8) 164
17.6.1.10 pam_login_access(8) 164
17.6.1.11 pam_nologin(8) 164
17.6.1.12 pam_permit(8) 164
17.6.1.13 pam_radius(8) 164
17.6.1.14 pam_rhosts(8) 164
17.6.1.15 pam_rootok(8) 165
17.6.1.16 pam_securetty(8) 165
17.6.1.17 pam_self(8) 165
17.6.1.18 pam_ssh(8) 165
17.6.1.19 pam_unix(8) 165
17.6.2 FreeBSD-specific PAM Modules 166
17.6.2.1 pam_opie(8) 166
17.6.2.2 pam_opieaccess(8) 166

17.6.2.3 pam_passwdqc(8) 166
17.6.2.4 pam_tacplus(8) 166
17.6.3 NetBSD-specific PAM Modules 166
17.6.3.1 pam_skey(8) 166
17.7 PAM Application Programming 166
17.8 PAM Module Programming 166
17.9 Sample PAM Application 167
17.10 Sample PAM Module 170
17.11 Sample PAM Conversation Function 172
17.12 Further Reading 174
Bibliography 174
18 Tuning NetBSD 175
18.1 Introduction 175
18.1.1 Overview 175
18.1.1.1 What is Performance Tuning? 175
18.1.1.2 When does one tune? 175
18.1.1.3 What these Documents Will Not Cover 176
18.1.1.4 How Examples are Laid Out 176
18.2 Tuning Considerations 176
18.2.1 General System Configuration 176
18.2.1.1 Filesystems and Disks 176
18.2.1.2 Swap Configuration 177
18.2.2 System Services 177
18.2.3 The NetBSD Kernel 178
18.2.3.1 Removing Unrequired Drivers 178
18.2.3.2 Configuring Options 178
18.2.3.3 System Settings 178
18.3 Visual Monitoring Tools 178
18.3.1 The top Process Monitor 179
18.3.1.1 Other Neat Things About Top 180
18.3.2 The sysstat utility 180
18.4 Monitoring Tools 181
18.4.1 fstat 182
18.4.2 iostat 182
18.4.3 ps 183
18.4.4 vmstat 184
18.5 Network Tools 185
18.5.1 ping 185
18.5.2 traceroute 186
18.5.3 netstat 187
18.5.4 tcpdump 189
18.5.4.1 Specific tcpdump Usage 189
18.6 Accounting 190
18.6.1 Accounting 190
18.6.2 Reading Accounting Information 190
18.6.2.1 lastcomm 190
18.6.2.2 sa 191
18.6.3 How to Put Accounting to Use 192

18.7 Kernel Profiling 192
18.7.1 Getting Started 192
18.7.1.1 Using kgmon 192
18.7.2 Interpretation of kgmon Output 193
18.7.2.1 Flat Profile 193
18.7.2.2 Call Graph Profile 194
18.7.3 Putting it to Use 195
18.7.4 Summary 196
18.8 System Tuning 196
18.8.1 Using sysctl 196
18.8.2 memfs & softdeps 197
18.8.2.1 Using memfs 197
18.8.2.2 Using softdeps 197
18.9 Kernel Tuning 198
18.9.1 Preparing to Recompile a Kernel 198
18.9.2 Configuring the Kernel 198
18.9.2.1 Some example Configuration Items 198
18.9.2.2 Some Drivers 199
18.9.2.3 Multi Pass 201
18.9.3 Building the New Kernel 201
18.9.4 Shrinking the NetBSD kernel 201
18.9.4.1 Removing ELF sections and debug information 201
18.9.4.2 Compressing the Kernel 202
19 NetBSD Veriexec subsystem 204
19.1 How it works 204
19.2 Signatures file 204
19.3 Generating fingerprints 204
19.4 Strict levels 206
19.5 Kernel configuration 206
20 Miscellaneous operations 208
20.1 Creating a custom install/boot floppies for i386 208
20.2 Synchronizing the system clock with NTP 208
20.3 Installing the boot manager 210
20.4 Deleting the disklabel 210
20.5 Speaker 211
20.6 Forgot root password? 211
20.7 Adding a new hard disk 212
20.8 Password file is busy? 214
20.9 How to rebuild the devices in /dev 214
IV. Networking and related issues 216
21 Introduction to TCP/IP Networking 217
21.1 Audience 217
21.2 Supported Networking Protocols 217
21.3 Supported Media 218
21.3.1 Serial Line 218
21.3.2 Ethernet 218
21.4 TCP/IP Address Format 219

21.5 Subnetting and Routing 221
21.6 Name Service Concepts 223
21.6.1 /etc/hosts 224
21.6.2 Domain Name Service (DNS) 224
21.6.3 Network Information Service (NIS/YP) 225
21.6.4 Other 225
21.7 Next generation Internet protocol - IPv6 226
21.7.1 The Future of the Internet 226
21.7.2 What good is IPv6? 226
21.7.2.1 Bigger Address Space 227
21.7.2.2 Mobility 227
21.7.2.3 Security 227
21.7.3 Changes to IPv4 227
21.7.3.1 Addressing 227
21.7.3.2 Multiple Addresses 230
21.7.3.3 Multicasting 230
21.7.3.4 Name Resolving in IPv6 231
22 Setting up TCP/IP on NetBSD in practice 233
22.1 A walk through the kernel configuration 233
22.2 Overview of the network configuration files 237
22.3 Connecting to the Internet with a modem 238
22.3.1 Getting the connection information 238
22.3.2 resolv.conf and nsswitch.conf 238
22.3.3 Creating the directories for pppd 239
22.3.4 Connection script and chat file 239
22.3.5 Authentication 240
22.3.5.1 PAP/CHAP authentication 240
22.3.5.2 Login authentication 241
22.3.6 pppd options 241
22.3.7 Testing the modem 241
22.3.8 Activating the link 242
22.3.9 Using a script for connection and disconnection 243
22.3.10 Running commands after dialin 243
22.4 Creating a small home network 244
22.5 Setting up an Internet gateway with IPNAT 246
22.5.1 Configuring the gateway/firewall 247
22.5.2 Configuring the clients 248
22.5.3 Some useful commands 248
22.6 A common LAN setup 249
22.7 Connecting two PCs through a serial line 249
22.7.1 Connecting NetBSD with BSD or Linux 249
22.7.2 Connecting NetBSD and Windows NT 250
22.7.3 Connecting NetBSD and Windows 95 251
23 The Internet Super Server inetd 252
23.1 Overview 252
23.2 What is inetd? 252
23.3 Configuring inetd - /etc/inetd.conf 252
23.4 Services - /etc/services 254

23.5 Protocols - /etc/protocols 254
23.6 Remote Procedure Calls (RPC) - /etc/rpc 254
23.7 Allowing and denying hosts - /etc/hosts.{allow,deny} 255
23.8 Adding a Service 255
23.9 When to use or not to use inetd 256
23.10 Other Resources 257
24 The Domain Name System 258
24.1 DNS Background and Concepts 258
24.1.1 Naming Services 258
24.1.2 The DNS namespace 258
24.1.3 Resource Records 259
24.1.4 Delegation 260
24.1.5 Delegation to multiple servers 261
24.1.6 Secondaries, Caching, and the SOA record 261
24.1.7 Name Resolution 262
24.1.8 Reverse Resolution 262
24.2 The DNS Files 263
24.2.1 /etc/namedb/named.conf 264
24.2.1.1 options 266
24.2.1.2 zone “diverge.org” 266
24.2.2 /etc/namedb/localhost 267
24.2.3 /etc/namedb/zone.127.0.0 268
24.2.4 /etc/namedb/diverge.org 268
24.2.5 /etc/namedb/1.168.192 269
24.2.6 /etc/namedb/root.cache 269
24.3 Using DNS 270
24.4 Setting up a caching only name server 272
24.4.1 Testing the server 272
25 Mail and news 274
25.1 sendmail 276
25.1.1 Configuration with genericstable 277
25.1.2 Testing the configuration 279
25.1.3 Using an alternative MTA 281
25.2 fetchmail 281
25.3 Reading and writing mail with mutt 281
25.4 Strategy for receiving mail 282
25.5 Strategy for sending mail 283
25.6 Advanced mail tools 283
25.7 News with tin 285
26 Miscellaneous networking topics 287
26.1 Bridge 287
26.1.1 Bridge example 287
26.2 Network File System (NFS) 287
26.2.1 NFS setup example 288
26.3 Setting up NFS automounting for /net with amd(8) 289
26.3.1 Introduction 289
26.3.2 Actual setup 290
26.4 IPv6 Connectivity & Transition via 6to4 290

26.4.1 Getting 6to4 IPv6 up & running 291
26.4.2 Obtaining IPv6 Address Space for 6to4 291
26.4.3 How to get connected 292
26.4.4 Security Considerations 292
26.4.5 Data Needed for 6to4 Setup 293
26.4.6 Kernel Preparation 293
26.4.7 6to4 Setup 293
26.4.8 Quickstart using pkgsrc/net/hf6to4 295
26.4.9 Known 6to4 Relay Routers 297
26.4.10 Tunneling 6to4 through an IPFilter firewall 297
26.4.11 Conclusion & Further Reading 298
V. Building the system 300
27 Obtaining the sources 301
27.1 Preparing directories 301
27.2 Terminology 301
27.3 Downloading tarballs 301
27.3.1 Downloading a NetBSD release 302
27.3.2 Downloading snapshots from a NetBSD stable branch 302
27.3.3 Downloading the NetBSD-current development branch 303
27.3.4 Downloading a pkgsrc-200xQy stable branch 303
27.3.5 Downloading the pkgsrc-current development branch 304
27.4 Fetching by CVS 304
27.4.1 Getting CVS 305
27.4.2 Fetching a NetBSD release 305
27.4.3 Fetching a NetBSD stable branch 305
27.4.4 Fetching the NetBSD-current development branch 306
27.4.5 Fetching a pkgsrc-200xQy stable branch 306
27.4.6 Fetching the pkgsrc-current development branch 307
27.4.7 Saving some cvs(1) options 307
27.5 Sources on CD (ISO) 307
28 Crosscompiling NetBSD with `build.sh` 309
28.1 Building the crosscompiler 309
28.2 Configuring the kernel manually 311
28.3 Crosscompiling the kernel manually 311
28.4 Crosscompiling the kernel with `build.sh` 312
28.5 Crosscompiling the userland 313
28.6 Crosscompiling the X Window System 313
28.7 Changing build behaviour 314
28.7.1 Changing the Destination Directory 314
28.7.2 Static Builds 314
28.7.3 Using `build.sh` options 314
28.7.4 make(1) variables used during build 316
29 Compiling the kernel 322
29.1 Requirements and procedure 322
29.2 Installing the kernel sources 322
29.3 Creating the kernel configuration file 323
29.4 Building the kernel manually 324

29.4.1 Configuring the kernel manually 325
29.4.2 Generating dependencies and recompiling manually 325
29.5 Building the kernel using `build.sh` 325
29.6 Installing the new kernel 326
29.7 If something went wrong 326
30 Console drivers 328
30.1 wscons 328
30.1.1 Virtual consoles 328
30.1.2 50 lines text mode with wscons 329
30.1.3 Keyboard mappings 330
30.1.4 Cut&paste on the console with wsmoused 331
30.1.5 Enable scrollback on the console 331
30.2 pccons 331
VI. The pkgsrc user's guide 333
31 Where to get pkgsrc 334
31.1 As tar file 334
31.2 Via SUP 334
31.3 Via CVS 334
32 Using pkgsrc on systems other than NetBSD 335
32.1 Bootstrapping pkgsrc 335
32.2 Platform-specific notes 336
32.2.1 Darwin (Mac OS X) 336
32.2.1.1 Using a disk image 336
32.2.1.2 Using a UFS partition 336
32.2.2 FreeBSD 337
32.2.3 Interix 337
32.2.3.1 When installing Interix/SFU 337
32.2.3.2 What to do if Interix/SFU is already installed 338
32.2.3.3 Important notes for using pkgsrc 338
32.2.4 IRIX 339
32.2.5 Linux 339
32.2.6 OpenBSD 340
32.2.7 Solaris 340
32.2.7.1 If you are using gcc 341
32.2.7.2 If you are using Sun WorkShop 341
33 Using pkgsrc 342
33.1 Working with binary packages 342
33.1.1 Where to get binary packages 342
33.1.2 How to use binary packages 342
33.1.3 A word of warning 343
33.2 Building packages from source 343
33.2.1 Requirements 343
33.2.2 Fetching distfiles 343
33.2.3 How to build and install 343
33.2.4 Selecting the compiler 345
34 Configuring pkgsrc 346
34.1 General configuration 346

34.2 Variables affecting the build process 346
34.3 Developer/advanced settings 347
34.4 Selecting Build Options 347
35 Creating binary packages 349
35.1 Building a single binary package 349
35.2 Settings for creation of binary packages 349
35.3 Doing a bulk build of all packages 349
35.3.1 Configuration 349
35.3.1.1 /etc/mk.conf 349
35.3.1.2 `build.conf` 350
35.3.1.3 `pre-build.local` 350
35.3.2 Other environmental considerations 350
35.3.3 Operation 350
35.3.4 What it does 351
35.3.5 Disk space requirements 351
35.3.6 Setting up a sandbox for chrooted builds 352
35.3.7 Building a partial set of packages 353
35.3.8 Uploading results of a bulk build 353
35.4 Creating a multiple CD-ROM packages collection 355
35.4.1 Example of cdpack 355
36 Frequently Asked Questions 356
36.1 Are there any mailing lists for pkg-related discussion? 356
36.2 Where's the pkgviews documentation? 356
36.3 Utilities for package management (pkgtools) 356
36.4 How to use pkgsrc as non-root 357
36.5 How to resume transfers when fetching distfiles? 358
36.6 How can I install/use XFree86 from pkgsrc? 358
36.7 How can I install/use X.org from pkgsrc? 358
36.8 How to fetch files from behind a firewall 358
36.9 How do I tell **make fetch** to do passive FTP? 359
36.10 How to fetch all distfiles at once 359
36.11 What does "Don't know how to make /usr/share/tmac/tmac.andoc" mean? 360
36.12 What does "Could not find bsd.own.mk" mean? 360
36.13 Using 'sudo' with pkgsrc 360
36.14 How do I change the location of configuration files? 360
36.15 Automated security checks 361
VII. The pkgsrc developer's guide 362
37 Package components - files, directories and contents 363
37.1 `Makefile` 363
37.2 `distinfo` 365
37.3 patches/* 365
37.4 Other mandatory files 366
37.5 Optional files 367
37.6 `work*` 367
37.7 `files/*` 367
38 Programming in `Makefiles` 369
38.1 `Makefile` variables 369

38.1.1 Naming conventions....370
38.2 Code snippets....370
38.2.1 Adding things to a list....370
38.2.2 Converting an internal list into an external list....370
38.2.3 Passing variables to a shell command....371
38.2.4 Quoting guideline....371
38.2.5 Workaround for a bug in BSD Make....372
39 PLIST issues....374
39.1 RCS ID....374
39.2 Semi-automatic PLIST generation....374
39.3 Tweaking output of **make print-PLIST**....374
39.4 Variable substitution in PLIST....374
39.5 Man page compression....375
39.6 Changing PLIST source with PLIST_SRC....376
39.7 Platform-specific and differing PLISTs....376
39.8 Sharing directories between packages....376
40 Buildlink methodology....378
40.1 Converting packages to use buildlink3....378
40.2 Writing buildlink3.mk files....379
40.2.1 Anatomy of a buildlink3.mk file....379
40.2.2 Updating BUILDLINK_DEPENDS.*pkg* in buildlink3.mk files....381
40.3 Writing builtin.mk files....381
40.3.1 Anatomy of a builtin.mk file....382
40.3.2 Global preferences for native or pkgsrc software....383
41 The pkginstall framework....384
41.1 Files and directories outside the installation prefix....384
41.1.1 Directory manipulation....384
41.1.2 File manipulation....385
41.2 Configuration files....385
41.2.1 How PKG_SYSCONFDIR is set....385
41.2.2 Telling the software where configuration files are....386
41.2.3 Patching installations....386
41.2.4 Disabling handling of configuration files....387
41.3 System startup scripts....387
41.3.1 Disabling handling of system startup scripts....388
41.4 System users and groups....388
41.5 System shells....388
41.5.1 Disabling handling of configuration files....388
42 Options handling....390
42.1 Global default options....390
42.2 Converting packages to use bsd.options.mk....390
42.3 Option Names....392
43 The build process....393
43.1 Program location....393
43.2 Main targets....394
43.3 Other helpful targets....396
44 Notes on fixes for packages....401
44.1 General operation....401

44.1.1 How to pull in variables from /etc/mk.conf 401
44.1.2 Where to install documentation 401
44.1.3 Restricted packages 401
44.1.4 Handling dependencies 402
44.1.5 Handling conflicts with other packages 404
44.1.6 Packages that cannot or should not be built 404
44.1.7 Packages which should not be deleted, once installed 404
44.1.8 Handling packages with security problems 405
44.1.9 How to handle compiler bugs 405
44.1.10 How to handle incrementing versions when fixing an existing package 405
44.1.11 Portability of packages 405
44.1.11.1 ${INSTALL}, ${INSTALL_DATA_DIR}, ... 406
44.2 Possible downloading issues 406
44.2.1 Packages whose distfiles aren't available for plain downloading 406
44.2.2 How to handle modified distfiles with the 'old' name 406
44.3 Configuration gotchas 406
44.3.1 Shared libraries - libtool 407
44.3.2 Using libtool on GNU packages that already support libtool 408
44.3.3 GNU Autoconf/Automake 408
44.4 Building considerations 409
44.4.1 CPP defines 409
44.5 Package specific actions 410
44.5.1 User interaction 410
44.5.2 Handling licenses 410
44.5.3 Installing score files 411
44.5.4 Packages containing perl scripts 411
44.5.5 Packages with hardcoded paths to other interpreters 411
44.5.6 Packages installing perl modules 412
44.5.7 Packages installing info files 412
44.5.8 Packages installing GConf2 data files 413
44.5.9 Packages installing scrollkeeper data files 413
44.5.10 Packages installing X11 fonts 413
44.5.11 Packages installing GTK2 modules 414
44.5.12 Packages installing SGML or XML data 414
44.5.13 Packages installing extensions to the MIME database 414
44.5.14 Packages using intltool 415
44.5.15 Packages installing startup scripts 415
44.6 Feedback to the author 415
45 Debugging 416
46 Submitting and Committing 418
46.1 Submitting your packages 418
46.2 General notes when adding, updating, or removing packages 418
46.3 Committing: Importing a package into CVS 418
46.4 Updating a package to a newer version 419
46.5 Moving a package in pkgsrc 419

A. Information **421**
A.1 Where to get this document 421
A.2 Guide history 421
B. A simple example package: bison **422**
B.1 files 422
B.1.1 Makefile 422
B.1.2 DESCR 422
B.1.3 PLIST 422
B.1.4 Checking a package with **pkglint** 423
B.2 Steps for building, installing, packaging 423
C. Build logs **426**
C.1 Building figlet 426
C.2 Packaging figlet 427
D. Layout of the FTP server's package archive **429**
E. Editing guidelines for the pkgsrc guide **431**
E.1 Targets 431
E.2 Procedure 431
F. Contributing to the NetBSD guide **433**
F.1 Translating the guide 433
F.1.1 What you need to start a translation 433
F.1.2 Writing XML/DocBook 434
F.2 Sending contributions 435
F.3 XML/DocBook template 435
G. Getting started with XML/DocBook **437**
G.1 What is XML/DocBook 437
G.2 Installing the necessary tools 437
G.3 Using the tools 438
G.4 Language-specific notes 439
G.4.1 Enabling hyphenation for the Italian language 439
G.5 Links 439
H. Acknowledgements **441**
H.1 Original acknowledgements 441
H.2 Current acknowledgements 441
H.3 Licenses 442
H.3.1 Federico Lupi's original license of this guide 442
H.3.2 Networks Associates Technology's license on the PAM article 442
I. Bibliography **444**
Bibliography 444

List of Tables

17-1. PAM chain execution summary....162
19-1. Veriexec fingerprints tools....204
19-2. Veriexec access type aliases....205

Purpose of this guide

This guide actually consists of two seperate guides, the NetBSD Guide and the pkgsrc Guide.

The NetBSD guide describes the installation and the configuration of the NetBSD operating system as well as setup and administration of some of its subsystems. It addresses mainly people coming from other operating systems in hope of being useful for the solution of the many small problems found when one starts using a new tool.

The pkgsrc Guide documents pkgsrc, the NetBSD Packages Collection which can be used to add third party software after installing the base operating system. The pkgsrc uer's guide covers the basics of pkgsrc, setup and installation by compiling from source and from binary packages is supported. The pkgsrc developer's guide goes into details that are useful when you want to start creating packages on your own.

This guide is not a Unix tutorial: a basic knowledge of some concepts and tools is required to understand it. You should know, for example, what a file and a directory are and how to use an editor. There are plenty of books explaining these things so, if you don't know them, I suggest that you buy an introductory text. I think that it is better to choose a general book and avoid titles like "Learning Unix-XYZ, version 1.2.3.4 in 10 days", but this is a matter of personal taste. If you install a BSD system, sooner or later you will be confronted with the vi editor: without some documentation this could be an insurmountable obstacle. When you finish installing your system, you will be able to install whatever editor and programs you like.

Still a lot of work is required to finish this introduction to NetBSD: some chapters are not finished (some are not even started) and some subjects still need testing (yes, a guide must also be tested). I'll try to work on it and improve it in my spare time but if you want to help, you're welcome: you can write new chapters (or parts of) or send corrections for existing subjects.

This guide is currently maintained by the NetBSD www team (<www@NetBSD.org>). Corrections and suggestions should be sent to that address. See also Appendix F.

I. About NetBSD and pkgsrc

Chapter 1
What is NetBSD?

NetBSD is a free, secure, highly portable Unix-like operating system available for many platforms, from 64bit Opteron servers and desktop systems to handheld and embedded devices. Its clean design and advanced features make it excellent in both production and research environments and it is user-supported with complete source. Many applications are easily available through the NetBSD Packages Collection.

1.1 The story of NetBSD

The first version of NetBSD (0.8) dates back to 1993 and springs from the 4.3BSD Lite operating system, a version of Unix developed at the University of California, Berkeley (BSD = Berkeley Software Distribution), and from the 386BSD system, the first BSD port to the Intel 386 CPU. In the following years, the modifications from the 4.4BSD Lite release (the last release of the Berkeley group) have been integrated in the system. The BSD branch of Unix has had a great importance and influence in the history of this operating system, to which it has contributed many tools, ideas and improvements which are now standard in all Unix environments: the vi editor, the C shell, job control, the Berkeley fast file system, reliable signals, support for virtual memory and TCP/IP, just to name a few. This tradition of research and development survives today in the BSD systems (free and commercial) and, in particular, in NetBSD.

1.2 NetBSD features

NetBSD operates on a vast range of hardware platforms and is very portable, probably the most portable operating system in the world. The full source to the NetBSD kernel and userland is available for all the supported platforms; please see the details on the official site of the NetBSD Project (http://www.NetBSD.org/).

A detailed list of NetBSD features can be found at: http://www.NetBSD.org/Misc/features.html.

The basic features of NetBSD are:

- Portability (more than 50 platforms are supported)
- Security can be taken for granted
- Code quality and correctness
- Adherence to industry standards
- Research and innovation

These characteristics bring also indirect advantages. For example, if you work on just one platform you could think that you're not interested in portability. But portability is tied to code quality; without a well written and well organized code base it would be impossible to support that many platforms. And code

quality is the base of any good and solid software systems, though surprisingly few people seem to understand it. The attention to architectural and quality issues is rewarded with the great potentiality of NetBSD's code and the quality of its drivers.

One of the key characteristics of NetBSD is not to be satisfied with partial implementations. Some systems seem to have the philosophy of "If it works, it's right". In that light NetBSD could be described as "It doesn't work unless it's right". Think about how many overgrown programs are nowadays sadly collapsing under their own weight and "features" and you'll understand why NetBSD wants to avoid this situation at all costs.

1.3 Supported platforms

NetBSD supports over 50 platforms, including the popular PC platform (i386), Opteron system, SPARC and UltraSPARC, Alpha, Amiga, Atari, m68k and PowerPC based Apple Macintosh platforms. Technical details for all of them can be found on the NetBSD site (http://www.NetBSD.org/).

1.4 NetBSD's target users

The NetBSD site states that: "The NetBSD Project provides a freely available and redistributable system that professionals, hobbyists, and researchers can use in whatever manner they wish". I would add that NetBSD is also an ideal system if you want to learn Unix, mainly because of its adherence to standards (one of the project goals) and because it works equally well on the latest PC hardware as well as on hardware which is considered obsolete by most other operating systems; we could say "to learn and use Unix you don't need to buy expensive hardware; you can reuse the old PC or Mac that you have in your attic", still NetBSD will of course rock even more on modern hardware! Also, if you need a Unix system which runs consistently on a variety of platforms, NetBSD is probably your best (only) choice.

1.5 Applications for NetBSD

When you install NetBSD you have a rich set of programs and applications that you can install on your system. Besides having all the standard Unix productivity tools, editors, formatters, C/C++ compilers and debuggers and so on, there is a huge (and constantly growing, currently over 5,000) number of packages that can be installed both from source and in pre-compiled form. All the packages that you expect to find on a well configured system are available for NetBSD for free and there is also a number of commercial applications. In addition, NetBSD provides binary emulation for various other *nix operating systems, thusly allowing you to run non-native applications. Linux emulation is probably the most relevant example, lots of efforts have gone into it and it is used by almost all NetBSD users; you can run the Linux version of

- Netscape
- Acrobat Reader
- Doom, Quake
- Adobe FrameMaker
- many other programs

NetBSD is also capable of emulating FreeBSD, BSDI, Solaris and other systems' binaries.

1.6 The philosophy of NetBSD

Differently from many contemporary operating systems, the NetBSD installation is rich in features, but not huge in size, because it strives to produce a stable and complete base system without being redundant. After the installation you get a fully working base system which can be tuned for various applications then, for example GNOME or KDE and a web browser and other productivity tools for a desktop machine, Apache for a webserver, PostgreSQL or MySQL for a database server, etc. - you have the freedom to decide which programs to install on your machine and the installation of new programs is very easy with the pkgsrc (http://www.pkgsrc.org/) system.

Another advantage of this approach is that the base system will work without these applications; if you decide to upgrade your version of Perl you needn't be afraid to break some parts of your system. When you install NetBSD you don't find huge pre-packaged collections of applications; you may now see this as a disadvantage but when you start understanding the philosophy behind this you will find that it gives you freedom. When you install these software collections (which someone else has decided for you) you fill your hard disk with tons of programs, most of which will stay unused (and unknown) and only waste space (and possibly make the system less stable); this is something which the typical BSD user doesn't want to do.

Even when you start knowing NetBSD, there is always something that will continue to amaze you, the extreme consistency and logic of the system and the attention to the details; nothing appears the result of chance and everything is well thought out. Yes, that's what quality is about and, in my opinion, this is the most distinguishing feature of NetBSD.

We could spend days arguing on the relative merits of operating systems (and some people like to do it) but if you don't try something seriously you can't really judge. I am convinced, because I saw it many times in the mailing lists, that if you try NetBSD you'll be conquered by the perfect balance between complexity and effectiveness; all problems have more than one solution; NetBSD is not happy with *a* solution but always tries to find the easiest and most elegant one. NetBSD is a tool that enables you to do your work without getting in your way. In this light it is an optimal tool; it's like using a pen; you work hard to learn how to use it but once you've learned you can write or draw and completely forget about the pen.

1.7 How to get NetBSD

NetBSD is an Open Source operating system, and as such it's freely available for download from ftp.NetBSD.org (ftp://ftp.NetBSD.org) and its mirrors.

There is no "official" supplier of NetBSD CD-ROMs but there are various resellers. You can find the most up to date list on the relevant page (http://www.NetBSD.org/Sites/cdroms.html) on the NetBSD site.

Chapter 2
What is pkgsrc?

2.1 Introduction

There is a lot of software freely available for Unix-based systems, which usually runs on NetBSD and other Unix-flavoured systems, too, sometimes with some modifications. The NetBSD Packages Collection (pkgsrc) incorporates any such changes necessary to make that software run, and makes the installation (and de-installation) of the software package easy by means of a single command.

Once the software has been built, it is manipulated with the **pkg_*** tools so that installation and de-installation, printing of an inventory of all installed packages and retrieval of one-line comments or more verbose descriptions are all simple.

pkgsrc currently contains several thousand packages, including:

- `www/apache` - The Apache web server
- `www/mozilla` - The Mozilla web browser
- `meta-pkgs/gnome` - The GNOME Desktop Environment
- `meta-pkgs/kde3` - The K Desktop Environment

...just to name a few.

pkgsrc has built-in support for handling varying dependencies, such as pthreads and X11, and extended features such as IPv6 support on a range of platforms.

pkgsrc was derived from FreeBSD's ports system, and initially developed for NetBSD only. Since then, pkgsrc has grown a lot, and now supports the following platforms:

- Darwin (http://developer.apple.com/darwin/) (Mac OS X (http://www.apple.com/macosx/))
- DragonFlyBSD (http://www.DragonFlyBSD.org/)
- FreeBSD (http://www.FreeBSD.org/)
- Microsoft Windows, via Interix (http://www.microsoft.com/windows/sfu/)
- IRIX (http://www.sgi.com/software/irix6.5/)
- Linux (http://www.linux.org/)
- NetBSD (http://www.NetBSD.org/) (of course)
- Tru64 (http://h30097.www3.hp.com/) (Digital UNIX, OSF1)
- OpenBSD (http://www.openbsd.org/)
- Solaris (http://www.sun.com/solaris/)

2.2 Overview

This document is divided into two parts. The first, *The pkgsrc user's guide*, describes how one can use one of the packages in the Package Collection, either by installing a precompiled binary package, or by building one's own copy using the NetBSD package system. The second part, *The pkgsrc developer's guide*, explains how to prepare a package so it can be easily built by other NetBSD users without knowing about the package's building details.

This document is available in various formats:

- HTML (index.html)
- PDF (pkgsrc.pdf)
- PS (pkgsrc.ps)
- TXT (pkgsrc.txt)

2.3 Terminology

There has been a lot of talk about "ports", "packages", etc. so far. Here is a description of all the terminology used within this document.

Package

A set of files and building instructions that describe what's necessary to build a certain piece of software using pkgsrc. Packages are traditionally stored under `/usr/pkgsrc`.

The NetBSD package system

This is the former name of "pkgsrc". It is part of the NetBSD operating system and can be bootstrapped to run on non-NetBSD operating systems as well. It handles building (compiling), installing, and removing of packages.

Distfile

This term describes the file or files that are provided by the author of the piece of software to distribute his work. All the changes necessary to build on NetBSD are reflected in the corresponding package. Usually the distfile is in the form of a compressed tar-archive, but other types are possible, too. Distfiles are usually stored below `/usr/pkgsrc/distfiles`.

Port

This is the term used by FreeBSD and OpenBSD people for what we call a package. In NetBSD terminology, "port" refers to a different architecture.

Precompiled/binary package

A set of binaries built with pkgsrc from a distfile and stuffed together in a single `.tgz` file so it can be installed on machines of the same machine architecture without the need to recompile. Packages are usually generated in `/usr/pkgsrc/packages`; there is also an archive on ftp.NetBSD.org (ftp://ftp.NetBSD.org/pub/NetBSD/packages/).

Sometimes, this is referred to by the term “package” too, especially in the context of precompiled packages.

Program

The piece of software to be installed which will be constructed from all the files in the distfile by the actions defined in the corresponding package.

2.4 Typography

When giving examples for commands, shell prompts are used to show if the command should/can be issued as root, or if “normal” user privileges are sufficient. We use a `#` for root’s shell prompt, and a `%` for users’ shell prompt, assuming they use the C-shell or tcsh.

II. System installation and related issues

Chapter 3
Installation

3.1 Documentation

The documentation of NetBSD is mostly in the format for manual pages and makes up an excellent technical reference to the system. I won't deny that it is unsuited as a tutorial (not to mention the fact that you can't read it until you install NetBSD); these are the reasons for the existence of this guide.

> **Note:** As a matter of fact you could read the man pages through the web interface, but I don't think it is a practical way to learn the system.

After installation you will find some BSD manuals in the `/usr/share/doc` directory. They are divided in three main sections, *psd* (UNIX Programmer's Supplementary Documents), *smm* (UNIX System Manager's Manual) and *usd* (UNIX User's Supplementary Documents). You can read the text on the terminal with, for example:

```
$ cd /usr/share/doc/smm/09.sendmail
$ nroff -me 09.sendmail/intro.me | more
```

or you can generate Postscript output using the `Makefile`s.

It's undeniable that there is a lack of HOWTOs and for this reason you should make the most of the existing ones; the NetBSD release contains some documents in text format and on the NetBSD web site you can find further information and FAQ's.

> **Original documentation:** The NetBSD site contains several pages with documentation and HOWTOs both generic and platform specific. This information is well written and usually easy to understand; for example you can find:
>
> - how to access a DOS/Windows partition from NetBSD
> - how to start NetBSD from the Windows NT boot loader
> - ...

All the versions of NetBSD contain the following files:

`INSTALL.txt`, `INSTALL.ps`, `INSTALL.html`

installation notes. This is the most important document and you should read (and reread it) carefully; it contains a description of the NetBSD system, a list of the supported hardware and, most notably, the installation instructions.

The release(7) manpage

describes the structure of the NetBSD release you are installing.

On the NetBSD web site you can find, amongst the others, the following documentation:

NetBSD FAQ

general information and pointers to other FAQ.

NetBSD/i386 FAQ

NetBSD/i386 specific FAQ.

Basic NetBSD Networking

Guide to network and PPP configuration.

3.2 The layout of a NetBSD installation

The layout of the files of a NetBSD installation is described in the aforementioned `INSTALL.*` files. For example, for the i386 platform the system binaries are in the `i386/binary/sets` directory and the sources are in the `source/sets` directory. The `source/patches` directory contains patches to the base release which usually fix bugs or security related problems discovered after the release.

3.3 Installation

The first thing to do before installing NetBSD is to read the release information and installation notes in one of the `INSTALL.*` files: this is the official description of the installation procedure.

Next you should decide the installation media that you will use; you can choose between:

- FTP
- NFS
- CD-ROM/DVD
- floppy disc
- unmounted filesystem
- local directory

3.3.1 Keyboard

The NetBSD install program sysinst will allow you to change the keyboard layout during the installation. If for some reason this does not work for you, you can use the map in the following table.

US	IT	DE	FR
-	’	ß	)
/	-	-	!

US	IT	DE	FR
=	ì	’	-
:	ç	Ö	M
;	ò	ö	m
#	£	§	3
"	°	Ä	%
*	(	(	8
(	)	)	9
)	=	=	0
’	à	ä	ù
‘	\	^	@
\	ù	#	‘

If you use a non US keyboard, you will need to change the keyboard layout after installation.

3.3.2 Geometries

The installation program mentions two types of hard disk geometries; you should understand what they mean:

- real geometry
- BIOS geometry

real geometry is the real geometry of the hard disk, detected by the system. *BIOS geometry* is the geometry used by the BIOS and it could be different from the real one (for example, BIOS could remap the disk using LBA).

The disk used in the installation example is an IDE disk with the following geometries:

```
real:  6232 cyl,   16 heads,  63 sec
BIOS:   779 cyl,  128 heads,  63 sec   (LBA)
```

As you can see the BIOS remaps the disk using LBA, effectively reducing the number of cylinders and increasing the number of tracks (but the result is the same: 6232 * 16 = 779 * 128 = 99712). A sector contains 512 bytes, which means that the disk size is 6232 * 16 * 63 * 512 = 3 GB. NetBSD does not need to remap the disk geometry (and in fact won’t do it). During the installation it is possible to change manually the geometry if sysinst got it wrong.

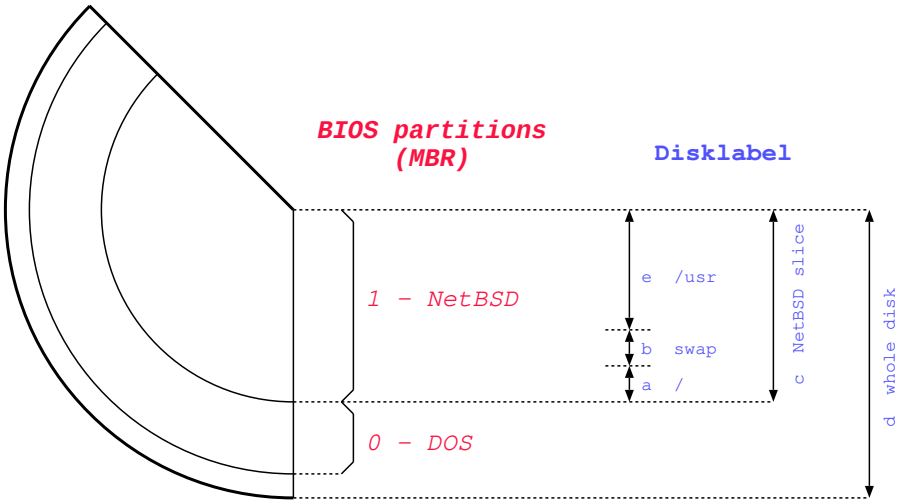
3.3.3 Partitions

The terminology used by NetBSD for partitioning is different from the typical DOS/Windows terminology; in fact, there are two partitioning schemes. NetBSD installs in one of the four primary BIOS partitions (the partitions defined in the hard disk partition table).

Within its BIOS partition (also called *slice*) NetBSD defines the BSD partitions using a *disklabel*: these partitions can be seen only by NetBSD and are identified by lowercase letters (starting with “a”). For example, wd0a refers to the “a” partition of the first IDE disk (wd0) and sd0a refers to the “a” partition

of the first SCSI disk. In Figure 3-1 there are two primary BIOS partitions, one used by DOS and the other by NetBSD. NetBSD describes the disk layout through the disklabel.

Figure 3-1. Partitions



Note: The meaning of partitions “c” and “d” is typical of the i386 port. Other ports use different conventions (e.g. “c” represents the whole disk.)

Note: If NetBSD shares the hard disk with another operating system (like in the previous example) you will probably need to install a *boot manager*, i.e. a program which enables you to choose the OS to start at boot time. sysinst can do this for you and install and configure a simple but effective boot manager.

3.3.4 Hard disk space requirements

The space required by a NetBSD installation depends on the use that you plan to do with it (e.g. server or workstation). For example, consider a home desktop system with a 420 MB hard disk (rather small by today’s standards) with X, the kernel sources and some applications (Netscape, ...). The swap partition is 32 MB. **df** shows the following:

```
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd1a       31887    16848    13444    56%    /
/dev/wd1e      363507   173202   172129    50%    /usr
```

As you can see, NetBSD takes about 225 MB altogether and there are 120 MB left on the `/usr` file system for your and your users’ files.

3.3.5 Retry

When you install an OS for the first time it is seldom a success and NetBSD is no exception. Even if everything goes well, as soon as you start using the system you usually realize that (for example) you could have chosen a better layout for your partitions. It is important not to give up; if you try again you'll realize that what was difficult to understand the first time gradually becomes clearer by virtue of the accumulated experience and numerous rereads of the `INSTALL.*` document.

During the first installations it is wiser to accept the defaults suggested by sysinst and avoid, for example, changing the disklabel.

Chapter 4
Example Installation

4.1 Introduction

This chapter shows an example installation for a two common cases: installation from CD-ROM, and via FTP. The concepts are the same for all types of installation; the only difference is in the way the binary sets are found by sysinst. Please note that some details of the installation differ depending on the NetBSD release: this example was created with release 2.0.

This set of choices gives the impression that the installation is very complicated and requires a lot of work: remember that if you accept the defaults everything is much simpler. On the other hand, a tutorial which explains only the "easy" parts is not very useful (except from the marketing point of view).

4.2 Preparing the installation

Before installing, it is a good idea to make a detailed plan of the steps that you will need to perform. First, read the `INSTALL` file (I promise it's the last time that I say it) reading the description of the installation and checking the hardware compatibilities. Next, if there is already something on the hard disk, think how you can free some space for NetBSD; if NetBSD will share the disk with other operating systems you will probably need to create a new partition (which you will do with sysinst) and, maybe, to resize an existing one. It is not possible to resize an existing partition with sysinst, but there are some commercial products (like Partition Magic) and some free tools (FIPS, pfdisk) available for this.

The installation is divided logically in two steps. In the first part you create a partition for NetBSD and you write the disklabel for that partition. In the second part you decide which binary sets you want to install and extract the files in the newly created partitions. The first part is independent of the installation method (CD-ROM, ftp, NFS, ...); at the end of the first part nothing has yet been written to the hard disk and you are prompted to confirm the installation. If you confirm, the installation goes on, else you are brought back to the main menu and the hard disk remains unchanged.

4.3 Creating the installation floppy

> **Note:** if you have a bootable NetBSD CD-ROM you don't need to create an installation floppy: enable the "boot from CD-ROM" in your BIOS settings, insert the CD and reboot the machine. This option is probably not available on older machines.

Before installing you need to create the installation floppies, i.e. to copy the floppy image from the CD-ROM to a diskette. To perform this operation in DOS you can use the rawrite program in the

`i386/installation/misc` directory, for Windows there's a version in `rawr32.zip`. The image files are `i386/installation/floppy/boot1.fs` and `i386/installation/floppy/boot2.fs` for installation of a "normal" PC, and `i386/installation/floppy/bootlap1.fs` and `i386/installation/floppy/bootlap2.fs` for a laptop. A number of other floppies are available that are described in more detail in the `INSTALL.*` document.

> **Note:** Before creating the installation disks always check that the floppies are good: this simple step is often overlooked and can save you a lot of trouble!

The procedure to write floppies is:

1. Format the floppy.
2. Go to the `I386\INSTALLATION\FLOPPY` directory of the CD-ROM.
3. Run the **`..\MISC\RAWRITE`** program (or extract `..\MISC\RAWR32.ZIP` if you're on a Windows system, and run the RAWRITE32 program in that file). The "Source file"s are `BOOT1.FS` and `BOOT2.FS` (etc., see above) and the "Destination drive" is A:

If you create the boot floppy in a Unix environment, you can use the **dd** command. For example:

```
# cd i386/installation/floppy
# dd if=boot.fs of=/dev/fd0a bs=36b
```

dd copies blocks of 512 bytes: the `bs=36b` option copies 36 blocks at a time, effectively making the operation faster.

> **Note:** A 1440K floppy contains 1474560 bytes and is made up of 80 cylinders, 2 tracks, 18 sectors and 512 bytes per sector, i.e. 80 * 2 * 18 = 2880 blocks. Thus `bs=36b` copies one cylinder (18 * 2 blocks) at a time and repeats the operation 80 times instead of 2880.

4.4 Last preparatory steps

Everything is now ready for the installation but, before beginning, it is better to gather some information on the hardware of the PC.

The most important thing to check is the type of hard disk (IDE, SCSI) and its geometry. You can find this information on the hard disk manual or using a diagnostic program. Some hard disks have a label on which this data is written. Another option is to connect to the web site of the producer of your disk and look for the product info.

If you install via ftp or NFS remember to check your network card settings: if the installation kernel expects your card to be on an IRQ but the card's settings are different you won't be able to install. For example, the install kernel can recognize an NE2000 compatible network card with one of the following two settings:

```
ne0     at isa? port 0x280 irq 9        # NE[12]000 ethernet cards
ne1     at isa? port 0x300 irq 10
```

If your NE2000 network card has different settings it will not be detected. (After the installation you will be able to compile a customized kernel with your own settings.)

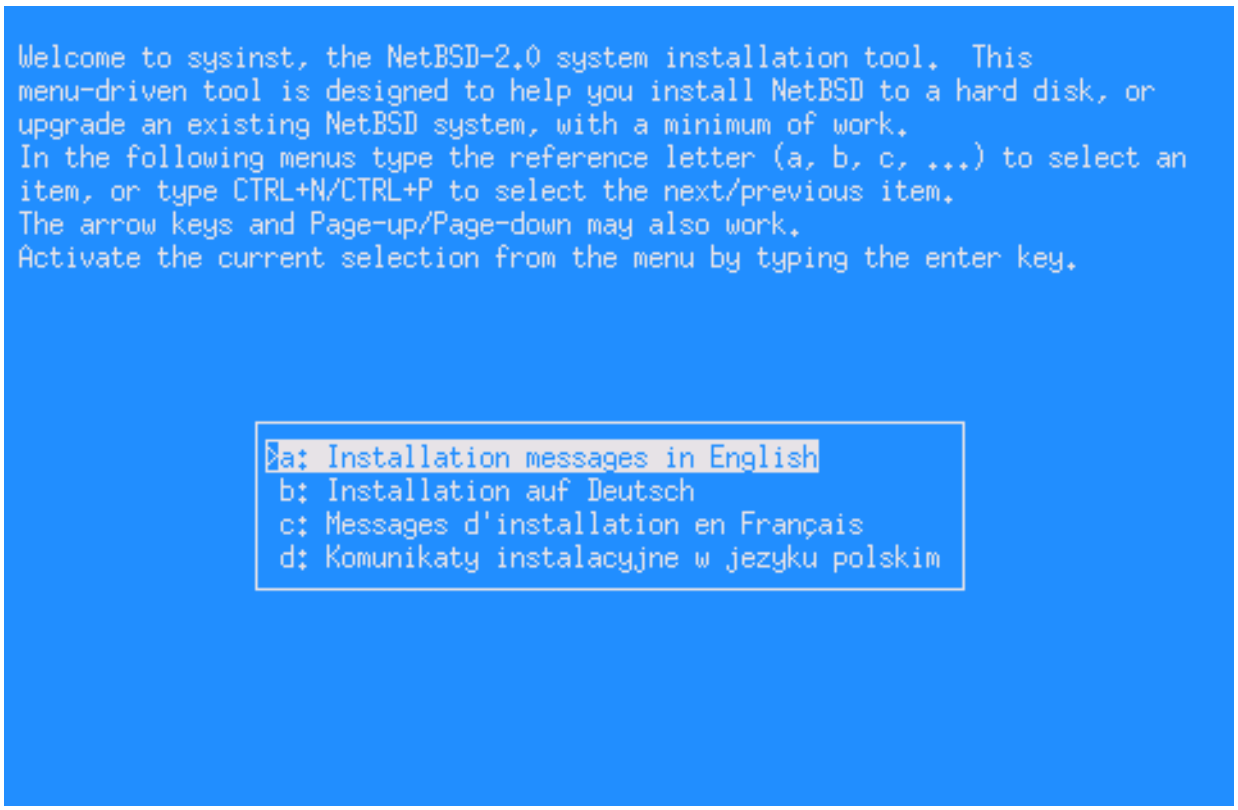
While you are at it you should check some other hardware details like, for example, the number of serial and parallel ports, etc.; this is not required for installation but it can turn out useful later. Check your settings (IRQ, I/O ports, ...) against the ones written in the `INSTALL` file.

> **Note:** you can install even if you don't know the hard disk geometry as well as any of the other details. In this case you must trust sysinst, which automatically determines the geometry and (usually) gets it right.

4.5 Beginning the installation

Insert the newly created installation floppy in drive A: and reboot the computer (or boot from CD-ROM). The kernel on the floppy is booted and starts displaying a lot of messages on the screen, most of which say something about hardware not being found or not being configured. This is normal: the kernel on the floppy tries to detect almost all the hardware supported by NetBSD; you probably (!) don't have all these devices in your machine.

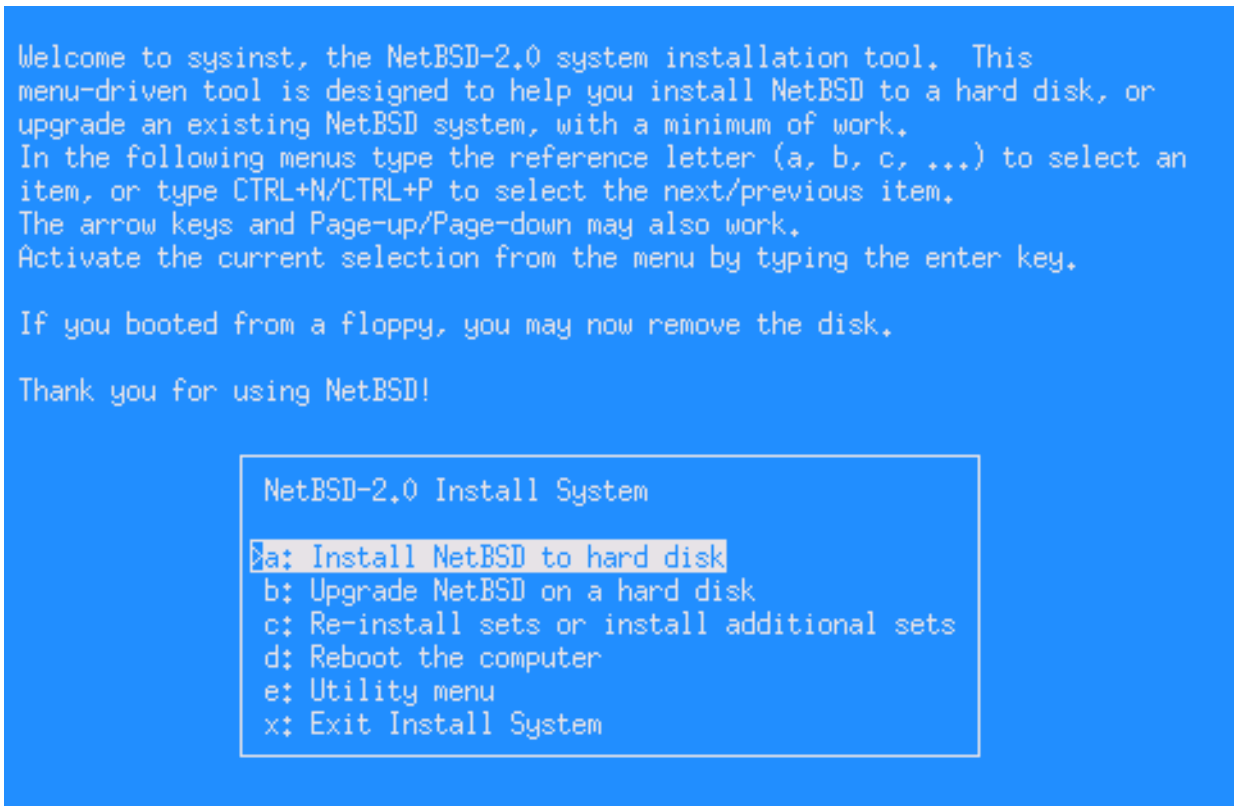
Figure 4-1. Selecting the language



When the boot procedure is over you will find yourself in the NetBSD installation program, sysinst, shown in Figure 4-1. Don't be deceived by the spartan look of sysinst: it is a rather powerful and flexible program. From here on you should follow the instructions displayed on the screen, using the `INSTALL.*` document as a reference. The sysinst screens all have more or less the same layout: the upper part of the screen shows a short description of the current operation or a short help message; the central part of the screen shows the current settings as detected by NetBSD; the bottom part displays a menu of available choices. To make a choice, either use the cursor keys, the "Ctrl+N" (next) and "Ctrl+P" (previous) keys, or press one of the letters displayed left of each choice, and confirm your choice by pressing the Return key.

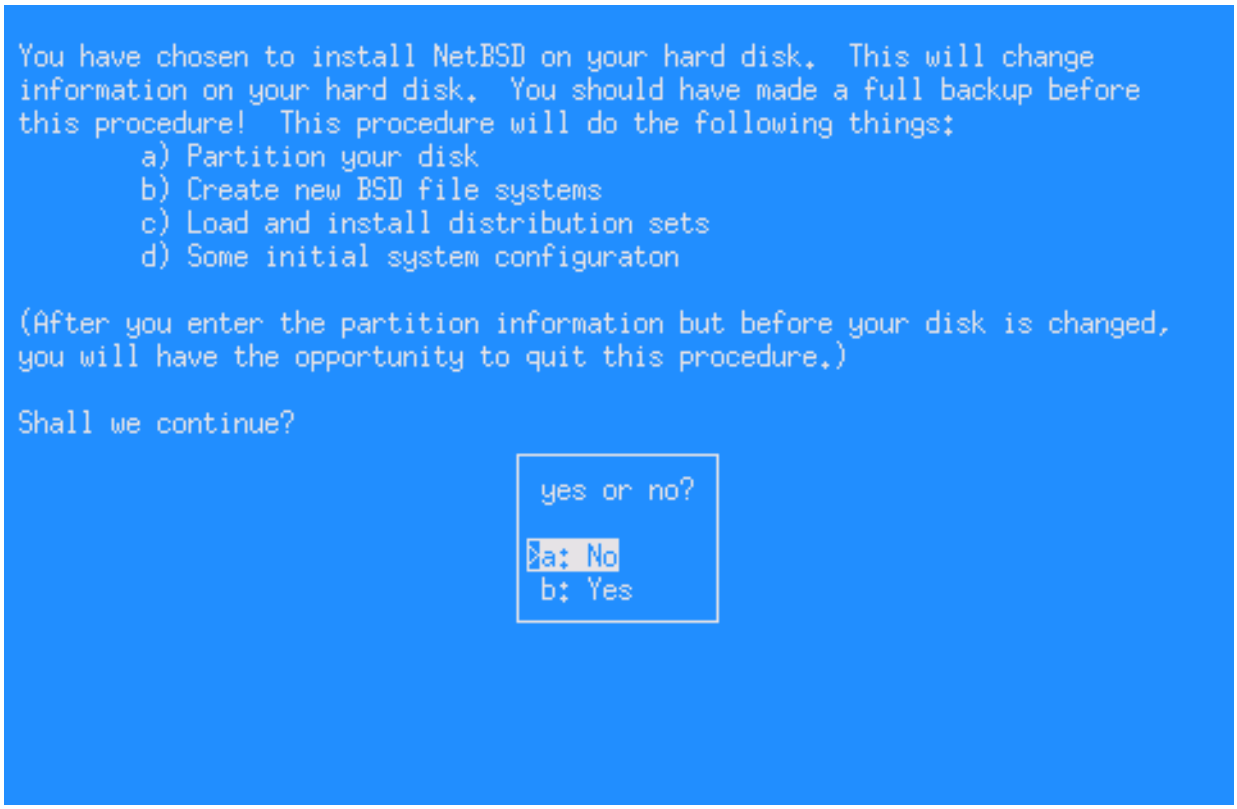
You can now go ahead, and select the language you prefer to do the installation in.

Figure 4-2. The main menu of the installation program



This will bring you to in the main menu of the installation program, as shown in (Figure 4-2. Choosing the Install option brings you to the next screen (Figure 4-3), where you can confirm the operation.

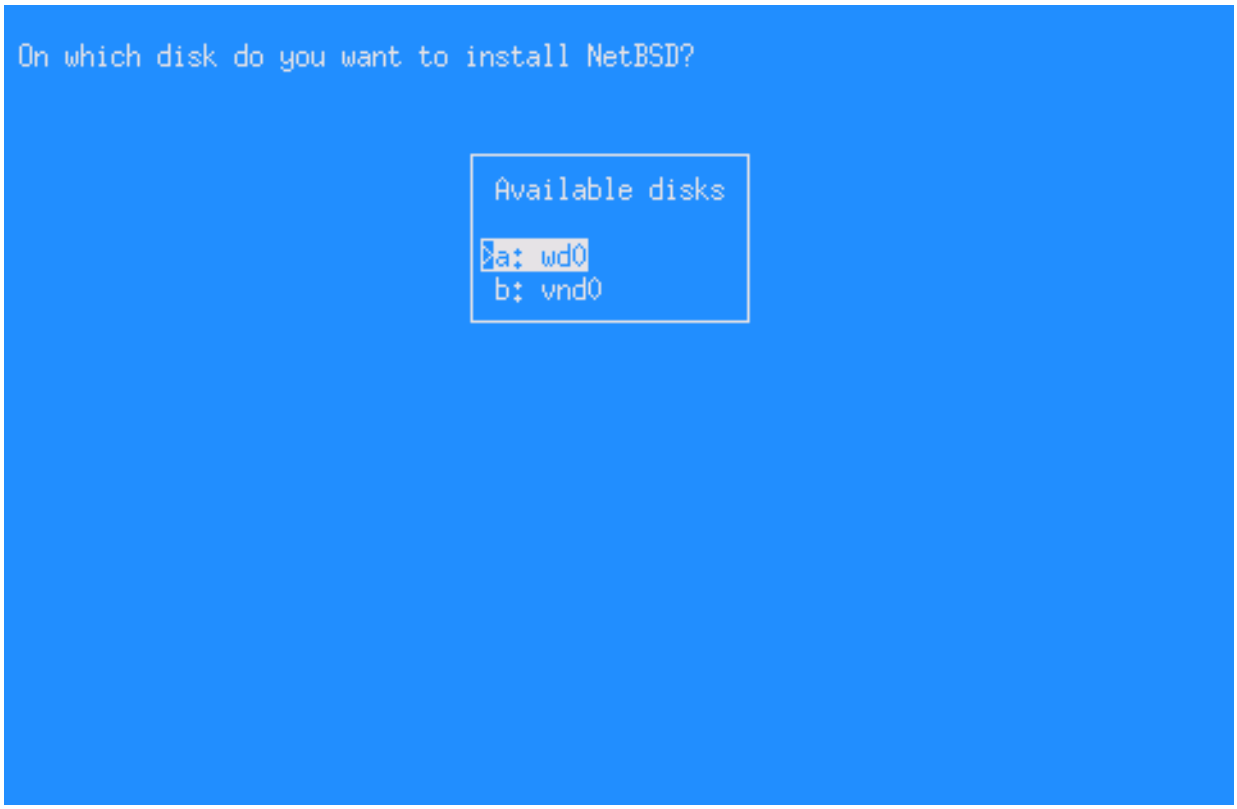
Figure 4-3. Confirming you want to install NetBSD



After choosing to continue with “Yes”, it is time to select on which hard disk you want to install NetBSD. If more than one disk is available, sysinst displays a list of disks from which you can pick one. In the example given in Figure 4-4, there are two disks, and NetBSD will be installed on “wd0”, the first IDE disk found. If you have SCSI disks, they will be named “sd0”, “sd1” and so on.

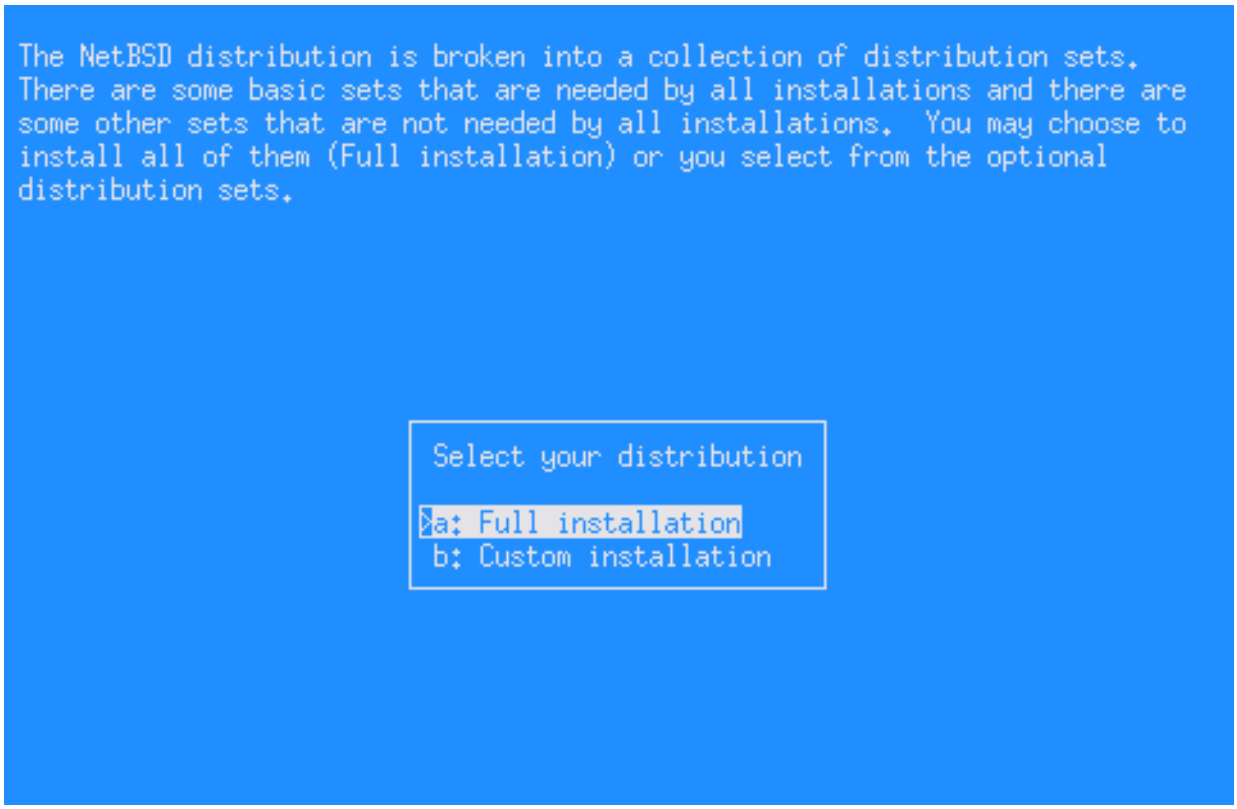
> **Note:** The information in this screen will be different depending on the type and number of hard disks installed on the system.

Figure 4-4. Choosing a hard disk



Sysinst will then ask whether you want to do a full installation, or a custom installation. NetBSD is broken into a collection of distributions sets. “Full installation” will install all sets; if you choose “Custom installation” you can choose which sets you would like to have installed. This step is shown in Figure 4-5.

Figure 4-5. Full or custom installation



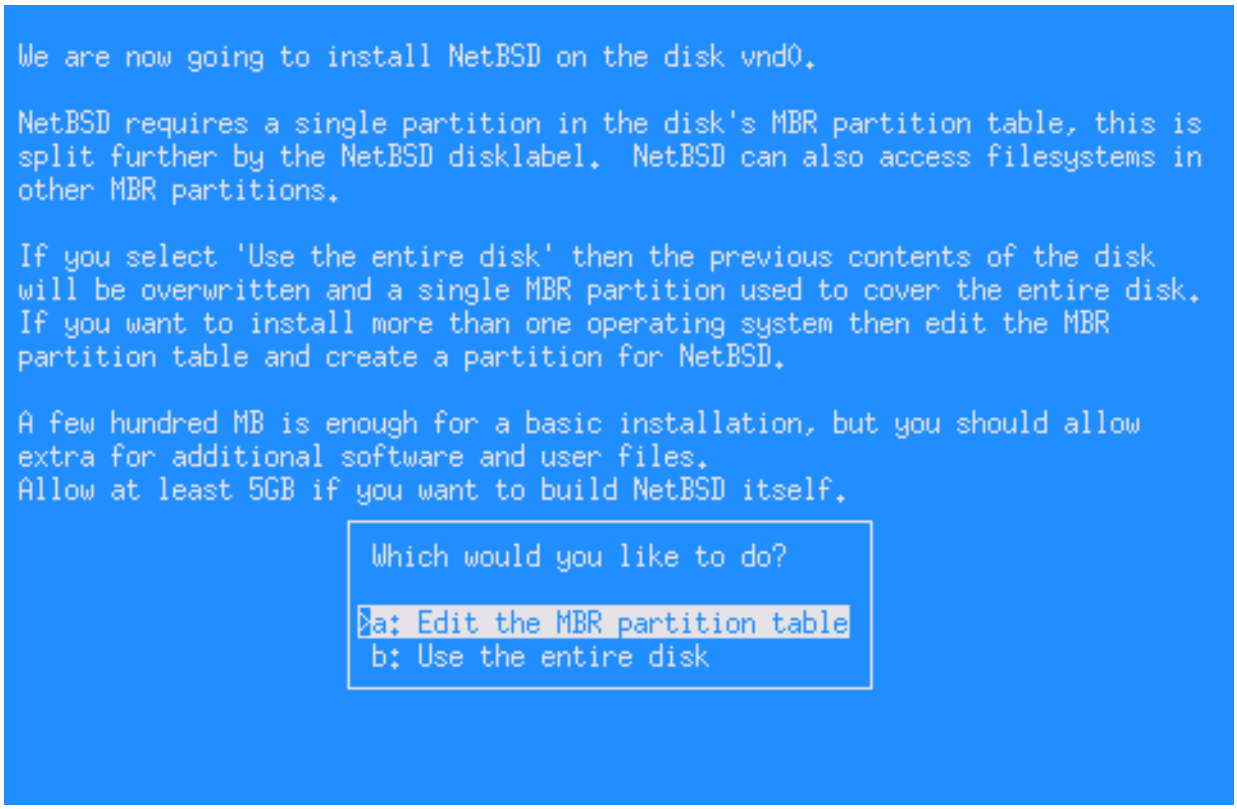
If you chose to do a custom installation, sysinst allows you to choose which distribution sets to install, as shown in Figure 4-6. You will at least need a kernel, “Base” and “System (`/etc`)” for a functional installation.

Figure 4-6. Selecting distribution sets

4.6 Partitions

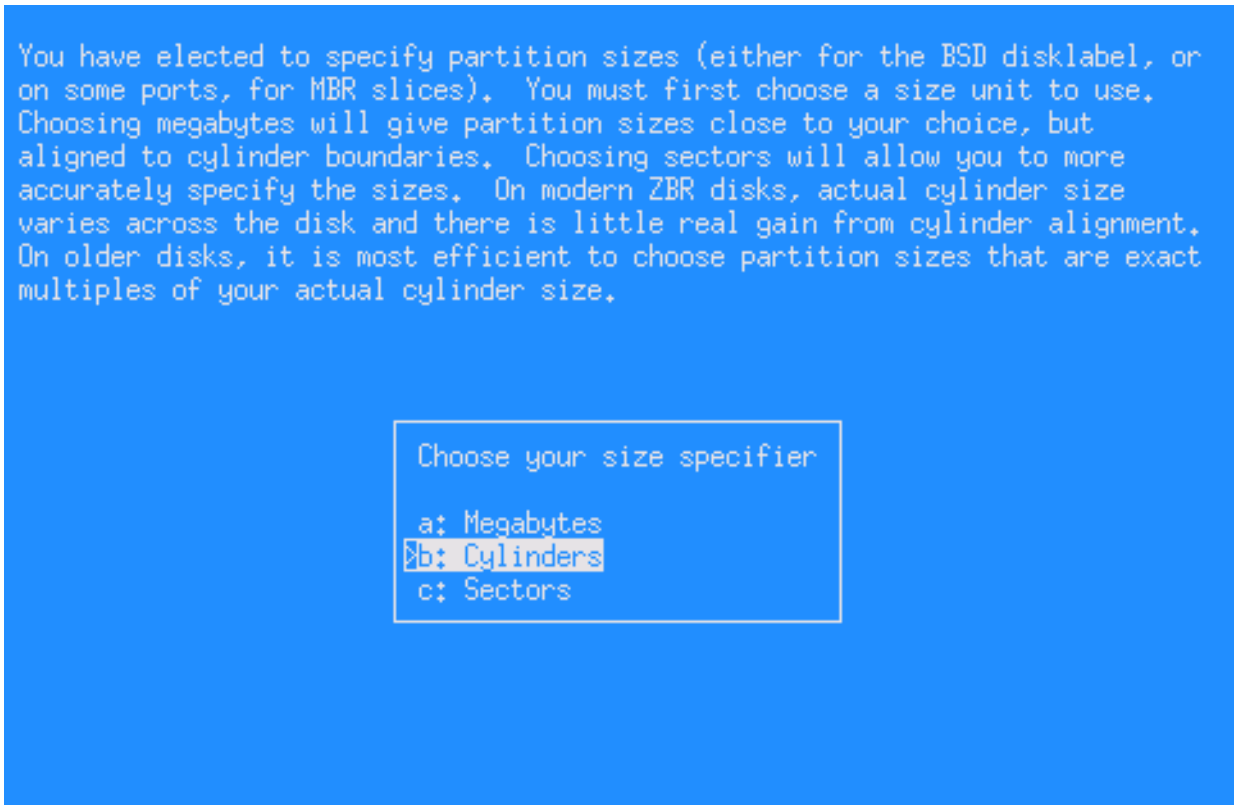
The first important step of the installation has come: the partitioning of the hard disk. First you must specify if NetBSD will use a partition (suggested choice) or the whole disk (“dangerous” choice). In the former case it is still possible to create a partition that uses the whole hard disk (Figure 4-7) so we recommend to select this option which keeps the BIOS partition table in a format compatible with other operating systems.

Figure 4-7. Choosing the partitioning scheme



The next step, depicted in Figure 4-8, is the selection of a unit of measure to be used for hard disk partitioning: sectors give the most flexibility and precision (note that it is usually better to align partition on cylinder boundaries for performance reasons, at least on older hard disks.) Megabytes are easier to use because they don't require manual calculations and are more "intuitive".

Figure 4-8. Choosing a unit of measure



For most people, “Megabytes” will be the most logical choice, because it is the unit to specify partition sizes. After making a choice you are taken to the fdisk interface screen.

Figure 4-9. fdisk

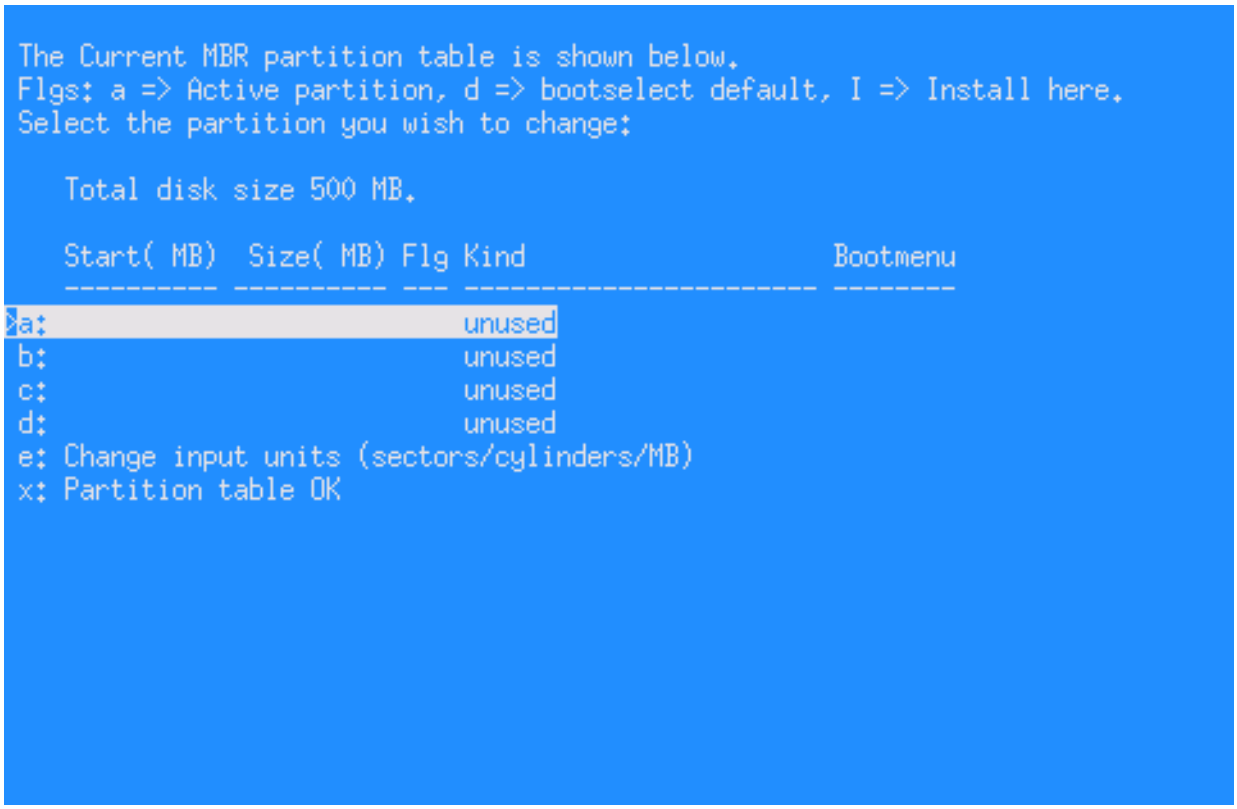
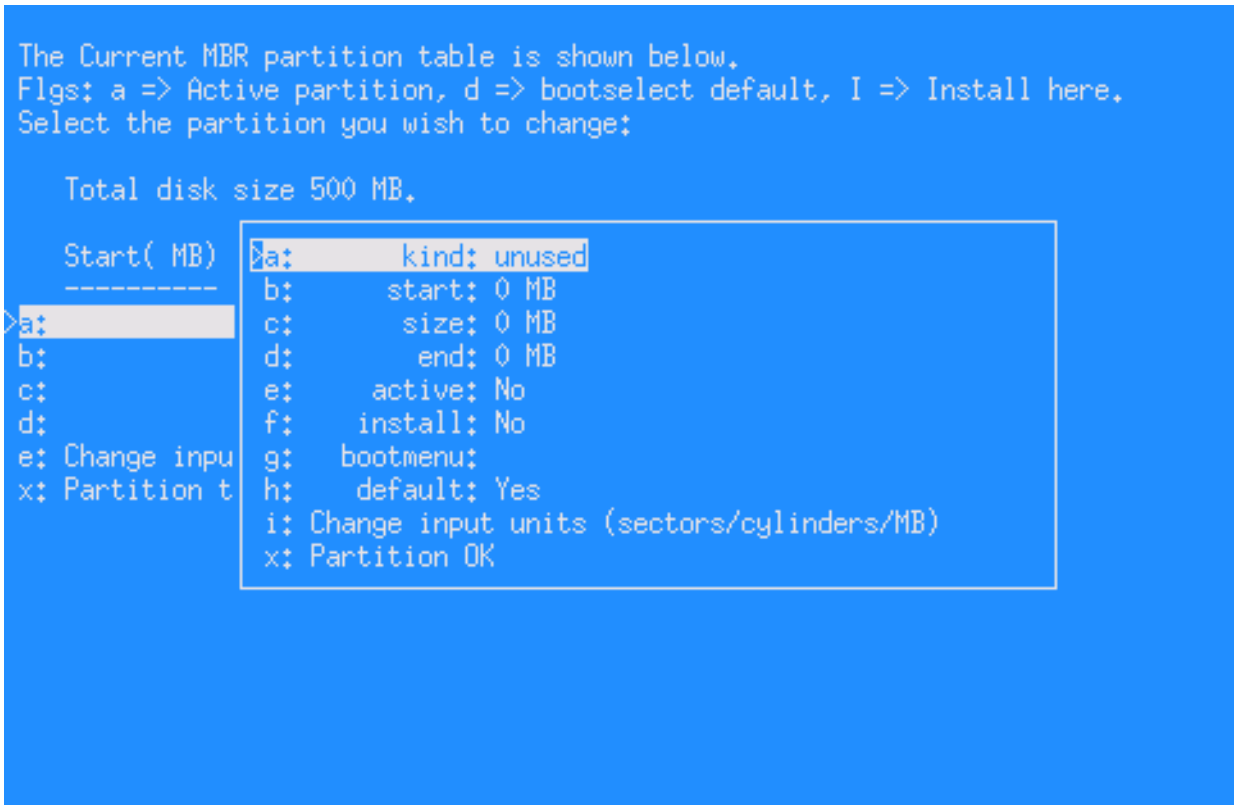


Figure 4-9 shows the current situation of the hard disk before the installation of NetBSD; there are four primary partitions, as you can see this disk is currently empty. If you do have other partitions you can leave them around, and install NetBSD on a partition that is currently unused, or you can wipe out a partition for NetBSD.

Deleting a partition is simple; after selecting the partition a menu with options for that partition appears (Figure 4-10), change the partition kind to “Delete partition” to remove the partition. Of course, if you want to use the partition for NetBSD you can set the partition kind to “NetBSD” right-away.

You can create a partition for NetBSD by selecting the partition you want to install NetBSD to. The partition names “a” to “d” correspond to the four primary partitions on other operating systems. After selecting a partition a menu with options for that partition will show up as shown in Figure 4-10.

Figure 4-10. Partition options



To create a new partition the following information must be supplied:

- the type (kind) of the new partition
- the first (start) sector of the new partition
- the size of the new partition

Choose the partition type “NetBSD” for the new partition (option “Kind”). The installation program will automatically try to guess option “Start”, by starting after the end of the preceding partition. Change this if necessary. The same thing applies to the “Size” option; the installation program will try to fill in the space that is available till the next partition or the end of the disk (depending on which comes first). Change this value if it is incorrect, or if you do not want NetBSD to use the suggested space.

After you have set up the partition kind, start and size, it is also a good idea to set the name that should be used in the boot menu. You can do this by selecting the “bootmenu” option, and filling in how NetBSD should appear in the bootmenu, e.g. “NetBSD”. It is a good idea to repeat this step for other bootable partitions so you can boot both NetBSD and Windows using the NetBSD bootselector. If you are satisfied with the partition options, you can confirm your choice by selecting “Partition OK”. You can choose the same option in the fdisk interface when you have finished partitioning your disk.

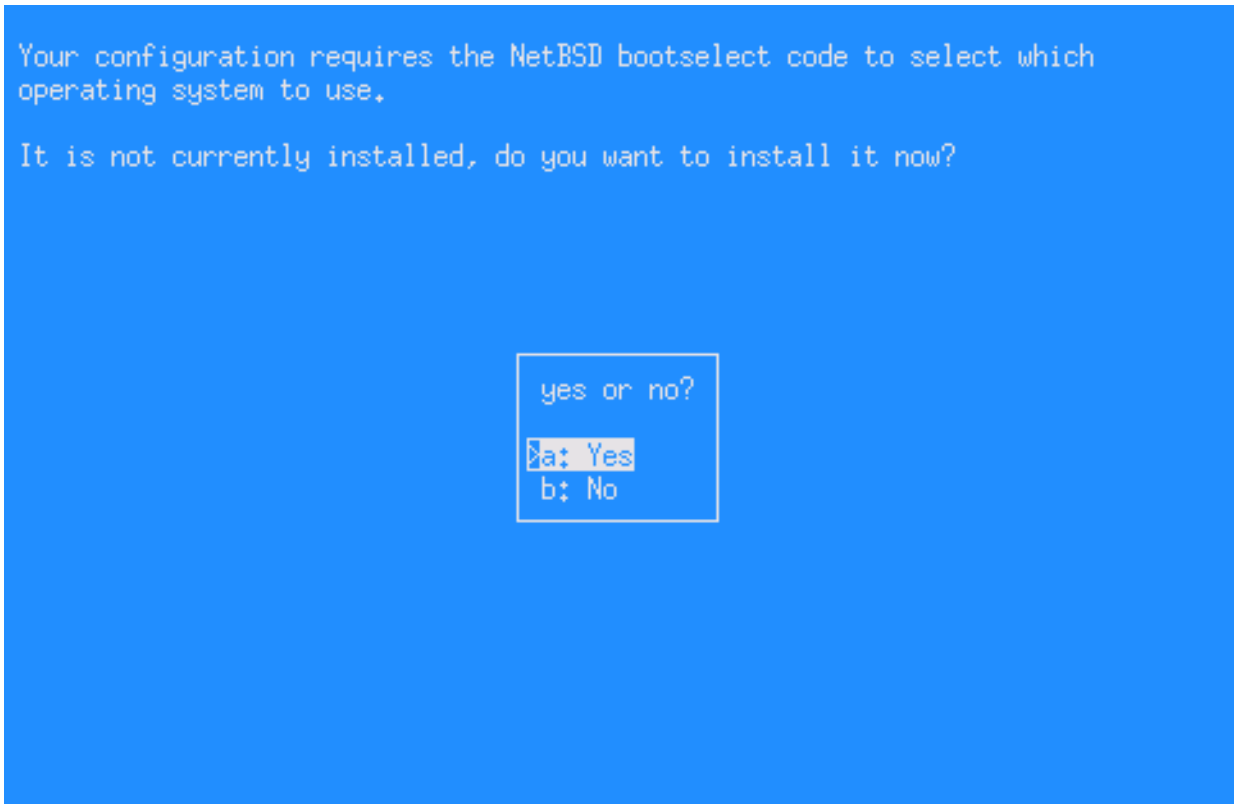
If you have made an error in partitioning (for example you have created overlapping partitions) sysinst will display a message and suggest to go back to the fdisk menu (you are also allowed to continue). If the data is correct but the NetBSD partition lies outside the range of sectors which is bootable by the BIOS,

sysinst warns you and asks if you want to proceed anyway. This could lead to problems on older PCs: the PC used in the example received this warning but boots perfectly. It is not possible to give a general rule (it is BIOS dependent); if the PC is not very old I suggest to ignore the warning and continue.

> **Note:** This is not a limitation of NetBSD; some old BIOSes cannot boot a partition which lies outside the first 1024 cylinders. To fully understand the problem you should study the different type of BIOSes and the many addressing schemes that they use (physical CHS, logical CHS, LBA, ...). These topics are not described in this guide.
>
> With the most recent BIOS, supporting int13 extensions, it is possible to install NetBSD in partitions that live outside the first 8 GB of the hard disk, provided that the NetBSD boot selector is installed.

If the data is correct, sysinst will offer to install a boot selector on the hard disk. This screen is shown in Figure 4-11.

Figure 4-11. Installing the boot selector



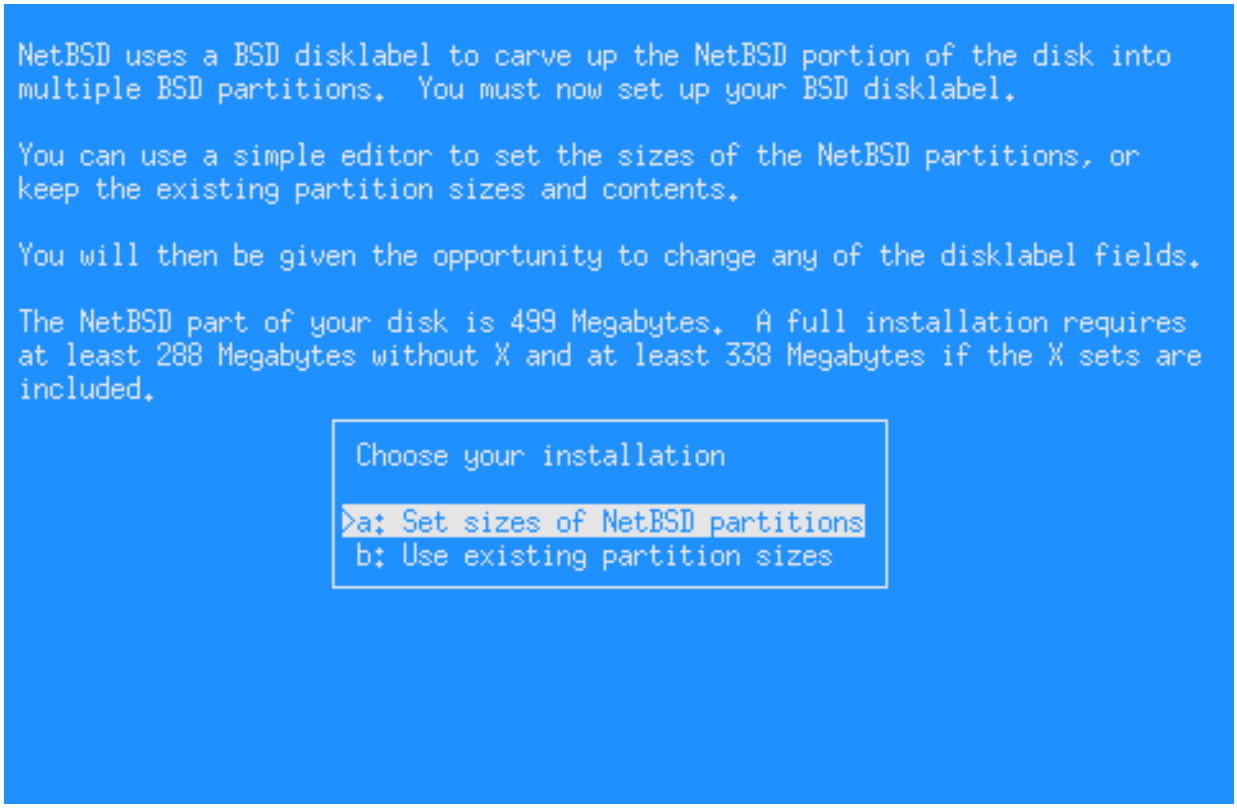
At this point, the first part of the installation, namely disk partitioning, is over.

The BIOS partitions, also called *slices* by BSD, have been created. It's time to define the BSD partitions.

4.7 Disklabel

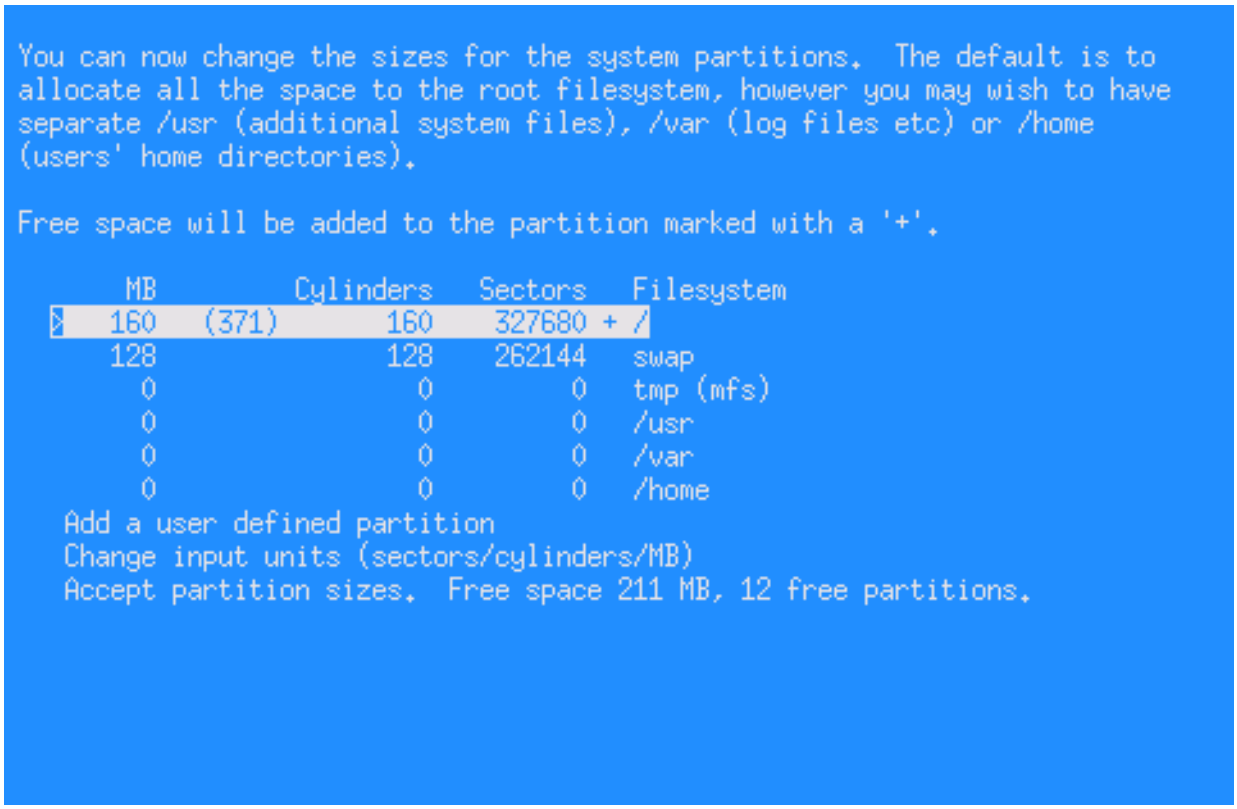
Some platforms, like PC systems (i386), use (DOS-style) MBR partitions to separate filesystems. The NetBSD MBR partition you created earlier in the installation process exists to make sure other operating systems do not overwrite the diskspace that you allocated to NetBSD. NetBSD uses its own partition scheme, named a disklabel, which is stored at the start of NetBSD's MBR partition. In the next few steps you will be able to create a disklabel and set the sizes of NetBSD partitions, or use existing partition sizes, as shown in Figure 4-12.

Figure 4-12. Choosing if partitions should be edited



When choosing to set sizes of NetBSD partitions and before editing the disklabel, you can predefine what partitions you would like to create. The installation program will generate a disklabel based on these settings. This installation screen is shown in Figure 4-13.

Figure 4-13. Setting partition sizes

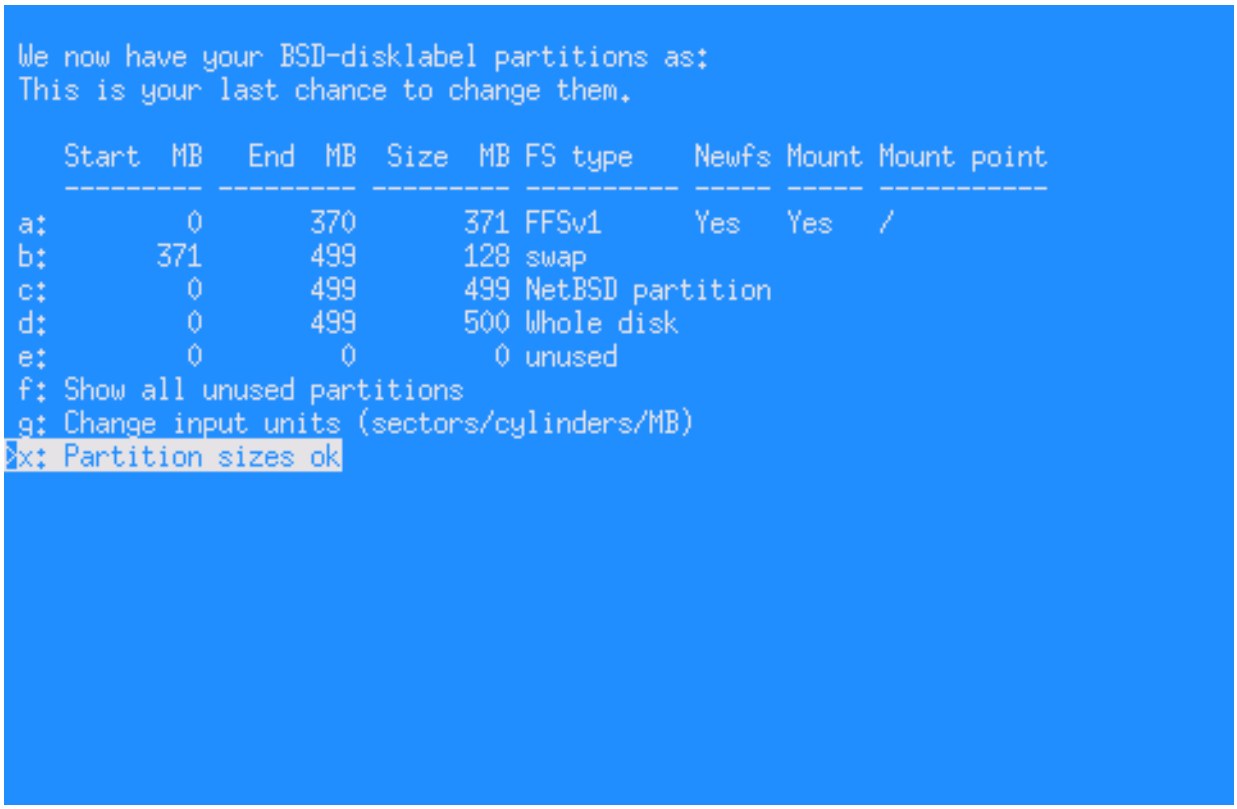


The default partition scheme of just using a big `/` (root) file system (plus swap) works fine with NetBSD, and there is little need to change this. Changing `/tmp` to reside on a RAM disk (mfs) for extra speed may be a good idea. Other partition schemes may use separate partitions for `/var`, `/usr` and/or `/home`, but you may use your own experience to decide if you need this.

4.8 Creating a disklabel

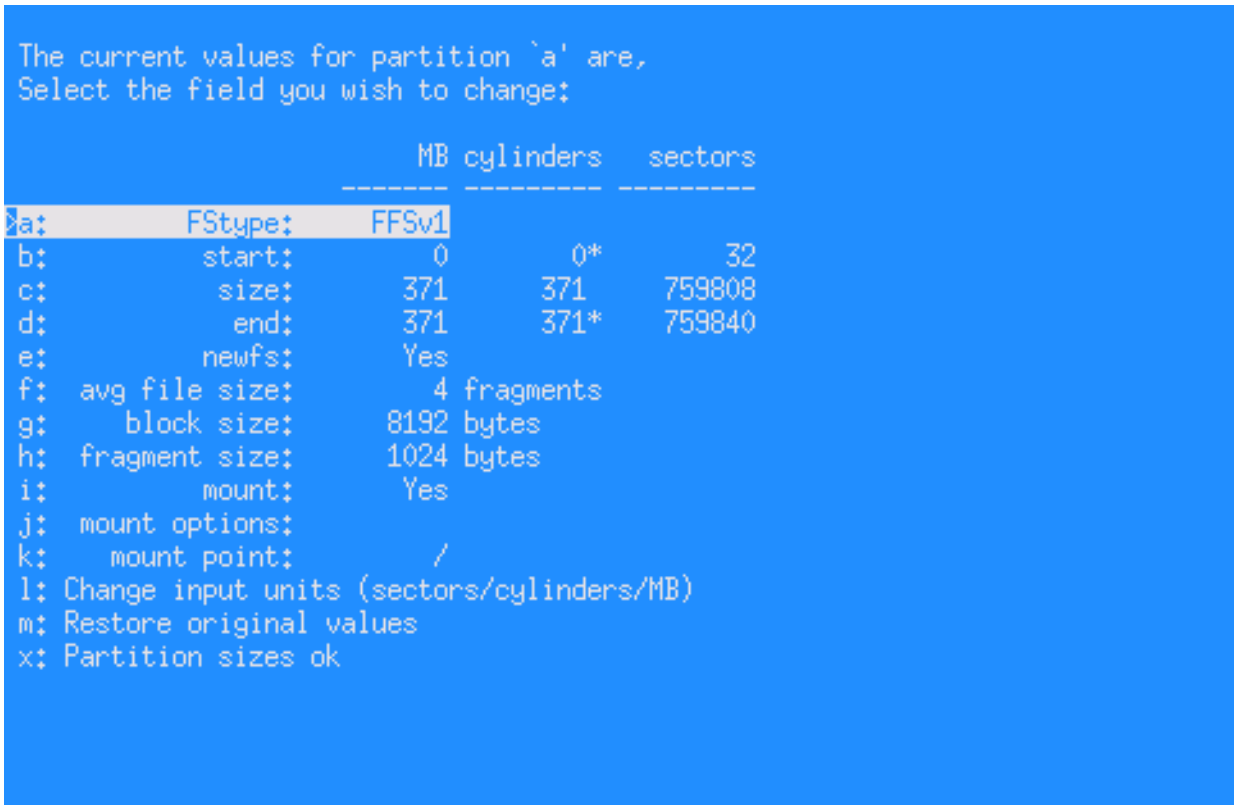
The next step is to create a disklabel, as shown in Figure 4-14. If you predefined the partition sizes in the previous step, the resulting disklabel will probably fit your wishes. In these case you can complete the process immediately by selecting “Partition sizes ok”.

Figure 4-14. The disklabel editor



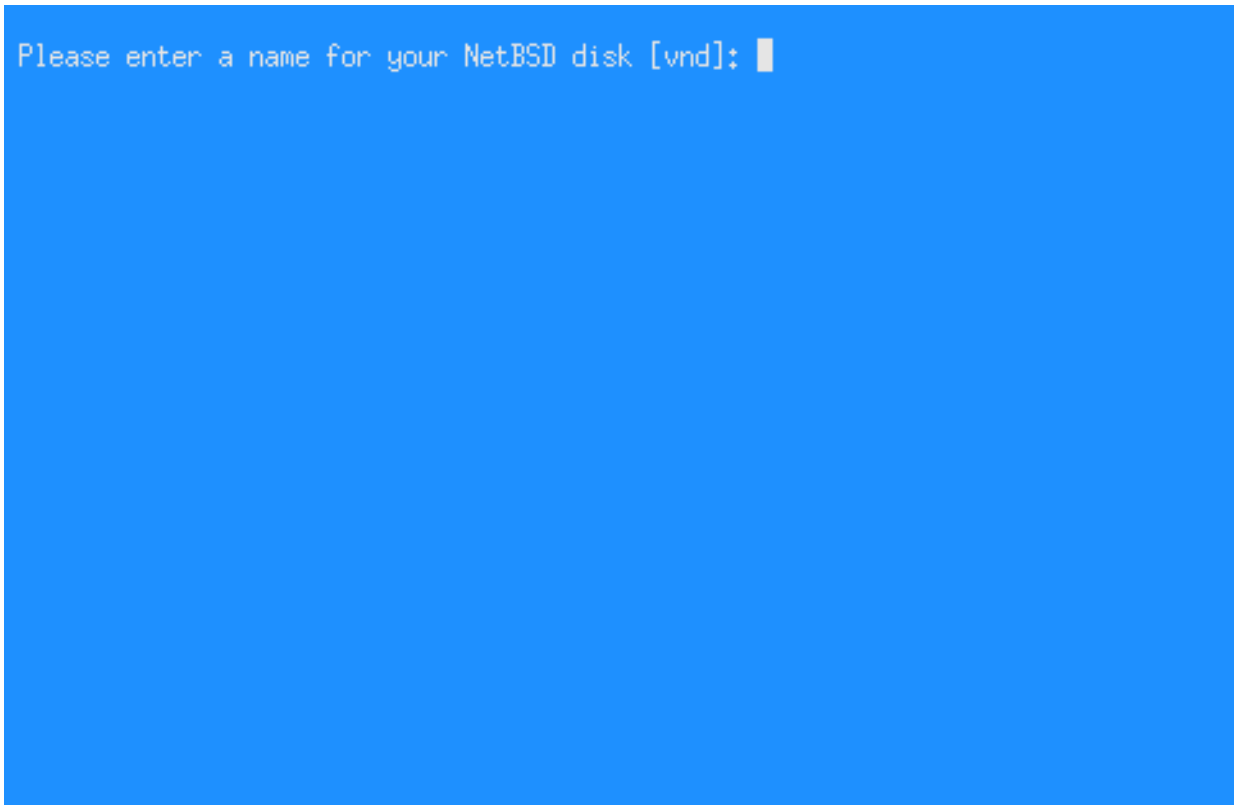
As you can see in Figure 4-14 there are two reserved partitions, “c” (representing the NetBSD partition), and “d” (representing the whole disk). You can edit all other partition by selecting it using the cursor keys and pressing the return key. You can add a partition by selecting an unused slot, and setting parameters for that partition. The partition editing screen is shown in Figure 4-15

Figure 4-15. The disklabel editor



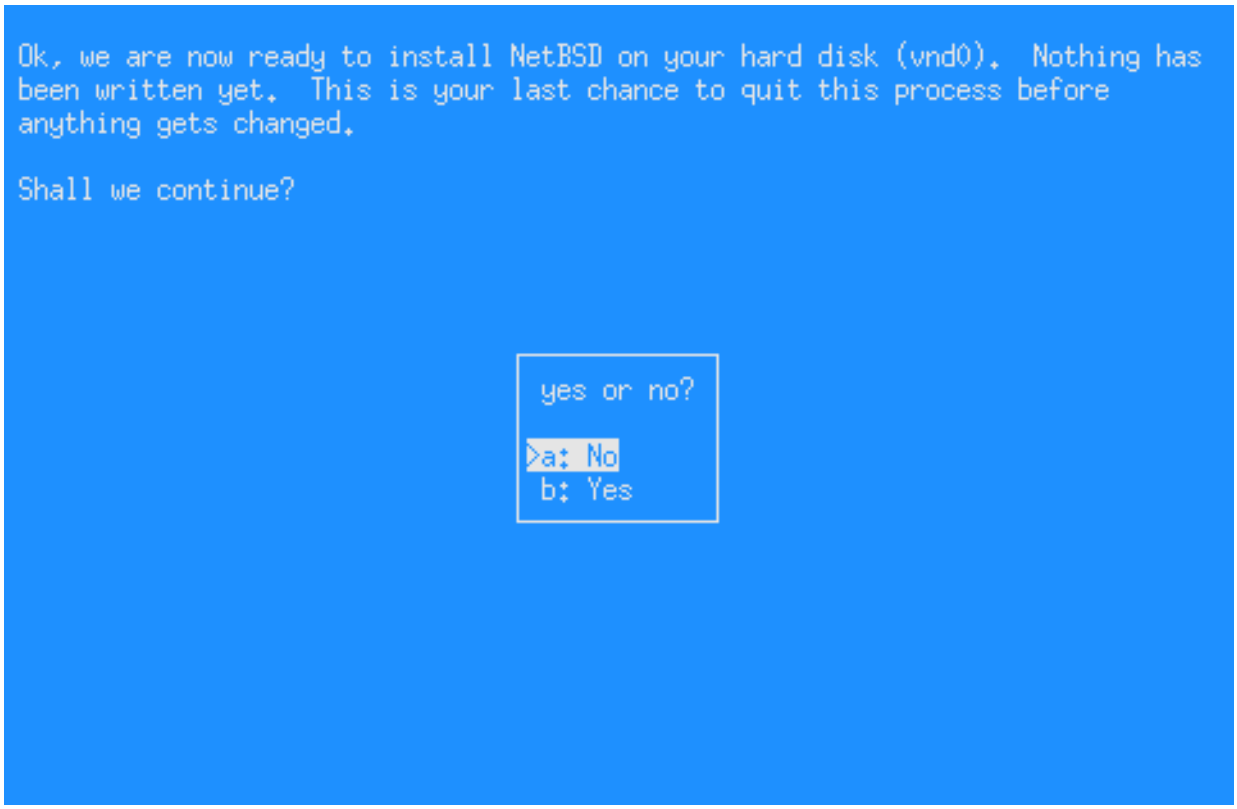
After defining all the data for the new disklabel, the last item is to enter a name for the NetBSD disk as shown in Figure 4-16. This can be used later to distinguish between disklabels of otherwise identical disks.

Figure 4-16. Naming the NetBSD disk



The NetBSD installer now has all the data to prepare the disk for the installation. Nothing has been written to the disk at this point, and now is the last chance to abort the installation process before actually writing data to the disk, as shown in Figure 4-17. Choose “no” to abort the installation process and return to the main menu, or bravely step ahead by selecting “yes”.

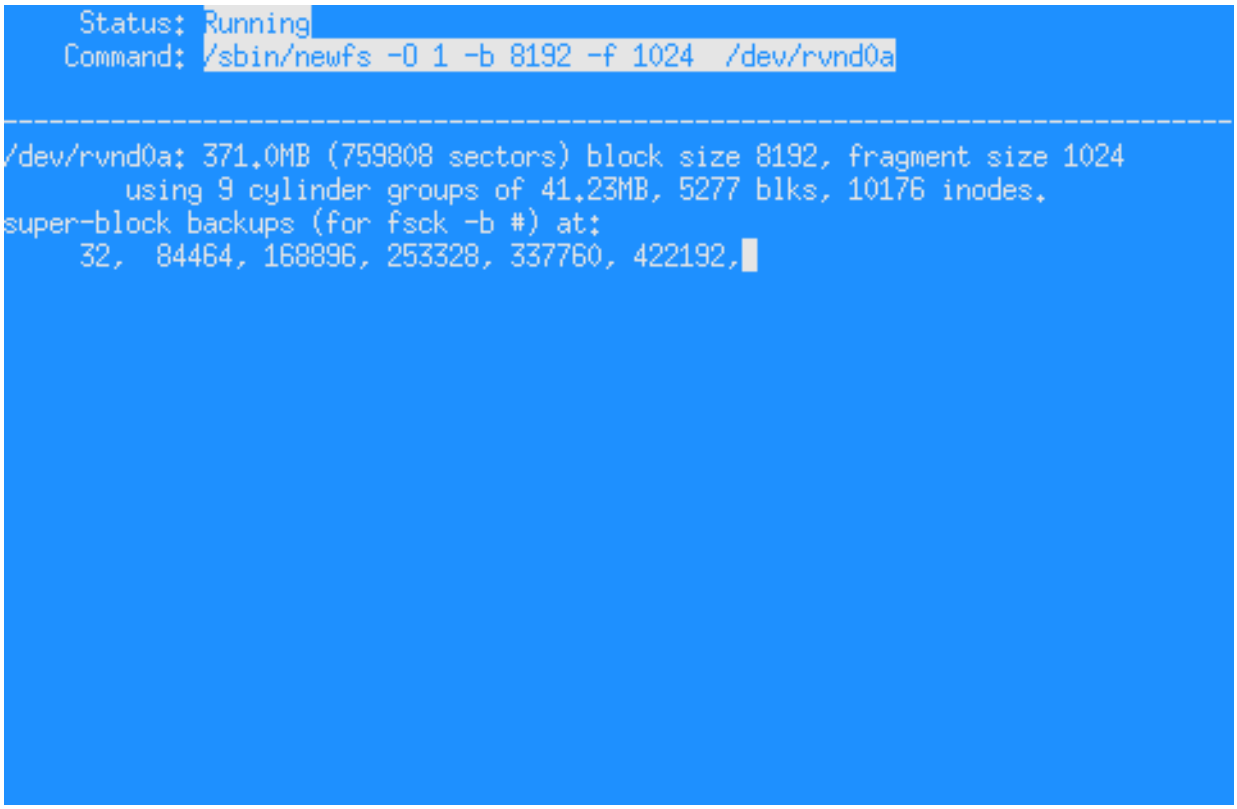
Figure 4-17. Last chance before hitting the disk



4.9 The disk preparation process

After confirming that sysinst should prepare the disk, it will run disklabel(8) to define NetBSD partition layout and newfs(8) to create the file systems on the disk, as shown in Figure 4-18.

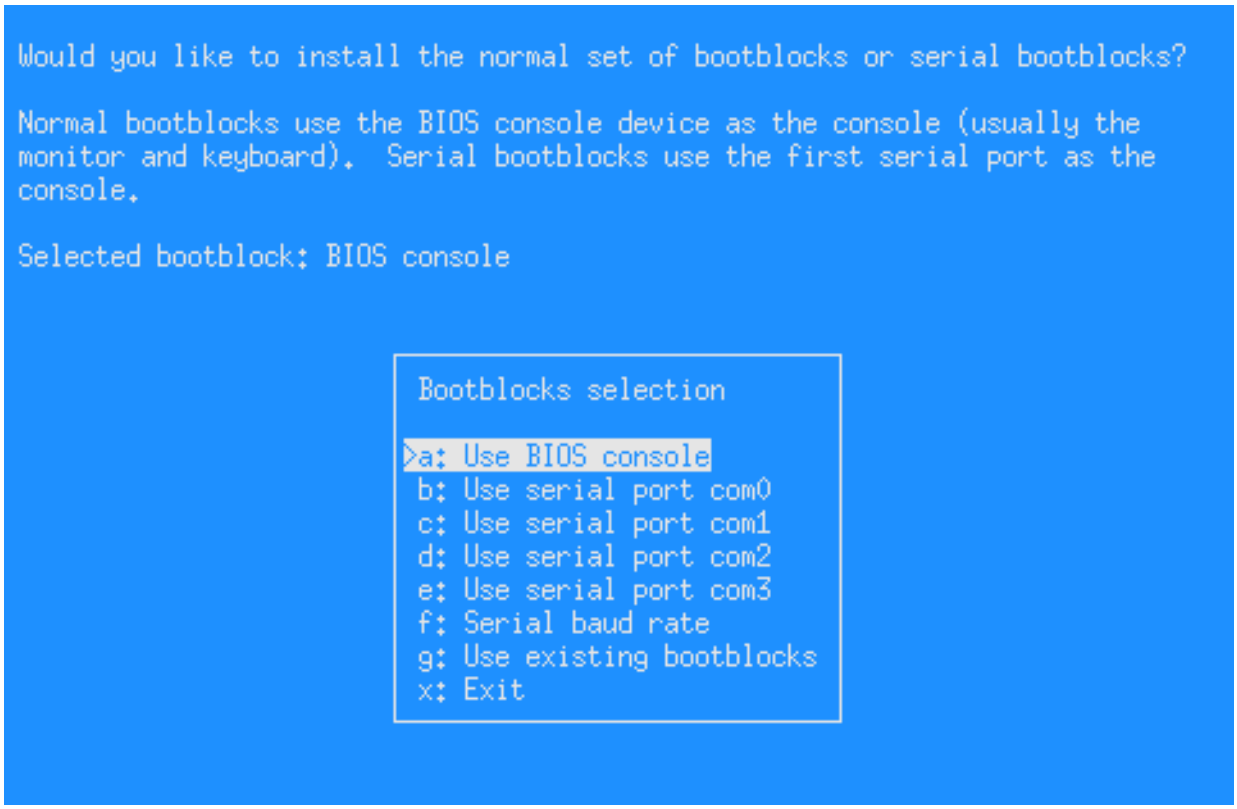
Figure 4-18. Creating the filesystems



After preparing the partitions and filesystems, the next question shown in Figure 4-19 is which bootblock to install. Usually you will choose the default to use the BIOS console, i.e. show boot messages on your PCs display.

If you run a farm of machines without monitor, it may be more convenient to opt for a serial console running on one of the serial ports. The menu also allows changing the serial port's baud rate from the default of 9600 baud, 8 data bits, no parity and one stopbit.

Figure 4-19. Selecting a bootblock

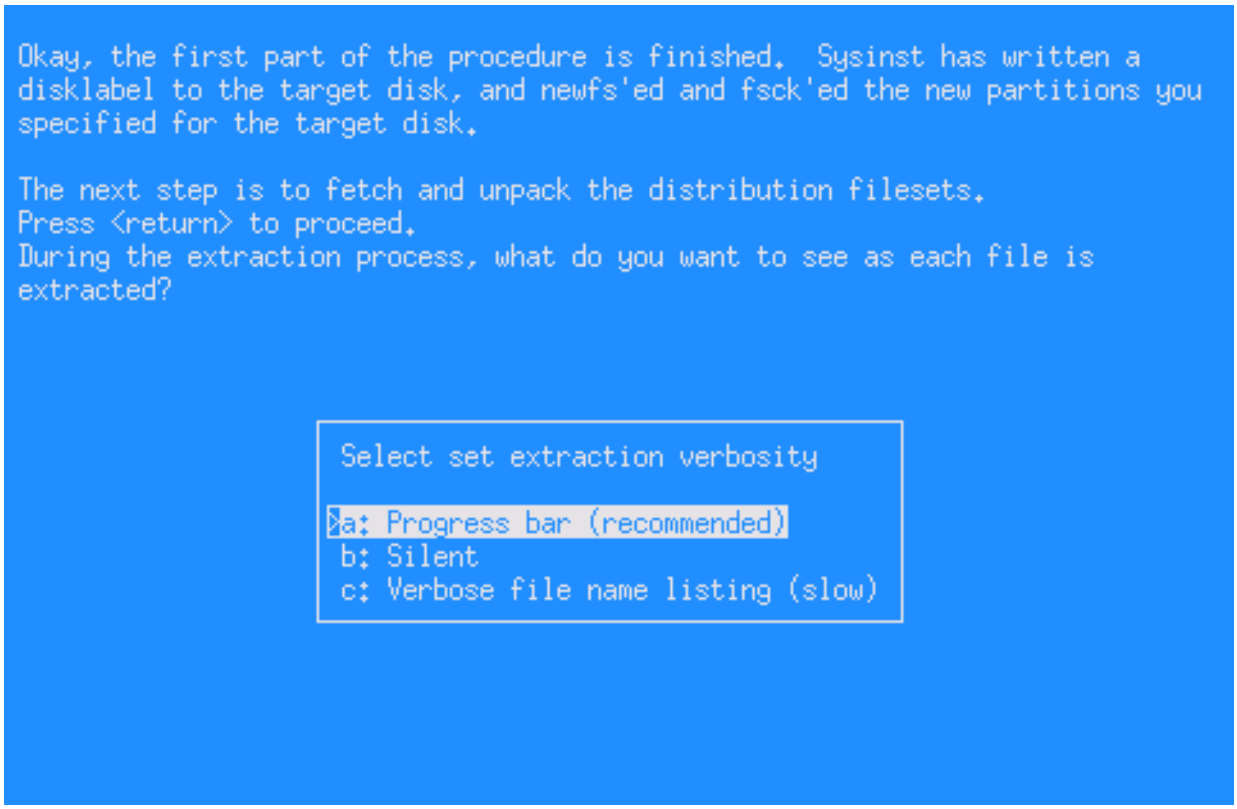


4.10 Choosing the installation media

Halftime - you have finished the first and most difficult part of the installation!

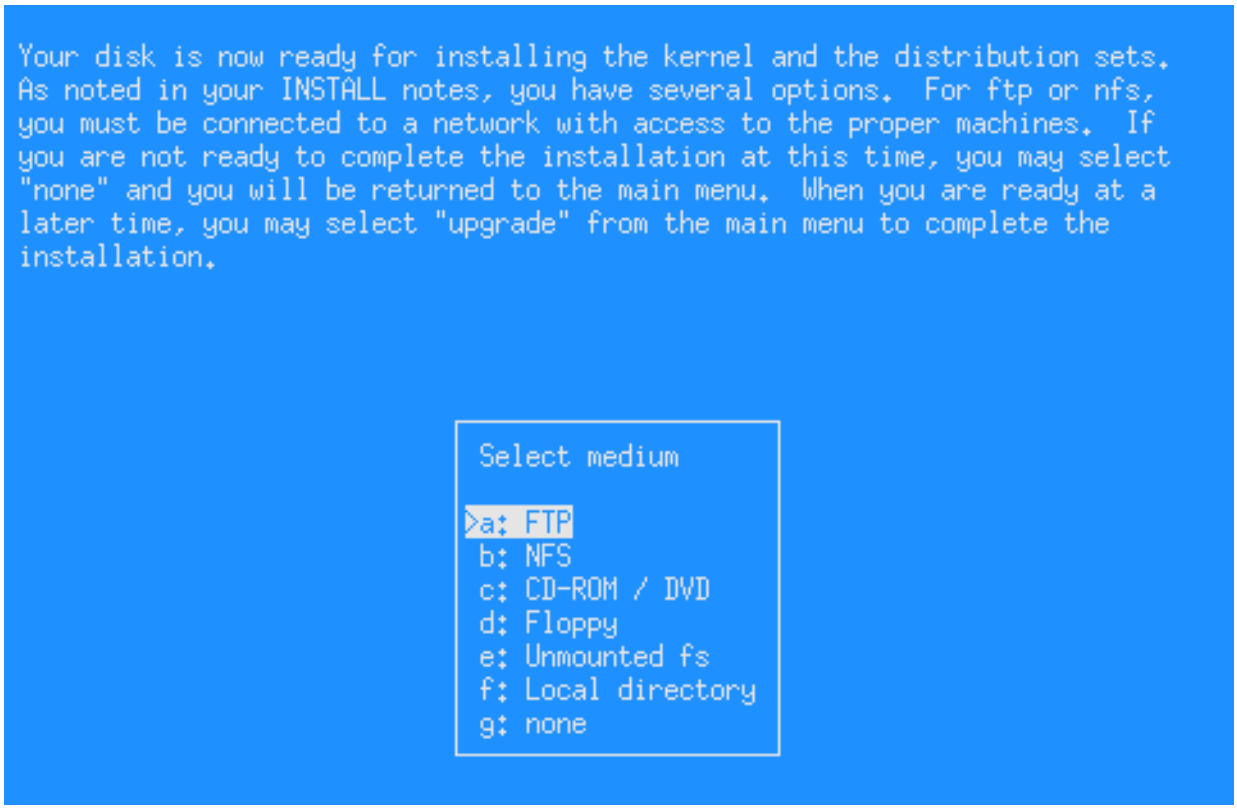
The second half of the installation process consists of populating the filesystems with the NetBSD operating system's files by extracting the "sets" that you have selected before (base, etc, comp, etc.). For unpacking the sets, sysinst asks what information you would like to see during that process, as shown in (Figure 4-20). You can choose to let *sysinst* either show a progress bar, be quiet, or show the name of each extracted file.

Figure 4-20. Choosing the verbosity of the extraction process



Now sysinst needs to find the NetBSD sets (the `.tgz` files) and you must supply this information. The menu offers several choices as shown in Figure 4-21.

Figure 4-21. Installation media



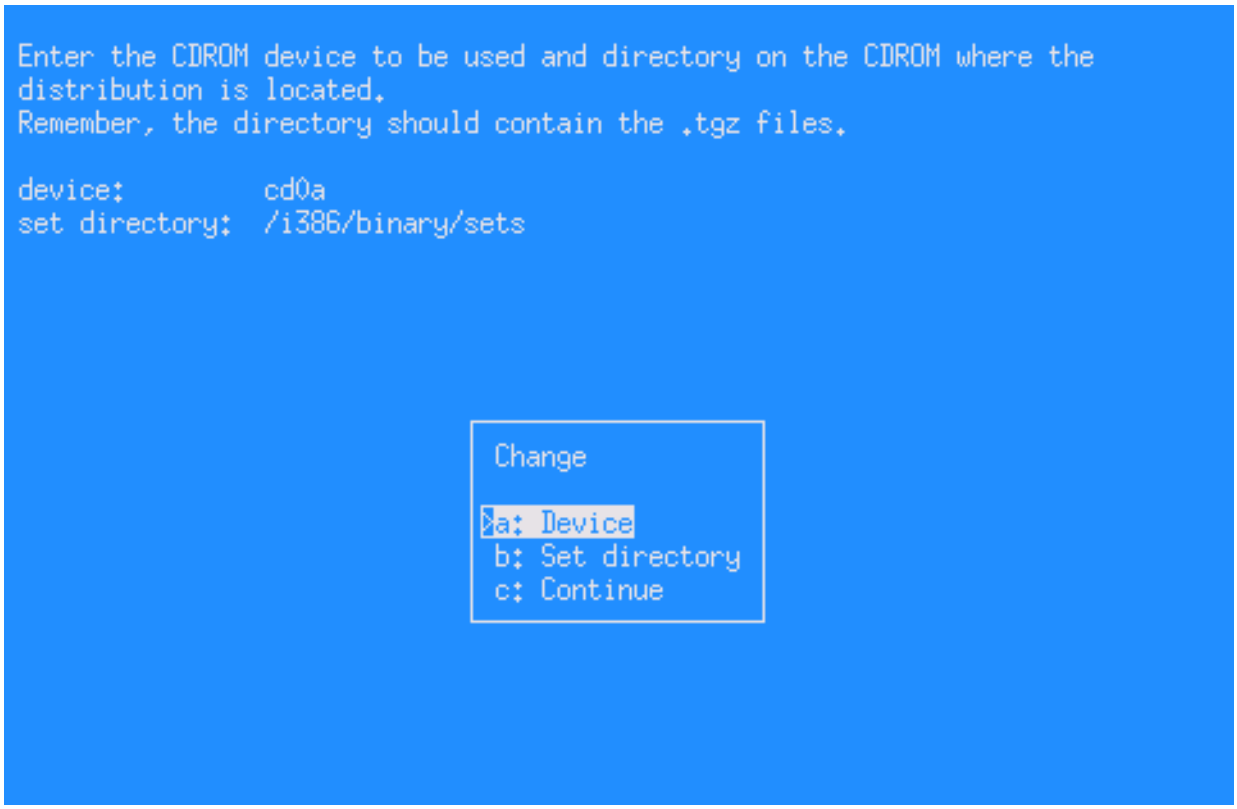
The options are explained in detail in the `INSTALL.*` document.

It is possible to install from an unmounted filesystem (provided that it is of a type recognised by the install kernel) so for example it is possible to copy all the sets to an existing MS-DOS partition and install from there.

4.11 Installing from CD-ROM and DVD

When selecting “CD-ROM / DVD”, sysinst asks the name of the CD-ROM or DVD device and the directory in which the set files are stored, see Figure 4-22. The device is usually `cd0` for the first CD-ROM or DVD drive, regardless if it is IDE or SCSI.

Figure 4-22. CD-ROM/DVD installation



Note: if you are using a non US keyboard you'll have to be careful when you type the "/" character. See Section 3.3.1.

The CD-ROM / DVD device name: if you don't know the name of the CD-ROM / DVD device, you can find it in the following way:

1. Press Ctrl-Z to pause sysinst and go to the shell prompt (that's a nice feature!)
2. Type the command:

   ```
   # cat /kern/msgbuf
   ```

   This will show the kernel startup messages, including the name of the CD-ROM device, for example *cd0*.
3. If the display scrolls too quickly, you can also use **more**:

   ```
   # more /kern/msgbuf
   ```
4. Go back to the installation program with the command:

   ```
   # fg
   ```

4.12 Installing via FTP

If you choose to install via FTP, sysinst will configure the system's network connection, download the selected set files to a temporary directory and extract them.

NetBSD 2.0 currently supports installation via ethernet, ethernet-over-USB and wireless LAN, installation via DSL (PPP over Ethernet) is not supported during installation and needs to be setup manually after installation.

The first step shown in Figure 4-23 consists of selecting which network card to configure. sysinst will determine a list of network interfaces available in your hardware, present them and ask which one it shall use.

Note: The exact names of your network interfaces depends on the hardware you use, example interfaces are "fxp0" for Intel i8255x based ethernet cards, "tlp0" for TULIP-based ethernet cards, "wi0" for Lucent WaveLAN and "ath0" for Atheros based wireless cards. This list is by no means complete, and NetBSD supports many more network devices.

To get a list of network interfaces available on your system (or rather, a list of all the network interfaces which NetBSD detected), interrupt the installation process by pressing "Ctrl+Z", then enter

```
# ifconfig -a
fxp0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:06:0d:c6:73:d5
        media: Ethernet autoselect (10baseT)
        status: active
        inet 0.0.0.0 netmask 0xffffff00 broadcast 0.0.0.0
        inet6 fe80::206:dff:fec6:73d5%fxp0 prefixlen 64 scopeid 0x1
lo0: flags=8009<UP,LOOPBACK,MULTICAST> mtu 33196
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
ppp1: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
strip0: flags=0 mtu 1100
strip1: flags=0 mtu 1100
```

for a list of all network interfaces (fxp0, lo0, ppp0, ...), and their current state (which is probably mostly unconfigured at this time). To get more information about all the devices found during system startup, including network devices, type

```
# more /kern/msgbuf
```

To only get information about a single device, for example "fxp0", run:

```
# grep ^fxp0 /kern/msgbuf
fxp0 at pci1 dev 8 function 0: Intel PRO/100 VE (MOB) Network Controller, rev 131
fxp0: interrupting at irq 11
fxp0: Ethernet address 00:06:0d:c6:73:d5
```

You can return to the NetBSD installation by typing

```
# fg
```

Figure 4-23. Which network interface to configure

Next, here is a chance to configure options for your network medium as shown in Figure 4-24, like duplex settings for ethernet, and various settings for wireless LAN cards.

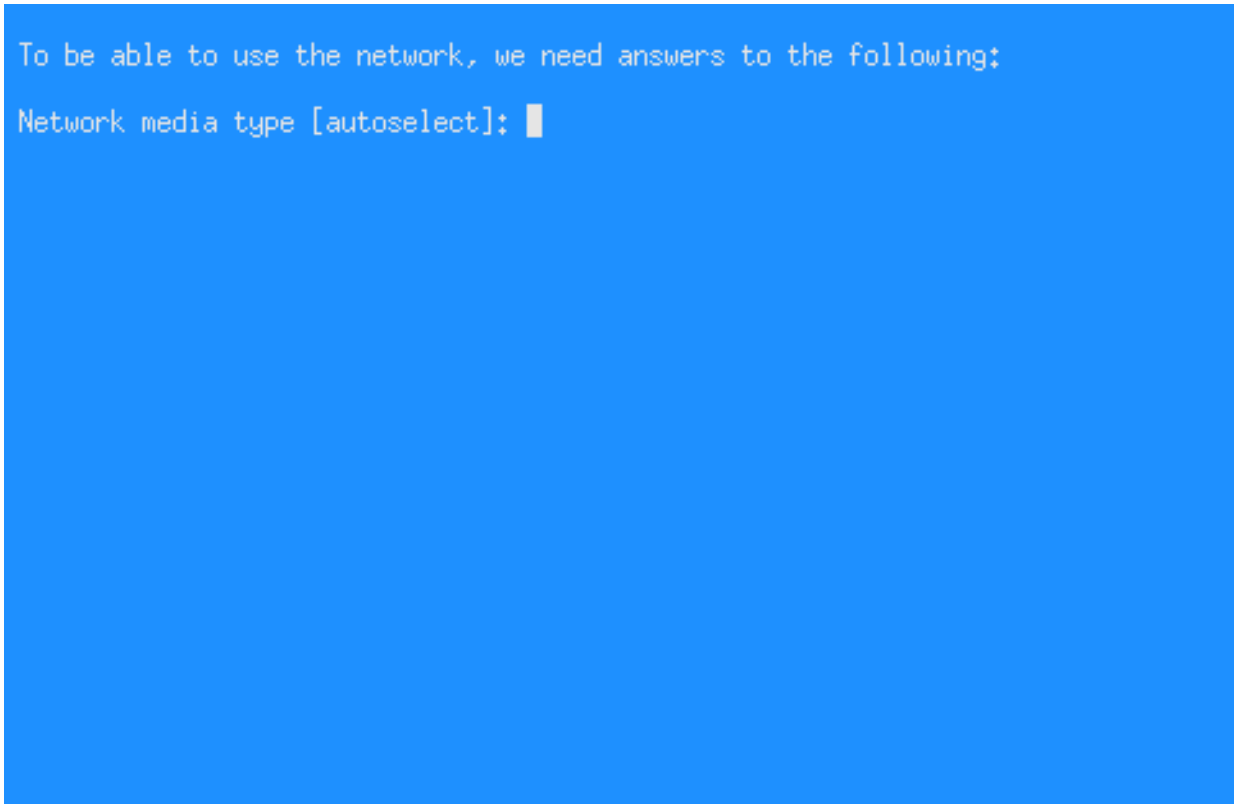
> **Note:** It is unlikely that you will need to enter anything other than the default here. If you experience problems like very slow transfers or timeouts, you may for example force different duplex settings for ethernet cards here. To get a list of supported media and media options for a given network device (say: “fxp0”), escape from sysinst by pressing “Ctrl+Z”, then enter:

```
# ifconfig -m fxp0
fxp0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:03:0d:c6:73:d5
        media: Ethernet autoselect (10baseT)
        status: active
        supported Ethernet media:
                media 10baseT
                media 10baseT mediaopt full-duplex
                media 100baseTX
                media 100baseTX mediaopt full-duplex
                media autoselect
        inet 0.0.0.0 netmask 0xffffff00 broadcast 0.0.0.0
        inet6 fe80::206:dff:fec6:73d5%fxp0 prefixlen 64 scopeid 0x1
```

> The various values given after “media” may be of interest here, including keywords like “autoselect” but also including any “mediaopt” settings.
>
> Return to the installation by typing

```
# fg
```

Figure 4-24. Choosing network media options



The next question will be if you want to perform DHCP autoconfiguration as shown in Figure 4-25. Answer “Yes” if you have a server for the Dynamic Host Configuration Protocol (DHCP) running somewhere on your network, and sysinst will fetch a number of defaults from it. Answer “No”, and you will have to enter all the values manually.

To explain things, we will assume you answered “No” and go into all the questions asked in detail.

Figure 4-25. Using DHCP for network configuration

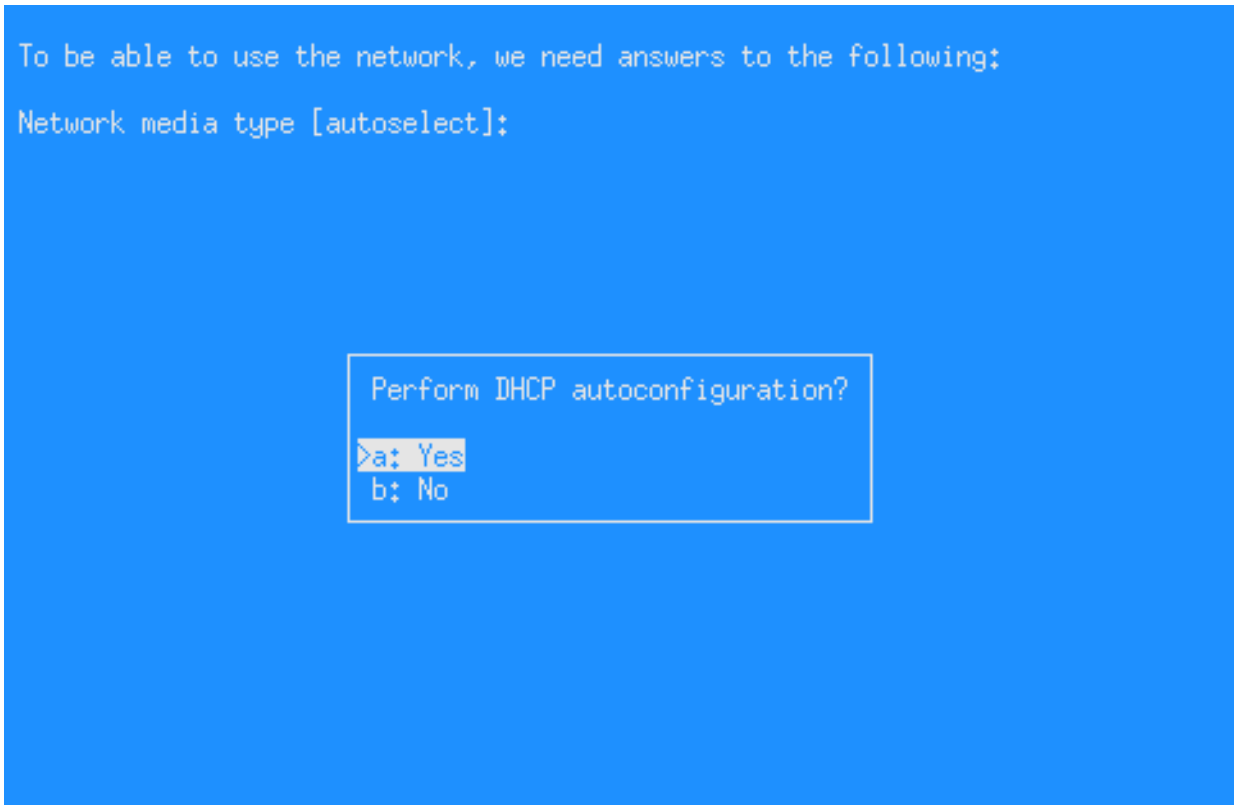


Figure 4-26 shows the questions asked for the network configuration. The values asked for are:

Your DNS Domain:

This is the name of the domain you're in WRT the Domain Name System (and which has *nothing* to do with any NIS/YP domain you may be in!).

Your host name:

The name by which other machines can usually address your computer. Not really used during installation.

Your IPv4 number:

Enter your numerical Internet Protocoll address in "dotted quad" notation here.

IPv4 Netmask:

The netmask for your network, either given as a hex value ("0xffffff00") or also in dotted-quad notation ("255.255.255.0").

IPv4 gateway:

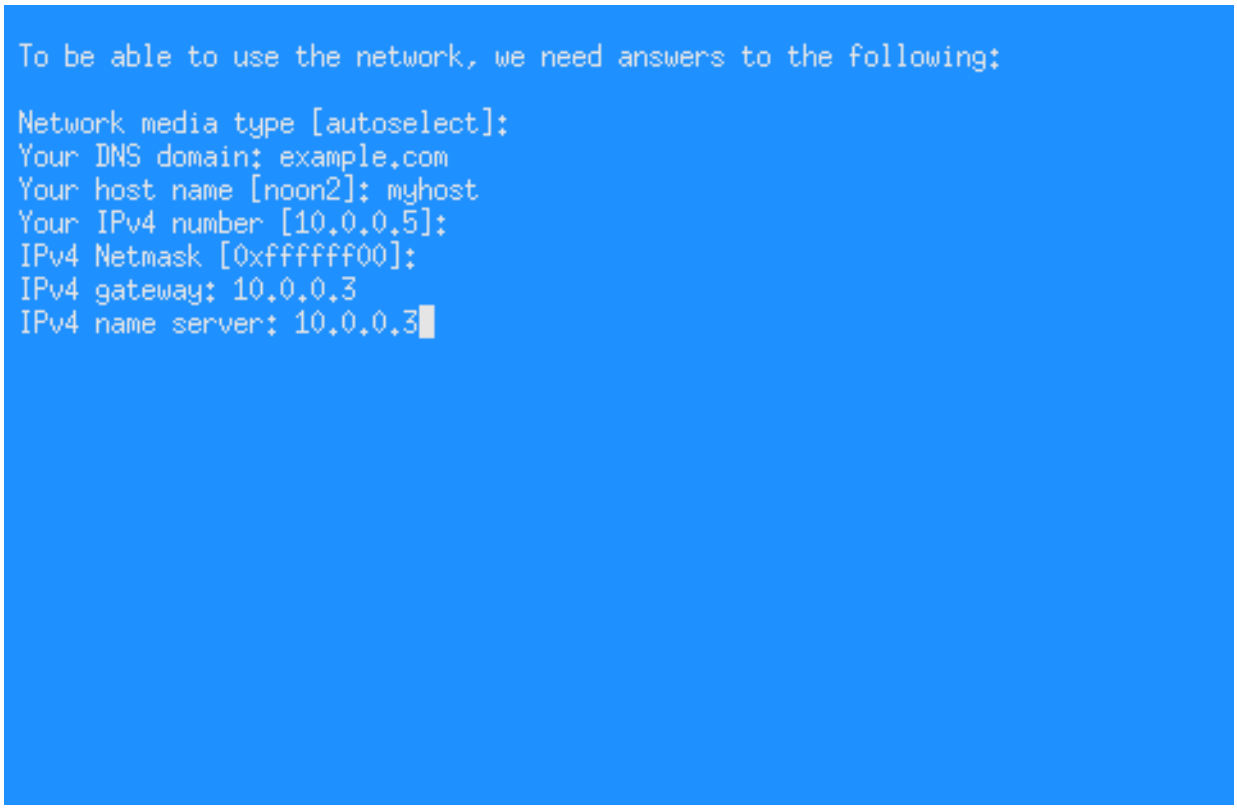
Your default router's IP address. Do not use a hostname here.

IPv4 name server:

> Your (first) DNS server's IP address. Again, don't use a hostname here to avoid some nasty problems.

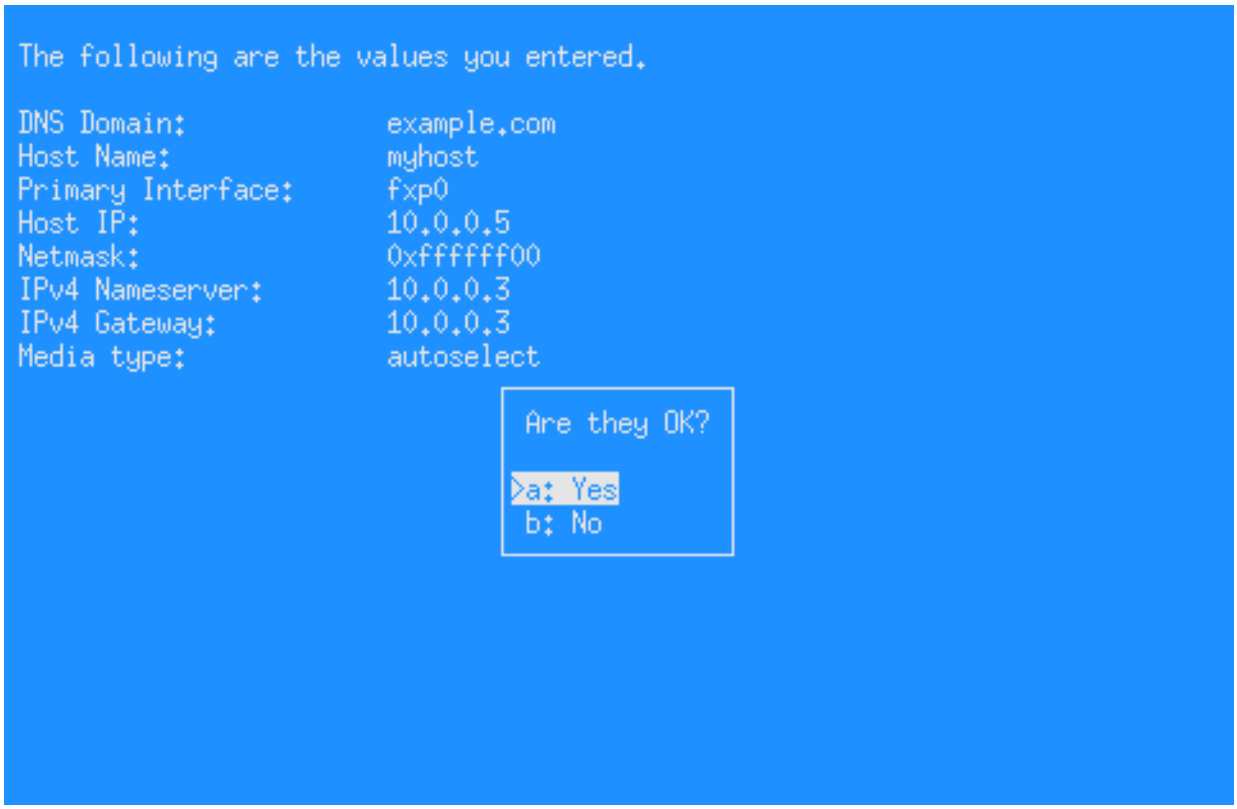
> **Note:** Please note that most of the network configuration parameters are *not* saved in the system you install. It can be argued that it should, but NetBSD was designed not to do so. Configuring network settings has to happen with other configuration works needed for NetBSD after installation, see Chapter 5, Chapter 6 and basically the whole rest of this guide. Yes, NetBSD allows you a lot of choices!

Figure 4-26. Entering and configuring network data



After answering all questions for network configuration, they will be printed again with a chance to go back and re-enter them, see Figure 4-27. When selecting "No". Choose "Yes" if you are satisfied with your settings to proceed with the installation.

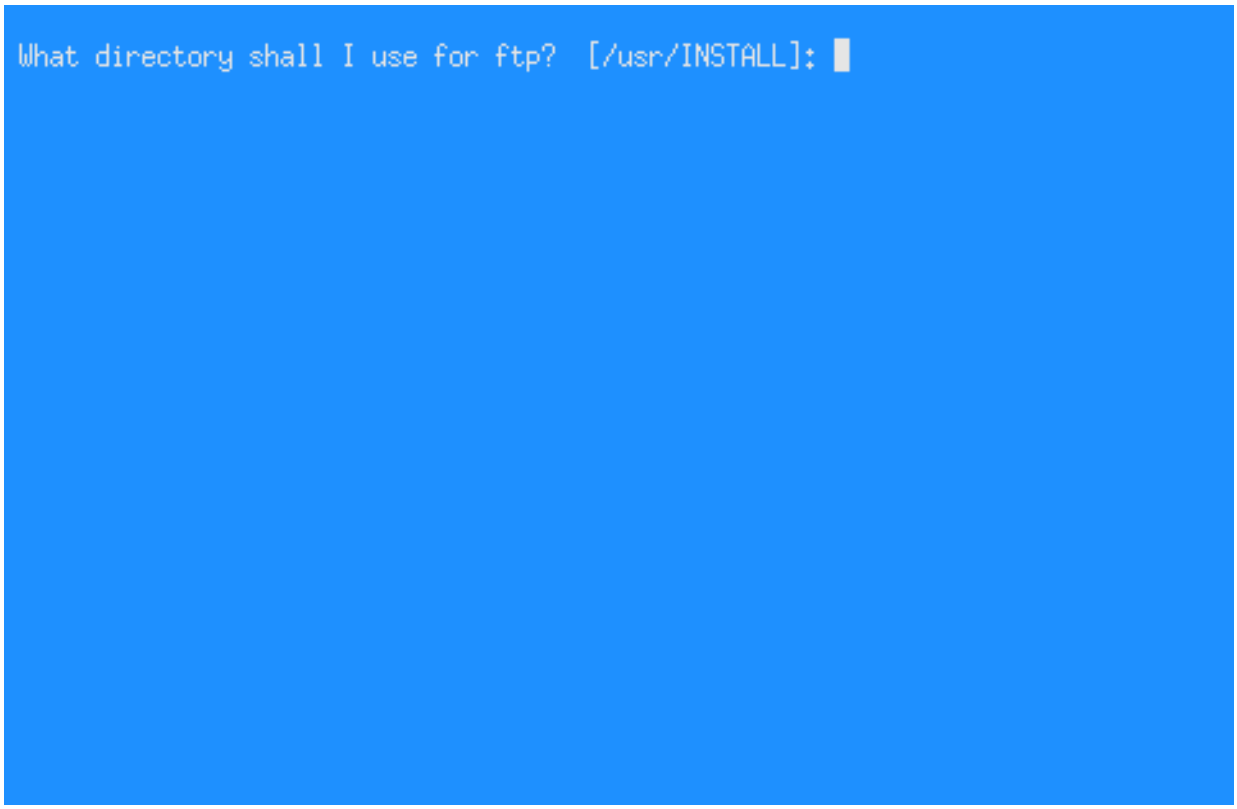
Figure 4-27. Confirming network parameters



sysinst will now run a few commands (not displayed in detail here) to configure the network: flushing the routing table, setting the default route, and testing if the network connection is operational.

The next question asked is about the place where files should be stored during download, see Figure 4-28. The default directory is usually ok, it is located on the NetBSD file system.

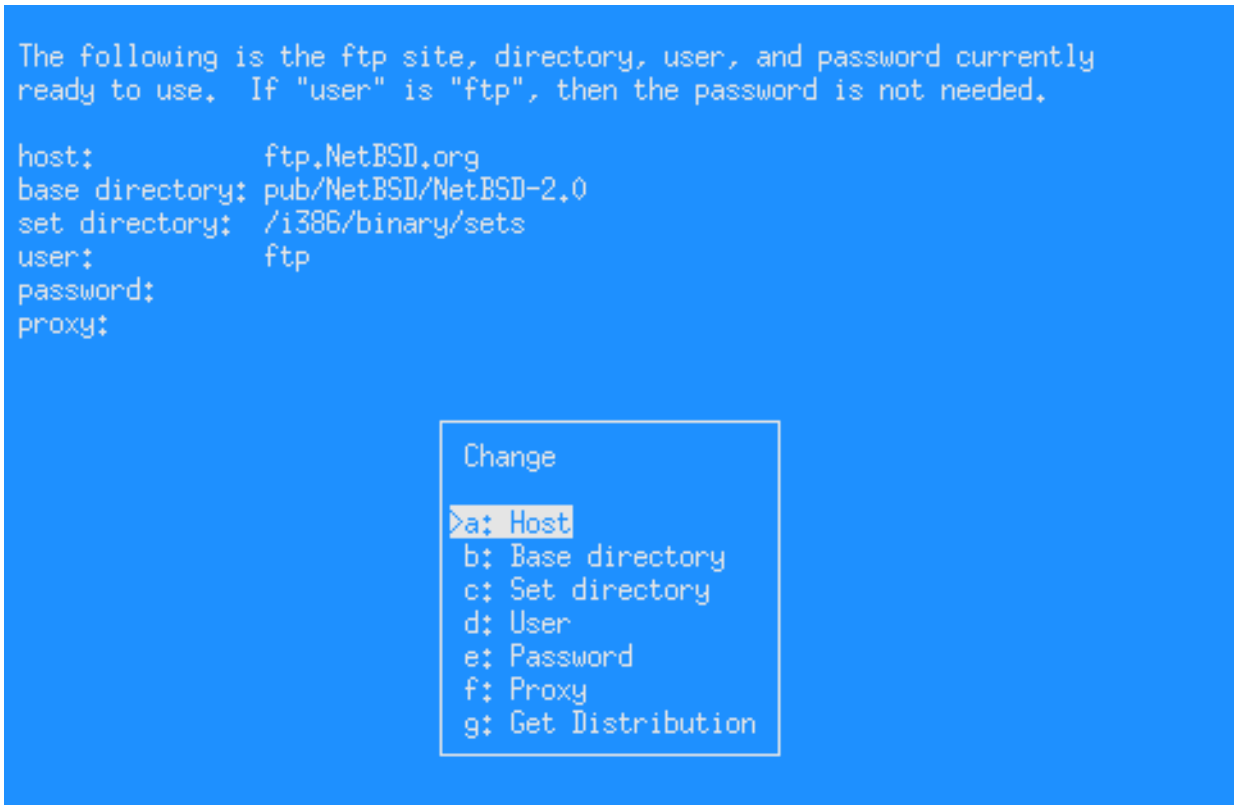
Figure 4-28. Which directory to use for downloading



After the installer knows where to download files, what network connection to use for downloading them, and on which partition and file system to extract them, the last data missing is the place where to download the install sets from, which is what the next dialogue shown in Figure 4-29 allows to change. You can adjust the host where the sets are fetched from, the base directory of the NetBSD release you want to install and the set directory (relative to the base directory), which usually contains the architecture you want to install. You can also change the FTP user's login name ("user") and password here, if needed. If you want to use a FTP proxy for downloading, enter its URL here as well.

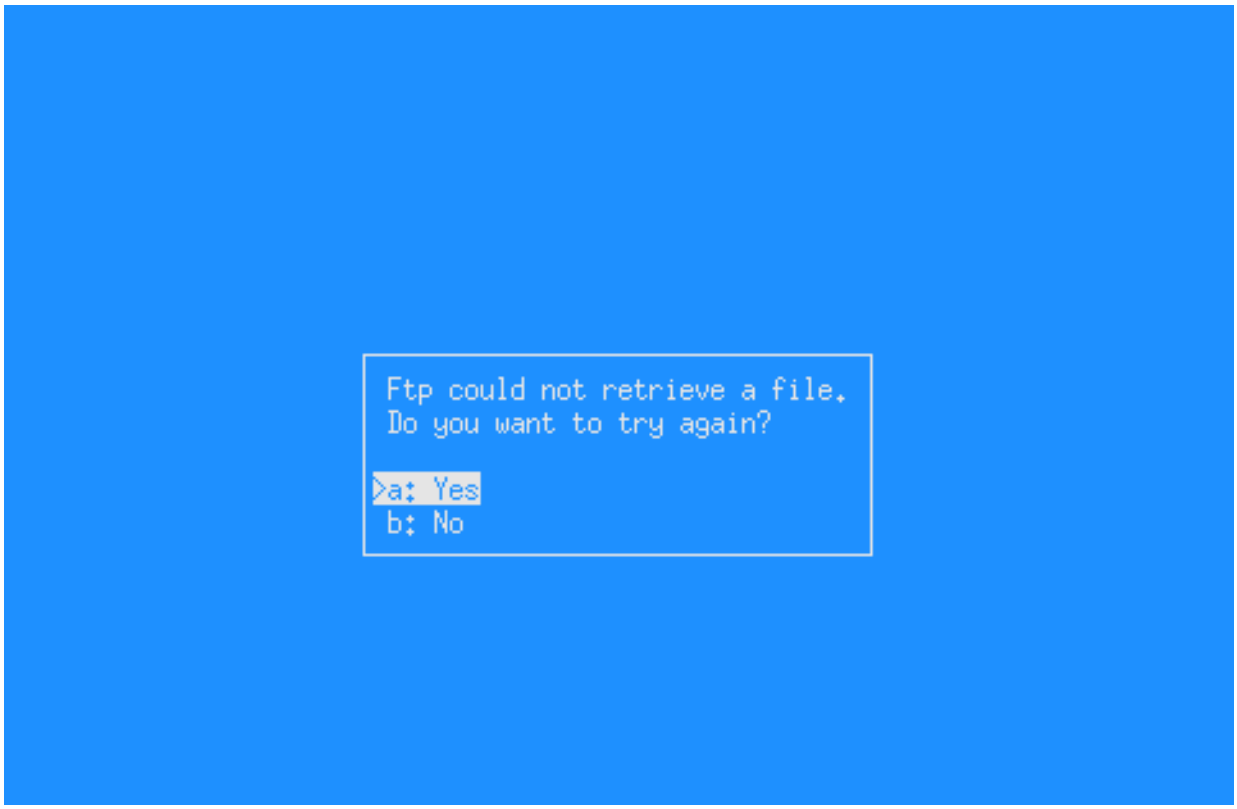
When you're satisfied with your settings (the defaults work most of the time), choose "Get Distribution" to continue.

Figure 4-29. Defining the set's source place



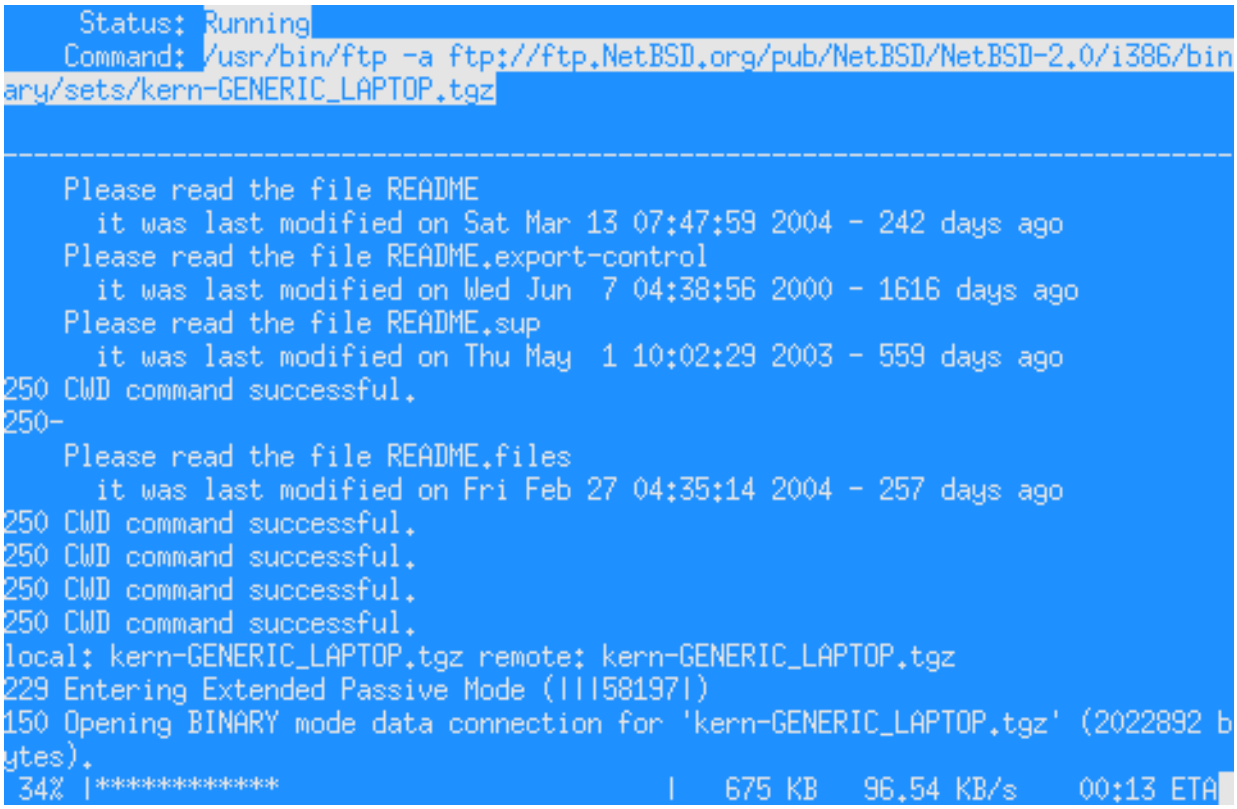
If you made an error and sysinst cannot fine a file, it will tell you so as shown in Figure 4-30 and you will have a chance to go back and adjust the place where it will download the sets from.

Figure 4-30. An error happened during download



If the data given for the download was correct, the set files will be downloaded and stored in the directory given before, as shown in Figure 4-31.

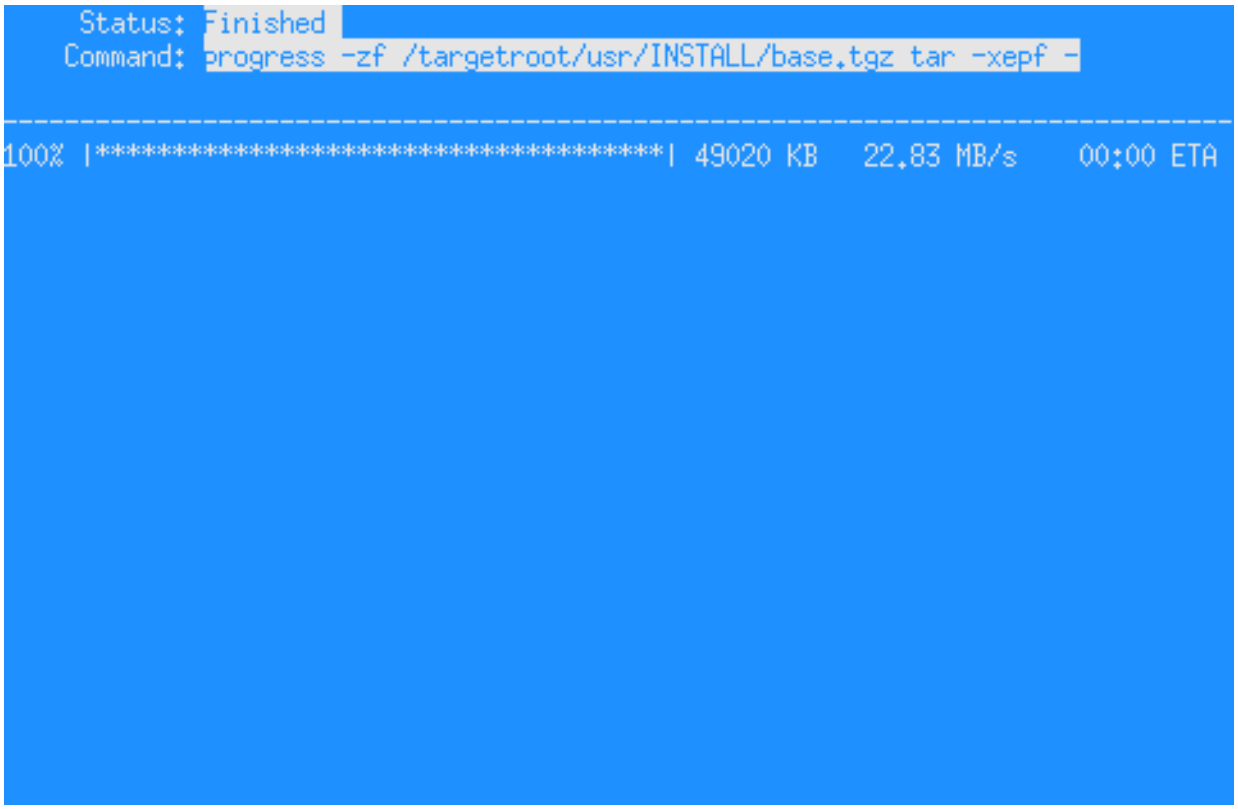
Figure 4-31. Downloading installation sets



4.13 Extracting sets

After all sets are available at this step - either from a CD-ROM/DVD or in a directory where the set files were downloaded into, they will be extracted into the new NetBSD file system next. Figure 4-32 shows an example extraction progress with a progress bar, as selected before.

Figure 4-32. Extracting set files



After extracting all selected sets, sysinst will create device nodes in the `/dev` directory and then displays a message saying that everything went well.

If the set files were downloaded from the net, you can delete them after extraction, or keep them around. sysinst will ask and do as you please - select “No” to keep the files, and “Yes” to delete them, as shown in Figure 4-33.

Figure 4-33. Deleting sets after extraction?

Another message (see Figure 4-34) will let you know that the set extraction is now completed, and that you will have an opportunity to configure some essential things before finishing the NetBSD installation.

Figure 4-34. Set extraction completed

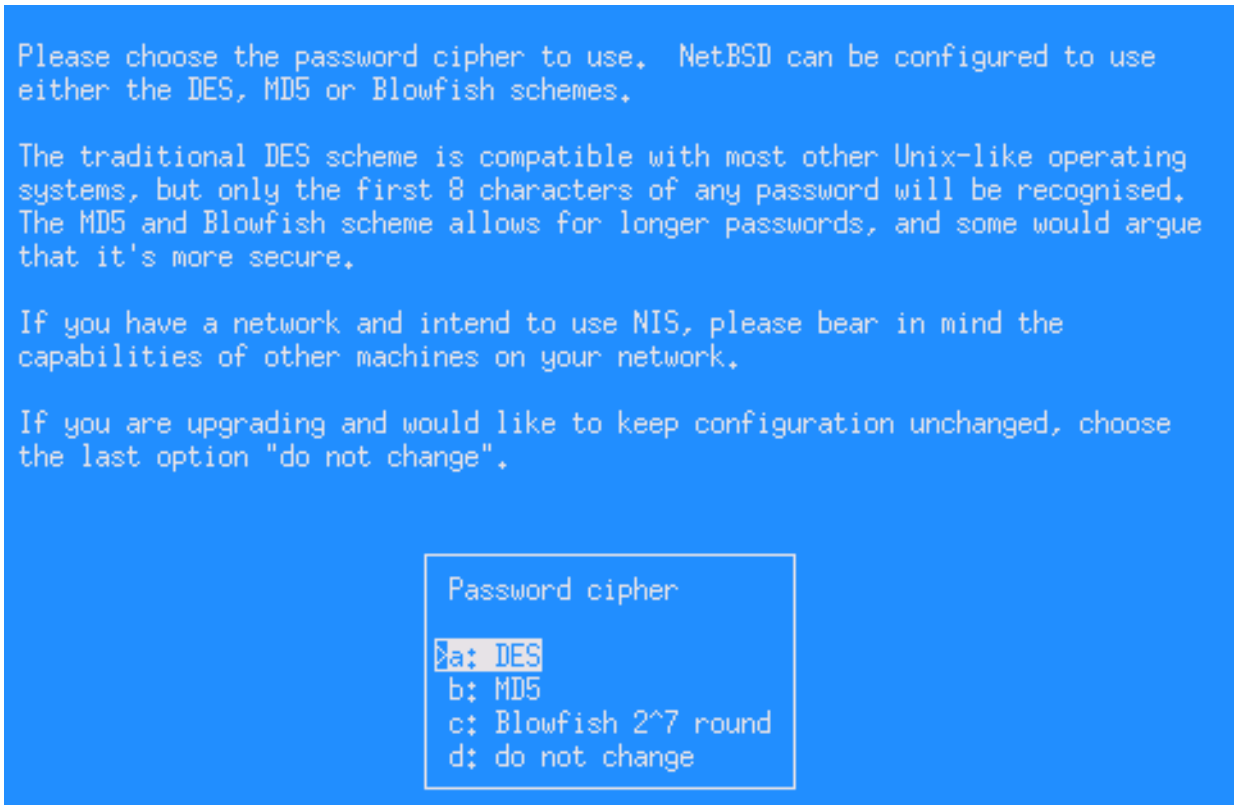
4.14 System configuration

The first thing you can adjust is the timezone in which the system resides. It is Universal Time Coordinated (UTC) by default, and you can use the two-level menu of continents/countries and cities shown in Figure 4-35 to determine your timezone. If you are satisfied with your choice, press the Return key to update the display of your local time. Press “x” followed by Return to exit timezone selection.

Figure 4-35. Selecting your system's time zone

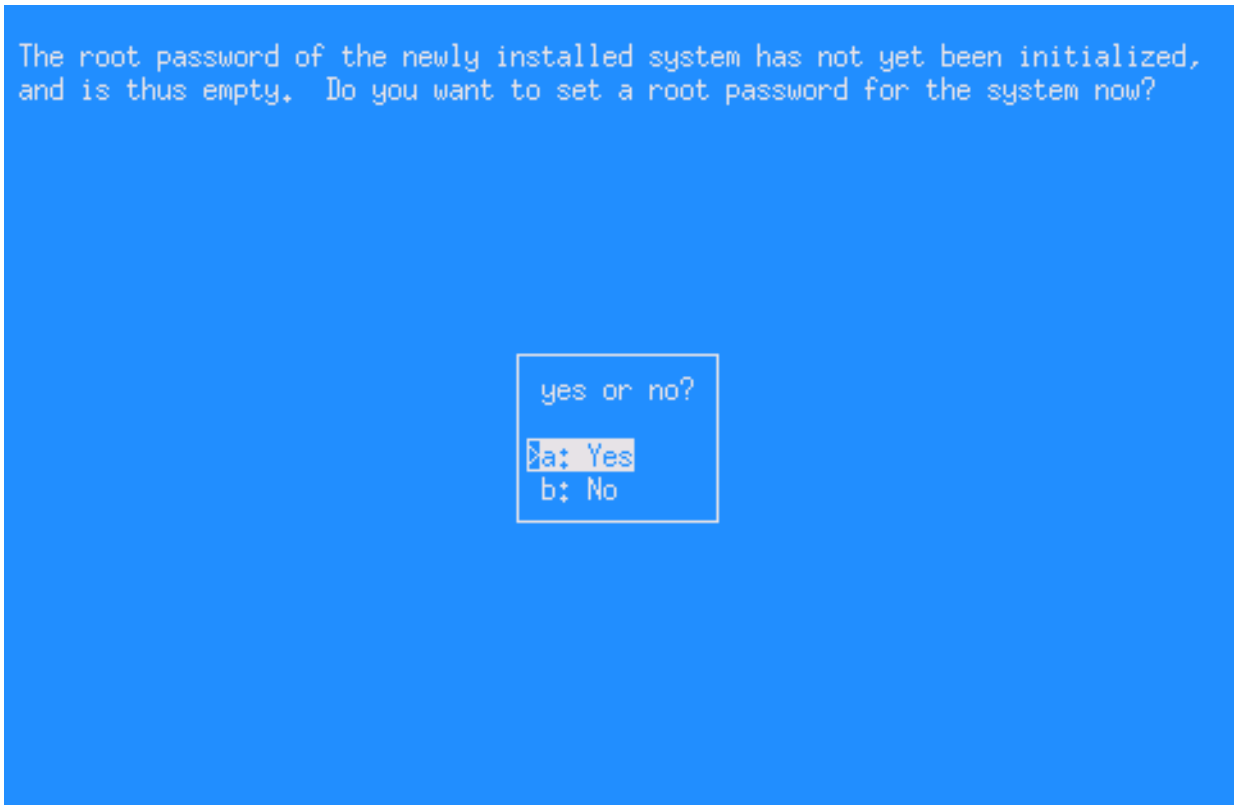
The next thing that is asked is which algorithm shall be used to encrypt the password file (Figure 4-36). While DES is the standard algorithm used on most Unix systems, MD5 and Blowfish allow longer passwords than DES, which only uses the first eight characters of the password that is entered. DES is still useful for interoperability with other operating systems (e.g. for NIS).

Figure 4-36. Selecting a password encryption scheme



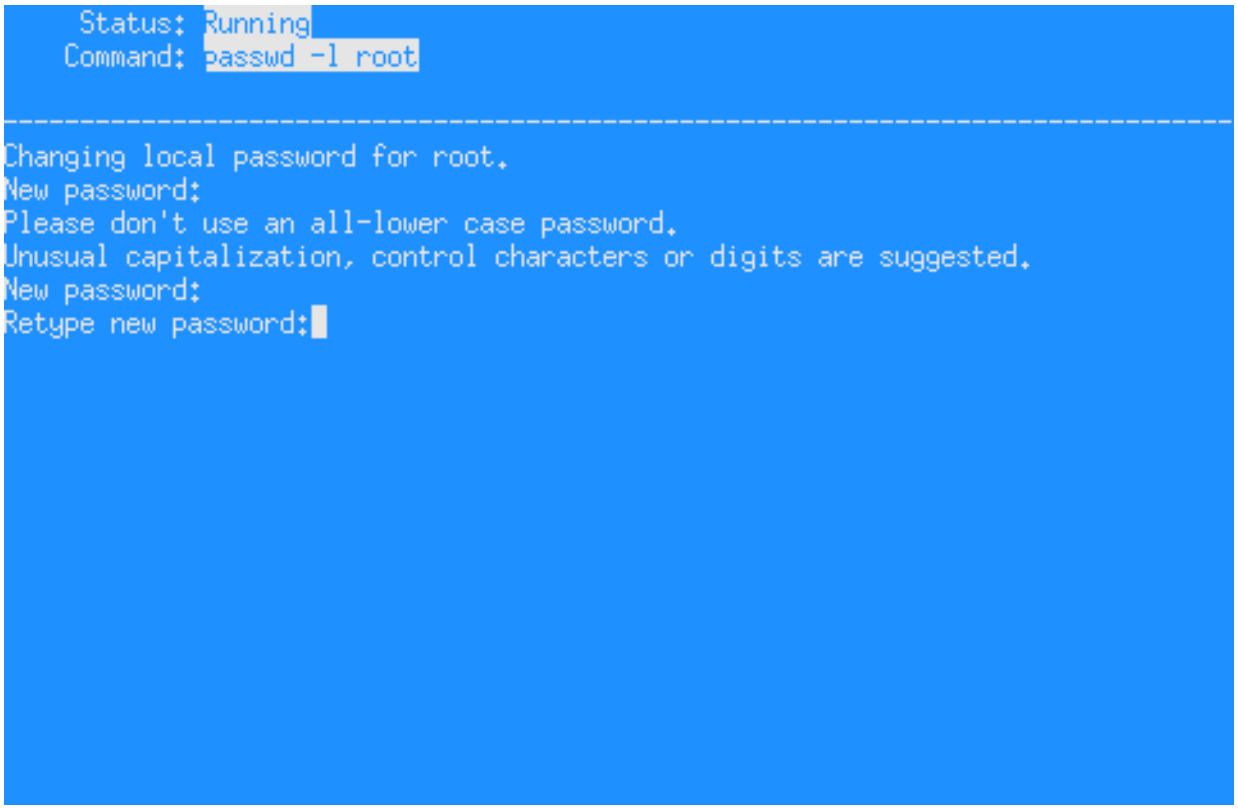
After choosing the password cipher you are asked if you want to set the root password, see Figure 4-37. NetBSD doesn't start any services when booting up after installation, yet it is still recommended to set a root password right here for security reasons.

Figure 4-37. Setting the root password now or later?



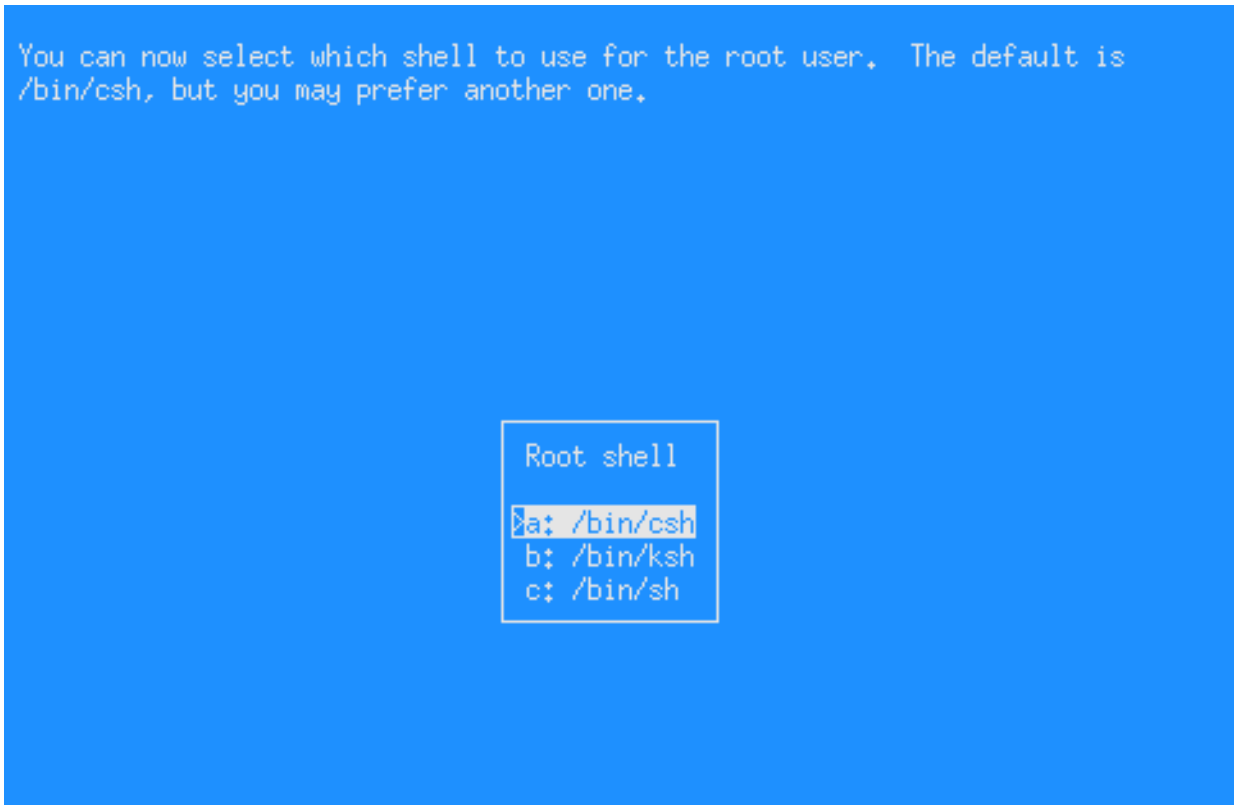
When you agree to set a root password, sysinst will run the passwd(1) utility for you, and you should enter your new root password (twice). Please note that the password is not echoed, and if you enter a very simple password, the system will warn you about this, see Figure 4-38. If you insist on entering the same simple password again, NetBSD will give in and let you have your will, providing you with all the rope you need to hang yourself (metaphorically speaking, of course).

Figure 4-38. Setting the root password



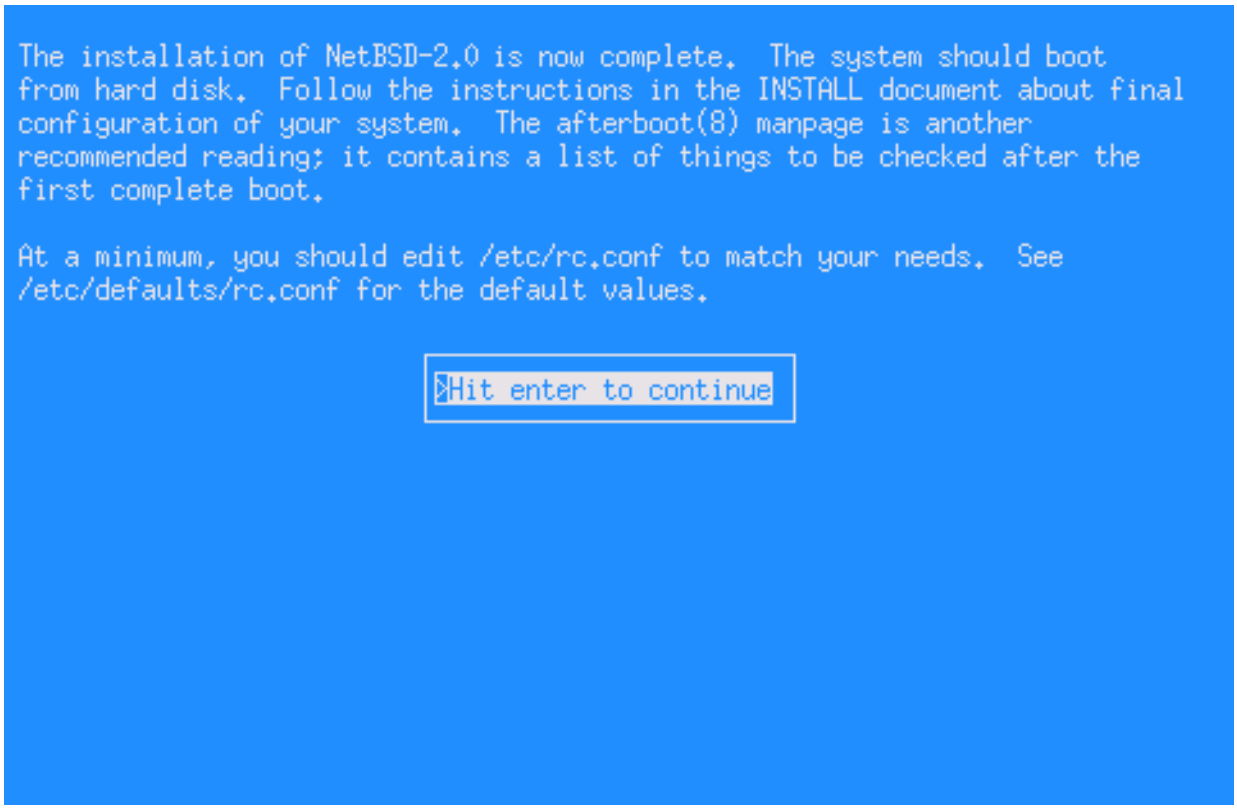
Next item to tweak is to choose which command line interpreter - also known as “shell” in Unix - to use for the root account. As printed in Figure 4-39, the default is the C shell (csh), other choices are Korn shell (ksh) and the classic Bourne shell (sh). While BSD systems have traditionally shipped with “csh” as login shell for the system manager, modern systems tend to come with a Bourne shell (or variantes thereof, like ksh or bash), and it may be useful to chose this if you have experience on such systems. Else, the default will be fine, and it can always be changed later (see Section 5.6).

Figure 4-39. Choosing a shell for the root account



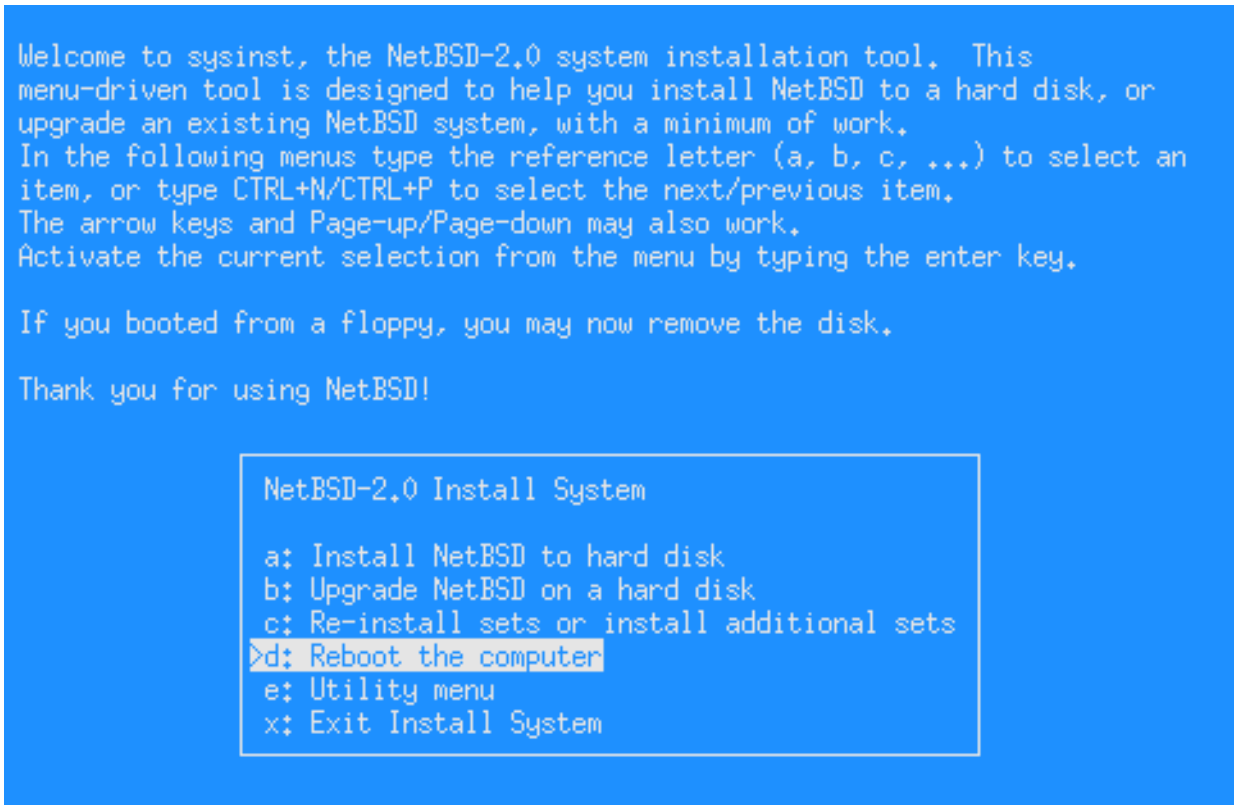
At this point the installation is finished, see Figure 4-40.

Figure 4-40. Congratulations



After passing the dialog that confirms the installation, sysinst will return to the main sysinst menu. Remove any installation media (floppy, CDROM) from your drives, and choose “Reboot the computer” to boot your NetBSD installation as shown in Figure 4-41.

Figure 4-41. Reboot the system to finish installation



Chapter 5
The first boot

After installing and rebooting, the computer will boot from the hard disk: if everything went well you'll be looking at the login prompt within a few seconds (or minutes, depending on your hardware). The system is not yet configured but don't worry: configuration is very easy and the approach offered by NetBSD gives you a lot of flexibility. You'll see how to quickly configure everything and, in the meantime, you'll learn how the system works; in the future, in case of trouble you'll know where to look.

5.1 If something went wrong

If the system doesn't boot it could be that the boot manager was not installed correctly or that there is a problem with the MBR (Master Boot Record). Reboot the machine from the boot floppy and when you see the prompt:

```
booting fd0a:netbsd - starting in ...
```

press the space bar during the 5 second countdown; the boot stops and a prompt is displayed. You can have a basic help with the "?" key or with the "help" command.

```
type "?" or "help" for help.
> ?
commands are:
boot [xdNx:][filename] [-adrs]
     (ex. "sd0a:netbsd.old -s")
ls [path]
dev xd[N[x]]:
help|?
quit
> boot wd0a:netbsd
```

The system should now boot from the hard disk instead of the floppy. If NetBSD boots correctly from the hard disk, there is probably a Master Boot Record problem: you can install the boot manager or modify its configuration with the **fdisk -B** command. See Section 20.3 for a detailed description.

5.2 Login

For the first login you will use the `root` superuser, which is the only user defined at the end of the installation. At the password prompt type the password for root that you have defined during the installation. If you haven't defined a password, just press Enter.

```
NetBSD/i386 (Amnesiac) (ttyE0)
login: root
password
```

```
...
We recommend creating a non-root account and using su(1) for root access.
#
```

5.3 Changing the keyboard layout

The keyboard still has the US layout; if you have a different keyboard it's better to change its layout now, before starting to configure the system. For example, to use the italian keyboard, give the following command:

```
# wsconsctl -k -w encoding=it
encoding -> it
```

See Section 30.1.3 for a list of keymaps available as well as how to make these settings permanent.

5.4 The man command

If you have never used a Unix(-like) operating system before, your best friend is now the **man** command, which displays a manual page: the NetBSD manual pages are amongst the best and most detailed you can find, although they are very technical.

man *name* shows the man page of the "*name*" command and **man -k *name*** shows a list of man pages dealing with "*name*" (you can also use the **apropos** command).

To learn the basics of the **man** command, type:

```
# man man
```

The manual is divided into nine sections, containing not only basic information on commands but also the descriptions of some NetBSD features and structures. For example, take a look at the hier(7) man page, which describes in detail the layout of the filesystem used by NetBSD.

```
# man hier
```

Other similar pages are release(7) and packages(7). Each section of the manual has an intro(8) man page describing its content. For example, try:

```
# man 8 intro
```

Manual pages are divided in several sections, depending on what they document:

1. general commands (tools and utilities), see intro(1)
2. system calls and error numbers, see intro(2)
3. C libraries, see intro(3)
4. special files and hardware support, see intro(4)
5. file formats, see intro(5)
6. games, see intro(6)

7. miscellaneous information pages, see intro(7)
8. system maintenance and operation commands, see intro(8)
9. kernel internals, see intro(9)

A subject may appear in more than one section of the manual; to view a specific page, supply the section number as an argument to the man command. For example, *time* appears in section 1 (the time user command), in section 3 (the time function of the C library) and in section 9 (the time system variable). To see the man page for the time C function, write:

```
# man 3 time
```

To see all the available pages:

```
# man -w time
# man -a time
```

5.5 Changing the `root` password

If you haven't defined a password for `root` during installation, you should use the **passwd** command to do so now.

```
# passwd
Changing local password for root.
New password:
Retype new password:
```

Passwords are not displayed on the screen while you type. Later we will see how to add other accounts on the system.

5.6 Changing the shell

The default shell for `root` can be chosen during system installation time, and it usually is **csh**; if this doesn't mean anything to you, you should begin studying csh with csh(1): it's a good interactive shell although it lacks history editing (have a look at *tcsh*, bash or even the NetBSD `/bin/sh` for this). If you want to change your shell, use the chsh(1) command. The shells available on NetBSD after installation are:

- csh
- sh
- ksh

The new shell will come into effect the next time you login. In the meantime, you can issue the following command if you use **csh**:

```
# set filec
```

which enables filename completion on the command line (with the ESC key; use Ctrl+D for a list of possible completions.)

You can also install other shells on the system, if you want to: tcsh, bash, zsh and other shells are available in the package collection (which we shall examine later).

If you're familiar with the shell you chose, this is a good time to create the shell's initialization files (`.chsrc`, `.login`, ...)

5.7 System time

NetBSD, like all Unix systems, uses a system clock based on Greenwich time (GMT) and this is what you should set your system clock to. If you want to keep the system clock set to the local time (because, for example, you have a dual boot system with Windows installed), you must notify NetBSD, modifying the `kern.rtc_offset` sysctl-variable:

```
# echo kern.rtc_offset=-60 >>/etc/sysctl.conf
# sh /etc/rc.d/sysctl restart
```

The value supplied (-60) is the number of minutes west of GMT.

To display the current setting of the `kern.rtc_offset` variable:

```
# sysctl kern.rtc_offset
kern.rtc_offset = -60
```

Now the kernel knows how to convert the time of the PC clock in the GMT system time but you must still configure the system for your local time zone (which you will find in the `/usr/share/zoneinfo` directory). If you have already done this during the installation you can skip this step (although it is better to check that the setting is correct.) For example, for Italy:

```
# rm -f /etc/localtime
# ln -s /usr/share/zoneinfo/Europe/Rome /etc/localtime
```

Once everything is set up correctly, you can change the time with the following command:

```
# date [[[[[cc]yy]mm]dd]hh]mm
```

5.8 Basic configuration in `/etc/rc.conf`

NetBSD uses the `/etc/rc.conf` for system configuration at startup: this file determines what will be executed when the system boots. Understanding this file is very important.

The `/etc/defaults/rc.conf` file contains the default values for a lot of settings, and to override a default value, the new value must be put into `/etc/rc.conf`: the definitions there override the one in `/etc/defaults/rc.conf` (which should stay unchanged).

Understanding the `/etc/rc.conf` file is very important. The rc.conf(5) manual page contains a detailed description of all the options.

```
# man rc.conf
```

The first modifications are:

- Set "`rc_configured=yes`" (this modification might already have been done by the installation software.)
- Set "`dhclient=yes`" to configure your system's network using DHCP.
- Set "`sshd=yes`" to start the Secure Shell server daemon upon system start.
- Define a *hostname* for your machine (use a fully qualified hostname, i.e. one including domain). If you have a standalone machine you can use any name (for example, woody.toys.net). If your machine is connected to a network, you should supply the correct network name.

> **Note:** Make sure that the hostname is resolvable, either using DNS or `/etc/hosts`; some programs do not work with an unresolvable hostname.

5.9 Enabling FFS soft-dependencies

This might be a good point to enable soft-dependencies for FFS filesystems. By default NetBSD writes FFS meta-data synchronously, which means that meta-data is written to disk immediately. Soft-dependencies is a mechanism which does not write meta-data immediately, but it is written in an ordered fashion, which keeps the filesystem consistent. This provides performance comparable to asynchronous mounts, without the same risks. Soft-dependencies can be enabled by adding `softdep` to the filesystem options in `/etc/fstab`. Let's look at an example of `/etc/fstab`:

```
/dev/wd0a / ffs rw 1 1
/dev/wd0b none swap sw 0 0
/dev/wd0e /var ffs rw 1 2
/dev/wd0f /tmp ffs rw 1 2
/dev/wd0g /usr ffs rw 1 2
```

Suppose we want to enable soft-dependencies for all filesystems, except for the `/` partition. We would change it to (changes are emphasized):

```
/dev/wd0a / ffs rw 1 1
/dev/wd0b none swap sw 0 0
/dev/wd0e /var ffs rw,softdep 1 2
/dev/wd0f /tmp ffs rw,softdep 1 2
/dev/wd0g /usr ffs rw,softdep 1 2
```

These are some references for more information about filesystems and soft-dependencies:

- The wikipedia filesystem page: http://en.wikipedia.org/wiki/File_system
- The Filesystems HOWTO: http://www.tldp.org/HOWTO/Filesystems-HOWTO.html
- Marshall Kirk McKusick's page about soft-dependencies: http://www.mckusick.com/softdep/

5.10 Rebooting the system

In this first session you have:

- Configured the keyboard
- Changed the `root` password (optional)
- Changed `root`'s shell (optional)
- Changed the system time and the RTC offset
- Defined the local time
- Configured `/etc/rc.conf`

Now it's time to reboot the system, with the following command:

```
# reboot
```

Chapter 6
The second boot

During the first boot you have set up a basic system configuration. This chapter describes some common commands and operations.

6.1 dmesg

At system startup the kernel displays a long sequence of messages on the screen: these messages give information about the kernel status (for example, available memory) and the peripherals that have been detected on the system. This information is very important for diagnosing hardware or configuration problems, and for determining the name of the devices for the peripherals (for example you can check if your network card has been detected as `ne0` or `ne1`). Usually these messages scroll on the screen too fast to be useful, but you can use the dmesg(8) command to view them again.

```
# dmesg | more
```

If something on your system doesn't appear to work correctly and you ask for help on one of the NetBSD mailing lists, always remember to include the relevant dmesg output in your post: it will help other people diagnose your problem. Note that during the boot process NetBSD also writes a copy of the dmesg output to `/var/run/dmesg.out`. This feature is useful because the system will scroll "old" messages out of the dmesg buffer over time.

6.2 Mounting the CD-ROM

New users are often surprised by the fact that although the installation program recognized and mounted their CD-ROM perfectly, the installed system seems to have "forgotten" how to use the CD-ROM. There is no special magic for using a CD-ROM: you can mount it as any other file system, all you need to know is the device name and some options to the mount(8) command. You can find the device name with the aforementioned dmesg(8) command. For example, if dmesg(8) displays:

```
# dmesg | grep ^cd
cd0 at atapibus0 drive 1: <ASUS CD-S400/A, , V2.1H> type 5 cdrom removable
```

the device name is `cd0`, and you can mount the CD-ROM with the following commands:

```
# mkdir /cdrom
# mount -t cd9660 -o ro /dev/cd0a /cdrom
```

To make things easier, you can add a line to the `/etc/fstab` file:

```
/dev/cd0a /cdrom cd9660 ro,noauto 0 0
```

Without the need to reboot, you can now mount the cdrom with:

```
# mount /cdrom
```

When the cdrom is mounted you can't eject it manually; you'll have to unmount it before you can do that:

```
# umount /cdrom
```

There is also a software command which unmounts the cdrom and ejects it:

```
# eject /dev/cd0a
```

6.3 Mounting the floppy

To mount a floppy you must know the name of the floppy device and the file system type of the floppy. Read the fdc(4) manpage for more information about device naming, as this will differ depending on the exact size and kind of your floppy disk. For example, to read and write a floppy in MS-DOS format you use the following command:

```
# mount -t msdos /dev/fd0a /mnt
```

Instead of `/mnt`, you can use another directory of your choice; you could, for example, create a `/floppy` directory like you did for the cdrom. If you do a lot of work with MS-DOS floppies, you will want to install the mtools package, which enables you to access a MS-DOS floppy (or hard disk partition) without the need to mount it. It is very handy for quickly copying a file from/to floppy:

```
# mcopy foo bar a:
# mcopy a:baz.txt baz
# mcopy a:\*.jpg .
```

6.4 Accessing a DOS/Windows partition

If NetBSD shares the hard disk with MS-DOS or Windows, it is possible modify the disklabel and make the Windows partitions visible from NetBSD. First, you must determine the geometry of the Windows partition, for example using fdisk(8).

```
# fdisk wd0
NetBSD disklabel disk geometry:
cylinders: 77520, heads: 16, sectors/track: 63 (1008 sectors/cylinder)
...
Partition table:
0: OS/2 HPFS or NTFS or QNX2 or Advanced UNIX (sysid 7)
    bootmenu: WinXP
    start 63, size 20643462 (10080 MB, Cyls 0-1285), Active
1: NetBSD (sysid 169)
    start 20643525, size 57496635 (28075 MB, Cyls 1285-4864)
2: <UNUSED>
3: <UNUSED>
```

Note: This example uses the `wd0` hard disk: substitute the device for your hard disk.

The output of the fdisk command shows that the Windows partition uses NTFS ("OS/2 HPFS or NTFS or QNX2 or Advanced UNIX (sysid 7)"), if the partition was FAT it would have said "Primary 'big' DOS, 16-bit FAT (>32MB) (sysid 6)".

The Windows partition is currently only known in the MBR partition table, in order to mount it from NetBSD it also needs to be in the NetBSD disk's disklabel. There are two ways to do so, editing the disklabel manually using **disklabel -e**, or using the mbrlabel(8) command, which is what we will describe first.

When running mbrlabel(8), it needs a disk which it will search for partitions that are in the MBR but not in the disklabel, and will then add them to the disklabel:

```
# disklabel wd0
...
 d:  78140160         0     unused      0     0        # (Cyl.      0 -  77519)
#
# mbrlabel -rw wd0
Found NTFS partition; size 20643462 (10079 MB), offset 63
  adding NTFS partition to slot e.
Found 4.2BSD partition; size 57496572 (28074 MB), offset 20643588
  skipping existing unused partition at slot c.

16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  57236256  20643588     4.2BSD   1024  8192 46920  # (Cyl.  20479*-  77261*)
 b:    260316  77879844       swap                     # (Cyl.  77261*-  77519)
 c:  57496572  20643588     unused      0     0        # (Cyl.  20479*-  77519)
 d:  78140160         0     unused      0     0        # (Cyl.      0 -  77519)
 e:  20643462        63       NTFS                     # (Cyl.      0*-  20479*)

Updating in-core and on-disk disk label.
#
# disklabel wd0
...
 d:  78140160         0     unused      0     0        # (Cyl.      0 -  77519)
 e:  20643462        63       NTFS                     # (Cyl.      0*-  20479*)
```

If you can't or don't want to use mbrlabel(8) for some reason, you will have to edit the disklabel manually with data from the **fdisk**-command above. The partition with the NTFS filesystem begins at sector 63 and has a size of 20643462 sectors. The NetBSD partition begins at sector 20643525 (20643525 = 20643462 + 63). You will use this data to modify the BSD disklabel: all you have to do is add one line which defines the position and type of the NTFS partition, choosing one of the still unused partition id letters. Use the **disklabel** command to modify the disklabel. If you give the `-e` option to **disklabel** it will invoke your favourite editor ($EDITOR) to modify the disklabel. For example:

```
# disklabel -e wd0
...
#        size   offset     fstype  [fsize bsize  cpg]
...
```

```
 d:  78140160         0     unused      0     0        # (Cyl.      0 -  77519)
 e:  20643462        63       NTFS
```

The partitions from “a” to “d” were already used, and the first available id was “e”. The “size” and “offset” fields have been filled with the previously calculated numbers. Next, the mount point must be created. For example:

```
# mkdir /c
```

finally, a line will be added to the `/etc/fstab` file.

```
/dev/wd0e /c ntfs ro,noauto 1 3
```

If you want to mount a MS-DOS "FAT" partition instead of a NTFS partition, use MSDOS as the "fstype" in the disklabel and "msdos" as filesystem type (3rd column) in `/etc/fstab`. You can also mount FAT-filesystems read/write, i.e. you can use "rw" instead of "ro" in `/etc/fstab` too. For details about NetBSD support for the MSDOS and NTFS filesystems, see mount_msdos(8) and mount_ntfs(8).

Now the Windows partition can be mounted with a simple command:

```
# mount /c
```

With this method you can mount NFS, FAT and FAT32 partitions. If you want to mount the partition(s) automatically at startup, remove the `noauto` option from `/etc/fstab`.

```
/dev/wd0e /c ntfs ro 1 3
```

6.5 Adding users

It’s time to add new users to the system, since you don’t want to use the root account for your daily work (yes, we’re serious about that!). NetBSD offers the useradd(8) utility to create user accounts. For example, to create a new user:

```
# useradd -m joe
```

The defaults for the useradd(8) command can be changed; see the useradd(8) man page.

Accounts that can su(1) to root are required to be in the "wheel" group. This can be done when the account is created by specifying a secondary group:

```
# useradd -m -G wheel joe
```

As an alternative, the usermod(8) command can be used to add a user to an existing group:

```
# usermod -G wheel joe
```

> **Note:** You can edit `/etc/group` directly to add users to groups, but do *not* edit `/etc/passwd` directly! All changes made to that file will get lost, see Section 6.6.

Note: If the system uses ssh, direct root access via ssh is disabled by default. Check the sshd_config(5) and `/etc/ssh/sshd_config` to change this behaviour.

6.6 Shadow passwords

Shadow passwords are enabled by default on NetBSD: all the passwords in `/etc/passwd` contain an "*"; the encrypted passwords are stored in another file, `/etc/master.passwd`, that can be read only by root. When you start vipw(8) to edit the password file, the program opens a copy of `/etc/master.passwd`; when you exit, vipw(8) checks the validity of the copy, creates a new `/etc/passwd` and installs the new `/etc/master.passwd` file. Finally, vipw(8) launches pwd_mkdb(8), which creates the files `/etc/pwd.db` and `/etc/spwd.db`, two databases which are equivalent to `/etc/passwd` and `/etc/master.passwd` but faster to process.

As you can see, passwords are handled automatically by NetBSD; if you use vipw(8) to edit the password file you don't need any special administration procedure.

It's very important to *always* use **vipw** and the other tools for account administration (chfn(1), chsh(1), chpass(1), passwd(1)) and to *never* modify directly `/etc/master.passwd` or `/etc/passwd`.

6.7 Stopping and rebooting the system

Use one of the following two shutdown(8) commands to halt and/or reboot the system:

```
# shutdown -h now
# shutdown -r now
```

Two other commands perform the same tasks are:

```
# halt
# reboot
```

halt(8), reboot(8), and shutdown(8) are not synonyms: the latter is more sophisticated. On a multiuser system you should really use shutdown(8); this will allow you to schedule a shutdown time, notify users, and it will also take care to shutdown database processes etc. properly without simply kill(1)ing them. For a more detailed description, see the shutdown(8), halt(8) and reboot(8) manpages.

Chapter 7
rc.d System

As of NetBSD 1.5, the startup of the system changed to using rc-scripts for controlling services, similar to the init-system System V and Linux use, but without runlevels. This chapter is an overview of the rc-system and its configuration on NetBSD.

7.1 The rc.d Configuration

The startup files for the system reside under `/etc`, they are:

- `/etc/rc`
- `/etc/rc.conf`
- `/etc/rc.d/*`
- `/etc/rc.lkm`
- `/etc/rc.local`
- `/etc/rc.shutdown`
- `/etc/rc.subr`
- `/etc/defaults/*`
- `/etc/rc.conf.d/*`

First, a look at controlling and supporting scripts, also documented in rc(8):

- When the kernel has initialized all devices on startup, it usually starts init(8), which in turn runs `/etc/rc`
- `/etc/rc` sorts the scripts in `/etc/rc.d` using rcorder(8), and runs them in that order. See the rcorder(8) manpage for more details on how the order of `/etc/rc.d` scripts is determined.
- `/etc/rc.subr` contains common functions used by many `/etc/rc.d/*` scripts.
- When shutting down the system with shutdown(8), `/etc/rc.shutdown` is run which runs the scripts in `/etc/rc.d` in reverse order (as defined by rcorder(8)).

Additional scripts outside of the `rc.d` directory:

- `/etc/rc.lkm` loads or unloads Loadable Kernel Modules (LKMs), see modload(8) and `/etc/rc.d/lkm[123]`.
- `/etc/rc.local` is almost the last script called at boot up. This script can be edited by the administrator to start local daemons that don't follow the rc-concept.

For example, packages installed pkgsrc usually add their startup files to /usr/pkg/etc/rc.d, and it's left as a decision to the system administrator on enabling them, either by manually copying/linking them to /etc/rc.d, or by adding them to /etc/rc.local. The following is the example from the system for an apache web server added to /etc/rc.local:

```
if [ -f /usr/pkg/etc/rc.d/apache ]; then
 /usr/pkg/etc/rc.d/apache start
fi
```

There's a central config file for bootscripts, rc.conf(5) located in /etc/rc.conf. /etc/rc.conf loads its defaults from /etc/defaults/rc.conf, the latter of which should not be touched. In order to alter a default setting, an override may be installed in /etc/rc.conf.

For example, if you wanted to enable the Secure Shell Daemon:

```
# cd /etc; grep ssh defaults/rc.conf
sshd=NO                 sshd_flags=""
# echo "sshd=YES" >> rc.conf
```

Or just edit /etc/rc.conf with your favorite editor. The same can be done with any default that needs to be changed. A common sequence often done after installing a fresh NetBSD system is:

```
# cat /etc/defaults/rc.conf >>/etc/rc.conf
# vi /etc/rc.conf
```

Be careful to use ">>" and not ">" else you will destroy the default contents in /etc/rc.conf, which are critical to remain there! After you have copied the defaults that way, modify anything you need to in /etc/rc.conf. Be sure to consult the rc.conf(5) manpage to explain all the settings in detail.

Last and not least, the /etc/rc.conf.d/ directory can be used for scripts-snippets from third party software, allowing setting only one or few settings per file.

7.2 The rc.d Scripts

The actual scripts that control services are in /etc/rc.d. Once a service has been activated or told not to activate in /etc/rc.conf it can be also be modified by calling the rc script from the command line, for example if an administrator needed to start the secure shell daemon:

```
# /etc/rc.d/sshd start
Starting sshd.
```

The rc scripts must receive one of the following arguments:

- start
- stop
- restart
- kill

An example might be when a new record has been added to the named database on a named server:

```
# /etc/rc.d/named restart
```

```
Stopping named.
Starting named.
```

A slightly more complex example is when a series of settings have been changed, for instance a firewall's ipfilter rules, ipnat configuration, and the secure shell server has switched encryption type:

```
# sh /etc/rc.d/ipfilter restart
# sh /etc/rc.d/ipnat restart
# sh /etc/rc.d/sshd restart
```

7.3 The Role of rcorder and rc Scripts

The startup system of every Unix system basically determines the order in which services are started one way or another. On some Unix systems this is done by numbering the files and/or putting them in separate run level directories. (Solaris relies on wildcards like `/etc/rc[23].d/S*` being sorted numerically when expanded.) Or they simply put all the commands that should be started at system boot time into a single monolithic script, which can be messy. (This is what ancient BSD and NetBSD did before the rc-system). On NetBSD this is done by the rc-scripts and their contents. Please note that NetBSD does not have multiple runlevels as found e.g. in System V systems like Solaris, or Linux.

At the beginning of each of the rc-scripts in `/etc/rc.d/*`, there is a series of comment-lines that have one of the following items in them:

- REQUIRE
- PROVIDE
- BEFORE
- KEYWORD

These dictate the dependencies of that particular rc script and hence rcorder can easily work either "up" or "down" as the situation requires. Following is an example of the `/etc/rc.d/nfsd` script:

```
...
 PROVIDE: nfsd
 REQUIRE: mountd

. /etc/rc.subr
...
```

Here we can see that this script provides the "nfsd" service, however, it requires "mountd" to be running first. The rcorder(8) utility will be used at system startup time to read through all the rc-scripts, and determine the correct order in which to run the rc-scripts (hence its name).

7.4 Additional Reading

There are other resources available pertaining to the rc.d system:

- One of the principal designers of rc.d, Luke Mewburn, gave a presentation on the system at USENIX 2001. It is available in PDF (http://www.mewburn.net/luke/papers/rc.d.pdf) format.
- Will Andrews wrote a Daemonnews (http://www.daemonnews.org/) article called The NetBSD rc.d System (http://ezine.daemonnews.org/200108/rcdsystem.html).

III. System configuration, administration and tuning

Chapter 8
Editing

8.1 Introducing vi

It is not like the vi editor needs introducing to seasoned UNIX users. The vi editor, originally developed by Bill Joy of Sun Microsystems, is an endlessly extensible, easy to use *light* ASCII editor and the bane of the newbie existence. This section will introduce the vi editor to the newbie and perhaps toss in a few ideas for the seasoned user as well.

The first half of this section will overview editing, saving, yanking/putting and navigating a file within a vi session. The second half will be a step by step sample vi session to help get started.

This is intended as a primer for using the vi editor, it is *not by any means* a thorough guide. It is meant to get the first time user up and using vi with enough skills to make changes to and create files.

8.1.1 The vi interface

Using the vi editor really is not much different than any other terminal based software with one exception, it does not use a tab type (or curses if you will) style interface, although many versions of vi *do use* curses it does not give the same look and feel of the typical curses based interface. Instead it works in two modes, *command* and *edit*. While this may seem strange, it is not much different than windows based editing if you think about it. Take this as an example, if you are using say gedit and you take the mouse, highlight some text, select cut and then paste, the whole time you are using the mouse you are not editing (even though you can). In vi, the same action is done by simply deleting the whole line with **dd** in command mode, moving to the line you wish to place it below and hitting **p** in command mode. One could almost say the analogy is “mouse mode vs. command mode” (although they are not exactly identical, conceptually the idea is similar).

To start up a vi session, one simply begins the way they might with any terminal based software:

```
$ vi filename
```

One important note to remember here is that when a file is edited, it is loaded into a memory buffer. The rest of the text will make reference to the buffer and file in their proper context. A file *only* changes when the user has committed changes with one of the write commands.

8.1.2 Switching to Edit Mode

The vi editor sports a range of options one can provide at start up, for the time being we will just look at the default startup. When invoked as shown above, the editors default startup mode is command mode, so in essence you cannot commence to typing into the buffer. Instead you must switch out out of command mode to enter text. The following text describes edit start modes:

a Append after cursor.
A Append to end of line.
C Change the rest of current line.
cw Change the current word.
i Insert before cursor.
I Insert before first non blank line.
o Open a line below for insert
O Open a line above for insert.

8.1.3 Switching Modes & Saving Buffers to Files

Of course knowing the edit commands does not do much good if you can't switch back to command mode and save a file, to switch back simply hit the **ESC** key. To enter certain commands, the colon must be used. Write commands are one such set of commands. To do this, simply enter **:**.

Hitting the colon then will put the user at the colon (or *command* if you will) prompt at the bottom left corner of the screen. Now let us look at the save commands:

:w Write the buffer to file.
:wq Write the buffer to file and quit.

8.1.4 Yanking and Putting

What good is an editor if you cannot manipulate blocks of text? Of course vi supports this feature as well and as with most of the vi commands it somewhat intuitive. To yank a line but *not* delete it, simply enter **yy** or **Y** in command mode and the current line will be copied into a buffer. To put the line somewhere, navigate to the line above where the line is to be put and hit the **p** key for the "put" command. To move a line, simply delete the whole line with the **dd** command, navigate and put.

8.1.4.1 Oops I Did Not Mean to do that!

Undo is pretty simple, **u** undoes the last action and **U** undoes the last line deleted or changes made on the last line.

8.1.5 Navigation in the Buffer

Most vi primers or tutorials start off with navigation, however, not unlike most editors in order to navigate a file there must be something to navigate to and from (hence why this column sort of went in reverse). Depending on your flavor of vi (or if it even *is* vi and not say elvis, nvi or vim) you can navigate in both edit and command mode.

For the beginner I feel that switching to command mode and then navigating is a bit safer until one has practiced for awhile. The navigation keys for terminals that are not recognized or do not support the use of arrow keys are the following:

k Moves the cursor up one line.
j Moves the cursor down one line.
l Moves the cursor right one character.
h Moves the cursor left one character.

If the terminal is recognized and supports them, the arrow keys can be used to navigate the buffer in command mode.

In addition to simple "one spot navigation" vi supports jumping to a line by simply typing in the line number at the colon prompt. For example, if you wanted to jump to line 223 the keystrokes from editor mode would look like so:

```
ESC
:223
```

8.1.6 Searching a File, the Alternate Navigational Aid

The vi editor supports searching using regular expression syntax, however, it is slightly different to invoke from command mode. One simply hits the **/** key in command mode and enters what they are searching for, as an example let us say I am searching for the expression *foo*:

```
/foo
```

That is it, to illustrate a slightly different expression, let us say I am looking for *foo bar*:

```
/foo bar
```

8.1.6.1 Additional Navigation Commands

Searching and scrolling are not the only ways to navigate a vi buffer. Following is a list of succinct navigation commands available for vi:

0 Move to beginning of line.
$ Move to end of line.
b Back up one word.
w Move forward one word.
G Move to the bottom of the buffer.
H Move to the top line on the screen.
L Move to the last line on the screen.
M Move the cursor to the middle of the screen.
N Scan for next search match but opposite direction.
n Scan for next search match in the same direction.

8.1.7 A Sample Session

Now that we have covered the basics, let us run a sample session using a couple of the items discussed so far. First, we open an empty file into the buffer from the command line like so:

```
# vi foo.txt
```

Next we switch to edit mode and enter two lines separated by an empty line, remember our buffer is empty so we hit the **i** key to insert before cursor and enter some text:

```
This is some text

there we skipped a line
~
~
~
~
```

Now hit the **ESC** key to switch back into command mode.

Now that we are in command mode, let us save the file. First, hit the **:** key, the cursor should be sitting in the lower left corner right after a prompt. At the **:** prompt enter **w** and hit the **ENTER** or **RETURN** key. The file has just been saved. There should have been a message to that effect, some vi editors will also tell you the name, how many lines and the size of the file as well.

It is time to navigate, the cursor should be sitting wherever it was when the file was saved. Try using the arrow keys to move around a bit. If they do not work (or you are just plain curious) try out the **hjkl** keys to see how they work.

Finally, let us do two more things, first, navigate up to the first line and then to the first character. Try out some of the other command mode navigation keys on that line, hit the following keys a couple of times:

```
$
0
$
0
```

The cursor should hop to the end of line, back to the beginning and then to the end again.

Next, search for an expression by hitting the **/** key and an expression like so:

```
/we
```

The cursor should jump to the *first occurrence* of *we*.

Now save the file and exit using write and quit:

```
:wq
```

8.2 Configuring vi

The standard editor supplied with NetBSD is, needless to say, vi, the most loved and hated editor in the world. If you don't use vi, skip this section, otherwise read it before installing other versions of vi. NetBSD's vi (*nvi*) was written by Keith Bostic of UCB to have a freely redistributable version of this editor and has many powerful extensions worth learning while being still very compatible with the original vi. Nvi has become the standard version of vi for BSD.

Amongst the most interesting extensions are:

- Extended regular expressions (egrep style), enabled with option `extended`.
- Tag stacks.
- Infinite undo (to undo, press **u**; to continue undoing, press **.**).
- Incremental search, enabled with the option `searchincr`.
- Left-right scrolling of lines, enabled with the option `leftright`; the number of columns to scroll is defined by the `sidescroll` option.
- Command line history editing, enabled with the option `cedit`.
- Filename completion, enabled by the `filec` option.
- Backgrounded screens and displays.
- Split screen editing.

8.2.1 Extensions to `.exrc`

The following example shows a `.exrc` file with some extended options enabled.

```
set showmode ruler
set filec=^[
set cedit=^[
```

The first line enables the display of the cursor position (row and column) and of the current mode (Command, Insert, Append) on the status line. The second line (where ^[is the ESC character) enables filename completion with the ESC character. The third line enables command line history editing (also with the ESC character.) For example, writing ":" and then pressing ESC opens a window with a list of the previous commands which can be edited and executed (pressing Enter on a command executes it.)

8.2.2 Documentation

The source *tarball* (`src.tgz`) contains a lot of useful documentation on (n)vi and ex, in the `/usr/src/usr.bin/vi/docs` directory. For example:

- Edit: A tutorial
- Ex Reference Manual
- Vi man page
- An Introduction to Display Editing with Vi by William Joy and Mark Horton

- Ex/Vi Reference Manual by Keith Bostic
- Vi Command & Function Reference
- Vi tutorial (beginner and advanced)

If you have never used vi, the “Vi tutorial” is a good starting point. It is meant to be read using vi and it gradually introduces the reader to all the vi commands, which can be tested while reading. *An Introduction to Display Editing with Vi* by William Joy and Mark Horton is also a very good starting point.

If you want to learn more about vi and the nvi extensions you should read the *Ex/Vi Reference Manual* by Keith Bostic which documents all the editor’s commands and options.

8.3 Using tags with vi

This topic is not directly related to NetBSD but it can be useful, for example, for examining the kernel sources.

When you examine a set of sources in a tree of directories and subdirectories you can simplify your work using the *tag* feature of vi. The method is the following:

1. **cd** to the base directory of the sources.

   ```
   $ cd /path
   ```

2. Write the following commands:

   ```
   $ find . -name "*.[ch]" > filelist
   $ cat filelist | xargs ctags
   ```

3. Add the following line to `.exrc`

   ```
   set tags=/path/tags
   ```

   (substitute the correct path instead of *path*.)

Chapter 9
X

9.1 What is X?

The X Window System is the graphical subsystem available for NetBSD and many Unix (and non Unix) systems. In fact it is much more than that: thanks to the usage of the X protocol, the X Window System is "network transparent" and can run distributed applications (client-server). This means, roughly, that you can run an application on one host (client) and transparently display the graphical output on another host (server); transparently means that you don't have to modify the application to achieve this result. The X Window System is produced and maintained by the X Consortium and the current release is X11R6. The flavour of X used by NetBSD is XFree86, a freely redistributable open source implementation of the X Window System.

Please note that the X Window System is a rather bare bones framework which acts again as a base for modern desktop environments like GNOME, KDE or XFCE, but they are not part of the X Windows System, and while NetBSD ships with the X Window System, it does not include these desktop environments. They can be added easily via the pkgsrc system, though, if needed.

When you start using X you'll find many new terms which you'll probably find confusing at first. The basic elements to use X are:

- *Video hardware* supported by XFree86, i.e. your video card.
- An *X server* running on top of the hardware. The X server provides a standard way to open windows, do graphics (including fonts for text display), and get mouse/keyboard/other input. X is network-transparent, so that you can run X clients on one machine, and the X server (i.e., the display, with video hardware) on another machine.
- A *window manager* running on the X server. The window manager is essentially a special client that is allowed to control placement of windows. It also "decorates" windows with standard "widgets" (usually these provide window-motion, resizing, iconifying, and perhaps a few other actions). A window manager also may provide backdrops, etc. Window managers can also let you kill windows/programs by clicking on their windows, and so forth.
- A *desktop environment* (optional.) KDE and GNOME, for example, are desktops: they are suites of more-or-less integrated software designed to give you a well-defined range of software and a more or less common interface to each of the programs. These include a help browser of some kind, a "desktop-metaphor" access to your filesystem, custom terminals to replace xterm, software development environments, audio, picture/animation viewres, etc.
- Any other applications (3rd party X clients) that you have. These talk to the X server and to the window manager. Unless the window manager is part of the desktop (if any), the desktop probably doesn't get involved in much of anything that these applications do. (However, e.g., GNOME may be able to detect that you've installed the GIMP, for example, and so offer a menu to launch the GIMP.)

To summarize: in order to use a graphical environment you need

- the XFree86 system
- a window manager (XFree86 already comes with a very basic window manager called twm.)
- If you prefer a more sophisticated environment you'll probably want to install a desktop too, although this is not necessary. Desktops have some nice features that are helpful to users who come from environments such as Macintosh or MS-WINDOWS (the KDE desktop, for example, has a very similar flavour to MS-WINDOWS.)

  Note: By now it should be clear that desktops like GNOME and KDE do not provide X servers. They run on top of an existing X server supplied by XFree86. KDE and GNOME can make use of their own window manager or of a separately installed window manager.

Normally, you can run at most one window manager at any given time on a given X server. (But you can run multiple X servers on a single computer.) If you are not running a window manager of your choosing, and start KDE/GNOME, then that desktop environment will run a window manager for you.

9.2 Configuration

If you haven't chosen a minimal configuration during installation, X is already installed and ready to run on your computer. Depending on the exact hardware platform you run NetBSD and X on, you may or may not need to configure your X server. While most workstation ports (sparc, pmax, ...) will just work without further configuration if you use the right X-server, which is what `/usr/X11R6/bin/X` is usually linked to.

On PCs (i386, amd64), Shark and some other platforms, you will have to tune the X server first by create the menacing `/etc/X11/XF86Config` file. To get an idea of what this file looks like, examine the `/usr/X11R6/lib/X11/XF86Config.eg` file. The structure of the configuration file is described formally in XF86Config(5), which can be examined with the following command:

```
# man XF86Config
```

Before configuring the system it is advisable to carefully read the documentation found in `/usr/X11R6/lib/X11/doc`: there are various README's for the video cards, for the mouse and even a NetBSD specific one (`README.NetBSD`.) I suggest to start by reading `QuickStart.doc`. You might have the feeling that other systems let you start more quickly and with less effort, but the time spent reading this documentation is not wasted: the knowledge of X and of your configuration that you gain will turn out very useful on many future occasions and you'll be able to get the most from your hardware (and software too.)

You can create the `/etc/X11/XF86Config` file manually with an editor or you can generate it automatically with an interactive configuration program. The best known programs are xf86config, XF86Setup (XFree86 3.x) and xf86cfg (XFree86 4.x). Both xf86config and xf86cfg are installed by default with X; XF86Setup is a graphical configuration tool which can be installed from pkgsrc.

You may find that a mixed approach is better: first create the XF86Config with one of the two programs and then check it and tune it manually with an editor. E.g. for the GUI based **xf86cfg**:

```
# xf86cfg
# configure to your will, and at the end save to /etc/X11/XF86Config
# vi /etc/X11/XF86Config
```

or for the screen-oriented, non-graphical **xf86config**:

```
# xf86config
# configure to your will, and at the end save to /etc/X11/XF86Config
# vi /etc/X11/XF86Config
```

The interface of the two programs is different but they both require the same set of information:

- the mouse type and the mouse device to be used
- the keyboard type and its layout
- the type of video card
- the type of monitor

Before configuring the system you should collect the required information.

9.3 The mouse

The first thing to check is the type of mouse you are using (for example, serial or PS/2, ...) and the mouse device (for example, *wsmouse* requires a different protocol). If you are using a serial mouse, choose the required protocol and specify the serial port to which it is connected.

For example, PS/2 and USB mice usually are attached to the wsmouse device, and as such you can use:

```
Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
        Option      "Protocol" "wsmouse"
        Option      "Device" "/dev/wsmouse"
EndSection
```

If you use a mouse with a scroll wheel, scrolling up and down is handled as mouse buttons 4 and 5 being pressed (respectively). Many applications like xterm or Firefox handle these button presses. To enable the scroll wheel, add the following lines to the "Pointer" section:

```
Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
        Option      "Protocol" "wsmouse"
        Option      "Device" "/dev/wsmouse"
        Option      "ZAxisMapping" "4 5"
EndSection
```

For a serial mouse on the first serial port, try something like:

```
Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
```

```
        Option      "Protocol" "auto"
        Option      "Device" "/dev/tty00"
EndSection
```

In this example. `/dev/tty00` is the first serial port here, use `/dev/tty01` for the second and so on. Protocol "auto" will try to automatically detect the protocol of your serial mouse. If this doesn't work, try values like "Microsoft", "ItelliMouse" or "Logitech", see `/usr/X11R6/lib/X11/XF86Config.eg` and `/usr/X11R6/lib/X11/doc/README.mouse` for more information.

9.4 The keyboard

Even if you have already configured your keyboard for wscons, you need to configure it for X too, at least if you want to get a non US layout.

An easy solution is to use the XKB protocol, specifying the keyboard type and layout.

This is one area in which that configuration programs are weak and you may want to choose the standard layout and modify the generated configuration file manually:

```
Section "InputDevice"
        Identifier  "Keyboard0"
        Driver      "keyboard"
        Option      "XkbRules" "xfree86"
        Option      "XkbModel" "pc102"
        Option      "XkbLayout" "de"
        Option      "XkbOptions" "ctrl:nocaps"
EndSection
```

If you want to use the "Windows" keys on your keyboard, use "pc105" instead of "pc102" for `XkbModel`.

9.5 The monitor

It is very important to correctly specify the values of the horizontal and vertical frequency of the monitor: a correct definition shields the monitor from damages deriving from an incompatible setup of the video card. This information can be found in the monitor's manual. In the X documentation directory there is a file containing the settings of many monitors; it can be used as a starting point to customize your own settings.

9.6 The video card

The video card can be chosen from the database of the configuration programs; the program will take care of all the needed setups. Video card support is slightly different between XFree86 3.x and 4.x.

XFree86 3.x has multiple servers for different categories of video card chipsets. XFree86 4.x has only one server. Different video chipsets are supported via platform independent driver modules, which can be found in `/usr/X11R6/lib/modules/drivers`.

9.6.1 XFree 3.x

When you have selected the correct video card you must choose the X server for the card. Usually, the configuration programs can automatically determine the correct server, but some video cards can be driven by more than one server (for example, S3 Virge is supported by the SVGA and S3V servers); in this case, study the documentation of the servers to decide which one you need: different servers usually have different capabilities and a different degree of support for the video cards.

9.6.2 XFree86 4.x

After selecting the correct video card the configuration program will automatically select the appropriate driver or suggest it. If you have not selected a card you can configure your video card by selecting the required module.

9.7 Starting X

When you exit the configuration program, it creates the file `/etc/X11/XF86Config`, which can be further examined and modified by hand.

Before starting X you should:

- check that the symbolic link `/usr/X11R6/bin/X` points to the correct X server:

  ```
  # ls -l /usr/X11R6/bin/X
  ```

- Verify that the configuration is correct. Launch:

  ```
  # X -probeonly
  ```

  and examine carefully the output.

Now you can start X with the following command:

```
# startx
```

If X doesn't fire up there is probably some error in the configuration file.

If X starts but doesn't work as expected (for example, you can't move the mouse pointer) you can exit quickly with the Ctrl-Alt-Backspace key combination (not available on all ports). If everything worked correctly you are left in the X environment with the default window manager (twm): although it is a simple window manager many users feel that it is enough for their needs. If you want a highly configurable window manager with many bells and whistles, you have many choices in the package collection, see Section 9.9 below.

To start customizing X, try giving the following command in an xterm to change the background color:

```
# xsetroot -solid DarkSeaGreen
```

9.8 Customizing X

The look of the X environment can be customized in several ways. The easiest method is to copy the default `.xinitrc` file in your home directory and modify it, or create a simple, new one from scratch. For example:

```
$ cp /usr/X11R6/lib/X11/xinit/xinitrc $HOME/.xinitrc
$ vi $HOME/.xinitrc
```

The following example shows how to start the window manager (twm), open an instance of the xclock program in the lower right part of the screen and two xterm windows. The “Bisque4” color is used for the background.

```
The first part of the file is the same
...
# start some nice programs
twm &
xclock -geometry 50x50-1-1 &
xterm -geometry 80x34-1+1 -bg OldLace &
xsetroot -solid Bisque4 &
exec xterm -geometry 80x44+0+0 -bg AntiqueWhite -name login
```

With this type of setup, to exit X you must close the last xterm (the one with the “login” title - just type "exit" in it, e.g.).

Even with this simple configuration X has a considerably nicer look. To give an even better look to the environment you can install some utility program from the package collection. For example:

xcolorsel

: displays all the colors defined in `rgb.txt`. Use it to choose background colors for the root window or for xterms.

xpmroot

: lets you use a pixmap for the background.

xscreensaver

: X screen saver.

xdaemon

: no desktop can be complete without this package, which displays a moveable bitmap of the BSD daemon in two selectable sizes.

9.9 Other window managers

If you don’t like twm, which is a very simple window manager lacking many features and not very configurable, you can choose another window manager from the package collection. Some of the most popular are: fvwm2, olwm/olvwm (Open Look Window Manager), WindowMaker, Enlightenment, AfterStep.

In the rest of this section the installation of WindowMaker is described as an example. WindowMaker is a very nice looking and highly configurable window manager. To add the program the `windowmaker-0.60.tgz` precompiled package will be used, which depends on some other packages which must be installed. As usual, both **pkg_add** and **make install** will fetch the needed packages automatically, so there is no need to go through the dependencies manually.

```
# cd /usr/pkgsrc/x11/windowmaker
# make depends-list
xpm-3.4k
jpeg-6b
pkglibtool-1.2p2
giflib-3.0
libproplist-0.9.1
tiff-3.5.2
```

> **Note:** You can also see the dependencies with the following command:
>
> ```
> # pkg_info -f windowmaker-0.61.0.tgz | grep depends
> ```

After adding the required packages, WindowMaker and some preconfigured themes can be added:

```
# pkg_add windowmaker-0.61.0.tgz wmthemes-0.6x.tgz
```

WindowMaker is now installed; to start it you must modify your `.xinitrc` and/or `.xsession` file: substitute the line which calls `twm` with a line which calls `wmaker`. For example:

```
# start some nice programs
# start WindowMaker
wmaker &
xclock -geometry 50x50-1-1 &
xdaemon2 -geometry +0-70 &
...
```

In this example the xdaemon program is also started automatically.

Before starting WindowMaker the configuration program must be run:

```
$ wmaker.inst
$ startx
```

In all the examples above, choosing “Quit” or similar from the window manager’s menu will quit the window manager, but not log you out as you may expect. To do so, run the window manager *last and with no &* in your `.xinitrc`:

```
...
xclock -geometry 50x50-1-1 &
xdaemon2 -geometry +0-70 &
wmaker # no & here!
```

9.10 Graphical login with xdm

If you always use X for your work and the first thing you do after you log in is run **startx**, you can set up a graphical login for your workstation which does this automatically. It is very easy:

1. Create the `.xsession` file in your home directory. This file is similar to `~/.xinitrc` and can, in fact, be a link to the latter.

   ```
   $ cd $HOME
   $ ln -s .xinitrc .xsession
   ```

2. Modify `/etc/rc.conf`:

   ```
   xdm=YES        xdm_flags=""        # x11 display manager
   ```

   If you prefer you can add the following line at the end of `/etc/rc.local` instead of modifying rc.conf:

   ```
   /usr/X11R6/bin/xdm
   ```

   This method can be used to start, for example, kdm or gdm instead of xdm.

The configuration files for xdm are in the `/usr/X11R6/lib/X11/xdm` directory. In the `Xservers` file X is started by default on "vt05", which is the console you reach via "Ctrl+Alt+F5"; if you want to use another virtual console instead, this is the right place to modify the setting. In order to avoid keyboard contention between getty and xdm it is advisable to start xdm on a virtual terminal where getty is disabled. For example if in `Xservers` you have:

```
:0 local /usr/X11R6/bin/X :0 vt04
```

in `/etc/ttys` you should have

```
ttyE3   "/usr/libexec/getty Pc"         vt220   off secure
```

(Please note that vt04 corresponds to ttyE3 because vt start at 1 and ttyE start at 0).

If you want a nice look for your xdm login screen, you can modify the xdm configuration file. For example, to change the background color you can add the following line the the `Xsetup_0` file:

```
xsetroot -solid SeaGreen
```

Instead of setting a color, you can put an image on the background using the xpmroot program: For example:

```
xpmroot /path_to_xpm/netbsd.xpm
```

If you experiment a little with the configuration file you can achieve many nice looking effects and build a pleasing login screen. Note that other display managers like gdm and kdm offer different ways of configuration, usually GUI based.

Chapter 10
Linux emulation

The NetBSD port for i386, alpha, mac68k, macppc and many others can execute a great number of native Linux programs, using the Linux emulation layer. Generally, when you think about emulation you imagine something slow and inefficient because, often, emulations must reproduce hardware instructions and even architectures (usually from old machines) in software. In the case of the Linux emulation this is radically different: it is only a thin software layer, mostly for system calls which are already very similar between the two systems. The application code itself is processed at the full speed of your CPU, so you don't get a degraded performance with the Linux emulation and the feeling is exactly the same as for native NetBSD applications.

This chapter explains how to configure the Linux emulation with an example: the installation of the well known Acrobat Reader version 4 program.

10.1 Emulation setup

The installation of the Linux emulation is described in the compat_linux(8) man page; using the package system only two steps are needed.

1. Configuring the kernel.
2. Installing the Linux libraries.
3. Installing Linux applications like Acrobat Reader

10.1.1 Configuring the kernel

If you use a GENERIC kernel you don't need to do anything because Linux compatibility is already enabled.

If you use a customized kernel, check that the following options are enabled:

```
option COMPAT_LINUX
option EXEC_ELF32
```

when you have compiled a kernel with the previous options you can start installing the necessary software.

10.1.2 Installing the Linux libraries

Usually, applications are linked against shared libraries, and for Linux applications, Linux shared libraries are needed. You can get the shared libraries from any Linux distribution, provided it's not too

old, but the suggested method is to use the package system and install the libraries automatically (which uses SuSE libraries). When you install the libraries, the following happens:

- A *secondary root directory* is created which will be used for Linux programs. This directory is `/emul/linux/`. The Linux programs in emulation mode will use this directory as their root directory and use files there. If a required file is not found, it will be searched with `/` as root directory.

  For example, if a Linux application opens `/etc/ld.so.conf`, it will first be searched in `/emul/linux/etc/ld.so.conf`, and if not found there in `/etc/ld.so.conf`.

- The shared libraries for Linux are installed. Most applications are linked dynamically and expect to find the necessary libraries on the system. For example, for Acrobat Reader, if you go to the `/usr/pkgsrc/print/acroread` and give the **make depends** command, you get the following message:

  ```
  ===>  acroread-4.0 requires Linux glibc2 libraries - see compat_linux(8).
  ```

Both operations will be handled automatically by the package system, without the need of manual intervention from the user (I suppose that, by now, you have already begun to love the package system...). Note that this section describes manual installation of the Linux libraries.

To install the libraries, a program must be installed that handles the RPM format: it is `rpm`, which will be used to extract the SuSE libraries. Execute **make** and **make install** in the `/usr/pkgsrc/misc/rpm/` directory to build and install **rpm**.

Next the `suse_base` package must be installed. The SuSE RPM files can be downloaded by the package system or, if you have a SuSE CD, you can copy them in the `/usr/pkgsrc/distfiles/suse` directory and then run **make** and **make install** after going to the `/usr/pkgsrc/emulators/suse_base` directory.

With the same method install `suse_compat` and `suse_x11`. The final configuration is:

```
# pkg_info -a | grep suse
suse_base-7.3nb5    Linux compatibility package
suse_x11-7.3nb2     Linux compatibility package for X11 binaries
suse_compat-7.3     Linux compatibility package with old shared libraries
```

10.1.3 Installing Acrobat Reader

Now everything is ready for the installation of the Acrobat Reader program (or other Linux programs). Change to `/usr/pkgsrc/print/acroread` and give the usual commands.

```
# make
# make install
```

> **Note:** To download and install Acrobat Reader you need to add the line "ACCEPTABLE_LICENSES+=adobe-acrobat-license" to `/etc/mk.conf` to accept the Acrobat Reader license, simply follow the instructions given after **make**.

10.2 Directory structure

If we examine the outcome of the installation of the Linux libraries and programs we find that `/emul/linux` is a symbolic link pointing to `/usr/pkg/emul/linux`, where the following directories have been created:

bin/
boot/
cdrom/
dev/
etc/
floppy/
home/
lib/
mnt/
opt/
proc/
root/
sbin/
usr/

> **Note:** Please always refer to `/emul/linux` and not to `/usr/pkg/emul/linux`. The latter is an implementation detail and may change in the future.

How much space is required for the Linux emulation software? On my system I get the following figure:

```
# cd /usr/pkg/emul
# du -k linux
...
214049   linux/
```

Acrobat Reader, the program, has been installed in the usual directory for package binaries: `/usr/pkg/bin`. It can be run just as any other program:

```
$ acroread netbsd.pdf
```

10.3 Emulating /proc

Some Linux programs rely on a Linux-like `/proc` filesystem. The NetBSD procfs filesystem can emulate a `/proc` filesystem that contains Linux-specific pseudo-files. To accomplish this you can mount the procfs with the “linux”-option:

```
# mount_procfs -o linux procfs /emul/linux/proc
```

In this example a Linux-like proc filesystem will be mounted to the `/emul/linux/proc` directory. You can also let NetBSD mount it automatically during the booting process of NetBSD, by adding the following line to `/etc/fstab`:

```
procfs /emul/linux/proc procfs ro,linux
```

Chapter 11
Audio

This chapter is a short introduction to the usage of audio devices on NetBSD (who wants a dumb computer, anyway?)

11.1 Basic hardware elements

In order to make audio work on your system you must know what audio card is installed. Sadly it is often not enough to know the brand and model of the card, because many cards use chipsets manufactured from third parties. Therefore knowing the chipset installed on the audio card can sometimes be useful. The NetBSD kernel can recognize many chipsets and a quick look at **dmesg** is enough most of the time.

Therefore, type the following command:

```
# dmesg | more
```

and look for the audio card and chipset. If you're lucky you won't need to do anything because NetBSD automatically detects and configures many audio cards.

Sometimes audio doesn't work because the card is not supported or because you need to do some work in order for the card to be detected by NetBSD. Many audio cards are nowadays very cheap, and it is worth considering buying a different card, but before doing this you can try some simple steps to make the card work with NetBSD.

11.2 BIOS settings

This section is useful only to the owners of i386 PCs; on other architectures (e.g. Amiga) there are no such features. The most important thing to determine in order to use the audio card with NetBSD is the type of bus supported by the card.

The most common interfaces are ISA and PCI.

ISA Plug and Play cards are usually more tricky to configure mostly because of the interaction with the BIOS of the computer.

On the newer machines (those produced after 1997) there is a BIOS option which causes many headaches for the configuration of ISA Plug and Play audio cards (but not only audio cards): this option is usually named "PNP OS Installed" and is commonly found in the "PNP/PCI Configuration" (the names can be different in your BIOS.) As a general rule it is usually better to disable (i.e. set it to "NO") this option for NetBSD.

> **Note:** On many systems everything works fine even if this option is enabled. This is highly system dependent.

11.3 Configuring the audio device

During the installation of NetBSD the devices are created in the `/dev` directory. We are primarily interested in:

```
/dev/audio
/dev/sound
/dev/mixer
```

If they are not present they can be created like this:

```
# cd /dev
# ./MAKEDEV all
```

This command creates all the devices, including the audio devices.

The audio card is now probably ready to be used without further work.

You can make a quick test and send an audio file to the device (audio files usually have the `.au` extension), but if you don't have an audio file you can just send a text or binary file (of course you won't hear anything useful...). Use `/dev/audio` or `/dev/sound`:

```
# cat filename > /dev/audio
```

or

```
# cat filename > /dev/sound
```

If you hear something it means that the card is supported by NetBSD and was recognized and configured by the kernel at boot; otherwise you must configure the kernel settings for the audio device installed on the system (assuming the card/chipset is supported.)

11.4 Configuring the kernel audio devices

NetBSD supports a wide range of audio cards and the GENERIC kernel already enables and configures most of them. Sometimes it is necessary to manually set up the IRQ and DMA for non-PnP ISA cards.

> **Note:** When you create a custom kernel it is better to work on a copy of the GENERIC file, as described in Chapter 29.

If you still have problems you can try enabling all the devices, because some audio cards can be made to work only by emulating another card.

Many chipset make use of the SoundBlaster and OPL compatibility, but a great number of them work with the WSS emulation.

OPL is a MIDI synthesizer produced by Yamaha; there are many OPL variants (e.g. OPL2, OPL3SA, OPL3SA2, etc.). Many audio cards rely on this component or on a compatible one. For example, the chips produced by Crystal (and amongst them the very common CS423x) all have this chipset, and that's why they work with NetBSD.

WSS is not a microchip; it is the acronym of Windows Sound System. WSS is the name of the NetBSD kernel driver which supports the audio system of Microsoft Windows. Many audio cards work with Windows because they adhere to this standard (WSS) and the same holds for NetBSD.

Of the many audio cards that I tested with NetBSD, a good number work only if `opl*` and `wss*` are enabled in the kernel.

You should have no problem to get the Creative SoundBlaster cards to work with NetBSD: almost all of them are supported, including the Sound Blaster Live 1024!

When everything works you can disable in the kernel configuration file the devices that you don't need.

11.5 Advanced commands

NetBSD comes with a number of commands that deal with audio devices. They are:

- audioctl(1)
- mixerctl(1)
- audioplay(1)
- audiorecord(1)

11.5.1 audioctl(1)

audioctl(1) made its appearance in NetBSD 1.3 and is used to manually set some variables regarding audio I/O, like the frequencies for playing and recording. The available parameters can be displayed with the following command:

```
# audioctl -a | more
```

For example, to listen to CD quality music you can use the following command.

```
# audioctl -w play=44100,2,16,slinear_le
```

This command sets the frequency to 44100Hz, 2 audio channels, 16 bit, slinear_le encoding.

You can see the supported encodings with:

```
# audioctl encodings
```

This command displays the list of all the encodings supported by the audio card on your system.

11.5.2 mixerctl(1)

This command is used to configure the audio mixing and has an interface similar to that of audioctl(1).

11.5.3 audioplay(1)

With this command you can play audio files in simple formats like ULAW and WAV. For more sophisticated needs you might want to install one of the many programs available in the package system which let you play audio files in different formats (e.g. MP3, etc.)

11.5.4 audiorecord(1)

Not unsurprisingly this command is used to record audio files.

Chapter 12
Printing

This chapter describes a simple configuration for printing, using an HP Deskjet 690C connected to the first parallel port as and the lpd printing system that comes with NetBSD an example. First, the system will be configured to print text documents, and next the configuration will be extended to print PostScript documents using the Ghostscript program. Please note that there are other, alternative printing systems available in the packages collection, like lprng and the Common Unix Printing System (CUPS), which are not covered here.

12.1 Enabling the printer daemon

After installation it is not yet possible to print, because the **lpd** printer spooler daemon is not enabled. To enable **lpd**, one line in the `/etc/rc.conf` file must be changed from:

```
lpd=NO
```

to

```
lpd=YES
```

The change will come into effect at the next boot, but the daemon can be started manually now:

```
# sh /etc/rc.d/lpd start
```

To check if **lpd** is active, type the following command:

```
# ps ax | grep lpd
  179 ??  Is     0:00.01 lpd
```

If you don't see an entry for lpd in the output of the previous command, the daemon is not active.

The lpd system is configured via `/etc/printcap`. Before configuring `/etc/printcap` it is a good idea to make a printer test, to check if the physical connection between your computer and the printer is working. The test sends out some data directly to the printer device. Assuming you use a printer connected to the parallel port, this is `/dev/lpt0`; if you use an USB printer try `/dev/ulpt0`. Please check the manpages of these devices (lpt(4), ulpt(4)) for more information!

In our example we have a printer attached to the parallel port, so we run this:

```
# lptest 70 5 > /dev/lpt0
```

To see what the output should look like, try the same command without redirecting the output to the printer:

```
# lptest 70 5
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdef
```

```
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefg
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
```

A frequent problem is that the output on the printer is not correctly aligned in columns but has a "staircase" configuration. This usually means that the printer is configured to begin a new line at the left margin after receiving both a <CR> (carriage return, ASCII 13) character and a <LF> (line feed, ASCII 10) character. NetBSD only sends a <LF> character. You can fix this problem in two ways:

- by changing the configuration of the printer
- by using a simple printer filter (described later)

> **Note:** In the previous example the lpd spooler is not involved because the program output is sent directly to the printer device (`/dev/lpt0`) and is not spooled.

12.2 Configuring `/etc/printcap`

This section explains how to configure the example printer to print text documents.

The printer must have an entry in the `/etc/printcap` file; the entry contains the printer id (the name of the printer) and the printer description. The *lp* id is the default used by many programs. Here is an example entry:

Example 12-1. `/etc/printcap`

```
lp|local printer|HP DeskJet 690C:\
        :lp=/dev/lpa0:sd=/var/spool/lpd/lp:lf=/var/log/lpd-errs:\
        :sh:pl#66:pw#80:if=/usr/local/libexec/lpfilter:
```

The file format and options are described in detail in the printcap(5) manpage. Please note that an *input filter* has been specified (with the *if* option) which will take care of eliminating the staircase problem:

```
if=/usr/local/libexec/lpfilter
```

> **Printer driver and HP printers:** Example 12-1 uses the *lpa0* device (polled driver) for the printer, instead of the *lpd0* (interrupt driven driver). Using interrupts there is a communication problem with some printers, and the HP Deskjet 690C is one of them: printing is very slow and one PostScript page can take hours. The problem is solved using the *lpa* driver. It is also possible to compile a custom kernel where lpt is polled.

The printcap entry for the printer also specifies a spool directory, which must be created; this directory will be used by the lpd daemon to accumulate the data to be printed:

```
# cd /var/spool/lpd
# mkdir lp
```

```
# chown daemon:daemon lp
# chmod 770 lp
```

The only missing part is the `lpfilter` input filter, which must be written. The only task performed by this filter is to configure the printer for the elimination of the staircase problem before sending the text to be printed. The printer used in this example requires the following initialization string: "`<ESC>&k2G`".

Example 12-2. `/usr/local/libexec/lpfilter`

```
#!/bin/sh
# Treat LF as CR+LF
printf "\033&k2G" && cat && exit 0
exit 2
```

After saving this script into the name you used in `/etc/printcap`, you need to make sure it's executable:

```
# chmod 755 /usr/local/libexec/lpfilter*
```

> **Note:** There is another filter that can be used:
>
> `if=/usr/libexec/lpr/lpf:`
>
> This filter is much more complex than the one presented before. It is written to process the output of **nroff** and handles underline and overprinting, expands tab characters and converts LF to CR + LF. The source to this filter program can be found in `/usr/src/usr.sbin/lpr/filters/lpf.c`.

After everything is in place now, the **lptest** command can be run again now, this time using the **lpr** command, which will first send the data to the lpd spooler, then runs the filter and sends the data off to the printer:

```
# lptest 70 5 | lpr -h
```

The **lpr** program prints text using the spooler to send data to the printer; the `-h` option turns off the printing of a banner page (not really necessary, because of the *sh* option in `/etc/printcap`). Users more familiar with the System V printing system can also use the lp(1) command that comes as an alternative to lpr(1).

12.3 Configuring Ghostscript

Now that basic printing works, the functionality for printing PostScript files can be added. The simple printer used in this example does not support native printing of PostScript files; a program must be used which is capable of converting a PostScript document in a sequence of commands that the printer understands. The Ghostscript program, which can be found in packages collection, can be used to this purpose (see Part VI in *The NetBSD & pkgsrc Guide*). This section explains how to configure lpd to use Ghostscript to print PostScript files on the HP Deskjet 690C.

A second id for the printer will be created in `/etc/printcap`: this new id will use a different input filter, which will call Ghostscript to perform the actual print of the PostScript document. Therefore, text

documents will be printed on the *lp* printer and PostScript documents on the *ps* printer: both entries use the same physical printer but have different printing filters.

The same result can be achieved using different configurations. For example, a single entry with only one filter could be used. For this, the filter should be able to automatically determine the format of the document being printed, and use the appropriate printing program. This approach is simpler but leads to a more complex filter; if you like it you should consider installing the magicfilter program from the packages collection: it does this and many other things automatically.

For our approach, the new `/etc/printcap` file looks like this:

Example 12-3. `/etc/printcap`

```
lp|local printer|HP DeskJet 690C:\
        :lp=/dev/lpa0:sd=/var/spool/lpd/lp:lf=/var/log/lpd-errs:\
        :sh:pl#66:pw#80:if=/usr/local/libexec/lpfilter:

ps|Ghostscript driver:\
        :lp=/dev/lpa0:sd=/var/spool/lpd/ps:lf=/var/log/lpd-errs:\
        :mx#0:sh:if=/usr/local/libexec/lpfilter-ps:
```

Option `mx#0` is very important for printing PostScript files because it eliminates size restrictions on the input file; PostScript documents tend to be very big. The `if` option points to the new filter. There is also a new spool directory.

The next steps are the creation of the new spool directory and of the filter program. The procedure for the spool directory is the same as above:

```
# cd /var/spool/lpd
# mkdir ps
# chown daemon:daemon ps
# chmod 770 ps
```

The filter program for PostScript output is more complex than the text base one: the file to be printed is fed to the interpreter which converts it into a sequence of commands in the printer's control language, and then sends that off to the printer. We have achieved to transform a cheap color printer in a device suitable for PostScript output, by virtue of the NetBSD operating system and some powerful freeware packages. The options used to configure Ghostscript are described in the Ghostscript documentation: `cdj550` is the device used to drive the HP printer.

Example 12-4. `/usr/local/libexec/lpfilter-ps`

```
#!/bin/sh
# Treat LF as CR+LF
printf "\033&k2G" || exit 2
# Print the postscript file
/usr/pkg/bin/gs -dSAFER -dBATCH -dQUIET -dNOPAUSE -q -sDEVICE=cdj550 \
-sOutputFile=- -sPAPERSIZE=a4 - && exit 0
exit 2
```

To summarize: two different printer names have been created on the system, which point to the same physical printer but use different options, different filters and different spool directories. Text files and

PostScript files can be printed. To print PostScript files the Ghostscript package must be installed on the system.

12.4 Printer management commands

This section lists some useful BSD commands for printer and print jobs administration. Besides the already mentioned **lpr** and **lpd** commands, we have:

lpq

examine the printer job queue.

lprm

delete jobs from the printer's queue.

lpc

check the printing system, enable/disable printers and printer features.

12.5 Remote printing

It is possible to configure the printing system in order to print on a printer connected to a remote host. Let's say that, for example, you work on the *wotan* host and you want to print on the printer connected to the *loge* host. The `/etc/printcap` file of loge is the one of Example 12-3. From wotan it will be possible to print Postscript files using Ghostscript on loge.

The first step is to accept the print jobs submitted from the wotan host to the loge host. To accomplish this, a line with the wotan host name must be added to the `/etc/hosts.lpd` file on loge:

```
# hostname
loge
# cat /etc/hosts.lpd
wotan
```

The format of this file is very simple: each line contains the name of a host which is permitted to print on the local system. By default the lpd daemon only listens on UNIX domain sockets for local connections, it won't accept any network connects. To ensure the daemon also accepts incoming network traffic, the following will need to be added to `/etc/rc.conf`:

```
lpd_flags=""
```

Next, the `/etc/printcap` file on wotan must be configured in order to send print jobs to loge. For example:

```
lp|line printer on loge:\
 :lp=:sd=/var/spool/lpd/lp:lf=/var/log/lp-errs:\
 :rm=loge:rp=lp

ps|Ghostscript driver on loge:\
 :lp=:sd=/var/spool/lpd/ps:lf=/var/log/lp-errs:\
```

```
:mx#0:\
:rm=loge:rp=ps
```

There are four main differences between this configuration and the one of Example 12-3.

1. The definition of “lp” is empty.
2. The “rm” (remote machine) entry defines the name of the host to which the printer is connected.
3. The “rp” (remote printer) entry defines the name of the printer connected to the remote host.
4. It is not necessary to specify input filters because the definitions on the loge host will be used.
5. The spool directories must still be created locally on wotan:

```
# cd /var/spool/lpd
# mkdir lp
# chown daemon:daemon lp
# chmod 770 lp
# mkdir ps
# chown daemon:daemon ps
# chmod 770 ps
```

Now the print jobs for the “lp” and “ps” queues on wotan will be sent automatically to the printer connected to loge.

Chapter 13
Using removable media

13.1 Initializing and using floppy disks

PC-style floppy disks work mostly like other disk devices like hard disks, except that you need to low-level format them first. To use an common 1440 KB floppy in the first floppy drive, first (as root) format it:

```
# fdformat -f /dev/rfd0a
```

Then create a single partition on the disk using disklabel(8):

```
# disklabel -rw /dev/rfd0a floppy3
```

Creating a small filesystem optimized for space:

```
# newfs -m 0 -o space -i 16384 -c 80 /dev/rfd0a
```

Now the floppy disk can be mounted like any other disk. Or if you already have a floppy disk with an MS-DOS filesystem on it that you just want to access from NetBSD, you can just do something like this:

```
# mount -t msdos /dev/fd0a /mnt
```

However, rather than using floppies like normal (bigger) disks, it is often more convenient to bypass the filesystem altogether and just splat an archive of files directly to the raw device. E.g.:

```
# tar cvfz /dev/rfd0a file1 file2 ...
```

A variation of this can also be done with MS-DOS floppies using the `sysutils/mtools` package which has the benefit of not going through the kernel buffer cache and thus not being exposed to the danger of the floppy being removed while a filesystem is mounted on it.

13.2 How to use a ZIP disk

1. See if your system has a ZIP drive:

   ```
   # dmesg | grep -i zip
   sd0 at atapibus0 drive 1: <IOMEGA  ZIP 100       ATAPI, , 14.A> type 0 direct removable
   ```

   Seems it has one, and it's recognized as sd0, just like any SCSI disk. The fact that the ZIP here is an ATAPI one doesn't matter - a SCSI ZIP will show up here, too. The ZIP is marked as "removable", which means you can eject it with:

```
# eject sd0
```

2. Insert ZIP disk
3. Check out what partitions are on the ZIP:

```
# disklabel sd0
# /dev/rsd0d:
type: ATAPI
 ...
8 partitions:
#        size   offset     fstype   [fsize bsize   cpg]
  d:   196608        0     unused        0     0         # (Cyl.    0 - 95)
  h:   196576       32      MSDOS                        # (Cyl.    0*- 95)
disklabel: boot block size 0
disklabel: super block size 0
```

   Partition d

   is the whole disk, as usual on i386.

   Partition h

   is what you want, and you can see it's a msdos filesystem even.

   Hence, use /dev/sd0h to access the zip's partition.

4. Mount it:

```
# mount -t msdos /dev/sd0h /mnt
```

5. Access your files:

```
# ls -la /mnt
total 40809
drwxr-xr-x   1 root  wheel     16384 Dec 31  1979 .
drwxr-xr-x  28 root  wheel      1024 Aug  2 22:06 ..
-rwxr-xr-x   1 root  wheel   1474560 Feb 23  1999 boot1.fs
-rwxr-xr-x   1 root  wheel   1474560 Feb 23  1999 boot2.fs
-rwxr-xr-x   1 root  wheel    548864 Feb 23  1999 boot3.fs
-rwxr-xr-x   1 root  wheel  38271173 Feb 23  1999 netbsd19990223.tar.gz
```

6. Unmount the ZIP:

```
# umount /mnt
#
```

7. Eject the ZIP:

```
# eject sd0
#
```

13.3 Reading data CDs with NetBSD

Data CDs can contain anything from programs, sound files (mp3, wav), movies (mp3, quicktime) to source code, text files, etc. Before accessing these files, a CD must be mounted on a directory, much like hard disks are. Just as hard disks can use different filesystems (ffs, lfs, ext2fs, ...), CDs have their own

filesystem, "cd9660". The NetBSD cd9660 filesystem can handle filesystems without and with Rockridge and Joliet extensions.

CD devices are named /dev/cd0a for both SCSI and IDE (ATAPI).

With this information, we can start:

1. See if your system has some CD drive:

   ```
   # dmesg | grep ^cd
       cd0 at atapibus0 drive 0: <CD-R/RW RW8040A, , 1.12> type 5 cdrom removable
       cd0: 32-bit data port
       cd0: drive supports PIO mode 4, DMA mode 0
       cd0(pciide0:1:0): using PIO mode 0, DMA mode 0 (using DMA data transfers)
   ```

   We have one drive here, "cd0". It is an IDE/ATAPI drive, as it is found on atapibus0. Of course the drive (rather, its medium) is removable, i.e., you can eject it. See below.

2. Insert a CD

3. Mount the CD manually:

   ```
   # mount -t cd9660 /dev/cd0a /mnt
   #
   ```

   This command shouldn't print anything. It instructs the system to mount the CD found on /dev/cd0a on /mnt, using the "cd9660" filesystem. The mountpoint "/mnt" must be an existing directory.

4. Check the contents of the CD:

   ```
   # ls /mnt
   INSTALL.html INSTALL.ps   TRANS.TBL    boot.catalog
   INSTALL.more INSTALL.txt  binary       installation
   #
   ```

   Everything looks fine! This is a NetBSD CD, of course. :)

5. Unmount the CD:

   ```
   # umount /mnt
   #
   ```

   If the CD is still accessed (e.g. some other shell's still "cd"'d into it), this will not work. If you shut down the system, the CD will be unmounted automatically for you, there's nothing to worry about there.

6. Making an entry in /etc/fstab:

   If you don't want to type the full "mount" command each time, you can put most of the values into a line in /etc/fstab:

   ```
   # Device        mountpoint      filesystem  mount options
   /dev/cd0a       /cdrom          cd9660      ro,noauto
   ```

   Make sure that the moutpoint, `/cdrom` in our example, exists:

   ```
   # mkdir /cdrom
   #
   ```

   Now you can mount the cd with the following command:

   ```
   # mount /cdrom
   ```

```
#
```

Access and unmount as before.

The CD is not mounted at boot time due to the "noauto" mount option - this is useful as you'll probably not have a CD in the drive all the time. See mount(8) and mount_cd9660(8) for some other useful options.

7. Eject the CD:

```
# eject cd0
#
```

If the CD is still mounted, it will be unmounted if possible, before being ejected.

13.4 Reading multi-session CDs with NetBSD

Use mscdlabel(8) to add all sessions to the CDs disklabel, and then use the appropriate device node to mount the session you want. You might have to create the corresponding device nodes in `/dev` manually. For example:

```
# mscdlabel cd1
track (ctl=4) at sector 142312
 adding as 'a'
track (ctl=4) at sector 0
 adding as 'b'
# ls -l /dev/cd1b
ls: /dev/cd1b: No such file or directory
# cd /dev
# ls -l cd1*
brw-r-----  1 root  operator       6,  8 Mar 18 21:55 cd1a
brw-r-----  1 root  operator       6, 11 Mar 18 21:55 cd1d
# mknod cd1b b 6 9
```

to create `/dev/cd1b`. Make sure you fix the permissions of any new device nodes you create:

```
# ls -l cd1*
brw-r-----  1 root  operator       6,  8 Mar 18 21:55 cd1a
brw-r--r--  1 root  wheel          6,  9 Mar 18 22:23 cd1b
brw-r-----  1 root  operator       6, 11 Mar 18 21:55 cd1d
# chgrp operator cd1b
# chmod 640 cd1b
# ls -l cd1*
brw-r-----  1 root  operator       6,  8 Mar 18 21:55 cd1a
brw-r-----  1 root  operator       6,  9 Mar 18 22:24 cd1b
brw-r-----  1 root  operator       6, 11 Mar 18 21:55 cd1d
```

Now you should be able to mount it.

```
# mount /dev/cd1b /mnt
```

13.5 Allowing normal users to access CDs

By default, NetBSD only allows "root" to mount a filesystem. If you want any user to be able to do this, perform the following steps:

- Give groups and other the access rights to the device.

  ```
  # chmod go+rw /dev/cd0a
  ```

- Ask NetBSD to let users mounting filesystems.

  ```
  # sysctl -w vfs.generic.usermount=1
  ```

  Note that this works for any filesystem and device, not only for CDs with a ISO 9660 filesystem.

To perform the mount operation after these commands, the user must own the mount point. So, for example:

```
$ cd $HOME
$ mkdir cdrom
$ mount -t cd9660 /dev/cd0a `pwd`/cdrom
```

Please also see mount(8) and as an alternative amd(8), for which example config files can be found in `/usr/share/examples/amd`.

13.6 Mounting an ISO image

Sometimes, it is interesting to mount an ISO9660 image file before you burn the CD; this way, you can examine its contents or even copy files to the outside. If you are a Linux user, you should know that this is done with the special *loop* filesystem. NetBSD does it another way, using the *vnode* pseudo-disk.

We will illustrate how to do this with an example. Suppose you have an ISO image in your home directory, called "mycd.iso":

1. Start by setting up a new vnode, "pointing" to the ISO file:

   ```
   # vnconfig -c vnd0 ~/mycd.iso
   ```

2. Now, mount the vnode:

   ```
   # mount -t cd9660 /dev/vnd0a /mnt
   ```

3. Yeah, image contents appear under `/mnt`! Go to that directory and explore the image.

4. When you are happy, you have to umount the image:

   ```
   # umount /mnt
   ```

5. And at last, deconfigure the vnode:

   ```
   # vnconfig -u vnd0
   ```

Note that these steps can also be used for any kind of file that contains a filesystem, not just ISO images.

See the vnd(4) and vnconfig(8) man pages for more information.

13.7 Using video CDs with NetBSD

To play MPEG Video streams as many DVD players can play them under NetBSD, mount the CD as you would do with any normal (data) CD (see Section 13.3), then use the `graphics/mtv`, `multimedia/xine-ui`, `multimedia/mplayer` or `multimedia/gmplayer` package to play the mpeg files stored on the CD.

13.8 Using audio CDs with NetBSD

There are two ways to handle audio CDs:

1. Tell the CD drive to play to the headphone or to a soundcard, to which CDROMs are usually connected internally. Use programs like cdplay(1), `audio/xmcd`, "kscd" from the `audio/kdemultimedia` package, mixer programs like mixerctl(1), `audio/xmix`, `audio/xmmix`, the Curses based `audio/cam`, or kmix, which is part of `audio/kdemultimedia`.

   This usually works well on both SCSI and IDE (ATAPI) CDROMs, CDRW and DVD drives.

2. To read ("rip") audio tracks in binary form without going through digital->analog conversion and back. There are several programs available to do this:

   - For most ATAPI, SCSI and several proprietary CDROM drives, the `audio/cdparanoia` package can be used. With cdparanoia the data can be saved to a file or directed to standard output in WAV, AIFF, AIFF-C or raw format. Currently the -g option is required by the NetBSD version of cdparanoia. A hypothetical example of how to save track 2 as a WAV file is as follows:

     ```
     $ cdparanoia -g /dev/rcd0d 2 track-02.wav
     ```

     If you want to grab all files from a CD, cdparanoia's batch mode is useful:

     ```
     $ cdparanoia -g /dev/rcd0d -B
     ```

   - For ATAPI or SCSI CD-ROMs the `audio/cdd` package can be used. To extract track 2 with cdd, type:

     ```
     # cdd -t 2 `pwd`
     ```

     This will put a file called `track-02.cda` in the current directory.

   - For SCSI CD-ROMS the `audio/tosha` package can be used. To extract track 2 with tosha, you should be able to type:

     ```
     # tosha -d CD-ROM-device -t 2 -o track-02.cda
     ```

   The data can then be post-processed e.g. by encoding it into MP3 streams (see Section 13.9) or by writing them to CD-Rs (see Section 13.11).

13.9 Creating an MP3 (MPEG layer 3) file from an audio CD

The basic steps in creating an MPEG layer 3 (MP3) file from an audio CD (using software from the NetBSD packages collection (http://www.NetBSD.org/Documentation/pkgsrc/)) are:

1. Extract (*rip*) the audio data of the CD as shown in Section 13.8.

2. Convert the CD audio format file to WAV format. You only need to perform this job if your ripping program (e.g. tosha, cdd) didn't already do the job for you!

   - Using the `audio/sox` package, type:

     ```
     $ sox -s -w -c 2 -r 44100 -t cdr track-02.cda track-02.wav
     ```

     This will convert `track-02.cda` in raw CD format to `track-02.wav` in WAV format, using **s**igned 16-bit **w**ords with 2 **c**hannels at a sampling **r**ate of 44100kHz.

3. Encode the WAV file into MP3 format.

   - Using the `audio/bladeenc` package, type:

     ```
     $ bladeenc -128 -QUIT track-02.wav
     ```

     This will encode `track-02.wav` into `track-02.mp3` in MP3 format, using a bit rate if **128**kBit/sec. The documentation for bladeenc describes bit-rates in more detail.

   - Using the `audio/lame` package, type:

     ```
     $ lame -p -o -v -V 5 -h track-02.wav track-02.mp3
     ```

     You may wish to use a lower quality, depending on your taste and hardware.

The resultant MP3 file can be played with any of the `audio/gqmpeg`, `audio/maplay`, `audio/mpg123` or `audio/splay` packages.

13.10 Using a CD-R writer with data CDs

The process of writing a CD consists of two steps: First, a "image" of the data must be generated, which can then be written to CD-R in a second step.

1. Reading an pre-existing ISO image

   ```
   # dd if=/dev/rcd0a of=filename.iso bs=2k
   #
   ```

   Alternatively, you can create a new ISO image yourself:

2. Generating the ISO image

   Put all the data you want to put on CD into one directory. Next you need to generate a disk-like ISO image of your data. The image stores the data in the same form as they're later put on CD, using the ISO 9660 format. The basic ISO9660 format only understands 8+3 filenames (max. eight letters for filename, plus three more for an extension). As this is not practical for Unix filenames, a so-called "Rockridge Extension" needs to be employed to get longer filenames. (A different set of such extension exists in the Microsoft world, to get their long filenames right; that's what's known as Joliet filesystem).

   The ISO image is created using the mkisofs command, which is part of the `sysutils/cdrecord` package.

   Example: if you have your data in /usr/tmp/data, you can generate a ISO image file in /usr/tmp/data.iso with the following command:

```
$ cd /usr/tmp
$ mkisofs -o data.iso -r data
Using NETBS000.GZ;1 for  data/binary/kernel/netbsd.INSTALL.gz (netbsd.INSTALL_TINY.gz)
Using NETBS001.GZ;1 for  data/binary/kernel/netbsd.GENERIC.gz (netbsd.GENERIC_TINY.gz)
  5.92% done, estimate finish Wed Sep 13 21:28:11 2000
 11.83% done, estimate finish Wed Sep 13 21:28:03 2000
 17.74% done, estimate finish Wed Sep 13 21:28:00 2000
 23.64% done, estimate finish Wed Sep 13 21:28:03 2000
 ...
 88.64% done, estimate finish Wed Sep 13 21:27:55 2000
 94.53% done, estimate finish Wed Sep 13 21:27:55 2000
Total translation table size: 0
Total rockridge attributes bytes: 5395
Total directory bytes: 16384
Path table size(bytes): 110
Max brk space used 153c4
84625 extents written (165 Mb)
$
```

Please see the mkisofs(8) man page for other options like noting publisher and preparer. The Bootable CD ROM How-To (../bootcd.html) explains how to generate a bootable CD.

3. Writing the ISO image to CD-R

   When you have the ISO image file, you just need to write it on a CD. This is done with the "cdrecord" command from the `sysutils/cdrecord` package. Insert a blank CD-R, and off we go:

   ```
   # cdrecord -v dev=/dev/rcd0d data.iso
   ...
   #
   ```

   After starting the command, 'cdrecord' shows you a lot of information about your drive, the disk and the image you're about to write. It then does a 10 seconds countdown, which is your last chance to stop things - type ^C if you want to abort. If you don't abort, the process will write the whole image to the CD and return with a shell prompt.

   Note that cdrecord(8) works on both SCSI and IDE (ATAPI) drives.

4. Test

   Mount the just-written CD and test it as you would do with any "normal" CD, see Section 13.3.

13.11 Using a CD-R writer to create audio CDs

If you want to make a backup copy of one of your audio CDs, you can do so by extracting ("ripping") the audio tracks from the CD, and then writing them back to a blank CD. Of course this also works fine if you only extract single tracks from various CDs, creating your very own mix CD!

The steps involved are:

1. Extract ("rip") the audio tracks as described as in Section 13.8 to get a couple of .wav files.
2. Write the .wav files using `sysutils/cdrecord`:

   ```
   # cdrecord -v dev=/dev/rcd0d -audio -pad *.wav
   ```

13.12 Creating an audio CD from mp3s

If you have converted all your audio CDs to mp3 and now want to make a mixed CD for your (e.g.) your car, you can do so by first converting the .mp3 files back to .wav format, then write them as a normal audio CD.

The steps involved here are:

1. Create .wav files from your .mp3 files:

   ```
   $ mpg123 -w foo.wav foo.mp3
   ```

   Do this for all of the mp3 files that you want to have on your audio CD. The .wav filenames you use don't matter.

2. Write the .wav files to CD as described under Section 13.11.

13.13 Copying an audio CD

To copy an audio CD while not introducing any pauses as mandated by the CDDA standard, you can use cdrdao for that:

```
# cdrdao read-cd --device /dev/rcd0d data.toc
# cdrdao write   --device /dev/rcd1d data.toc
```

13.14 Copying a data CD with two drives

If you have both a CD-R and a CD-ROM drive in your machine, you can copy a data CD with the following command:

```
# cdrecord dev=/dev/rcd1d /dev/rcd0d
```

Here the CD-ROM (cd0) contains the CD you want to copy, and the CD-R (cd1) contains the blank disk. Note that this only works with computer disks that contain some sort of data, it does *not* work with audio CDs! In practice you'll also want to add something like "*speed=8*" to make things a bit faster.

13.15 Using CD-RW rewritables

You can treat a CD-RW drive like a CD-R drive (see Section 13.10) in NetBSD, creating images with mkisofs(8) and writing them on a CD-RW medium with cdrecord(8).

If you want to blank a CD-RW, you can do this with cdrecord's "`blank`" option:

```
# cdrecord dev=/dev/rcd0d blank=fast
```

There are several other ways to blank the CD-RW, call cdrecord(8) with "*blank=help*" for a list. See the cdrecord(8) man page for more information.

13.16 DVD support

Currently, NetBSD supports DVD media through the ISO 9660 also used for CD-ROMs. The new UDF filesystem also present on DVDs is not supported, but almost all DVDs contain ISO 9660 filesystems as well. DVDs, DivX and many avi files be played with `multimedia/ogle` or `multimedia/gmplayer`.

For some hints on creating DVDs, see this postings about growisofs (http://mail-index.NetBSD.org/current-users/2004/01/06/0021.html) and this article about recording CDs and DVDs with NetBSD (http://www.mreriksson.net/blog/archive/15/).

13.17 Creating ISO images from a CD

To create an ISO image and save the checksum do this:

```
# readcd dev=/dev/cd0d f=/tmp/cd.iso
```

Here is an alternative using dd(1):

```
# dd if=/dev/cd0d of=/tmp/cd.iso bs=2048
```

If the CD has errors you can recover the rest with this:

```
# dd if=/dev/cd0d of=/tmp/cd.iso bs=2048 conv=noerror
```

To create an ISO image from a mounted data CD first, mount the CD disk by:

```
# mount -t cd9660 -r /dev/cd0d /mnt/cdrom
```

Second, get the image:

```
# mkhybrid -v -l -J -R -o /tmp/my_cd.iso /mnt/cdrom/
```

13.18 Getting volume information from CDs and ISO images

You can read the volume data from an unmounted CD with this command:

```
# file -s /dev/cd0d
```

You can read the volume data from an ISO image with this command:

```
# isoinfo -d -i /tmp/my_cd.iso
```

You can get the unique disk number from an unmounted CD with this:

```
# cd-discid /dev/cd0d
```

You can read the table of contents of an unmounted CD with this command:

```
# cdrecord -v dev=/dev/cd0d -toc
```

Chapter 14
The cryptographic device driver (CGD)

The `cgd` driver provides functionality which allows you to use disks or partitions for encrypted storage. After providing the appropriate key, the encrypted partition is accessible using `cgd` pseudo-devices.

14.1 Overview

People often store sensitive information on their hard disks and are concerned about this information falling into the wrong hands. This is particularly relevant to users of laptops and other portable devices, or portable media, which might be stolen or accidentally misplaced.

14.1.1 Why use disk encryption?

File-oriented encryption tools like GnuPG are great for encrypting individual files, which can then be sent across untrusted networks as well as stored encrypted on disk. But sometimes they can be inconvenient, because the file must be decrypted each time it is to be used; this is especially cumbersome when you have a large collection of files to protect. Any time a security tool is cumbersome to use, there's a chance you'll forget to use it properly, leaving the files unprotected for the sake of convenience.

Worse, readable copies of the encrypted contents might still exist on the hard disk. Even if you overwrite these files (using **rm -P**) before unlinking them, your application software might make temporary copies you don't know about, or have been paged to swapspace - and even your hard disk might have silently remapped failing sectors with data still in them.

The solution is to simply never write the information unencrypted to the hard disk. Rather than taking a file-oriented approach to encryption, consider a block-oriented approach - a virtual hard disk, that looks just like a normal hard disk with normal filesystems, but which encrypts and decrypts each block on the way to and from the real disk.

14.1.2 Logical Disk Drivers

The `cgd` device looks and behaves to the rest of the operating system like any other disk driver. Rather than driving real hardware directly, it provides a logical function layered on top of another block device. It has a special configuration program, **cgdconfig**, to create and configure a `cgd` device and point it at the underlying disk device that will hold the encrypted data.

NetBSD includes several other similar logical block devices, each of which provides some other function where `cgd` provides encryption. You can stack several of these logical block devices together: you can make an encrypted `raid` to protect your encrypted data against hard disk failure as well.

Once you have created a `cgd` disk, you can use **disklabel** to divide it up into partitions, **swapctl** to enable swapping to those partitions or **newfs** to make filesystems, then **mount** and use those filesystems, just like any other new disk.

14.1.3 Availability

The `cgd` driver was written by Roland C. Dowdeswell, and introduced in NetBSD-current between the 1.6 and 2.0 release branches. As a result, it is not in the 1.6 release series; it is in the 2.0 release.

14.2 Components of the Crypto-Graphic Disk system

A number of components and tools work together to make the `cgd` system effective.

14.2.1 Kernel driver pseudo-device

To use `cgd` you need a kernel with support for the `cgd` pseudo-device. Make sure the following line is in the kernel configuration file:

```
pseudo-device   cgd     4       # cryptographic disk driver
```

The number specifies how many `cgd` devices may be configured at the same time. After configuring the `cgd` pseudo-device you can recompile the kernel and boot it to enable `cgd` support.

14.2.2 Ciphers

The `cgd` driver provides the following encryption algorithms:

Encryption Methods

`aes-cbc`

AES (Rijndael). AES uses a 128 bit blocksize and accepts 128, 192 or 256 bit keys.

`blowfish-cbc`

Blowfish uses a 64 bit blocksize and accepts 128 bit keys

`3des-cbc`

Triple DES uses a 64 bit blocksize and accepts 192 bit keys (only 168 bits are actually used for encryption)

All three ciphers are used in CBC mode. This means each block is XORed with the previous encrypted block before encryption. This reduces the risk that a pattern can be found, which can be used to break the encryption.

14.2.3 Verification Methods

Another aspect of `cgd` that needs some attention are the verification methods **cgdconfig** provides. These verification methods are used to verify the passphrase is correct. The following verification methods are available:

Verification Methods

`none`

no verification is performed. This can be dangerous, because the key is not verified at all. When a wrong key is entered **cgdconfig** configures the `cgd` device as normal, but data which was available on the volume will be destroyed (decrypting blocks with a wrong key will result in random data, which will result in a regeneration of the disklabel with the current key).

`disklabel`

cgdconfig scans for a valid disklabel. If a valid disklabel is found with the key that is provided authentication will succeed.

`ffs`

cgdconfig scans for a valid FFS file system. If a valid FFS file system is found with the key that is provided authentication will succeed.

14.3 Example: encrypting your disk

This section works through a step-by-step example of converting an existing system to use `cgd`, performing the following actions:

1. Preparing the disk and partitions
2. Scrub off all data
3. Create the cgd
4. Adjust config-files
5. Restoring your backed-up files to the encrypted disk

14.3.1 Preparing the disk

First, decide which filesystems you want to move to an encrypted device. You're going to need to leave at least the small root (`/`) filesystem unencrypted, in order to load the kernel and run **init**, **cgdconfig** and the **rc.d** scripts that configure your `cgd`. In this example, we'll encrypt everything except the root (`/`) filesystem.

We are going to delete and re-make partitions and filesystems, and will require a backup to restore the data. So make sure you have a current, reliable backup stored on a different disk or machine. Do your backup in single-user mode, with the filesystems unmounted, to ensure you get a clean **dump**. Make sure you back up the disklabel of your hard disk as well, so you have a record of the partition layout before you started.

With the system at single user, / mounted read-write and everything else unmounted, use **disklabel** to delete all the data partitions you want to move into cgd.

Then make a single new partition in all the space you just freed up, say, *wd0e*. Set the partition type for this partition to ccd (there's no code specifically for cgd, but ccd is very similar. Though it doesn't really matter what it is, it will help remind you that it's not a normal filesystem later). When finished, label the disk to save the new partition table.

14.3.2 Scrubbing the disk

We have removed the partition table information, but the existing filesystems and data are still on disk. Even after we make a cgd device, create filesystems, and restore our data, some of these disk blocks might not yet be overwritten and still contain our data in plaintext. This is especially likely if the filesystems are mostly empty. We want to scrub the disk before we go further.

We could use **dd** to copy /dev/zero over the new *wd0e* partition, but this will leave our disk full of zeros, except where we've written encrypted data later. We might not want to give an attacker any clues about which blocks contain real data, and which are free space, so we want to write "noise" into all the disk blocks. So we'll create a temporary cgd, configured with a random, unknown key.

First, we configure a cgd to use a random key:

```
# cgdconfig -s cgd0 /dev/wd0e aes-cbc 128 < /dev/urandom
```

Now we can write zeros into the raw partition of our cgd (/dev/rcgd0d on NetBSD/i386, /dev/rcgd0c on most other platforms):

```
# dd if=/dev/zero of=/dev/rcgd0d bs=32k
```

The encrypted zeros will look like random data on disk. This might take a while if you have a large disk. Once finished, unconfigure the random-key cgd:

```
# cgdconfig -u cgd0
```

14.3.3 Creating the cgd

The **cgdconfig** program, which manipulates cgd devices, uses parameters files to store such information as the encryption type, key length, and a random password salt for each cgd. These files are very important, and need to be kept safe - without them, you will not be able to decrypt the data!

We'll generate a parameters file and write it into the default location (make sure the directory /etc/cgd exists and is mode 700):

```
# cgdconfig -g -V disklabel -o /etc/cgd/wd0e aes-cbc 256
```

This creates a parameters file /etc/cgd/wd0e describing a cgd using the aes-cbc cipher method, a key verification method of disklabel, and a key length of 256 bits. It will look something like this:

```
algorithm aes-cbc;
iv-method encblkno;
keylength 256;
```

```
verify_method disklabel;
keygen pkcs5_pbkdf2/sha1 {
        iterations 6275;
        salt AAAAgHTg/jKCd2ZJiOSGrgnadGw=;
};
```

Note: Remember, you'll want to save this file somewhere safe later.

Tip: When creating the parameters file, **cgdconfig** reads from `/dev/random` to create the password salt. This read may block if there is not enough collected entropy in the random pool. This is unlikely, especially if you just finished overwriting the disk as in the previous step, but if it happens you can press keys on the console and/or move your mouse until the `rnd` device gathers enough entropy.

Now it's time to create our `cgd`, for which we'll need a passphrase. This passphrase needs to be entered every time the `cgd` is opened, which is usually at each reboot. The encryption key is derived from this passphrase and the salt. Make sure you choose something you won't forget, and others won't guess.

The first time we configure the `cgd`, there is no valid disklabel on the logical device, so the validation mechanism we want to use later won't work. We override it this one time:

```
# cgdconfig -V re-enter cgd0 /dev/wd0e
```

This will prompt twice for a matching passphrase, just in case you make a typo, which would otherwise leave you with a `cgd` encrypted with a passphrase that's different to what you expected.

Now that we have a new `cgd`, we need to partition it and create filesystems. Recreate your previous partitions with all the same sizes, with the same letter names.

Tip: Remember to use the **disklabel -I** argument, because you're creating an initial label for a new disk.

Note: Although you want the sizes of your new partitions to be the same as the old, unencrypted ones, the offsets will be different because they're starting at the beginning of this virtual disk.

Then, use **newfs** to create filesystems on all the relevant partitions. This time your partitions will reflect the `cgd` disk names, for example:

```
# newfs /dev/rcgd0h
```

14.3.4 Modifying configuration files

We've moved several filesystems to another (logical) disk, and we need to update `/etc/fstab` accordingly. Each partition will have the same letter (in this example), but will be on `cgd0` rather than `wd0`. So you'll have `/etc/fstab` entries something like this:

```
/dev/wd0a   /     ffs     rw,softdep    1 1
/dev/cgd0b  none  swap    sw            0 0
/dev/cgd0b  /tmp  mfs     rw,-s=132m    0 0
/dev/cgd0e  /var  ffs     rw,softdep    1 2
/dev/cgd0f  /usr  ffs     rw,softdep    1 2
/dev/cgd0h  /home ffs     rw,softdep    1 2
```

> **Note:** `/tmp` should be a separate filesystem, either `mfs` or `ffs`, inside the `cgd`, so that your temporary files are not stored in plain text in the `/` filesystem.

Each time you reboot, you're going to need your `cgd` configured early, before **fsck** runs and filesystems are mounted.

Put the following line in `/etc/cgd/cgd.conf`:

```
cgd0    /dev/wd0e
```

This will use `/etc/cgd/wd0e` as config file for `cgd0`.

To finally enable cgd on each boot, put the following line into `/etc/rc.conf`:

```
cgd=YES
```

You should now be prompted for `/dev/cgd0`'s passphrase whenever `/etc/rc` starts.

14.3.5 Restoring data

Next, **mount** your new filesystems, and **restore** your data into them. It often helps to have `/tmp` mounted properly first, as **restore** can use a fair amount of temporary space when extracting a large dumpfile.

To test your changes to the boot configuration, **umount** the filesystems and unconfigure the `cgd`, so when you exit the single-user shell, **rc** will run like on a clean boot, prompting you for the passphrase and mounting your filesystems correctly. Now you can bring the system up to multi-user, and make sure everything works as before.

14.4 Example: encrypted CDs/DVDs

14.4.1 Introduction

This section explains how to create and use encrypted CDs/DVDs with NetBSD (all I say about "CDs" here does also apply to "DVDs"). I assume that you have basic knowledge of cgd(4), so I will not explain what cgd is or what's inside it in detail. The same applies to vnd(4). One can make use of encrypted CDs after reading this howto, but for more detailed information about different cgd configuration options, please read Chapter 14 or the manpages.

14.4.2 Creating an encrypted CD/DVD

cgd(4) provides highly secure encryption of whole partitions or disks. Unfortunately, creating "normal" CDs is not disklabling something and running newfs on it. Neither can you just put a CDR into the drive, configure cgd and assume it to write encrypted data when syncing. Standard CDs contain at least an ISO-9660 filesystem created with mkisofs(8) from the `sysutils/cdrecord` package. ISO images may *not* contain disklabels or cgd partitions.

But of course CD reader/writer hardware doesn't care about filesystems at all. You can write raw data to the CD if you like - or an encrypted FFS filesystem, which is what we'll do here. But be warned, there is NO way to read this CD with any OS except NetBSD - not even other BSDs due to the lack of cgd.

The basic steps when creating an encrypted CD are:

- Create an (empty) imagefile
- Register it as a virtual disk using vnd(4)
- Configure cgd inside the vnd disk
- Copy content to the cgd
- Unconfigure all (flush!)
- Write the image on a CD

The first step when creating an encrypted CD is to create a single image file with dd. The image may not grow, so make it large enough to allow all CD content to fit into. Note that the whole image gets written to the CD later, so creating a 700 MB image for 100 MB content will still require a 700 MB write operation to the CD. Some info on DVDs here: DVDs are only 4.7 GB in marketing language. 4.7GB = 4.7 x 1024 x 1024 x 1024 = 5046586573 bytes. In fact, a DVD can only approximately hold 4.7 x 1000 x 1000 x 1000 = 4700000000 bytes, which is about 4482 MB or about 4.37 GB. Keep this in mind when creating DVD images. Don't worry for CDs, they hold "real" 700 MB (734003200 Bytes).

Invoke all following commands as root!

For a CD:

```
# dd if=/dev/zero of=image.img bs=1m count=700
```

or, for a DVD:

```
# dd if=/dev/zero of=image.img bs=1m count=4482
```

Now configure a vnd(4)-pseudo disk with the image:

```
# vnconfig vnd0 image.img
```

In order to use cgd, a so-called parameter file, describing encryption parameters and a containing "password salt" must be generated. We'll call it `/etc/cgd/image` here. You can use one parameter file for several encrypted partitions (I use one different file for each host and a shared file `image` for all removable media, but that's up to you).

I'll use AES-CBC with a keylength of 256 bits. Refer to cgd(4) and cgdconfig(8) for details and alternatives.

The following command will create the parameter file as `/etc/cgd/image`. *YOU DO NOT WANT TO INVOKE THE FOLLOWING COMMAND AGAIN* after you burnt any CD, since a recreated parameter file is a lost parameter file and you'll never access your encrypted CD again (the "salt" this file contains will differ among each call). Consider this file being *HOLY, BACKUP IT* and *BACKUP IT AGAIN!* Use switch -V to specify verfication method "disklabel" for the CD (cgd cannot detect wheter you entered a valid password for the CD later when mounting it otherwise).

```
# cgdconfig -g -V disklabel aes-cbc 256 > /etc/cgd/image
```

Now it's time to configure a cgd for our vnd drive. (Replace slice "d" with "c" for all platforms that use "c" as the whole disk (where "**sysctl kern.rawpartition**" prints "2", not "3"); if you're on i386 or amd64, "d" is OK for you):

```
# cgdconfig -V re-enter cgd1 /dev/vnd0d /etc/cgd/image
```

The "`-V re-enter`" option is neccessary as long as the cgd doesn't have a disklabel yet so we can access and configure it. This switch asks for a password twice and uses it for encryption.

Now it's time to create a disklabel inside the cgd. The defaults of the label are ok, so invoking disklabel with

```
# disklabel -e -I cgd1
```

and leaving vi with "**:wq**" immediately will do.

Let's create a filesystem on the cgd, and finally mount it somewhere:

```
# newfs /dev/cgd1a
# mount /dev/cgd1a /mnt
```

The cgd is alive! Now fill `/mnt` with content. When finished, reverse the configuration process. The steps are:

1. Unmounting the cgd1a:

   ```
   # umount /mnt
   ```

2. Unconfiguring the cgd:

   ```
   # cgdconfig -u cgd1
   ```

3. Unconfiguring the vnd:

   ```
   # vnconfig -u vnd0
   ```

The following commands are examples to burn the images on CD or DVD. Please adjust the *`dev=`* for cdrecord or the `/dev/rcd0d` for growisofs. Note the "*`rcd0d`*" *is* necessary with NetBSD. Growisofs is available in the `sysutils/dvd+rw-tools` package. Again, use "`c`" instead of "`d`" if this is the raw partition on your platform.

Finally, write the image file to a CD:

```
# cdrecord dev=/dev/rcd0d -v image.img
```

...or to a DVD:

```
# growisofs -dvd-compat -Z /dev/rcd0d=image.img
```

Congratulations! You've just created a really secure CD!

14.4.3 Using an encrypted CD/DVD

After creating an encrypted CD as described above, we're not done yet - what about mounting it again? One might guess, configuring the cgd on `/dev/cd0d` is enough - no, it is not.

NetBSD cannot access FFS file systems on media that is not 512 bytes/sector format. It doesn't matter that the cgd on the CD is, since the CD's disklabel the cgd resides in has 2048 bytes/sector.

But the CD driver cd(4) is smart enough to grant "write" access to the (emulated) disklabel on the CD. So before configuring the cgd, let's have a look at the disklabel and modify it a bit:

```
# disklabel -e cd0
# /dev/rcd0d:
type: ATAPI
disk: mydisc
label: fictitious
flags: removable
bytes/sector: 2048    # -- Change to 512 (= orig / 4)
sectors/track: 100    # -- Change to 400 (= orig * 4)
tracks/cylinder: 1
sectors/cylinder: 100 # -- Change to 400 (= orig * 4)
cylinders: 164
total sectors: 16386  # -- Change to value of slice "d" (=65544)
rpm: 300
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

4 partitions:
#     size  offset  fstype [fsize bsize cpg/sgs]
 a:   65544   0     4.2BSD  0     0     0  # (Cyl. 0 - 655+)
 d:   65544   0     ISO9660 0     0        # (Cyl. 0 - 655+)
```

If you don't want to do these changes every time by hand, you can use Florian Stoehr's tool **neb-cd512** which is (at time of writing this) in pkgsrc-wip and will move to `sysutils/neb-cd512` soon. You can also download the neb-cd512 source from http://sourceforge.net/projects/neb-stoehr/ (http://sourceforge.net/projects/neb-stoehr/) (be sure to use neb-cd512, not neb-wipe!).

It is invoked with the disk name as parameter, by root:

```
# neb-cd512 cd0
```

Now as the disklabel is in 512 b/s format, accessing the CD is as easy as:

```
# cgdconfig cgd1 /dev/cd0d /etc/cgd/image
# mount -o ro /dev/cgd1a /mnt
```

Note that the cgd *MUST* be mounted read-only or you'll get illegal command errors from the cd(4) driver which can in some cases make even mounting a CD-based cgd impossible!

Now we're done! Enjoy your secure CD!

```
# ls /mnt
```

Remember you have to reverse all steps to remove the CD:

```
# umount /mnt
# cgdconfig -u cgd1
# eject cd0
```

14.5 Suggestions and Warnings

You now have your filesystems encrypted within a `cgd`. When your machine is shut down, the data is protected, and can't be decrypted without the passphrase. However, there are still some dangers you should be aware of, and more you can do with `cgd`. This section documents several further suggestions and warnings that will help you use `cgd` effectively.

- Use multiple `cgd`'s for different kinds of data, one mounted all the time and others mounted only when needed.
- Use a `cgd` configured on top of a `vnd` made from a file on a remote network fileserver (NFS, SMBFS, CODA, etc) to safely store private data on a shared system. This is similar to the procedure for using encrypted CDs and DVDs described in Section 14.4.

14.5.1 Using a random-key cgd for swap

You may want to use a dedicated random-key `cgd` for swap space, regenerating the key each reboot. The advantage of this is that once your machine is rebooted, any sensitive program memory contents that may have been paged out are permanently unrecoverable, because the decryption key is never known to you.

We created a temporary `cgd` with a random key when scrubbing the disk in the example above, using a shorthand **cgdconfig -s** invocation to avoid creating a parameters file.

The **cgdconfig** params file includes a "randomkey" keygen method. This is more appropriate for "permanent" random-key configurations, and facilitates the easy automatic configuration of these volumes at boot time.

For example, if you wanted to convert your existing `/dev/wd0b` partition to a dedicated random-key cgd1, use the following command to generate `/etc/cgd/wd0b`:

```
# cgdconfig -g -o /etc/cgd/wd0b -V none -k randomkey blowfish-cbc
```

When using the randomkey keygen method, only verification method "none" can be used, because the contents of the new `cgd` are effectively random each time (the previous data decrypted with a random key). Likewise, the new disk will not have a valid label or partitions, and **swapctl** will complain about configuring swap devices not marked as such in a disklabel.

In order to automate the process of labeling the disk, prepare an appropriate disklabel and save it to a file, for example `/etc/cgd/wd0b.disklabel`. Please refer to disklabel(8) for information about how to use **disklabel** to set up a swap partition.

On each reboot, to restore this saved label to the new `cgd`, create the `/etc/rc.conf.d/cgd` file as below:

```
swap_device="cgd0"
swap_disklabel="/etc/cgd/wd0b.disklabel"
start_postcmd="cgd_swap"

cgd_swap()
{
 if [ -f $swap_disklabel ]; then
  disklabel -R -r $swap_device $swap_disklabel
 fi
}
```

The same technique could be extended to encompass using **newfs** to re-create an `ffs` filesystem for `/tmp` if you didn't want to use `mfs`.

14.5.2 Warnings

Prevent cryptographic disasters by making sure you can always recover your passphrase and parameters file. Protect the parameters file from disclosure, perhaps by storing it on removable media as above, because the salt it contains helps protect against dictionary attacks on the passphrase.

Keeping the data encrypted on your disk is all very well, but what about other copies? You already have at least one other such copy (the backup we used during this setup), and it's not encrypted. Piping **dump** through file-based encryption tools like **gpg** can be one way of addressing this issue, but make sure you have all the keys and tools you need to decrypt it to **restore** after a disaster.

Like any form of software encryption, the `cgd` key stays in kernel memory while the device is configured, and may be accessible to privileged programs and users, such as `/dev/kmem` grovellers. Taking other system security steps, such as running with elevated securelevel, is highly recommended.

Once the `cgd` volumes are mounted as normal filesystems, their contents are accessible like any other file. Take care of file permissions and ensure your runnning system is protected against application and network security attack.

Avoid using suspend/resume, especially for laptops with a BIOS suspend-to-disk function. If an attacker can resume your laptop with the key still in memory, or read it from the suspend-to-disk memory image on the hard disk later, the whole point of using `cgd` is lost.

14.6 Further Reading

The following ressources contain more information on CGD:

Bibliography

NetBSD CGD Setup (http://www.s-mackie.com/notes/NetBSD-CGD-Setup.html), Stuart Mackie.

I want my cgd (http://www.nycbug.org/uploads/_netbsdcgd.html) aka: I want an encrypted psuedo-device on my laptop.

The original paper on The CryptoGraphic Disk Driver (http://www.imrryr.org/~elric/cgd/cgd.pdf), Roland Dowdeswell and John Ioannidis.

Chapter 15
Concatenated Disk Device (CCD) configuration

The CCD driver allows the user to "concatenate" several physical disks into one pseudo volume. While RAIDframe (see Chapter 16) also allows doing this to create RAID level 0 sets, it does not allow you to do striping across disks of different geometry, which is where CCD comes in handy. CCD also allows for an "interleave" to improve disk performance with a gained space loss. This example will not cover that feature.

The steps required to setup a CCD are as follows:

1. Install physical media
2. Configure kernel support
3. Disklabel each volume member of the CCD
4. Configure the CCD conf file
5. Initialize the CCD device
6. Create a filesystem on the new CCD device
7. Mount the CCD filesystem

This example features a CCD setup on NetBSD/sparc 1.5. The CCD will reside on 4 SCSI disks in a generic external Sun disk pack chassis connected to the external 50 pin SCSI port.

15.1 Install physical media

This step is at your own discretion, depending on your platform and the hardware at your disposal.

From my DMESG:

```
Disk #1:
  probe(esp0:0:0): max sync rate 10.00MB/s
  sd0 at scsibus0 target 0 lun 0: <SEAGATE, ST32430N SUN2.1G, 0444> SCSI2 0/direct fixed
  sd0: 2049 MB, 3992 cyl, 9 head, 116 sec, 512 bytes/sect x 4197405 sectors

Disk #2
  probe(esp0:1:0): max sync rate 10.00MB/s
  sd1 at scsibus0 target 1 lun 0: <SEAGATE, ST32430N SUN2.1G, 0444> SCSI2 0/direct fixed
  sd1: 2049 MB, 3992 cyl, 9 head, 116 sec, 512 bytes/sect x 4197405 sectors

Disk #3
  probe(esp0:2:0): max sync rate 10.00MB/s
```

```
  sd2 at scsibus0 target 2 lun 0: <SEAGATE, ST11200N SUN1.05, 9500> SCSI2 0/direct fixed
  sd2: 1005 MB, 1872 cyl, 15 head, 73 sec, 512 bytes/sect x 2059140 sectors

Disk #4
  probe(esp0:3:0): max sync rate 10.00MB/s
  sd3 at scsibus0 target 3 lun 0: <SEAGATE, ST11200N SUN1.05, 8808 > SCSI2 0
  sd3: 1005 MB, 1872 cyl, 15 head, 73 sec, 512 bytes/sect x 2059140 sectors
```

15.2 Configure Kernel Support

The following kernel configuration directive is needed to provide CCD device support. It is enabled in the GENERIC kernel:

```
pseudo-device  ccd  4    # concatenated disk devices
```

In my kernel config, I also hard code SCSI ID associations to `/dev` device entries to prevent bad things from happening:

```
sd0     at scsibus0 target 0 lun ?
# SCSI disk drives
sd1     at scsibus0 target 1 lun ?
# SCSI disk drives
sd2     at scsibus0 target 2 lun ?
# SCSI disk drives
sd3     at scsibus0 target 3 lun ?
# SCSI disk drives
sd4     at scsibus0 target 4 lun ?
# SCSI disk drives
sd5     at scsibus0 target 5 lun ?
# SCSI disk drives
sd6     at scsibus0 target 6 lun ?
# SCSI disk drives
```

15.3 Disklabel each volume member of the CCD

Each member disk of the CCD will need a special file system established. In this example, I will need to disklabel:

```
/dev/rsd0c
/dev/rsd1c
/dev/rsd2c
/dev/rsd3c
```

Note: Always remember to disklabel the character device, not the block device, in `/dev/r{s,w}d*`

Note: On all platforms, the `c` slice is symbolic of the entire NetBSD partition and is reserved.

You will probably want to remove any pre-existing disklabels on the disks in the CCD. This can be accomplished in one of two ways with the dd(1) command:

```
# dd if=/dev/zero of=/dev/rsd0c bs=8k count=1
# dd if=/dev/zero of=/dev/rsd1c bs=8k count=1
# dd if=/dev/zero of=/dev/rsd2c bs=8k count=1
# dd if=/dev/zero of=/dev/rsd3c bs=8k count=1
```

If your port uses a MBR (Master Boot Record) to partition the disks so that the NetBSD partitions are only part of the overall disk, and other OSs like Windows or Linux use other parts, you can void the MBR and all partitions on disk by using the command:

```
# dd if=/dev/zero of=/dev/rsd0d bs=8k count=1
# dd if=/dev/zero of=/dev/rsd1d bs=8k count=1
# dd if=/dev/zero of=/dev/rsd2d bs=8k count=1
# dd if=/dev/zero of=/dev/rsd3d bs=8k count=1
```

This will make all data on the entire disk inaccessible. Note that the entire disk is slice `d` on i386 (and some other ports), and `c` elsewhere (e.g. on sparc). See the “kern.rawpartition” sysctl - "3" means "d", "2" means "c".

The default disklabel for the disk will look similar to this:

```
# disklabel -r sd0
[...snip...]
bytes/sector: 512
sectors/track: 116
tracks/cylinder: 9
sectors/cylinder: 1044
cylinders: 3992
total sectors: 4197405
[..snip...]
3 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  c:  4197405        0     unused     1024  8192        # (Cyl.    0 - 4020*)
```

You will need to create one “slice” on the NetBSD partition of the disk that consumes the entire partition. The slice must begin at least one cylinder offset from the beginning of the disk/partition to provide space for the special CCD disklabel. The offset should be 1x sectors/cylinder (see following note). Therefore, the “size” value should be “total sectors” minus 1x “sectors/cylinder”. Edit your disklabel accordingly:

```
# disklabel -e sd0
```

Note: The offset of a slice of type “ccd” must be a multiple of the “sectors/cylinder” value.

Note: Be sure to **export EDITOR=[path to your favorite editor]** before editing the disklabels.

Note: The slice must be fstype `ccd`.

Because there will only be one slice on this partition, you can recycle the `c` slice (normally reserved for symbolic uses). Change your disklabel to the following:

```
3 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  c:  4196361     1044       ccd                     # (Cyl. 1 - 4020*)
```

Optionally you can setup a slice other than `c` to use, simply adjust accordingly below:

```
3 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:  4196361     1044       ccd                     # (Cyl. 1 - 4020*)
  c:  4197405        0     unused     1024  8192        # (Cyl. 0 - 4020*)
```

Be sure to write the label when you have completed. Disklabel will object to your disklabel and prompt you to re-edit if it does not pass its sanity checks.

15.4 Configure the CCD

Once all disks are properly labeled, you will need to generate a configuration file, `/etc/ccd.conf`. The file does not exist by default, and you will need to create a new one. The format is:

```
#ccd    ileave    flags    component    devices
```

Note: For the “ileave”, if a value of zero is used then the disks are concatenated, but if you use a value equal to the “sectors/track” number the disks are interleaved.

Example in this case:

```
# more /etc/ccd.conf
ccd0   0   none /dev/sd0c /dev/sd1c /dev/sd2c /dev/sd3c
```

Note: The CCD configuration file references the device file for the newly created CCD filesystems. Be sure not to reference the block device at this point; instead use the character device.

15.5 Initialize the CCD device

Once you are confident that your CCD configuration is sane, you can initialize the device using the ccdconfig(8) command: Configure:

```
# ccdconfig -c -f /etc/ccd.conf
```

Unconfigure:

```
# ccdconfig -u -f /etc/ccd.conf
```

Initializing the CCD device will activate `/dev` entries: `/dev/{,r}ccd#:`

```
# ls -la  /dev/{,r}ccd0*
brw-r-----  1 root  operator   9, 0 Apr 28 21:35 /dev/ccd0a
brw-r-----  1 root  operator   9, 1 Apr 28 21:35 /dev/ccd0b
brw-r-----  1 root  operator   9, 2 May 12 00:10 /dev/ccd0c
brw-r-----  1 root  operator   9, 3 Apr 28 21:35 /dev/ccd0d
brw-r-----  1 root  operator   9, 4 Apr 28 21:35 /dev/ccd0e
brw-r-----  1 root  operator   9, 5 Apr 28 21:35 /dev/ccd0f
brw-r-----  1 root  operator   9, 6 Apr 28 21:35 /dev/ccd0g
brw-r-----  1 root  operator   9, 7 Apr 28 21:35 /dev/ccd0h
crw-r-----  1 root  operator  23, 0 Jun 12 20:40 /dev/rccd0a
crw-r-----  1 root  operator  23, 1 Apr 28 21:35 /dev/rccd0b
crw-r-----  1 root  operator  23, 2 Jun 12 20:58 /dev/rccd0c
crw-r-----  1 root  operator  23, 3 Apr 28 21:35 /dev/rccd0d
crw-r-----  1 root  operator  23, 4 Apr 28 21:35 /dev/rccd0e
crw-r-----  1 root  operator  23, 5 Apr 28 21:35 /dev/rccd0f
crw-r-----  1 root  operator  23, 6 Apr 28 21:35 /dev/rccd0g
crw-r-----  1 root  operator  23, 7 Apr 28 21:35 /dev/rccd0h
```

15.6 Create a 4.2BSD/UFS filesystem on the new CCD device

You may now disklabel the new virtual disk device associated with your CCD:

```
# disklabel -e ccd0
```

Once again, there will be only one slice, so you may either recycle the `c` slice or create a separate slice for use.

```
# disklabel -r ccd0
# /dev/rccd0c:
type: ccd
disk: ccd
label: default label
flags:
bytes/sector: 512
sectors/track: 2048
tracks/cylinder: 1
sectors/cylinder: 2048
cylinders: 6107
total sectors: 12508812
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0
```

```
#        size    offset    fstype   [fsize bsize   cpg]
  c: 12508812         0    4.2BSD     1024  8192    16  # (Cyl. 0 - 6107*)
```

The filesystem will then need to be formatted:

```
# newfs /dev/rccd0c
Warning: 372 sector(s) in last cylinder unallocated
/dev/rccd0c:    12508812 sectors in 6108 cylinders of 1 tracks, 2048 sectors
        6107.8MB in 382 cyl groups (16 c/g, 16.00MB/g, 3968 i/g)

super-block backups (for fsck -b #) at:
[...]
```

15.7 Mount the filesystem

Once you have a created a file system on the CCD device, you can then mount the file system against a mount point on your system. Be sure to mount the slice labeled type `ffs` or `4.2BSD`:

```
# mount /dev/ccd0c /mnt
```

Then:

```
# export BLOCKSIZE=1024; df
Filesystem  1K-blocks     Used   Avail Capacity  Mounted on
/dev/sd6a      376155   320290   37057    89%    /
/dev/ccd0c    6058800        1  5755859     0%    /mnt
```

Congratulations, you now have a working CCD. To configure the CCD device at boot time, set `ccd=yes` in `/etc/rc.conf`. You can adjust `/etc/fstab` to get the filesystem mounted at boot:

```
/dev/ccd0c  /home ffs     rw,softdep    1 2
```

Chapter 16
NetBSD RAIDframe

16.1 RAIDframe Introduction

16.1.1 About RAIDframe

NetBSD uses the CMU RAIDframe (http://www.pdl.cmu.edu/RAIDframe/) subsystem. Although NetBSD is the primary platform for RAIDframe development, RAIDframe can also be found in OpenBSD and FreeBSD. NetBSD also has other in-kernel RAID systems called Vinum (http://www.vinumvm.org/) and Chapter 15, but they will not be discussed here. You should possess some basic knowledge (http://www.acnc.com/04_00.html) about RAID concepts and terminology before continuing. You
should also be at least familiar with the different levels of RAID - Adaptec provides an excellent reference (http://www.adaptec.com/worldwide/product/markeditorial.html?sess=no&prodkey=quick_explanation_of_raid), and the raid(4) manpage contains a short overview too.

16.1.2 A warning about Data Integrity, Backups, and High Availability

Firstly, because RAIDframe is a Software RAID implementation, as opposed to Hardware RAID, which needs special disk controllers some of which are supported by NetBSD System administrators should give a great deal of consideration to its implementation in “Mission Critical” applications. For such projects, you might consider the use of many of the hardware RAID devices supported by NetBSD (http://www.NetBSD.org/Hardware/). It is truly at your discretion what type of RAID you use, but I recommend you consider factors such as: manageability, commercial vendor support, load-balancing and failover, etc.

Secondly, depending on the RAID level used, RAIDframe does provide redundancy in the event of a hardware failure. However, it is *not* a replacement for reliable backups! Software and user-error can still cause data loss. RAIDframe may be used as a mechanism for facilitating backups in systems without backup hardware. Finally, with regard to "high availability", RAID is only a very small component to ensuring data availability.

Once more for good measure: *Back up your data!*

16.1.3 Getting Help

If you encounter problems using RAIDframe, you have several options for obtaining help.

1. Read the RAIDframe man pages: raid(4) and raidctl(8) thoroughly.

2. Search the mailing list archives. Unfortunately, there is no NetBSD list dedicated to RAIDframe support. Depending on the nature of the problem, posts tend to end up in a variety of lists. At a very minimum, search netbsd-help@NetBSD.org (http://mail-index.NetBSD.org/netbsd-help/), netbsd-users@NetBSD.org (http://mail-index.NetBSD.org/netbsd-users/), current-users@NetBSD.org (http://mail-index.NetBSD.org/current-users/). Also search the list for the NetBSD platform on which you are using RAIDframe: port-*${ARCH}*@NetBSD.org.

> **Caution**
>
> Because RAIDframe is constantly undergoing development, some information in mailing list archives has the potential of being dated and inaccurate.

3. Search the Problem Report database (http://www.NetBSD.org/Misc/send-pr.html).
4. If your problem persists: Post to the mailing list most appropriate (judgment call). Collect as much verbosely detailed information as possible before posting: Include your dmesg(8) output from `/var/run/dmesg.boot`, your kernel config(8) , your `/etc/raid[0-9].conf`, any relevant errors on `/dev/console`, `/var/log/messages`, or to `stdout/stderr` of raidctl(8) . Also include details on the troubleshooting steps you've taken thus far, exactly when the problem started, and any notes on recent changes that may have prompted the problem to develop. Remember to be patient when waiting for a response.

16.2 Setup RAIDframe Support

The use of RAID will require software and hardware configuration changes.

16.2.1 Kernel Support

Next we need to make sure we have RAID support in the kernel, which is the case for the GENERIC kernel. If you have already built a custom kernel for your environment, the kernel configuration must have the following options:

```
pseudo-device   raid            8       # RAIDframe disk driver
options         RAID_AUTOCONFIG         # auto-configuration of RAID components
```

The RAID support must be detected by the NetBSD kernel, which can be checked by looking at the output of the dmesg(8) command.

```
# dmesg|grep -i raid
Kernelized RAIDframe activated
```

Historically, the kernel must also contain static mappings between bus addresses and device nodes in `/dev`. This used to ensure consistency of devices within RAID sets in the event of a device failure after reboot. Since NetBSD 1.6, however, using the auto-configuration features of RAIDframe has been recommended over statically mapping devices. The auto-configuration features allow drives to move around on the system, and RAIDframe will automatically determine which components belong to which RAID sets.

16.2.2 Power Redundancy and Disk Caching

If your system has an Uninterruptible Power Supply (UPS), and/or if your system has redundant power supplies, you should consider enabling the read and write caches on your drives. On systems with redundant power, this will improve drive performance. On systems without redundant power, the write cache could endanger the integrity of RAID data in the event of a power loss.

For drives on the SCSI bus, you can use scsictl(8) utility:

```
# scsictl /dev/rsd0{c,d} getcache
/dev/rsd0d: no caches enabled
/dev/rsd0d: caching parameters are savable
# scsictl /dev/rsd0{c,d} setcache rw save
# scsictl /dev/rsd0{c,d} getcache
/dev/rsd0d: read cache enabled
/dev/rsd0d: write-back cache enabled
/dev/rsd0d: caching parameters are savable
```

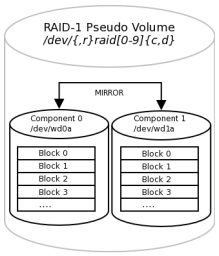
For drives on non-SCSI buses (EIDE, SATA, USB, IEEE1394), atactl(8) may be available or a virtual SCSI bus may be attached which should allow for access.

16.3 Example: RAID-1 Root Disk

This example explains how to setup RAID-1 root disk. With RAID-1 components are mirrored and therefore the server can be fully functional in the event of a single component failure. The goal is to provide a level of redundancy that will allow the system to encounter a component failure on either component disk in the RAID and:

- Continue normal operations until a maintenance window can be scheduled.
- Or, in the unlikely event that the component failure causes a system reboot, be able to quickly reconfigure the system to boot from the remaining component (platform dependant).

Figure 16-1. RAID-1 Disk Logical Layout



Because RAID-1 provides both redundancy and performance improvements, its most practical application is on critical "system" partitions such as `/`, `/usr`, `/var`, `swap`, etc., where read operations are more frequent than write operations. For other file systems, such as `/home` or `/var/{application}`, other RAID levels might be considered (see the references above). If one were simply creating a generic RAID-1 volume for a non-root file system, the cookie-cutter examples from the man page could be

followed, but because the root volume must be bootable, certain special steps must be taken during initial setup.

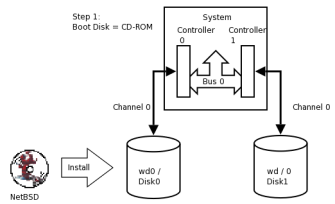
> **Note:** This example will outline a process that differs only slightly between the i386 and sparc64 platforms. In an attempt to reduce excessive duplication of content, where differences do exist and are cosmetic in nature, they will be pointed out using a section such as this. If the process is drastically different, the process will branch into separate, platform dependant steps.
>
> A bug in NetBSD 1.6.2 renders RAID-1 booting on sparc64 unusable, see this message (http://mail-index.NetBSD.org/port-sparc64/2004/06/22/0001.html). To use RAID-1 with full functionality on sparc64, please use NetBSD 2.0.

16.3.1 Pseudo-Process Outline

Although a much more refined process could be developed using a custom copy of NetBSD installed on custom-developed removable media, presently the NetBSD install media lacks RAIDframe tools and support, so the following pseudo process has become the de facto standard for setting up RAID-1 Root.

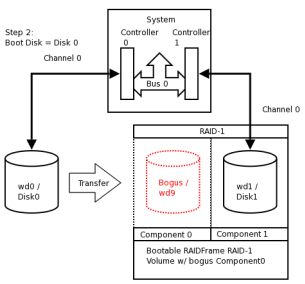
1. Install a stock NetBSD onto Disk0 of your system.

   Figure 16-2. Perform generic install onto Disk0/wd0

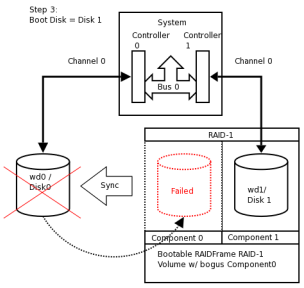
2. Use the installed system on Disk0/wd0 to setup a RAID Set composed of Disk1/wd1 only.

   Figure 16-3. Setup RAID Set

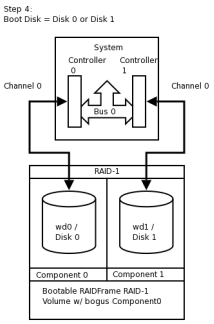
3. Reboot the system off the Disk1/wd1 with the newly created RAID volume.

Figure 16-4. Reboot using Disk1/wd1 of RAID



4. Add / re-sync Disk0/wd0 back into the RAID set.

Figure 16-5. Mirror Disk1/wd1 back to Disk0/wd0



16.3.2 Hardware Review

At present, the i386 and sparc64 NetBSD platforms support booting from RAID-1. Booting is not supported from any other RAID level. Booting from a RAID set is accomplished by teaching the 1st stage boot loader to understand both 4.2BSD/FFS and RAID partitions. The 1st boot block code only needs to know enough about the disk partitions and file systems to be able to read the 2nd stage boot blocks. Therefore, at any time, the system's BIOS / firmware must be able to read a drive with 1st stage boot blocks installed. On the i386 platform, configuring this is entirely dependant on the vendor of the controller card / host bus adapter to which your disks are connected. On sparc64 this is controlled by the IEEE 1275 Sun OpenBoot Firmware.

This article assumes two identical IDE disks (`/dev/wd{0,1}`) which we are going to mirror (RAID-1). These disks are identified as:

```
# grep ^wd /var/run/dmesg.boot
wd0 at atabus0 drive 0: <WDC WD100BB-75CLB0>
wd0: drive supports 16-sector PIO transfers, LBA addressing
wd0: 9541 MB, 19386 cyl, 16 head, 63 sec, 512 bytes/sect x 19541088 sectors
wd0: drive supports PIO mode 4, DMA mode 2, Ultra-DMA mode 5 (Ultra/100)
wd0(piixide0:0:0): using PIO mode 4, Ultra-DMA mode 2 (Ultra/33) (using DMA data transfers)

wd1 at atabus1 drive 0: <WDC WD100BB-75CLB0>
```

```
wd1: drive supports 16-sector PIO transfers, LBA addressing
wd1: 9541 MB, 19386 cyl, 16 head, 63 sec, 512 bytes/sect x 19541088 sectors
wd1: drive supports PIO mode 4, DMA mode 2, Ultra-DMA mode 5 (Ultra/100)
wd1(piixide0:1:0): using PIO mode 4, Ultra-DMA mode 2 (Ultra/33) (using DMA data transfers)
```

Note: If you are using SCSI, replace `/dev/{,r}wd{0,1}` with `/dev/{,r}sd{0,1}`

In this example, both disks are jumpered as Master on separate channels on the same controller. You would never want to have both disks on the same bus on the same controller; this creates a single point of failure. Ideally you would have the disks on separate channels on separate controllers. Some SCSI controllers have multiple channels on the same controller, however, a SCSI bus reset on one channel could adversely affect the other channel if the ASIC/IC becomes overloaded. The trade-off with two controllers is that twice the bandwidth is used on the system bus. For purposes of simplification, this example shows two disks on different channels on the same controller.

Note: RAIDframe requires that all components be of the same size. Actually, it will use the lowest common denominator among components of dissimilar sizes. For purposes of illustration, the example uses two disks of identical geometries. Also, consider the availability of replacement disks if a component suffers a critical hardware failure.

Tip: Two disks of identical vendor model numbers could have different geometries if the drive possesses "grown defects". Use a low-level program to examine the grown defects table of the disk. These disks are obviously suboptimal candidates for use in RAID and should be avoided.

16.3.3 Initial Install on Disk0/wd0

Perform a very generic installation onto your Disk0/wd0. Follow the INSTALL instructions for your platform. Install all the sets but do not bother customizing anything other than the kernel as it will be overwritten.

Tip: On i386, during the sysinst install, when prompted if you want to "use the entire disk for NetBSD", answer "yes".

- Chapter 3
- NetBSD/i386 Install Directions (ftp://ftp.NetBSD.org/pub/NetBSD/NetBSD-2.0/i386/INSTALL.html)
- NetBSD/sparc64 Install Directions (ftp://ftp.NetBSD.org/pub/NetBSD/NetBSD-2.0/sparc64/INSTALL.html)

Once the installation is complete, you should examine the disklabel(8) and fdisk(8) / sunlabel(8) outputs on the system:

```
# df
Filesystem  1K-blocks     Used     Avail Capacity  Mounted on
/dev/wd0a     9343708   191717   8684806     2%    /
```

On i386:

```
# disklabel -r wd0
type: unknown
disk: Disk00
label:
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 19386
total sectors: 19541088
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # microseconds
track-to-track seek: 0  # microseconds
drivedata: 0

16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19276992        63     4.2BSD   1024  8192 46568  # (Cyl.      0* - 19124*)
 b:    264033  19277055       swap                     # (Cyl.  19124* - 19385)
 c:  19541025        63     unused      0     0        # (Cyl.      0* - 19385)
 d:  19541088         0     unused      0     0        # (Cyl.      0 - 19385)

# fdisk /dev/rwd0d
Disk: /dev/rwd0d
NetBSD disklabel disk geometry:
cylinders: 19386, heads: 16, sectors/track: 63 (1008 sectors/cylinder)
total sectors: 19541088

BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 19541088

Partition table:
0: NetBSD (sysid 169)
    start 63, size 19541025 (9542 MB, Cyls 0-1216/96/1), Active
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
```

On Sparc64 the command / output differs slightly:

```
# disklabel -r wd0
type: unknown
disk: Disk0
[...snip...]
8 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19278000         0     4.2BSD   1024  8192 46568  # (Cyl.      0 -  19124)
 b:    263088  19278000       swap                     # (Cyl.  19125 -  19385)
 c:  19541088         0     unused      0     0        # (Cyl.      0 -  19385)

# sunlabel /dev/rwd0c
sunlabel> P
a: start cyl =      0, size = 19278000 (19125/0/0 - 9413.09Mb)
b: start cyl =  19125, size =   263088 (261/0/0 - 128.461Mb)
c: start cyl =      0, size = 19541088 (19386/0/0 - 9541.55Mb)
```

16.3.4 Preparing Disk1/wd1

Once you have a stock install of NetBSD on Disk0/wd0, you are ready to begin. Disk1/wd1 will be visible and unused by the system. To setup Disk1/wd1, you will use disklabel(8) to allocate the entire second disk to the RAID-1 set.

> **Tip:** The best way to ensure that Disk1/wd1 is completely empty is to 'zero' out the first few sectors of the disk with dd(1) . This will erase the MBR (i386) or Sun disk label (sparc64), as well as the NetBSD disk label. If you make a mistake at any point during the RAID setup process, you can always refer to this process to restore the disk to an empty state.

> **Note:** On sparc64, use `/dev/rwd1c` instead of `/dev/rwd1d`!

```
# dd if=/dev/zero of=/dev/rwd1d bs=8k count=1
1+0 records in
1+0 records out
8192 bytes transferred in 0.003 secs (2730666 bytes/sec)
```

Once this is complete, on i386, verify that both the MBR and NetBSD disk labels are gone. On sparc64, verify that the Sun Disk label is gone as well.

On i386:

```
# fdisk /dev/rwd1d

fdisk: primary partition table invalid, no magic in sector 0
Disk: /dev/rwd1d
NetBSD disklabel disk geometry:
cylinders: 19386, heads: 16, sectors/track: 63 (1008 sectors/cylinder)
total sectors: 19541088
```

```
BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 19541088

Partition table:
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.

# disklabel -r wd1

[...snip...]
16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 c:  19541025        63     unused      0     0        # (Cyl.      0* - 19385)
 d:  19541088         0     unused      0     0        # (Cyl.      0 - 19385)
```

On sparc64:

```
# sunlabel /dev/rwd1c

sunlabel: bogus label on '/dev/wd1c' (bad magic number)

# disklabel -r wd1

[...snip...]
3 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 c:  19541088         0     unused      0     0        # (Cyl.      0 -  19385)
disklabel: boot block size 0
disklabel: super block size 0
```

Now that you are certain the second disk is empty, on i386 you must establish the MBR on the second disk using the values obtained from Disk0/wd0 above. We must remember to mark the NetBSD partition active or the system will not boot. You must also create a NetBSD disklabel on Disk1/wd1 that will enable a RAID volume to exist upon it. On sparc64, you will need to simply disklabel(8) the second disk which will write the proper Sun Disk Label.

> **Tip:** disklabel(8) will use your shell' s environment variable `$EDITOR` variable to edit the disklabel. The default is vi(1)

On i386:

```
# fdisk -0ua /dev/rwd1d
fdisk: primary partition table invalid, no magic in sector 0
Disk: /dev/rwd1d
NetBSD disklabel disk geometry:
cylinders: 19386, heads: 16, sectors/track: 63 (1008 sectors/cylinder)
```

```
total sectors: 19541088

BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 19541088

Do you want to change our idea of what BIOS thinks? [n]

Partition 0:
<UNUSED>
The data for partition 0 is:
<UNUSED>
sysid: [0..255 default: 169]
start: [0..1216cyl default: 63, 0cyl, 0MB]
size: [0..1216cyl default: 19541025, 1216cyl, 9542MB]
bootmenu: []
Do you want to change the active partition? [n] y
Choosing 4 will make no partition active.
active partition: [0..4 default: 0] 0
Are you happy with this choice? [n] y

We haven't written the MBR back to disk yet.  This is your last chance.
Partition table:
0: NetBSD (sysid 169)
    start 63, size 19541025 (9542 MB, Cyls 0-1216/96/1), Active
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
Should we write new partition table? [n] y

# disklabel -r -e -I wd1
type: unknown
disk: Disk1
label:
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 19386
total sectors: 19541088
[...snip...]
16 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19541025        63       RAID                      # (Cyl.      0*-19385)
 c:  19541025        63     unused      0     0         # (Cyl.      0*-19385)
 d:  19541088         0     unused      0     0         # (Cyl.      0 -19385)
```

On sparc64:

```
# disklabel -r -e -I wd1
type: unknown
```

```
disk: Disk1
label:
flags:
bytes/sector: 512
sectors/track: 63
tracks/cylinder: 16
sectors/cylinder: 1008
cylinders: 19386
total sectors: 19541088
[...snip...]
3 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19541088         0       RAID                      # (Cyl.      0 -  19385)
 c:  19541088         0     unused      0     0         # (Cyl.      0 -  19385)

# sunlabel /dev/rwd1c
sunlabel> P
a: start cyl =      0, size = 19541088 (19386/0/0 - 9541.55Mb)
c: start cyl =      0, size = 19541088 (19386/0/0 - 9541.55Mb)
```

Note: On i386, the **c:** and **d:** slices are reserved. **c:** represents the NetBSD portion of the disk. **d:** represents the entire disk. Because we want to allocate the entire NetBSD MBR partition to RAID, and because **a:** resides within the bounds of **c:**, the **a:** and **c:** slices have same size and offset values and sizes. The offset must start at a track boundary (an increment of sectors matching the sectors/track value in the disk label). On sparc64 however, **c:** represents the entire NetBSD partition in the Sun disk label and **d:** is not reserved. Also note that sparc64's **c:** and **a:** require no offset from the beginning of the disk, however if they should need to be, the offset must start at a cylinder boundary (an increment of sectors matching the sectors/cylinder value).

16.3.5 Initializing the RAID Device

Next we create the configuration file for the RAID set / volume. Traditionally, RAIDframe configuration files belong in `/etc` and would be read and initialized at boot time, however, because we are creating a bootable RAID volume, the configuration data will actually be written into the RAID volume using the "auto-configure" feature. Therefore, files are needed only during the initial setup and should not reside in `/etc`.

```
# vi /var/tmp/raid0.conf
START array
1 2 0

START disks
/dev/wd9a
/dev/wd1a

START layout
128 1 1 1

START queue
```

```
fifo 100
```

Note that `wd9` is a non-existing disk. This will allow us to establish the RAID volume with a bogus component that we will substitute for Disk0/wd0 at a later time. Regardless, a device node in `/dev` for `wd9` must exist.

```
# cd /dev
# sh MAKEDEV wd9
# cd -
```

> **Tip:** On systems running NetBSD 2.0+, you may substitute a "bogus" component such as `/dev/wd9a` for a special disk name *"absent"*

Next we configure the RAID device and initialize the serial number to something unique. In this example we use a "YYYYMMDD*Revision*" scheme. The format you choose is entirely at your discretion, however the scheme you choose should ensure that no two RAID sets use the same serial number at the same time.

After that we initialize the RAID set for the first time, safely ignoring the errors regarding the bogus component.

```
# raidctl -v -C /var/tmp/raid0.conf raid0
raidlookup on device: /dev/wd9a failed!
raid0: Component /dev/wd9a being configured at col: 0
         Column: 0 Num Columns: 0
         Version: 0 Serial Number: 0 Mod Counter: 0
         Clean: No Status: 0
Number of columns do not match for: /dev/wd9a
/dev/wd9a is not clean!
raid0: Component /dev/wd1a being configured at col: 1
         Column: 0 Num Columns: 0
         Version: 0 Serial Number: 0 Mod Counter: 0
         Clean: No Status: 0
Column out of alignment for: /dev/wd1a
Number of columns do not match for: /dev/wd1a
/dev/wd1a is not clean!
raid0: There were fatal errors
raid0: Fatal errors being ignored.
raid0: RAID Level 1
raid0: Components: /dev/wd9a[**FAILED**] /dev/wd1a
raid0: Total Sectors: 19540864 (9541 MB)
# raidctl -v -I 2004082401 raid0
# raidctl -v -i raid0
Initiating re-write of parity
# tail -1 /var/log/messages
raid0: Error re-writing parity!
# raidctl -v -s raid0
Components:
           /dev/wd9a: failed
           /dev/wd1a: optimal
No spares.
```

```
/dev/wd9a status is: failed.  Skipping label.
Component label for /dev/wd1a:
   Row: 0, Column: 1, Num Rows: 1, Num Columns: 2
   Version: 2, Serial Number: 2004082401, Mod Counter: 7
   Clean: No, Status: 0
   sectPerSU: 128, SUsPerPU: 1, SUsPerRU: 1
   Queue size: 100, blocksize: 512, numBlocks: 19540864
   RAID Level: 1
   Autoconfig: No
   Root partition: No
   Last configured as: raid0
Parity status: DIRTY
Reconstruction is 100% complete.
Parity Re-write is 100% complete.
Copyback is 100% complete.
```

16.3.6 Setting up Filesystems

Caution

The root filesystem must begin at sector 0 of the RAID device. Else, the primary boot loader will be unable to find the secondary boot loader.

The RAID device is now configured and available. The RAID device is a pseudo disk-device. It will be created with a default disk label. You must now determine the proper sizes for disklabel slices for your production environment. For purposes of simplification in this example, our system will have 8.5 gigabytes dedicated to `/` as **/dev/raid0a** and the rest allocated to `swap` as **/dev/raid0b**.

Caution

This is an unrealistic disk layout for a production server; the NetBSD Guide can expand on proper partitioning technique. See Chapter 3

Note: Note that 1 GB is 2*1024*1024=2097152 blocks (1 block is 512 bytes, or 0.5 kilobytes). Despite what the underlying hardware composing a RAID set is, the RAID pseudo disk will always have 512 bytes/sector.

Note: In our example, the space allocated to the underlying `a:` slice composing the RAID set differed between i386 and sparc64, therefore the total sectors of the RAID volumes differs:

On i386:

```
 # disklabel -r -e -I raid0
type: RAID
disk: raid
```

```
label: fictitious
flags:
bytes/sector: 512
sectors/track: 128
tracks/cylinder: 8
sectors/cylinder: 1024
cylinders: 19082
total sectors: 19540864
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0 # microseconds
track-to-track seek: 0 # microseconds
drivedata: 0

#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19015680         0     4.2BSD      0     0     0  # (Cyl.      0 - 18569)
 b:    525184  19015680       swap                     # (Cyl.  18570 - 19082*)
 d:  19540864         0     unused      0     0        # (Cyl.      0 - 19082*)
```

On sparc64:

```
# disklabel -r -e -I raid0
[...snip...]
total sectors: 19539968
[...snip...]
3 partitions:
#        size    offset     fstype [fsize bsize cpg/sgs]
 a:  19251200         0     4.2BSD      0     0     0  # (Cyl.      0 -  18799)
 b:    288768  19251200       swap                     # (Cyl.  18800 -  19081)
 c:  19539968         0     unused      0     0        # (Cyl.      0 -  19081)
```

Next, format the newly created / partition as a 4.2BSD FFSv1 File System:

```
# newfs -O 1 /dev/raid0a
/dev/rraid0a: 9285.0MB (19015680 sectors) block size 16384, fragment size 2048
        using 51 cylinder groups of 182.06MB, 11652 blks, 22912 inodes.
super-block backups (for fsck -b #) at:
       32,   372896,   745760,  1118624,  1491488,  1864352,  2237216, 2610080,
  2982944,  3355808,  3728672,  4101536,  4474400,  4847264,  5220128, 5592992,
  5965856,  6338720,  6711584,  7084448,  7457312,  7830176,  8203040, 8575904,
  8948768,  9321632,  9694496, 10067360, 10440224, 10813088, 11185952,11558816,
 11931680, 12304544, 12677408, 13050272, 13423136, 13796000, 14168864,14541728,
 14914592, 15287456, 15660320, 16033184, 16406048, 16778912, 17151776,17524640,
 17897504, 18270368, 18643232,

# fsck -fy /dev/rraid0a
** /dev/rraid0a
** File system is already clean
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
```

```
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
1 files, 1 used, 4680062 free (14 frags, 585006 blocks, 0.0% fragmentation)
```

16.3.7 Setting up kernel dumps

The normal swap area in our case is on raid0b but this can not be used for crash dumps as process scheduling is stopped when dumps happen. Therefore we must use a real disk device. However, nothing stops us from defining a dump area which overlaps with raid0b. The trick here is to calculate the correct start offset for our crash dump area. This is dangerous and it is possible to destroy valuable data if we make a mistake in these calculations! Data corruption will happen when the kernel writes its memory dump over a normal filesystem. So we must be extra careful here. (The author destroyed his 100+ GB `/home` with a kernel crash dump!)

First we need to take a look at the disklabel for swap (raid0b) and the real physical disk (wd1).

On i386:

```
# disklabel raid0

8 partitions:
#        size    offset     fstype  [fsize bsize cpg/sgs]
 a:  19015680         0     4.2BSD   1024  8192    64
 b:    525184  19015680     swap
 d:  19540864         0     unused      0     0     0


# disklabel wd1

8 partitions:
#        size    offset     fstype  [fsize bsize cpg/sgs]
 a:  19541025        63     RAID
 c:  19541025        63     unused      0     0
 d:  19541088         0     unused      0     0
```

Each RAID set has a 64 block reserved area (see RF_PROTECTED_SECTORS in <dev/raidframe/raidframevar.h>) in the beginning of the set to store the internal RAID structures.

```
# dc
63              # offset of wd1a
64              # RF_PROTECTED_SECTORS
+
19015680        # offset of raid0b
+p
19015807        # offset of swap within wd1
q
```

We know now that real offset of the still-nonexisting wd1b is 19015807 and size is 525184. Next we need to add wd1b to wd1's disklabel.

```
# disklabel wd1 > disklabel.wd1
# vi disklabel.wd1

8 partitions:
#        size    offset     fstype  [fsize bsize cpg/sgs]
 a:  19541025        63       RAID
 b:   2097152  19015807       swap
 c:  19541025        63     unused      0     0
 d:  19541088         0     unused      0     0
```

Next we install the new disklabel.

```
# disklabel -R -r wd1 disklabel.wd1
```

Why isn't sizeof(raid0d) == (sizeof(wd1a) - RF_PROTECTED_SECTORS)? Size of raid0d is based on the largest multiple of the stripe size used for a RAID set. As an example, with stripe width of 128, size of raid0d is:

```
# dc
19541025    # size of wd1a
64          # RF_PROTECTED_SECTORS
-
128         # stripe width
/p
152663      # number of stripes
128         # number of blocks per stripe
*p
19540864    # size of raid0d
```

16.3.8 Migrating System to RAID

The new RAID filesystems are now ready for use. We mount them under `/mnt` and copy all files from the old system. This can be done using dump(8) or pax(1).

```
# mount /dev/raid0a /mnt
# df -h /mnt
Filesystem    Size     Used     Avail Capacity  Mounted on
/dev/raid0a   9.0G     2.0K      8.6G     0%    /mnt
# cd /; pax -v -X -rw -pe / /mnt
[...snip...]
```

The NetBSD install now exists on the RAID filesystem. We need to fix the mount-points in the new copy of `/etc/fstab` or the system will not come up correctly. Replace instances of `wd0` with `raid0`.

Note that the kernel crash dumps must not be saved on a RAID device but on a real physical disk (wd0b). This dump area was created in the previous chapter on the second disk (wd1b) but we will make wd0 an identical copy of wd1 later so wd0b and wd1b will have the same size and offset. If wd0 fails and is removed from the server wd1 becomes wd0 after reboot and crash dumps will still work as we are using wd0b in `/etc/fstab`. The only fault in this configuration is when the original, failed wd0 is replaces by

a new drive and we haven't initialized it yet with fdisk and disklabel. In this short period of time we can not make crash dumps in case of kernel panic. Note how the dump device has the "dp" keyword on the 4th field.

```
# vi /mnt/etc/fstab

/dev/raid0a  /  ffs  rw  1  1
/dev/raid0b  none  swap  sw  0  0
/dev/wd0b  none  swap  dw  0  0
kernfs    /kern  kernfs  rw
procfs    /proc  procfs  rw
```

The swap should be unconfigured upon shutdown to avoid parity errors on the RAID device. This can be done with a simple, one-line setting in `/etc/rc.conf`.

```
# vi /mnt/etc/rc.conf
swapoff=YES
```

Next the boot loader must be installed on Disk1/wd1. Failure to install the loader on Disk1/wd1 will render the system un-bootable if Disk0/wd0 fails making the RAID-1 pointless.

> **Tip:** Because the BIOS/CMOS menus in many i386 based systems are misleading with regard to device boot order. I highly recommend utilizing the "-o timeout=X" option supported by the i386 1st stage boot loader. Setup unique values for each disk as a point of reference so that you can easily determine from which disk the system is booting.

Caution

Although it may seem logical to install the 1st stage boot block into `/dev/rwd1{c,d}` (which is historically correct with NetBSD 1.6.x installboot(8) , this is no longer the case. If you make this mistake, the boot sector will become irrecoverably damaged and you will need to start the process over again.

On i386, install the boot loader into `/dev/rwd1a` :

```
# /usr/sbin/installboot -o timeout=30 -v /dev/rwd1a /usr/mdec/bootxx_ffsv1
File system:          /dev/rwd1a
File system type:     raw (blocksize 8192, needswap 1)
Primary bootstrap:    /usr/mdec/bootxx_ffsv1
Preserving 51 (0x33) bytes of the BPB
```

On sparc64, install the boot loader into `/dev/rwd1a` as well, however the "-o" flag is unsupported (and un-needed thanks to OpenBoot):

```
# /usr/sbin/installboot -v /dev/rwd1a /usr/mdec/bootblk
File system:          /dev/rwd1a
File system type:     raw (blocksize 8192, needswap 0)
Primary bootstrap:    /usr/mdec/bootblk
Bootstrap start sector: 1
Bootstrap byte count:   4915
```

```
Writing bootstrap
```

Finally the RAID set must be made auto-configurable and the system should be rebooted. After the reboot everything is mounted from the RAID devices.

```
# raidctl -v -A root raid0
raid0: Autoconfigure: Yes
raid0: Root: Yes
# tail -2 /var/log/messages
raid0: New autoconfig value is: 1
raid0: New rootpartition value is: 1
# raidctl -v -s raid0
[...snip...]
   Autoconfig: Yes
   Root partition: Yes
   Last configured as: raid0
[...snip...]
# shutdown -r now
```

> **Warning**
>
> Always use shutdown(8) when shutting down. Never simply use reboot(8). reboot(8) will not properly run shutdown RC scripts and will not safely disable swap. This will cause dirty parity at every reboot.

16.3.9 The first boot with RAID

At this point, temporarily configure your system to boot Disk1/wd1. See notes in Section 16.3.11 for details on this process. The system should boot now and all filesystems should be on the RAID devices. The RAID will be functional with a single component, however the set is not fully functional because the bogus drive (wd9) has failed.

```
# egrep -i "raid|root" /var/run/dmesg.boot
raid0: RAID Level 1
raid0: Components: component0[**FAILED**] /dev/wd1a
raid0: Total Sectors: 19540864 (9541 MB)
boot device: raid0
root on raid0a dumps on raid0b
root file system type: ffs

# df -h
Filesystem    Size     Used     Avail Capacity  Mounted on
/dev/raid0a   8.9G     196M      8.3G     2%    /
kernfs        1.0K     1.0K        0B   100%    /kern

# swapctl -l
Device      1K-blocks     Used    Avail Capacity  Priority
/dev/raid0b    262592        0   262592     0%    0
# raidctl -s raid0
Components:
```

```
          component0: failed
           /dev/wd1a: optimal
No spares.
component0 status is: failed.  Skipping label.
Component label for /dev/wd1a:
   Row: 0, Column: 1, Num Rows: 1, Num Columns: 2
   Version: 2, Serial Number: 2004082401, Mod Counter: 65
   Clean: No, Status: 0
   sectPerSU: 128, SUsPerPU: 1, SUsPerRU: 1
   Queue size: 100, blocksize: 512, numBlocks: 19540864
   RAID Level: 1
   Autoconfig: Yes
   Root partition: Yes
   Last configured as: raid0
Parity status: DIRTY
Reconstruction is 100% complete.
Parity Re-write is 100% complete.
Copyback is 100% complete.
```

16.3.10 Adding Disk0/wd0 to RAID

We will now add Disk0/wd0 as a component of the RAID. This will destroy the original file system structure. On i386, the MBR disklabel will be unaffected (remember we copied wd0's label to wd1 anyway) , therefore there is no need to "zero" Disk0/wd0. However, we need to relabel Disk0/wd0 to have an identical NetBSD disklabel layout as Disk1/wd1. Then we add Disk0/wd0 as "hot-spare" to the RAID set and initiate the parity reconstruction for all RAID devices, effectively bringing Disk0/wd0 into the RAID-1 set and "synching up" both disks.

```
# disklabel -r wd1 > /tmp/disklabel.wd1
# disklabel -R -r wd0 /tmp/disklabel.wd1
```

As a last-minute sanity check, you might want to use diff(1) to ensure that the disklabels of Disk0/wd0 match Disk1/wd1. You should also backup these files for reference in the event of an emergency.

```
# disklabel -r wd0 > /tmp/disklabel.wd0
# disklabel -r wd1 > /tmp/disklabel.wd1
# diff /tmp/disklabel.wd0 /tmp/disklabel.wd1
# fdisk /dev/rwd0 > /tmp/fdisk.wd0
# fdisk /dev/rwd1 > /tmp/fdisk.wd1
# diff /tmp/fdisk.wd0 /tmp/fdisk.wd1
# mkdir /root/RFbackup
# cp -p /tmp/{disklabel,fdisk}* /root/RFbackup
```

Once you are certain, add Disk0/wd0 as a spare component, and start reconstruction:

```
# raidctl -v -a /dev/wd0a raid0
/netbsd: Warning: truncating spare disk /dev/wd0a to 241254528 blocks
# raidctl -v -s raid0
Components:
          component0: failed
           /dev/wd1a: optimal
```

```
Spares:
           /dev/wd0a: spare
[...snip...]
# raidctl -F component0 raid0
RECON: initiating reconstruction on col 0 -> spare at col 2
 11% |****                                   | ETA:    04:26 \
```

Depending on the speed of your hardware, the reconstruction time will vary. You may wish to watch it on another terminal:

```
# raidctl -S raid0
Reconstruction is 0% complete.
Parity Re-write is 100% complete.
Copyback is 100% complete.
Reconstruction status:
  17% |******                                 | ETA: 03:08 -
```

After reconstruction, both disks should be “optimal”.

```
# tail -f /var/log/messages
raid0: Reconstruction of disk at col 0 completed
raid0: Recon time was 1290.625033 seconds, accumulated XOR time was 0 us (0.000000)
raid0:  (start time 1093407069 sec 145393 usec, end time 1093408359 sec 770426 usec)
raid0: Total head-sep stall count was 0
raid0: 305318 recon event waits, 1 recon delays
raid0: 1093407069060000 max exec ticks

# raidctl -v -s raid0
Components:
           component0: spared
           /dev/wd1a: optimal
Spares:
     /dev/wd0a: used_spare
     [...snip...]
```

When the reconstruction is finished we need to install the boot loader on the Disk0/wd0. On i386, install the boot loader into `/dev/rwd0a`:

```
# /usr/sbin/installboot -o timeout=15 -v /dev/rwd0a /usr/mdec/bootxx_ffsv1
File system:         /dev/rwd1a
File system type:    raw (blocksize 8192, needswap 1)
Primary bootstrap:   /usr/mdec/bootxx_ffsv1
Preserving 51 (0x33) bytes of the BPB
```

On sparc64:

```
# /usr/sbin/installboot -v /dev/rwd0a /usr/mdec/bootblk
File system:         /dev/rwd0a
File system type:    raw (blocksize 8192, needswap 0)
Primary bootstrap:   /usr/mdec/bootblk
Bootstrap start sector: 1
Bootstrap byte count:   4915
Writing bootstrap
```

And finally, reboot the machine one last time before proceeding. This is required to migrate Disk0/wd0 from status "used_spare" as "Component0" to state "optimal". Refer to notes in the next section regarding verification of clean parity after each reboot.

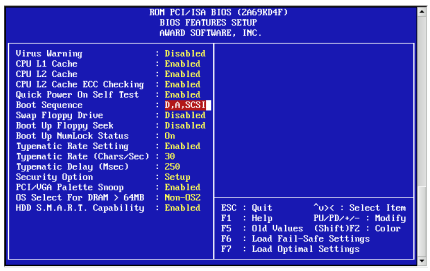
```
# shutdown -r now
```

16.3.11 Testing Boot Blocks

At this point, you need to ensure that your system's hardware can properly boot using the boot blocks on either disk. On i386, this is a hardware- dependant process that may be done via your motherboard CMOS/BIOS menu or your controller card's configuration menu.

On i386, use the menu system on your machine to set the boot device order / priority to Disk1/wd1 before Disk0/wd0. The examples here depict a generic Award BIOS.

Figure 16-6. Award BIOS i386 Boot Disk1/wd1

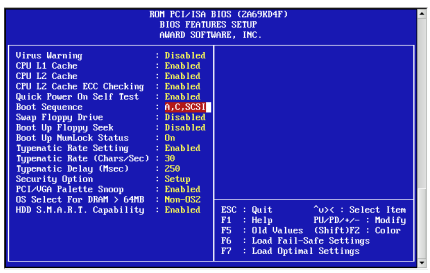


Save changes and exit.

```
>> NetBSD/i386 BIOS Boot, Revision 3.1
>> (seklecki@localhost, Fri Aug 13 08:08:47 EDT 2004)
>> Memory: 640/31744 k
Press return to boot now, any other key for boot menu
booting hd0a:netbsd - starting in 30
```

You can determine that the BIOS is reading Disk1/wd1 because the timeout of the boot loader is 30 seconds instead of 15. After the reboot, re-enter the BIOS and configure the drive boot order back to the default:

Figure 16-7. Award BIOS i386 Boot Disk0/wd0



Save changes and exit.

```
>> NetBSD/i386 BIOS Boot, Revision 3.1
>> (seklecki@localhost, Fri Aug 13 08:08:47 EDT 2004)
>> Memory: 640/31744 k
Press return to boot now, any other key for boot menu
booting hd0a:netbsd - starting in 15
```

Notice how your custom kernel detects controller/bus/drive assignments independent of what the BIOS assigns as the boot disk. This is the expected behavior.

On sparc64, use the Sun OpenBoot **devalias** to confirm that both disks are bootable:

```
Sun Ultra 5/10 UPA/PCI (UltraSPARC-IIi 400MHz), No Keyboard
OpenBoot 3.15, 128 MB memory installed, Serial #nnnnnnnn.
Ethernet address 8:0:20:a5:d1:3b, Host ID: nnnnnnnn.

ok devalias
[...snip...]
cdrom /pci@1f,0/pci@1,1/ide@3/cdrom@2,0:f
disk /pci@1f,0/pci@1,1/ide@3/disk@0,0
disk3 /pci@1f,0/pci@1,1/ide@3/disk@3,0
disk2 /pci@1f,0/pci@1,1/ide@3/disk@2,0
disk1 /pci@1f,0/pci@1,1/ide@3/disk@1,0
disk0 /pci@1f,0/pci@1,1/ide@3/disk@0,0
[...snip...]

ok boot disk0 netbsd
Initializing Memory [...]
Boot device /pci/pci/ide@3/disk@0,0 File and args: netbsd
NetBSD IEEE 1275 Bootblock
>> NetBSD/sparc64 OpenFirmware Boot, Revision 1.8
>> (lavalamp@j8, Thu Aug 19: 15:45:42 EDT 2004)
loadfile: reading header
elf64_exec: Booting [...]
symbols @ [....]
 Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001
     The NetBSD Foundation, Inc.  All rights reserved.
 Copyright (c) 1982, 1986, 1989, 1991, 1993
     The Regents of the University of California.  All rights reserved.
[...snip...]
```

And the second disk:

```
ok boot disk2 netbsd
Initializing Memory [...]
Boot device /pci/pci/ide@3/disk@2,0: File and args:netbsd
NetBSD IEEE 1275 Bootblock
>> NetBSD/sparc64 OpenFirmware Boot, Revision 1.8
>> (lavalamp@j8, Thu Aug 19: 15:45:42 EDT 2004)
loadfile: reading header
elf64_exec: Booting [...]
symbols @ [....]
 Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001
```

```
     The NetBSD Foundation, Inc.  All rights reserved.
 Copyright (c) 1982, 1986, 1989, 1991, 1993
     The Regents of the University of California.  All rights reserved.
[...snip...]
```

At each boot, the following should appear in the NetBSD kernel dmesg(8) :

```
raid0: RAID Level 1
raid0: Components: /dev/wd0a /dev/wd1a
raid0: Total Sectors: 19540864 (9541 MB)
boot device: raid0
root on raid0a dumps on raid0b
root file system type: ffs
```

Once you are certain that both disks are bootable, verify the RAID parity is clean after each reboot:

```
# raidctl -v -s raid0
Components:
          /dev/wd0a: optimal
          /dev/wd1a: optimal
No spares.
[...snip...]
Component label for /dev/wd0a:
   Row: 0, Column: 0, Num Rows: 1, Num Columns: 2
   Version: 2, Serial Number: 2004082401, Mod Counter: 67
   Clean: No, Status: 0
   sectPerSU: 128, SUsPerPU: 1, SUsPerRU: 1
   Queue size: 100, blocksize: 512, numBlocks: 19540864
   RAID Level: 1
   Autoconfig: Yes
   Root partition: Yes
   Last configured as: raid0
Component label for /dev/wd1a:
   Row: 0, Column: 1, Num Rows: 1, Num Columns: 2
   Version: 2, Serial Number: 2004082401, Mod Counter: 67
   Clean: No, Status: 0
   sectPerSU: 128, SUsPerPU: 1, SUsPerRU: 1
   Queue size: 100, blocksize: 512, numBlocks: 19540864
   RAID Level: 1
   Autoconfig: Yes
   Root partition: Yes
   Last configured as: raid0
Parity status: clean
Reconstruction is 100% complete.
Parity Re-write is 100% complete.
Copyback is 100% complete.
```

16.4 Testing kernel dumps

It is also important to test the kernel crash dumps so that they work correctly and do not overwrite any important filesystems (like the raid0e filesystem).

Press Ctrl+Alt+Esc to test the kernel crash dump. This will invoke the kernel debugger. Type **sync** or **reboot 0x104** and press Enter. This will save the current kernel memory to the dump area (wd0b) for further analysis. Most likely the offset and/or size of wd0b is wrong if the system will not come up correctly after reboot (unable to mount /home, corrupted super-blocks, etc). It is very important to test this now, not when we have lots of valuable files in `/home`. As an example, the author destroyed his 100+ GB `/home` partition with a kernel crash dump! No real harm was caused by this because of up-to-date backups (backup was made just before converting to RAID-1). One more time: take a backup of all your files before following these instructions!

Chapter 17
Pluggable Authentication Modules (PAM)

17.1 About

This article describes the underlying principles and mechanisms of the Pluggable Authentication Modules (PAM) library, and explains how to configure PAM, how to integrate PAM into applications, and how to write PAM modules.

17.2 Introduction

The Pluggable Authentication Modules (PAM) library is a generalized API for authentication-related services which allows a system administrator to add new authentication methods simply by installing new PAM modules, and to modify authentication policies by editing configuration files.

PAM was defined and developed in 1995 by Vipin Samar and Charlie Lai of Sun Microsystems, and has not changed much since. In 1997, the Open Group published the X/Open Single Sign-on (XSSO) preliminary specification, which standardized the PAM API and added extensions for single (or rather integrated) sign-on. At the time of this writing, this specification has not yet been adopted as a standard.

Although this article focuses primarily on FreeBSD 5.x and NetBSD 3.x, which both use OpenPAM, it should be equally applicable to FreeBSD 4.x, which uses Linux-PAM, and other operating systems such as Linux and Solaris™.

17.3 Terms and conventions

17.3.1 Definitions

The terminology surrounding PAM is rather confused. Neither Samar and Lai's original paper nor the XSSO specification made any attempt at formally defining terms for the various actors and entities involved in PAM, and the terms that they do use (but do not define) are sometimes misleading and ambiguous. The first attempt at establishing a consistent and unambiguous terminology was a whitepaper written by Andrew G. Morgan (author of Linux-PAM) in 1999. While Morgan's choice of terminology was a huge leap forward, it is in this author's opinion by no means perfect. What follows is an attempt, heavily inspired by Morgan, to define precise and unambiguous terms for all actors and entities involved in PAM.

account

The set of credentials the applicant is requesting from the arbitrator.

applicant

The user or entity requesting authentication.

arbitrator

The user or entity who has the privileges necessary to verify the applicant's credentials and the authority to grant or deny the request.

chain

A sequence of modules that will be invoked in response to a PAM request. The chain includes information about the order in which to invoke the modules, what arguments to pass to them, and how to interpret the results.

client

The application responsible for initiating an authentication request on behalf of the applicant and for obtaining the necessary authentication information from him.

facility

One of the four basic groups of functionality provided by PAM: authentication, account management, session management and authentication token update.

module

A collection of one or more related functions implementing a particular authentication facility, gathered into a single (normally dynamically loadable) binary file and identified by a single name.

policy

The complete set of configuration statements describing how to handle PAM requests for a particular service. A policy normally consists of four chains, one for each facility, though some services do not use all four facilities.

server

The application acting on behalf of the arbitrator to converse with the client, retrieve authentication information, verify the applicant's credentials and grant or deny requests.

service

A class of servers providing similar or related functionality and requiring similar authentication. PAM policies are defined on a per-service basis, so all servers that claim the same service name will be subject to the same policy.

session

The context within which service is rendered to the applicant by the server. One of PAM's four facilities, session management, is concerned exclusively with setting up and tearing down this context.

token

A chunk of information associated with the account, such as a password or passphrase, which the applicant must provide to prove his identity.

transaction

A sequence of requests from the same applicant to the same instance of the same server, beginning with authentication and session set-up and ending with session tear-down.

17.3.2 Usage examples

This section aims to illustrate the meanings of some of the terms defined above by way of a handful of simple examples.

17.3.2.1 Client and server are one

This simple example shows `alice` su(1)'ing to `root`.

```
$ whoami
alice
$ ls -l `which su`
-r-sr-xr-x  1 root  wheel  10744 Dec  6 19:06 /usr/bin/su
$ su -
Password: xi3kiune
# whoami
```

```
root
```

- The applicant is `alice`.
- The account is `root`.
- The su(1) process is both client and server.
- The authentication token is `xi3kiune`.
- The arbitrator is `root`, which is why su(1) is setuid `root`.

17.3.2.2 Client and server are separate

The example below shows `eve` try to initiate an ssh(1) connection to `login.example.com`, ask to log in as `bob`, and succeed. Bob should have chosen a better password!

```
$ whoami
eve
$ ssh bob@login.example.com
bob@login.example.com's password: god
Last login: Thu Oct 11 09:52:57 2001 from 192.168.0.1
NetBSD 3.0 (LOGIN) #1: Thu Mar 10 18:22:36 WET 2005

Welcome to NetBSD!
$
```

- The applicant is `eve`.
- The client is Eve's ssh(1) process.
- The server is the sshd(8) process on `login.example.com`
- The account is `bob`.
- The authentication token is `god`.
- Although this is not shown in this example, the arbitrator is `root`.

17.3.2.3 Sample policy

The following is FreeBSD's default policy for `sshd`:

```
sshd auth  required pam_nologin.so no_warn
sshd auth  required pam_unix.so no_warn try_first_pass
sshd account  required pam_login_access.so
sshd account  required pam_unix.so
sshd session  required pam_lastlog.so no_fail
sshd password required pam_permit.so
```

- This policy applies to the `sshd` service (which is not necessarily restricted to the sshd(8) server.)

- `auth`, `account`, `session` and `password` are facilities.
- `pam_nologin.so`, `pam_unix.so`, `pam_login_access.so`, `pam_lastlog.so` and `pam_permit.so` are modules. It is clear from this example that `pam_unix.so` provides at least two facilities (authentication and account management.)

There are some differences between FreeBSD and NetBSD PAM policies:

- By default, every configuration is done under `/etc/pam.d`.
- If configuration is non-existent, you will not have access to the system, in contrast with FreeBSD that has a default policy of allowing authentication.
- For authentication, NetBSD forces at least one `required`, `requisite` or `binding` module to be present.

17.4 PAM Essentials

17.4.1 Facilities and primitives

The PAM API offers six different authentication primitives grouped in four facilities, which are described below.

`auth`

Authentication. This facility concerns itself with authenticating the applicant and establishing the account credentials. It provides two primitives:

- pam_authenticate(3) authenticates the applicant, usually by requesting an authentication token and comparing it with a value stored in a database or obtained from an authentication server.
- pam_setcred(3) establishes account credentials such as user ID, group membership and resource limits.

`account`

Account management. This facility handles non-authentication-related issues of account availability, such as access restrictions based on the time of day or the server's work load. It provides a single primitive:

- pam_acct_mgmt(3) verifies that the requested account is available.

`session`

Session management. This facility handles tasks associated with session set-up and tear-down, such as login accounting. It provides two primitives:

- pam_open_session(3) performs tasks associated with session set-up: add an entry in the `utmp` and `wtmp` databases, start an SSH agent, etc.

- pam_close_session(3) performs tasks associated with session tear-down: add an entry in the `utmp` and `wtmp` databases, stop the SSH agent, etc.

`password`

Password management. This facility is used to change the authentication token associated with an account, either because it has expired or because the user wishes to change it. It provides a single primitive:

- pam_chauthtok(3) changes the authentication token, optionally verifying that it is sufficiently hard to guess, has not been used previously, etc.

17.4.2 Modules

Modules are a very central concept in PAM; after all, they are the "M" in "PAM". A PAM module is a self-contained piece of program code that implements the primitives in one or more facilities for one particular mechanism; possible mechanisms for the authentication facility, for instance, include the UNIX® password database, NIS, LDAP and Radius.

17.4.2.1 Module Naming

FreeBSD and NetBSD implement each mechanism in a single module, named `pam_`*`mechanism`*`.so` (for instance, `pam_unix.so` for the UNIX® mechanism.) Other implementations sometimes have separate modules for separate facilities, and include the facility name as well as the mechanism name in the module name. To name one example, Solaris™ has a `pam_dial_auth.so.1` module which is commonly used to authenticate dialup users. Also, almost every module has a man page with the same name, i.e.: pam_unix(8) explains how the `pam_unix.so` module works.

17.4.2.2 Module Versioning

FreeBSD's original PAM implementation, based on Linux-PAM, did not use version numbers for PAM modules. This would commonly cause problems with legacy applications, which might be linked against older versions of the system libraries, as there was no way to load a matching version of the required modules.

OpenPAM, on the other hand, looks for modules that have the same version number as the PAM library (currently 2 in FreeBSD and 0 in NetBSD), and only falls back to an unversioned module if no versioned module could be loaded. Thus legacy modules can be provided for legacy applications, while allowing new (or newly built) applications to take advantage of the most recent modules.

Although Solaris™ PAM modules commonly have a version number, they're not truly versioned, because the number is a part of the module name and must be included in the configuration.

17.4.2.3 Module Path

There isn't a common directory for storing PAM modules. Under FreeBSD, they are located at `/usr/lib` and, under NetBSD, you can find them in `/usr/lib/security`.

17.4.3 Chains and policies

When a server initiates a PAM transaction, the PAM library tries to load a policy for the service specified in the pam_start(3) call. The policy specifies how authentication requests should be processed, and is defined in a configuration file. This is the other central concept in PAM: the possibility for the admin to tune the system security policy (in the wider sense of the word) simply by editing a text file.

A policy consists of four chains, one for each of the four PAM facilities. Each chain is a sequence of configuration statements, each specifying a module to invoke, some (optional) parameters to pass to the module, and a control flag that describes how to interpret the return code from the module.

Understanding the control flags is essential to understanding PAM configuration files. There are a number of different control flags:

`binding`

If the module succeeds and no earlier module in the chain has failed, the chain is immediately terminated and the request is granted. If the module fails, the rest of the chain is executed, but the request is ultimately denied.

This control flag was introduced by Sun in Solaris™ 9 (SunOS™ 5.9), and is also supported by OpenPAM.

`required`

If the module succeeds, the rest of the chain is executed, and the request is granted unless some other module fails. If the module fails, the rest of the chain is also executed, but the request is ultimately denied.

`requisite`

If the module succeeds, the rest of the chain is executed, and the request is granted unless some other module fails. If the module fails, the chain is immediately terminated and the request is denied.

`sufficient`

If the module succeeds and no earlier module in the chain has failed, the chain is immediately terminated and the request is granted. If the module fails, the module is ignored and the rest of the chain is executed.

As the semantics of this flag may be somewhat confusing, especially when it is used for the last module in a chain, it is recommended that the `binding` control flag be used instead if the implementation supports it.

`optional`

The module is executed, but its result is ignored. If all modules in a chain are marked `optional`, all requests will always be granted.

When a server invokes one of the six PAM primitives, PAM retrieves the chain for the facility the primitive belongs to, and invokes each of the modules listed in the chain, in the order they are listed, until it reaches the end, or determines that no further processing is necessary (either because a `binding` or `sufficient` module succeeded, or because a `requisite` module failed.) The request is granted if and only if at least one module was invoked, and all non-optional modules succeeded.

Note that it is possible, though not very common, to have the same module listed several times in the same chain. For instance, a module that looks up user names and passwords in a directory server could be invoked multiple times with different parameters specifying different directory servers to contact. PAM treat different occurrences of the same module in the same chain as different, unrelated modules.

17.4.4 Transactions

The lifecycle of a typical PAM transaction is described below. Note that if any of these steps fails, the server should report a suitable error message to the client and abort the transaction.

1. If necessary, the server obtains arbitrator credentials through a mechanism independent of PAM—most commonly by virtue of having been started by `root`, or of being setuid `root`.
2. The server calls pam_start(3) to initialize the PAM library and specify its service name and the target account, and register a suitable conversation function.
3. The server obtains various information relating to the transaction (such as the applicant's user name and the name of the host the client runs on) and submits it to PAM using pam_set_item(3).
4. The server calls pam_authenticate(3) to authenticate the applicant.
5. The server calls pam_acct_mgmt(3) to verify that the requested account is available and valid. If the password is correct but has expired, pam_acct_mgmt(3) will return `PAM_NEW_AUTHTOK_REQD` instead of `PAM_SUCCESS`.
6. If the previous step returned `PAM_NEW_AUTHTOK_REQD`, the server now calls pam_chauthtok(3) to force the client to change the authentication token for the requested account.
7. Now that the applicant has been properly authenticated, the server calls pam_setcred(3) to establish the credentials of the requested account. It is able to do this because it acts on behalf of the arbitrator, and holds the arbitrator's credentials.
8. Once the correct credentials have been established, the server calls pam_open_session(3) to set up the session.
9. The server now performs whatever service the client requested—for instance, provide the applicant with a shell.
10. Once the server is done serving the client, it calls pam_close_session(3) to tear down the session.
11. Finally, the server calls pam_end(3) to notify the PAM library that it is done and that it can release whatever resources it has allocated in the course of the transaction.

17.5 PAM Configuration

17.5.1 PAM policy files

17.5.1.1 The `/etc/pam.conf` file

The traditional PAM policy file is `/etc/pam.conf`. This file contains all the PAM policies for your system. Each line of the file describes one step in a chain, as shown below:

```
login   auth    required        pam_nologin.so  no_warn
```

The fields are, in order: service name, facility name, control flag, module name, and module arguments. Any additional fields are interpreted as additional module arguments.

A separate chain is constructed for each service / facility pair, so while the order in which lines for the same service and facility appear is significant, the order in which the individual services and facilities are listed is not. The examples in the original PAM paper grouped configuration lines by facility, and the Solaris™ stock `pam.conf` still does that, but FreeBSD's stock configuration groups configuration lines by service. Either way is fine; either way makes equal sense.

17.5.1.2 The `/etc/pam.d` directory

OpenPAM and Linux-PAM support an alternate configuration mechanism, which is the preferred mechanism in FreeBSD and NetBSD. In this scheme, each policy is contained in a separate file bearing the name of the service it applies to. These files are stored in `/etc/pam.d/`.

These per-service policy files have only four fields instead of `pam.conf`'s five: the service name field is omitted. Thus, instead of the sample `pam.conf` line from the previous section, one would have the following line in `/etc/pam.d/login`:

```
auth    required        pam_nologin.so  no_warn
```

As a consequence of this simplified syntax, it is possible to use the same policy for multiple services by linking each service name to a same policy file. For instance, to use the same policy for the `su` and `sudo` services, one could do as follows:

```
# cd /etc/pam.d
# ln -s su sudo
```

This works because the service name is determined from the file name rather than specified in the policy file, so the same file can be used for multiple differently-named services.

Since each service's policy is stored in a separate file, the `pam.d` mechanism also makes it very easy to install additional policies for third-party software packages.

17.5.1.3 The policy search order

As we have seen above, PAM policies can be found in a number of places. If no configuration file is found for a particular service, the `/etc/pam.d/other` is used instead. If that file does not exist, `/etc/pam.conf` is searched for entries matching he specified service or, failing that, the "other" service.

It is essential to understand that PAM's configuration system is centered on chains.

17.5.2 Breakdown of a configuration line

As explained in the *PAM policy files* section, each line in `/etc/pam.conf` consists of four or more fields: the service name, the facility name, the control flag, the module name, and zero or more module arguments.

The service name is generally (though not always) the name of the application the statement applies to. If you are unsure, refer to the individual application's documentation to determine what service name it uses.

Note that if you use `/etc/pam.d/` instead of `/etc/pam.conf`, the service name is specified by the name of the policy file, and omitted from the actual configuration lines, which then start with the facility name.

The facility is one of the four facility keywords described in the *Facilities and primitives* section.

Likewise, the control flag is one of the four keywords described in the *Chains and policies* section, describing how to interpret the return code from the module. Linux-PAM supports an alternate syntax that lets you specify the action to associate with each possible return code, but this should be avoided as it is non-standard and closely tied in with the way Linux-PAM dispatches service calls (which differs greatly from the way Solaris™ and OpenPAM do it.) Unsurprisingly, OpenPAM does not support this syntax.

17.5.3 Policies

To configure PAM correctly, it is essential to understand how policies are interpreted.

When an application calls pam_start(3), the PAM library loads the policy for the specified service and constructs four module chains (one for each facility.) If one or more of these chains are empty, the corresponding chains from the policy for the `other` service are substituted.

When the application later calls one of the six PAM primitives, the PAM library retrieves the chain for the corresponding facility and calls the appropriate service function in each module listed in the chain, in the order in which they were listed in the configuration. After each call to a service function, the module type and the error code returned by the service function are used to determine what happens next. With a few exceptions, which we discuss below, the following table applies:

Table 17-1. PAM chain execution summary

	PAM_SUCCESS	**PAM_IGNORE**	**other**
binding	if (!fail) break;	-	fail = true;
required	-	-	fail = true;
requisite	-	-	fail = true; break;
sufficient	if (!fail) break;	-	-
optional	-	-	-

If `fail` is true at the end of a chain, or when a "break" is reached, the dispatcher returns the error code returned by the first module that failed. Otherwise, it returns `PAM_SUCCESS`.

The first exception of note is that the error code `PAM_NEW_AUTHTOK_REQD` is treated like a success, except that if no module failed, and at least one module returned `PAM_NEW_AUTHTOK_REQD`, the dispatcher will return `PAM_NEW_AUTHTOK_REQD`.

The second exception is that pam_setcred(3) treats `binding` and `sufficient` modules as if they were `required`.

The third and final exception is that pam_chauthtok(3) runs the entire chain twice (once for preliminary

checks and once to actually set the password), and in the preliminary phase it treats `binding` and `sufficient` modules as if they were `required`.

17.6 PAM modules

17.6.1 Common Modules

17.6.1.1 pam_deny(8)

The pam_deny(8) module is one of the simplest modules available; it responds to any request with `PAM_AUTH_ERR`. It is useful for quickly disabling a service (add it to the top of every chain), or for terminating chains of `sufficient` modules.

17.6.1.2 pam_echo(8)

The pam_echo(8) module simply passes its arguments to the conversation function as a `PAM_TEXT_INFO` message. It is mostly useful for debugging, but can also serve to display messages such as “Unauthorized access will be prosecuted” before starting the authentication procedure.

17.6.1.3 pam_exec(8)

The pam_exec(8) module takes its first argument to be the name of a program to execute, and the remaining arguments are passed to that program as command-line arguments. One possible application is to use it to run a program at login time which mounts the user’s home directory.

17.6.1.4 pam_ftpusers(8)

The pam_ftpusers(8) module successes if and only if the user is listed in `/etc/ftpusers`. Currently, in NetBSD, this module doesn’t understand the extended syntax of ftpd(8), but this will be fixed in the future.

17.6.1.5 pam_group(8)

The pam_group(8) module accepts or rejects applicants on the basis of their membership in a particular file group (normally `wheel` for su(1)). It is primarily intended for maintaining the traditional behaviour of BSD su(1), but has many other uses, such as excluding certain groups of users from a particular service.

In NetBSD, there is an argument called `authenticate` in which the user is asked to authenticate using his own password.

17.6.1.6 pam_guest(8)

The pam_guest(8) module allows guest logins using fixed login names. Various requirements can be placed on the password, but the default behaviour is to allow any password as long as the login name is that of a guest account. The pam_guest(8) module can easily be used to implement anonymous FTP logins.

17.6.1.7 pam_krb5(8)

The pam_krb5(8) module provides functions to verify the identity of a user and to set user specific credentials using Kerberos 5. It prompts the user for a password and obtains a new Kerberos TGT for the principal. The TGT is verified by obtaining a service ticket for the local host. The newly acquired credentials are stored in a credential cache and the environment variable KRB5CCNAME is set appropriately. The credentials cache should be destroyed by the user at logout with kdestroy(1).

17.6.1.8 pam_ksu(8)

The pam_ksu(8) module provides only authentication serivices for Kerberos 5 to determine whether or not the applicant is authorized to obtain the privileges of the target account.

17.6.1.9 pam_lastlog(8)

The pam_lastlog(8) module provides only session management services. It records the session in utmp(5), utmpx(5), wtmp(5), wtmpx(5), lastlog(5) and lastlogx(5) databases.

17.6.1.10 pam_login_access(8)

The pam_login_access(8) module provides an implementation of the account management primitive which enforces the login restrictions specified in the login.access(5) table.

17.6.1.11 pam_nologin(8)

The pam_nologin(8) module refuses non-root logins when `/var/run/nologin` exists. This file is normally created by shutdown(8) when less than five minutes remain until the scheduled shutdown time.

17.6.1.12 pam_permit(8)

The pam_permit(8) module is one of the simplest modules available; it responds to any request with `PAM_SUCCESS`. It is useful as a placeholder for services where one or more chains would otherwise be empty.

17.6.1.13 pam_radius(8)

The pam_radius(8) module provides authentication services based upon the RADIUS (Remote Authentication Dial In User Service) protocol.

17.6.1.14 pam_rhosts(8)

The pam_rhosts(8) module provides only authentication services. It reports success if and only if the target user's ID is not 0 and the remote host and user are listed in `/etc/hosts.equiv` or in the target user's `~/.rhosts`.

17.6.1.15 pam_rootok(8)

The pam_rootok(8) module reports success if and only if the real user id of the process calling it (which is assumed to be run by the applicant) is 0. This is useful for non-networked services such as su(1) or passwd(1), to which the `root` should have automatic access.

17.6.1.16 pam_securetty(8)

The pam_securetty(8) module provides only account services. It is used when the applicant is attemping to authenticate as superuser, and the process is attached to an insecure TTY.

17.6.1.17 pam_self(8)

The pam_self(8) module reports success if and only if the names of the applicant matches that of the target account. It is most useful for non-networked services such as su(1), where the identity of the applicant can be easily verified.

17.6.1.18 pam_ssh(8)

The pam_ssh(8) module provides both authentication and session services. The authentication service allows users who have passphrase-protected SSH secret keys in their `~/.ssh` directory to authenticate themselves by typing their passphrase. The session service starts ssh-agent(1) and preloads it with the keys that were decrypted in the authentication phase. This feature is particularly useful for local logins, whether in X (using xdm(1) or another PAM-aware X login manager) or at the console.

This module implements what is fundamentally a password authentication scheme. Care should be taken to only use this module over a secure session (secure TTY, encrypted session, etc.), otherwise the user's SSH passphrase could be compromised.

Additional consideration should be given to the use of pam_ssh(8). Users often assume that file permissions are sufficient to protect their SSH keys, and thus use weak or no passphrases. Since the system administrator has no effective means of enforcing SSH passphrase quality, this has the potential to expose the system to security risks.

17.6.1.19 pam_unix(8)

The pam_unix(8) module implements traditional UNIX® password authentication, using getpwnam(3) under FreeBSD or getpwnam_r(3) under NetBSD to obtain the target account's password and compare it with the one provided by the applicant. It also provides account management services (enforcing account and password expiration times) and password-changing services. This is probably the single most useful module, as the great majority of admins will want to maintain historical behaviour for at least some services.

17.6.2 FreeBSD-specific PAM Modules

17.6.2.1 pam_opie(8)

The pam_opie(8) module implements the opie(4) authentication method. The opie(4) system is a challenge-response mechanism where the response to each challenge is a direct function of the challenge and a passphrase, so the response can be easily computed "just in time" by anyone possessing the passphrase, eliminating the need for password lists. Moreover, since opie(4) never reuses a challenge that has been correctly answered, it is not vulnerable to replay attacks.

17.6.2.2 pam_opieaccess(8)

The pam_opieaccess(8) module is a companion module to pam_opie(8). Its purpose is to enforce the restrictions codified in opieaccess(5), which regulate the conditions under which a user who would normally authenticate herself using opie(4) is allowed to use alternate methods. This is most often used to prohibit the use of password authentication from untrusted hosts.

In order to be effective, the pam_opieaccess(8) module must be listed as `requisite` immediately after a `sufficient` entry for pam_opie(8), and before any other modules, in the `auth` chain.

17.6.2.3 pam_passwdqc(8)

The pam_passwdqc(8) module is a simple password strength checking module for PAM. In addition to checking regular passwords, it offers support for passphrases and can provide randomly generated passwords.

17.6.2.4 pam_tacplus(8)

The pam_tacplus(8) module provides authentication services based upon the TACACS+ protocol for the PAM (Pluggable Authentication Module) framework.

17.6.3 NetBSD-specific PAM Modules

17.6.3.1 pam_skey(8)

The pam_skey(8) module implements S/Key One Time Password (OTP) authentication methods, using the `/etc/skeykeys` database.

17.7 PAM Application Programming

This section has not yet been written.

17.8 PAM Module Programming

This section has not yet been written.

17.9 Sample PAM Application

The following is a minimal implementation of su(1) using PAM. Note that it uses the OpenPAM-specific openpam_ttyconv(3) conversation function, which is prototyped in `security/openpam.h`. If you wish build this application on a system with a different PAM library, you will have to provide your own conversation function. A robust conversation function is surprisingly difficult to implement; the one presented in the *Sample PAM Conversation Function* sub-chapter is a good starting point, but should not be used in real-world applications.

```
#include <sys/param.h>
#include <sys/wait.h>

#include <err.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>

#include <security/pam_appl.h>
#include <security/openpam.h> /* for openpam_ttyconv() */

extern char **environ;

static pam_handle_t *pamh;
static struct pam_conv pamc;

static void
usage(void)
{

 fprintf(stderr, "Usage: su [login [args]]\n");
 exit(1);
}

int
main(int argc, char *argv[])
{
 char hostname[MAXHOSTNAMELEN];
 const char *user, *tty;
 char **args, **pam_envlist, **pam_env;
 struct passwd *pwd;
 int o, pam_err, status;
 pid_t pid;

 while ((o = getopt(argc, argv, "h")) != -1)
```

```
 switch (o) {
 case 'h':
 default:
  usage();
 }

argc -= optind;
argv += optind;

if (argc > 0) {
 user = *argv;
 --argc;
 ++argv;
} else {
 user = "root";
}

/* initialize PAM */
pamc.conv = &openpam_ttyconv;
pam_start("su", user, &pamc, &pamh);

/* set some items */
gethostname(hostname, sizeof(hostname));
if ((pam_err = pam_set_item(pamh, PAM_RHOST, hostname)) != PAM_SUCCESS)
 goto pamerr;
user = getlogin();
if ((pam_err = pam_set_item(pamh, PAM_RUSER, user)) != PAM_SUCCESS)
 goto pamerr;
tty = ttyname(STDERR_FILENO);
if ((pam_err = pam_set_item(pamh, PAM_TTY, tty)) != PAM_SUCCESS)
 goto pamerr;

/* authenticate the applicant */
if ((pam_err = pam_authenticate(pamh, 0)) != PAM_SUCCESS)
 goto pamerr;
if ((pam_err = pam_acct_mgmt(pamh, 0)) == PAM_NEW_AUTHTOK_REQD)
 pam_err = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
if (pam_err != PAM_SUCCESS)
 goto pamerr;

/* establish the requested credentials */
if ((pam_err = pam_setcred(pamh, PAM_ESTABLISH_CRED)) != PAM_SUCCESS)
 goto pamerr;

/* authentication succeeded; open a session */
if ((pam_err = pam_open_session(pamh, 0)) != PAM_SUCCESS)
 goto pamerr;

/* get mapped user name; PAM may have changed it */
pam_err = pam_get_item(pamh, PAM_USER, (const void **)&user);
if (pam_err != PAM_SUCCESS || (pwd = getpwnam(user)) == NULL)
 goto pamerr;
```

```
 /* export PAM environment */
 if ((pam_envlist = pam_getenvlist(pamh)) != NULL) {
  for (pam_env = pam_envlist; *pam_env != NULL; ++pam_env) {
   putenv(*pam_env);
   free(*pam_env);
  }
  free(pam_envlist);
 }

 /* build argument list */
 if ((args = calloc(argc + 2, sizeof *args)) == NULL) {
  warn("calloc()");
  goto err;
 }
 *args = pwd->pw_shell;
 memcpy(args + 1, argv, argc * sizeof *args);

 /* fork and exec */
 switch ((pid = fork())) {
 case -1:
  warn("fork()");
  goto err;
 case 0:
  /* child: give up privs and start a shell */

  /* set uid and groups */
  if (initgroups(pwd->pw_name, pwd->pw_gid) == -1) {
   warn("initgroups()");
   _exit(1);
  }
  if (setgid(pwd->pw_gid) == -1) {
   warn("setgid()");
   _exit(1);
  }
  if (setuid(pwd->pw_uid) == -1) {
   warn("setuid()");
   _exit(1);
  }
  execve(*args, args, environ);
  warn("execve()");
  _exit(1);
 default:
  /* parent: wait for child to exit */
  waitpid(pid, &status, 0);

  /* close the session and release PAM resources */
  pam_err = pam_close_session(pamh, 0);
  pam_end(pamh, pam_err);

  exit(WEXITSTATUS(status));
 }

pamerr:
```

```
 fprintf(stderr, "Sorry\n");
err:
 pam_end(pamh, pam_err);
 exit(1);
}
```

17.10 Sample PAM Module

The following is a minimal implementation of pam_unix(8), offering only authentication services. It should build and run with most PAM implementations, but takes advantage of OpenPAM extensions if available: note the use of pam_get_authtok(3), which enormously simplifies prompting the user for a password.

```
#include <sys/param.h>

#include <pwd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <security/pam_modules.h>
#include <security/pam_appl.h>

#ifndef _OPENPAM
static char password_prompt[] = "Password:";
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN
#endif

PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{
#ifndef _OPENPAM
 struct pam_conv *conv;
 struct pam_message msg;
 const struct pam_message *msgp;
 struct pam_response *resp;
#endif
 struct passwd *pwd;
 const char *user;
 char *crypt_password, *password;
 int pam_err, retry;

 /* identify user */
 if ((pam_err = pam_get_user(pamh, &user, NULL)) != PAM_SUCCESS)
  return (pam_err);
 if ((pwd = getpwnam(user)) == NULL)
```

```
  return (PAM_USER_UNKNOWN);

 /* get password */
#ifndef _OPENPAM
 pam_err = pam_get_item(pamh, PAM_CONV, (const void **)&conv);
 if (pam_err != PAM_SUCCESS)
  return (PAM_SYSTEM_ERR);
 msg.msg_style = PAM_PROMPT_ECHO_OFF;
 msg.msg = password_prompt;
 msgp = &msg;
#endif
 for (retry = 0; retry < 3; ++retry) {
#ifdef _OPENPAM
  pam_err = pam_get_authtok(pamh, PAM_AUTHTOK,
      (const char **)&password, NULL);
#else
  resp = NULL;
  pam_err = (*conv->conv)(1, &msgp, &resp, conv->appdata_ptr);
  if (resp != NULL) {
   if (pam_err == PAM_SUCCESS)
    password = resp->resp;
   else
    free(resp->resp);
   free(resp);
  }
#endif
  if (pam_err == PAM_SUCCESS)
   break;
 }
 if (pam_err == PAM_CONV_ERR)
  return (pam_err);
 if (pam_err != PAM_SUCCESS)
  return (PAM_AUTH_ERR);

 /* compare passwords */
 if ((!pwd->pw_passwd[0] && (flags & PAM_DISALLOW_NULL_AUTHTOK)) ||
     (crypt_password = crypt(password, pwd->pw_passwd)) == NULL ||
     strcmp(crypt_password, pwd->pw_passwd) != 0)
  pam_err = PAM_AUTH_ERR;
 else
  pam_err = PAM_SUCCESS;
#ifndef _OPENPAM
 free(password);
#endif
 return (pam_err);
}

PAM_EXTERN int
pam_sm_setcred(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{

 return (PAM_SUCCESS);
```

```
}

PAM_EXTERN int
pam_sm_acct_mgmt(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{

 return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_open_session(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{

 return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_close_session(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{

 return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_chauthtok(pam_handle_t *pamh, int flags,
 int argc, const char *argv[])
{

 return (PAM_SERVICE_ERR);
}

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_unix");
#endif
```

17.11 Sample PAM Conversation Function

The conversation function presented below is a greatly simplified version of OpenPAM's openpam_ttyconv(3). It is fully functional, and should give the reader a good idea of how a conversation function should behave, but it is far too simple for real-world use. Even if you're not using OpenPAM, feel free to download the source code and adapt openpam_ttyconv(3) to your uses; we believe it to be as robust as a tty-oriented conversation function can reasonably get.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
#include <security/pam_appl.h>

int
converse(int n, const struct pam_message **msg,
 struct pam_response **resp, void *data)
{
 struct pam_response *aresp;
 char buf[PAM_MAX_RESP_SIZE];
 int i;

 data = data;
 if (n <= 0 || n > PAM_MAX_NUM_MSG)
  return (PAM_CONV_ERR);
 if ((aresp = calloc(n, sizeof *aresp)) == NULL)
  return (PAM_BUF_ERR);
 for (i = 0; i < n; ++i) {
  aresp[i].resp_retcode = 0;
  aresp[i].resp = NULL;
  switch (msg[i]->msg_style) {
  case PAM_PROMPT_ECHO_OFF:
   aresp[i].resp = strdup(getpass(msg[i]->msg));
   if (aresp[i].resp == NULL)
    goto fail;
   break;
  case PAM_PROMPT_ECHO_ON:
   fputs(msg[i]->msg, stderr);
   if (fgets(buf, sizeof buf, stdin) == NULL)
    goto fail;
   aresp[i].resp = strdup(buf);
   if (aresp[i].resp == NULL)
    goto fail;
   break;
  case PAM_ERROR_MSG:
   fputs(msg[i]->msg, stderr);
   if (strlen(msg[i]->msg) > 0 &&
       msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
    fputc('\n', stderr);
   break;
  case PAM_TEXT_INFO:
   fputs(msg[i]->msg, stdout);
   if (strlen(msg[i]->msg) > 0 &&
       msg[i]->msg[strlen(msg[i]->msg) - 1] != '\n')
    fputc('\n', stdout);
   break;
  default:
   goto fail;
  }
 }
 *resp = aresp;
 return (PAM_SUCCESS);
 fail:
       for (i = 0; i < n; ++i) {
               if (aresp[i].resp != NULL) {
```

```
                        memset(aresp[i].resp, 0, strlen(aresp[i].resp));
                        free(aresp[i].resp);
                }
        }
        memset(aresp, 0, n * sizeof *aresp);
 *resp = NULL;
 return (PAM_CONV_ERR);
}
```

17.12 Further Reading

Bibliography

Papers

Making Login Services Independent of Authentication Technologies (http://www.sun.com/software/solaris/pam/pam.external.pdf), Vipin Samar and Charlie Lai, Sun Microsystems.

X/Open Single Sign-on Preliminary Specification (http://www.opengroup.org/pubs/catalog/p702.htm), The Open Group, 1-85912-144-6, June 1997.

Pluggable Authentication Modules (http://www.kernel.org/pub/linux/libs/pam/pre/doc/current-draft.txt), Andrew G. Morgan, October 6, 1999.

User Manuals

PAM Administration (http://www.sun.com/software/solaris/pam/pam.admin.pdf), Sun Microsystems.

Related Web pages

OpenPAM homepage (http://openpam.sourceforge.net/), Dag-Erling Smørgrav, ThinkSec AS.

Linux-PAM homepage (http://www.kernel.org/pub/linux/libs/pam/), Andrew G. Morgan.

Solaris PAM homepage (http://wwws.sun.com/software/solaris/pam/), Sun Microsystems.

Chapter 18
Tuning NetBSD

18.1 Introduction

18.1.1 Overview

This section covers a variety of performance tuning topics. It attempts to span tuning from the perspective of the system administrator to systems programmer. The art of performance tuning itself is very old. To tune something means to make it operate more efficiently, whether one is referring to a NetBSD based technical server or a vacuum cleaner, the goal is to improve something, whether that be the way something is done, how it works or how it is put together.

18.1.1.1 What is Performance Tuning?

A view from 10,000 feet pretty much dictates that everything we do is task oriented, this pertains to a NetBSD system as well. When the system boots, it automatically begins to perform a variety of tasks. When a user logs in, they usually have a wide variety of tasks they have to accomplish. In the scope of these documents, however, performance tuning strictly means to improve how efficient a NetBSD system performs.

The most common thought that crops into someone's mind when they think "tuning" is some sort of speed increase or decreasing the size of the kernel - while those are ways to improve performance, they are not the only ends an administrator may have to take for increasing efficiency. For our purposes, performance tuning means this: *To make a NetBSD system operate in an optimum state.*

Which could mean a variety of things, not necessarily speed enhancements. A good example of this is filesystem formatting parameters, on a system that has a lot of small files (say like a source repository) an administrator may need to increase the number of inodes by making their size smaller (say down to 1024k) and then increasing the amount of inodes. In this case the number of inodes was increased, however, it keeps the administrator from getting those nasty out of inodes messages, which ultimately makes the system more efficient.

Tuning normally revolves around finding and eliminating bottlenecks. Most of the time, such bottlenecks are spurious, for example, a release of Mozilla that does not quite handle java applets too well can cause Mozilla to start crunching the CPU, especially applets that are not done well. Occasions when processes seem to spin off into nowhere and eat CPU are almost always resolved with a kill. There are instances, however, when resolving bottlenecks takes a lot longer, for example, say an rsynced server is just getting larger and larger. Slowly, performance begins to fade and the administrator may have to take some sort of action to speed things up, however, the situation is relative to say an emergency like an instantly spiked CPU.

18.1.1.2 When does one tune?

Many NetBSD users rarely have to tune a system. The GENERIC kernel may run just fine and the layout/configuration of the system may do the job as well. By the same token, as a pragma it is always good to know how to tune a system. Most often tuning comes as a result of a sudden bottleneck issue (which may occur randomly) or a gradual loss of performance. It does happen in a sense to everyone at some point, one process that is eating the CPU is a bottleneck as much as a gradual increase in paging. So, the question should not be when to tune so much as when to learn to tune.

One last time to tune is if you can tune in a preventive manner (and you think you might need to) then do it. One example of this was on a system that needed to be able to reboot quickly. Instead of waiting, I did everything I could to trim the kernel and make sure there was absolutely nothing running that was not needed, I even removed drivers that did have devices, but were never used (lp). The result was reducing reboot time by nearly two-thirds. In the long run, it was a smart move to tune it before it became an issue.

18.1.1.3 What these Documents Will Not Cover

Before I wrap up the introduction, I think it is important to note what these documents will not cover. This guide will pertain only to the core NetBSD system. In other words, it will not cover tuning a web server's configuration to make it run better; however, it might mention how to tune NetBSD to run better as a web server. The logic behind this is simple: web servers, database software, etc. are third party and almost limitless. I could easily get mired down in details that do not apply to the NetBSD system. Almost all third party software have their own documentation about tuning anyhow.

18.1.1.4 How Examples are Laid Out

Since there is ample man page documentation, only used options and arguments with examples are discussed. In some cases, material is truncated for brevity and not thoroughly discussed because, quite simply, there is too much. For example, every single device driver entry in the kernel will not be discussed, however, an example of determining whether or not a given system needs one will be. Nothing in this Guide is concrete, tuning and performance are very subjective, instead, it is a guide for the reader to learn what some of the tools available to them can do.

18.2 Tuning Considerations

Tuning a system is not really too difficult when pro-active tuning is the approach. This document approaches tuning from a "before it comes up" approach. While tuning in spare time is considerably easier versus say, a server that is almost completely bogged down to 0.1% idle time, there are still a few things that should be mulled over about tuning before actually doing it, hopefully, before a system is even installed.

18.2.1 General System Configuration

Of course, how the system is setup makes a big difference. Sometimes small items can be overlooked which may in fact cause some sort of long term performance problem.

18.2.1.1 Filesystems and Disks

How the filesystem is laid out relative to disk drives is very important. On hardware RAID systems, it is not such a big deal, but, many NetBSD users specifically use NetBSD on older hardware where hardware RAID simply is not an option. The idea of `/` being close to the first drive is a good one, but for example if there are several drives to choose from that will be the first one, is the best performing the one that `/` will be on? On a related note, is it wise to split off `/usr`? Will the system see heavy usage say in `/usr/pkgsrc`? It might make sense to slap a fast drive in and mount it under `/usr/pkgsrc`, or it might not. Like all things in performance tuning, this is subjective.

18.2.1.2 Swap Configuration

There are three schools of thought on swap size and about fifty on using split swap files with prioritizing and how that should be done. In the swap size arena, the vendor schools (at least most commercial ones) usually have their own formulas per OS. As an example, on a particular version of HP-UX with a particular version of Oracle the formula was:

2.5 GB * Number_of_processor

Well, that all really depends on what type of usage the database is having and how large it is, for instance if it is so large that it must be distributed, that formula does not fit well.

The next school of thought about swap sizing is sort of strange but makes some sense, it says, if possible, get a reference amount of memory used by the system. It goes something like this:

1. Startup a machine and estimate total memory needs by running everything that may ever be needed at once. Databases, web servers .... whatever. Total up the amount.
2. Add a few MB for padding.
3. Subtract the amount of physical RAM from this total.

If the amount leftover is 3 times the size of physical RAM, consider getting more RAM. The problem, of course, is figuring out what is needed and how much space it will take. There is also another flaw in this method, some programs do not behave well. A glaring example of misbehaved software is web browsers. On certain versions of Netscape, when something went wrong it had a tendency to runaway and eat swap space. So, the more spare space available, the more time to kill it.

Last and not least is the tried and true PHYSICAL_RAM * 2 method. On modern machines and even older ones (with limited purpose of course) this seems to work best.

All in all, it is hard to tell when swapping will start. Even on small 16MB RAM machines (and less) NetBSD has always worked well for most people until misbehaving software is running.

18.2.2 System Services

On servers, system services have a large impact. Getting them to run at their best almost always requires some sort of network level change or a fundamental speed increase in the underlying system (which of course is what this is all about). There are instances when some simple solutions can improve services. One example, an ftp server is becoming slower and a new release of the ftp server that is shipped with the

system comes out that, just happens to run faster. By upgrading the ftp software, a performance boost is accomplished.

Another good example where services are concerned is the age old question: “To use inetd or not to use inetd?” A great service example is pop3. Pop3 connections can conceivably clog up inetd. While the pop3 service itself starts to degrade slowly, other services that are multiplexed through inetd will also degrade (in some case more than pop3). Setting up pop3 to run outside of inetd and on its own may help.

18.2.3 The NetBSD Kernel

The NetBSD kernel obviously plays a key role in how well a system performs, while rebuilding and tuning the kernel is covered later in the text, it is worth discussing in the local context from a high level.

Tuning the NetBSD kernel really involves three main areas:

1. removing unrequired drivers
2. configuring options
3. system settings

18.2.3.1 Removing Unrequired Drivers

Taking drivers out of the kernel that are not needed achieves several results; first, the system boots faster since the kernel is smaller, second again since the kernel is smaller, more memory is free to users and processes and third, the kernel tends to respond quicker.

18.2.3.2 Configuring Options

Configuring options such as enabling/disabling certain subsystems, specific hardware and filesystems can also improve performance pretty much the same way removing unrequired drivers does. A very simple example of this is a FTP server that only hosts ftp files - nothing else. On this particular server there is no need to have anything but native filesystem support and perhaps a few options to help speed things along. Why would it ever need NTFS support for example? Besides, if it did, support for NTFS could be added at some later time. In an opposite case, a workstation may need to support a lot of different filesystem types to share and access files.

18.2.3.3 System Settings

System wide settings are controlled by the kernel, a few examples are filesystem settings, network settings and core kernel settings such as the maximum number of processes. Almost all system settings can be at least looked at or modified via the sysctl facility. Examples using the sysctl facility are given later on.

18.3 Visual Monitoring Tools

NetBSD ships a variety of performance monitoring tools with the system. Most of these tools are common on all UNIX systems. In this section some example usage of the tools is given with interpretation of the output.

18.3.1 The top Process Monitor

The top monitor does exactly what it says: it displays the CPU hogs on the system. To run the monitor, simply type top at the prompt. Without any arguments, it should look like:

```
load averages:  0.09,  0.12,  0.08                                      20:23:41
21 processes:  20 sleeping, 1 on processor
CPU states:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
Memory: 15M Act, 1104K Inact, 208K Wired, 22M Free, 129M Swap free

  PID USERNAME PRI NICE   SIZE   RES STATE     TIME   WCPU    CPU COMMAND
13663 root       2    0  1552K 1836K sleep     0:08  0.00%  0.00% httpd
  127 root      10    0   129M 4464K sleep     0:01  0.00%  0.00% mount_mfs
22591 root       2    0   388K 1156K sleep     0:01  0.00%  0.00% sshd
  108 root       2    0   132K  472K sleep     0:01  0.00%  0.00% syslogd
22597 jrf       28    0   156K  616K onproc    0:00  0.00%  0.00% top
22592 jrf       18    0   828K 1128K sleep     0:00  0.00%  0.00% tcsh
  203 root      10    0   220K  424K sleep     0:00  0.00%  0.00% cron
    1 root      10    0   312K  192K sleep     0:00  0.00%  0.00% init
  205 root       3    0    48K  432K sleep     0:00  0.00%  0.00% getty
  206 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
  208 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
  207 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
13667 nobody     2    0  1660K 1508K sleep     0:00  0.00%  0.00% httpd
 9926 root       2    0   336K  588K sleep     0:00  0.00%  0.00% sshd
  200 root       2    0    76K  456K sleep     0:00  0.00%  0.00% inetd
  182 root       2    0    92K  436K sleep     0:00  0.00%  0.00% portsentry
  180 root       2    0    92K  436K sleep     0:00  0.00%  0.00% portsentry
13666 nobody    -4    0  1600K 1260K sleep     0:00  0.00%  0.00% httpd
```

The top utility is great for finding CPU hogs, runaway processes or groups of processes that may be causing problems. The output shown above indicates that this particular system is in good health. Now, the next display should show some very different results:

```
load averages:  0.34,  0.16,  0.13                                      21:13:47
25 processes:  24 sleeping, 1 on processor
CPU states:  0.5% user,  0.0% nice,  9.0% system,  1.0% interrupt, 89.6% idle
Memory: 20M Act, 1712K Inact, 240K Wired, 30M Free, 129M Swap free

  PID USERNAME PRI NICE   SIZE   RES STATE     TIME   WCPU    CPU COMMAND
 5304 jrf       -5    0    56K  336K sleep     0:04 66.07% 19.53% bonnie
 5294 root       2    0   412K 1176K sleep     0:02  1.01%  0.93% sshd
  108 root       2    0   132K  472K sleep     1:23  0.00%  0.00% syslogd
  187 root       2    0  1552K 1824K sleep     0:07  0.00%  0.00% httpd
 5288 root       2    0   412K 1176K sleep     0:02  0.00%  0.00% sshd
 5302 jrf       28    0   160K  620K onproc    0:00  0.00%  0.00% top
```

```
 5295 jrf       18    0   828K 1116K sleep     0:00  0.00%  0.00% tcsh
 5289 jrf       18    0   828K 1112K sleep     0:00  0.00%  0.00% tcsh
  127 root      10    0   129M 8388K sleep     0:00  0.00%  0.00% mount_mfs
  204 root      10    0   220K  424K sleep     0:00  0.00%  0.00% cron
    1 root      10    0   312K  192K sleep     0:00  0.00%  0.00% init
  208 root       3    0    48K  432K sleep     0:00  0.00%  0.00% getty
  210 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
  209 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
  211 root       3    0    48K  424K sleep     0:00  0.00%  0.00% getty
  217 nobody     2    0  1616K 1272K sleep     0:00  0.00%  0.00% httpd
  184 root       2    0   336K  580K sleep     0:00  0.00%  0.00% sshd
  201 root       2    0    76K  456K sleep     0:00  0.00%  0.00% inetd
```

At first, it should seem rather obvious which process is hogging the system, however, what is interesting in this case is why. The bonnie program is a disk benchmark tool which can write large files in a variety of sizes and ways. What the previous output indicates is only that the bonnie program is a CPU hog, but not why.

18.3.1.1 Other Neat Things About Top

A careful examination of the manual page top(1) shows that there is a lot more that can be done with top, for example, processes can have their priority changed and killed. Additionally, filters can be set for looking at processes.

18.3.2 The sysstat utility

As the man page sysstat(1) indicates, the sysstat utility shows a variety of system statistics using the curses library. While it is running the screen is shown in two parts, the upper window shows the current load average while the lower screen depends on user commands. The exception to the split window view is when vmstat display is on which takes up the whole screen. Following is what sysstat looks like on a fairly idle system with no arguments given when it was invoked:

```
                /0   /1   /2   /3   /4   /5   /6   /7   /8   /9   /10
    Load Average   |

                      /0   /10  /20  /30  /40  /50  /60  /70  /80  /90  /100
               <idle> XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Basically a lot of dead time there, so now have a look with some arguments provided, in this case, **sysstat inet.tcp** which looks like this:

```
                /0   /1   /2   /3   /4   /5   /6   /7   /8   /9   /10
    Load Average   |

       0 connections initiated          19 total TCP packets sent
       0 connections accepted           11   data
       0 connections established         0   data (retransmit)
                                         8   ack-only
       0 connections dropped             0   window probes
       0   in embryonic state            0   window updates
```

```
        0   on retransmit timeout          0   urgent data only
        0   by keepalive                   0   control
        0   by persist
                                          29 total TCP packets received
       11 potential rtt updates           17   in sequence
       11 successful rtt updates           0   completely duplicate
        9 delayed acks sent                0   with some duplicate data
        0 retransmit timeouts              4   out of order
        0 persist timeouts                 0   duplicate acks
        0 keepalive probes                11   acks
        0 keepalive timeouts               0   window probes
                                           0   window updates
```

Now that is informative. The first poll is accumulative, so it is possible to see quite a lot of information in the output when sysstat is invoked. Now, while that may be interesting, how about a look at the buffer cache with **sysstat bufcache**:

```
                   /0   /1   /2   /3   /4   /5   /6   /7   /8   /9   /10
     Load Average

There are 1642 buffers using 6568 kBytes of memory.

File System          Bufs used   %   kB in use   %  Bufsize kB   %  Util %
/                          877  53        6171  93        6516  99       94
/var/tmp                     5   0          17   0          28   0       60

Total:                     882  53        6188  94        6544  99
```

Again, a pretty boring system, but great information to have available. While this is all nice to look at, it is time to put a false load on the system to see how sysstat can be used as a performance monitoring tool. As with top, bonnie++ will be used to put a high load on the I/O subsystems and a little on the CPU. The bufcache will be looked at again to see of there are any noticeable differences:

```
                   /0   /1   /2   /3   /4   /5   /6   /7   /8   /9   /10
     Load Average  |||

There are 1642 buffers using 6568 kBytes of memory.

File System          Bufs used   %   kB in use   %  Bufsize kB   %  Util %
/                          811  49        6422  97        6444  98       99

Total:                     811  49        6422  97        6444  98
```

First, notice that the load average shot up, this is to be expected of course, then, while most of the numbers are close, notice that utilization is at 99%. Throughout the time that bonnie++ was running the utilization percentage remained at 99, this of course makes sense, however, in a real troubleshooting situation, it could be indicative of a process doing heavy I/O on one particular file or filesystem.

18.4 Monitoring Tools

In addition to screen oriented monitors and tools, the NetBSD system also ships with a set of command line oriented tools. Many of the tools that ship with a NetBSD system can be found on other UNIX and UNIX-like systems.

18.4.1 fstat

The fstat(1) utility reports the status of open files on the system, while it is not what many administrators consider a performance monitor, it can help find out if a particular user or process is using an inordinate amount of files, generating large files and similar information.

Following is a sample of some fstat output:

```
USER     CMD          PID   FD MOUNT      INUM MODE         SZ|DV R/W
jrf      tcsh       21607   wd /         29772 drwxr-xr-x     512 r
jrf      tcsh       21607    3* unix stream c057acc0<-> c0553280
jrf      tcsh       21607    4* unix stream c0553280 <-> c057acc0
root     sshd       21597   wd /             2 drwxr-xr-x     512 r
root     sshd       21597    0 /         11921 crw-rw-rw-    null rw
nobody   httpd       5032   wd /             2 drwxr-xr-x     512 r
nobody   httpd       5032    0 /         11921 crw-rw-rw-    null r
nobody   httpd       5032    1 /         11921 crw-rw-rw-    null w
nobody   httpd       5032    2 /         15890 -rw-r--r--  353533 rw
...
```

The fields are pretty self explanatory, again, this tool while not as performance oriented as others, can come in handy when trying to find out information about file usage.

18.4.2 iostat

The iostat(8) command does exactly what it sounds like, it reports the status of the I/O subsystems on the system. When iostat is employed, the user typically runs it with a certain number of counts and an interval between them like so:

```
$ iostat 5 5
      tty             wd0             cd0             fd0             md0             cpu
 tin tout  KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s  us ni sy in id
   0    1  5.13   1 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
   0   54  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
   0   18  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
   0   18  8.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
   0   28  0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
```

The above output is from a very quiet ftp server. The fields represent the various I/O devices, the tty (which, ironically, is the most active because iostat is running), wd0 which is the primary IDE disk, cd0, the cdrom drive, fd0, the floppy and the memory filesystem.

Now, let's see if we can pummel the system with some heavy usage. First, a large ftp transaction consisting of a tarball of netbsd-current source along with the bonnie++ disk benchmark program running at the same time.

```
$ iostat 5 5
      tty             wd0             cd0             fd0             md0             cpu
 tin tout  KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s   KB/t t/s MB/s  us ni sy in id
   0    1  5.68   1 0.00   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  0  0 100
   0   54 61.03 150 8.92   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   1  0 18  4 78
   0   26 63.14 157 9.71   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   1  0 20  4 75
   0   20 43.58  26 1.12   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   0  0  9  2 88
   0   28 19.49  82 1.55   0.00   0 0.00   0.00   0 0.00   0.00   0 0.00   1  0  7  3 89
```

As can be expected, notice that wd0 is very active, what is interesting about this output is how the processor's I/O seems to rise in proportion to wd0. This makes perfect sense, however, it is worth noting that only because this ftp server is hardly being used can that be observed. If, for example, the cpu I/O subsystem was already under a moderate load and the disk subsystem was under the same load as it is now, it could appear that the cpu is bottlenecked when in fact it would have been the disk. In such a case, we can observe that "one tool" is rarely enough to completely analyze a problem. A quick glance at processes probably would tell us (after watching iostat) which processes were causing problems.

18.4.3 ps

Using the ps(1) command or process status, a great deal of information about the system can be discovered. Most of the time, the ps command is used to isolate a particular process by name, group, owner etc. Invoked with no options or arguments, ps simply prints out information about the user executing it.

```
$ ps
  PID TT STAT    TIME COMMAND
21560 p0 Is   0:00.04 -tcsh
21564 p0 I+   0:00.37 ssh jrf.odpn.net
21598 p1 Ss   0:00.12 -tcsh
21673 p1 R+   0:00.00 ps
21638 p2 Is+  0:00.06 -tcsh
```

Not very exciting. The fields are self explanatory with the exception of STAT which is actually the state a process is in. The flags are all documented in the man page, however, in the above example, I is idle, S is sleeping, R is runnable, the + means the process is in a foreground state, and the s means the process is a session leader. This all makes perfect sense when looking at the flags, for example, PID 21560 is a shell, it is idle and (as would be expected) the shell is the process leader.

In most cases, someone is looking for something very specific in the process listing. As an example, looking at all processes is specified with -a, to see all processes plus those without controlling terminals is -ax and to get a much more verbose listing (basically everything plus information about the impact processes are having) aux:

```
# ps aux
USER     PID %CPU %MEM    VSZ  RSS TT STAT STARTED    TIME COMMAND
root       0  0.0  9.6      0 6260 ?? DLs  16Jul02 0:01.00 (swapper)
root   23362  0.0  0.8    144  488 ?? S    12:38PM 0:00.01 ftpd -l
root   23328  0.0  0.4    428  280 p1 S    12:34PM 0:00.04 -csh
jrf    23312  0.0  1.8    828 1132 p1 Is   12:32PM 0:00.06 -tcsh
root   23311  0.0  1.8    388 1156 ?? S    12:32PM 0:01.60 sshd: jrf@ttyp1
jrf    21951  0.0  1.7    244 1124 p0 S+    4:22PM 0:02.90 ssh jrf.odpn.net
```

```
jrf     21947  0.0  1.7    828 1128 p0 Is    4:21PM 0:00.04 -tcsh
root    21946  0.0  1.8    388 1156 ?? S     4:21PM 0:04.94 sshd: jrf@ttyp0
nobody   5032  0.0  2.0   1616 1300 ?? I    19Jul02 0:00.02 /usr/pkg/sbin/httpd
...
```

Again, most of the fields are self explanatory with the exception of VSZ and RSS which can be a little confusing. RSS is the real size of a process in 1024 byte units while VSZ is the virtual size. This is all great, but again, how can ps help? Well, for one, take a look at this modified version of the same output:

```
# ps aux
USER      PID %CPU %MEM    VSZ  RSS TT STAT STARTED    TIME COMMAND
root        0  0.0  9.6      0 6260 ?? DLs  16Jul02 0:01.00 (swapper)
root    23362  0.0  0.8    144  488 ?? S    12:38PM 0:00.01 ftpd -l
root    23328  0.0  0.4    428  280 p1 S    12:34PM 0:00.04 -csh
jrf     23312  0.0  1.8    828 1132 p1 Is   12:32PM 0:00.06 -tcsh
root    23311  0.0  1.8    388 1156 ?? S    12:32PM 0:01.60 sshd: jrf@ttyp1
jrf     21951  0.0  1.7    244 1124 p0 S+    4:22PM 0:02.90 ssh jrf.odpn.net
jrf     21947  0.0  1.7    828 1128 p0 Is    4:21PM 0:00.04 -tcsh
root    21946  0.0  1.8    388 1156 ?? S     4:21PM 0:04.94 sshd: jrf@ttyp0
nobody   5032  9.0  2.0   1616 1300 ?? I    19Jul02 0:00.02 /usr/pkg/sbin/httpd
...
```

Given that on this server, our baseline indicates a relatively quiet system, the PID 5032 has an unusually large amount of %CPU. Sometimes this can also cause high TIME numbers. The ps command can be grepped on for PIDs, username and process name and hence help track down processes that may be experiencing problems.

18.4.4 vmstat

Using vmstat(1), information pertaining to virtual memory can be monitored and measured. Not unlike iostat, vmstat can be invoked with a count and interval. Following is some sample output using 5 5 like the iostat example:

```
# vmstat 5 5
 procs   memory     page                    disks         faults      cpu
 r b w   avm   fre  flt  re  pi   po   fr   sr w0 c0 f0 m0   in   sy  cs us sy id
 0 7 0 17716 33160    2   0   0    0    0    0  1  0  0  0  105   15   4  0  0 100
 0 7 0 17724 33156    2   0   0    0    0    0  1  0  0  0  109    6   3  0  0 100
 0 7 0 17724 33156    1   0   0    0    0    0  1  0  0  0  105    6   3  0  0 100
 0 7 0 17724 33156    1   0   0    0    0    0  0  0  0  0  107    6   3  0  0 100
 0 7 0 17724 33156    1   0   0    0    0    0  0  0  0  0  105    6   3  0  0 100
```

Yet again, relatively quiet, for posterity, the exact same load that was put on this server in the iostat example will be used. The load is a large file transfer and the bonnie benchmark program.

```
# vmstat 5 5
 procs   memory     page                    disks         faults      cpu
 r b w   avm   fre  flt  re  pi   po   fr   sr w0 c0 f0 m0   in   sy  cs us sy id
 1 8 0 18880 31968    2   0   0    0    0    0  1  0  0  0  105   15   4  0  0 100
 0 8 0 18888 31964    2   0   0    0    0    0 130  0  0  0 1804 5539 1094 31 22 47
 1 7 0 18888 31964    1   0   0    0    0    0 130  0  0  0 1802 5500 1060 36 16 49
 1 8 0 18888 31964    1   0   0    0    0    0 160  0  0  0 1849 5905 1107 21 22 57
```

```
 1 7 0 18888 31964    1   0   0    0    0    0 175  0  0  0 1893 6167 1082  1 25 75
```

Just a little different. Notice, since most of the work was I/O based, the actual memory used was not very much. Since this system uses mfs for `/tmp`, however, it can certainly get beat up. Have a look at this:

```
# vmstat 5 5
 procs    memory      page                          disks         faults      cpu
 r b w    avm    fre  flt  re  pi    po    fr    sr w0 c0 f0 m0   in    sy  cs us sy id
 0 2 0 99188    500    2   0   0     0     0     0  1  0  0  0  105    16   4  0  0 100
 0 2 0111596    436  592   0 587   624   586 1210 624  0  0  0  741   883 1088  0 11 89
 0 3 0123976    784  666   0 662   643   683 1326 702  0  0  0  828   993 1237  0 12 88
 0 2 0134692   1236  581   0 571   563   595 1158 599  0  0  0  722   863 1066  0  9 90
 2 0 0142860    912  433   0 406   403   405  808 429  0  0  0  552   602 768  0  7 93
```

Pretty scary stuff. That was created by running bonnie in `/tmp` on a memory based filesystem. If it continued for too long, it is possible the system could have started thrashing. Notice that even though the VM subsystem was taking a beating, the processors still were not getting too battered.

18.5 Network Tools

Sometimes a performance problem is not a particular machine, it is the network or some sort of device on the network such as another host, a router etc. What other machines that provide a service or some sort of connectivity to a particular NetBSD system do and how they act can have a very large impact on performance of the NetBSD system itself, or the perception of performance by users. A really great example of this is when a DNS server that a NetBSD machine is using suddenly disappears. Lookups take long and they eventually fail. Someone logged into the NetBSD machine who is not experienced would undoubtedly (provided they had no other evidence) blame the NetBSD system. One of my personal favorites, “the Internet is broke” usually means either DNS service or a router/gateway has dropped offline. Whatever the case may be, a NetBSD system comes adequately armed to deal with finding out what network issues may be cropping up whether the fault of the local system or some other issue.

18.5.1 ping

The classic ping(8) utility can tell us if there is just plain connectivity, it can also tell if host resolution (depending on how `nsswitch.conf` dictates) is working. Following is some typical ping output on a local network with a count of 3 specified:

```
# ping -c 3 marie
PING marie (172.16.14.12): 56 data bytes
64 bytes from 172.16.14.12: icmp_seq=0 ttl=255 time=0.571 ms
64 bytes from 172.16.14.12: icmp_seq=1 ttl=255 time=0.361 ms
64 bytes from 172.16.14.12: icmp_seq=2 ttl=255 time=0.371 ms

----marie PING Statistics----
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.361/0.434/0.571/0.118 ms
```

Not only does ping tell us if a host is alive, it tells us how long it took and gives some nice details at the very end. If a host cannot be resolved, just the IP address can be specified as well:

```
# ping -c 1 172.16.20.5
PING ash (172.16.20.5): 56 data bytes
64 bytes from 172.16.20.5: icmp_seq=0 ttl=64 time=0.452 ms

----ash PING Statistics----
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.452/0.452/0.452/0.000 ms
```

Now, not unlike any other tool, the times are very subjective, especially in regards to networking. For example, while the times in the examples are good, take a look at the localhost ping:

```
# ping -c 4 localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=0.091 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=255 time=0.129 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=255 time=0.120 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=255 time=0.122 ms

----localhost PING Statistics----
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.091/0.115/0.129/0.017 ms
```

Much smaller because the request never left the machine. Pings can be used to gather information about how well a network is performing. It is also good for problem isolation, for instance, if there are three relatively close in size NetBSD systems on a network and one of them simply has horrible ping times, chances are something is wrong on that one particular machine.

18.5.2 traceroute

The traceroute(8) command is great for making sure a path is available or detecting problems on a particular path. As an example, here is a trace between the example ftp server and ftp.NetBSD.org:

```
# traceroute ftp.NetBSD.org
traceroute to ftp.NetBSD.org (204.152.184.75), 30 hops max, 40 byte packets
 1  208.44.95.1 (208.44.95.1)  1.646 ms  1.492 ms  1.456 ms
 2  63.144.65.170 (63.144.65.170)  7.318 ms  3.249 ms  3.854 ms
 3  chcg01-edge18.il.inet.qwest.net (65.113.85.229)  35.982 ms  28.667 ms  21.971 ms
 4  chcg01-core01.il.inet.qwest.net (205.171.20.1)  22.607 ms  26.242 ms  19.631 ms
 5  snva01-core01.ca.inet.qwest.net (205.171.8.50)  78.586 ms  70.585 ms  84.779 ms
 6  snva01-core03.ca.inet.qwest.net (205.171.14.122)  69.222 ms  85.739 ms  75.979 ms
 7  paix01-brdr02.ca.inet.qwest.net (205.171.205.30)  83.882 ms  67.739 ms  69.937 ms
 8  198.32.175.3 (198.32.175.3)  72.782 ms  67.687 ms  73.320 ms
 9  so-1-0-0.orpa8.pf.isc.org (192.5.4.231)  78.007 ms  81.860 ms  77.069 ms
10  tun0.orrc5.pf.isc.org (192.5.4.165)  70.808 ms  75.151 ms  81.485 ms
11  ftp.NetBSD.org (204.152.184.75)  69.700 ms  69.528 ms  77.788 ms
```

All in all, not bad. The trace went from the host to the local router, then out onto the provider network and finally out onto the Internet looking for the final destination. How to interpret traceroutes, again, are subjective, but abnormally high times in portions of a path can indicate a bottleneck on a piece of

network equipment. Not unlike ping, if the host itself is suspect, run traceroute from another host to the same destination. Now, for the worst case scenario:

```
# traceroute www.microsoft.com
traceroute: Warning: www.microsoft.com has multiple addresses; using 207.46.230.220
traceroute to www.microsoft.akadns.net (207.46.230.220), 30 hops max, 40 byte packets
 1  208.44.95.1 (208.44.95.1)  2.517 ms  4.922 ms  5.987 ms
 2  63.144.65.170 (63.144.65.170)  10.981 ms  3.374 ms  3.249 ms
 3  chcg01-edge18.il.inet.qwest.net (65.113.85.229)  37.810 ms  37.505 ms  20.795 ms
 4  chcg01-core03.il.inet.qwest.net (205.171.20.21)  36.987 ms  32.320 ms  22.430 ms
 5  chcg01-brdr03.il.inet.qwest.net (205.171.20.142)  33.155 ms  32.859 ms  33.462 ms
 6  205.171.1.162 (205.171.1.162)  39.265 ms  20.482 ms  26.084 ms
 7  sl-bb24-chi-13-0.sprintlink.net (144.232.26.85)  26.681 ms  24.000 ms  28.975 ms
 8  sl-bb21-sea-10-0.sprintlink.net (144.232.20.30)  65.329 ms  69.694 ms  76.704 ms
 9  sl-bb21-tac-9-1.sprintlink.net (144.232.9.221)  65.659 ms  66.797 ms  74.408 ms
10  144.232.187.194 (144.232.187.194)  104.657 ms  89.958 ms  91.754 ms
11  207.46.154.1 (207.46.154.1)  89.197 ms  84.527 ms  81.629 ms
12  207.46.155.10 (207.46.155.10)  78.090 ms  91.550 ms  89.480 ms
13  * * *
.......
```

In this case, the Microsoft server cannot be found either because of multiple addresses or somewhere along the line a system or server cannot reply to the information request. At that point, one might think to try ping, in the Microsoft case, a ping does not reply, that is because somewhere on their network ICMP is most likely disabled.

18.5.3 netstat

Another problem that can crop up on a NetBSD system is routing table issues. These issues are not always the systems fault. The route(8) and netstat(1) commands can show information about routes and network connections (respectively).

The route command can be used to look at and modify routing tables while netstat can display information about network connections and routes. First, here is some output from route show:

```
# route show
Routing tables

Internet:
Destination      Gateway            Flags
default          208.44.95.1        UG
loopback         127.0.0.1          UG
localhost        127.0.0.1          UH
172.15.13.0      172.16.14.37       UG
172.16.0.0       link#2             U
172.16.14.8      0:80:d3:cc:2c:0    UH
172.16.14.10     link#2             UH
marie            0:10:83:f9:6f:2c   UH
172.16.14.37     0:5:32:8f:d2:35    UH
172.16.16.15     link#2             UH
loghost          8:0:20:a7:f0:75    UH
artemus          8:0:20:a8:d:7e     UH
```

```
ash              0:b0:d0:de:49:df   UH
208.44.95.0      link#1             U
208.44.95.1      0:4:27:3:94:20     UH
208.44.95.2      0:5:32:8f:d2:34    UH
208.44.95.25     0:c0:4f:10:79:92   UH

Internet6:
Destination      Gateway            Flags
default          localhost          UG
default          localhost          UG
localhost        localhost          UH
::127.0.0.0      localhost          UG
::224.0.0.0      localhost          UG
::255.0.0.0      localhost          UG
::ffff:0.0.0.0   localhost          UG
2002::           localhost          UG
2002:7f00::      localhost          UG
2002:e000::      localhost          UG
2002:ff00::      localhost          UG
fe80::           localhost          UG
fe80::%ex0       link#1             U
fe80::%ex1       link#2             U
fe80::%lo0       fe80::1%lo0        U
fec0::           localhost          UG
ff01::           localhost          U
ff02::%ex0       link#1             U
ff02::%ex1       link#2             U
ff02::%lo0       fe80::1%lo0        U
```

The flags section shows the status and whether or not it is a gateway. In this case we see U, H and G (U is up, H is host and G is gateway, see the man page for additional flags).

Now for some netstat output using the -r (routing) and -n (show network numbers) options:

```
Routing tables

Internet:
Destination        Gateway            Flags     Refs     Use    Mtu  Interface
default            208.44.95.1        UGS          0   330309   1500  ex0
127                127.0.0.1          UGRS         0        0  33228  lo0
127.0.0.1          127.0.0.1          UH           1     1624  33228  lo0
172.15.13/24       172.16.14.37       UGS          0        0   1500  ex1
172.16             link#2             UC          13        0   1500  ex1
...
Internet6:
Destination                   Gateway                   Flags     Refs     Use
  Mtu  Interface
::/104                        ::1                       UGRS         0        0
33228  lo0 =>
::/96                         ::1                       UGRS         0        0
```

The above output is a little more verbose. So, how can this help? Well, a good example is when routes between networks get changed while users are connected. I saw this happen several times when someone

was rebooting routers all day long after each change. Several users called up saying they were getting kicked out and it was taking very long to log back in. As it turned out, the clients connecting to the system were redirected to another router (which took a very long route) to reconnect. I observed the M flag or Modified dynamically (by redirect) on their connections. I deleted the routes, had them reconnect and summarily followed up with the offending technician.

18.5.4 tcpdump

Last, and definitely not least is tcpdump(8), the network sniffer that can retrieve a lot of information. In this discussion, there will be some sample output and an explanation of some of the more useful options of tcpdump.

Following is a small snippet of tcpdump in action just as it starts:

```
# tcpdump
tcpdump: listening on ex0
14:07:29.920651 mail.ssh > 208.44.95.231.3551: P 2951836801:2951836845(44) ack 2
476972923 win 17520 <nop,nop,timestamp 1219259 128519450> [tos 0x10]
14:07:29.950594 12.125.61.34 >  208.44.95.16: ESP(spi=2548773187,seq=0x3e8c) (DF)
14:07:29.983117 smtp.somecorp.com.smtp > 208.44.95.30.42828: . ack 420285166 win
16500 (DF)
14:07:29.984406 208.44.95.30.42828 > smtp.somecorp.com.smtp: . 1:1376(1375) ack 0
 win 7431 (DF)
...
```

Given that the particular server is a mail server, what is shown makes perfect sense, however, the utility is very verbose, I prefer to initially run tcpdump with no options and send the text output into a file for later digestion like so:

```
# tcpdump > tcpdump.out
tcpdump: listening on ex0
```

So, what precisely in the mish mosh are we looking for? In short, anything that does not seem to fit, for example, messed up packet lengths (as in a lot of them) will show up as improper lens or malformed packets (basically garbage). If, however, we are looking for something specific, tcpdump may be able to help depending on the problem.

18.5.4.1 Specific tcpdump Usage

These are just examples of a few things one can do with tcpdump.

Look for duplicate IP addresses:

```
tcpdump -e host ip-address
```

For example:

```
tcpdump -e host 192.168.0.2
```

Routing Problems:

```
tcpdump icmp
```

There are plenty of third party tools available, however, NetBSD comes shipped with a good tool set for tracking down network level performance problems.

18.6 Accounting

The NetBSD system comes equipped with a great deal of performance monitors for active monitoring, but what about long term monitoring? Well, of course the output of a variety of commands can be sent to files and re-parsed later with a meaningful shell script or program. NetBSD does, by default, offer some extraordinarily powerful low level monitoring tools for the programmer, administrator or really astute hobbyist.

18.6.1 Accounting

While accounting gives system usage at an almost userland level, kernel profiling with gprof provides explicit system call usage.

Using the accounting tools can help figure out what possible performance problems may be laying in wait, such as increased usage of compilers or network services for example.

Starting accounting is actually fairly simple, as root, use the accton(8) command. The syntax to start accounting is: **accton filename**

Where accounting information is appended to filename, now, strangely enough, the lastcomm command which reads from an accounting output file, by default, looks in `/var/account/acct` so I tend to just use the default location, however, lastcomm can be told to look elsewhere.

To stop accounting, simply type accton with no arguments.

18.6.2 Reading Accounting Information

To read accounting information, there are two tools that can be used:

- lastcomm(1)
- sa(8)

18.6.2.1 lastcomm

The lastcomm command shows the last commands executed in order, like all of them. It can, however, select by user, here is some sample output:

```
$ lastcomm jrf
last       -       jrf      ttyp3      0.00 secs Tue Sep  3 14:39 (0:00:00.02)
man        -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:01:49.03)
sh         -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:01:49.03)
less       -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:01:49.03)
lastcomm   -       jrf      ttyp3      0.02 secs Tue Sep  3 14:38 (0:00:00.02)
stty       -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:00:00.02)
tset       -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:00:01.05)
```

```
hostname   -       jrf      ttyp3      0.00 secs Tue Sep  3 14:38 (0:00:00.02)
ls         -       jrf      ttyp0      0.00 secs Tue Sep  3 14:36 (0:00:00.00)
...
```

Pretty nice, the lastcomm command gets its information from the default location of /var/account/acct, however, using the -f option, another file may be specified.

As may seem obvious, the output of lastcomm could get a little heavy on large multi user systems. That is where sa comes into play.

18.6.2.2 sa

The sa command (meaning "print system accounting statistics") can be used to maintain information. It can also be used interactively to create reports. Following is the default output of sa:

```
$ sa
      77       18.62re        0.02cp        8avio        0k
       3        4.27re        0.01cp       45avio        0k   ispell
       2        0.68re        0.00cp       33avio        0k   mutt
       2        1.09re        0.00cp       23avio        0k   vi
      10        0.61re        0.00cp        7avio        0k   ***other
       2        0.01re        0.00cp       29avio        0k   exim
       4        0.00re        0.00cp        8avio        0k   lastcomm
       2        0.00re        0.00cp        3avio        0k   atrun
       3        0.03re        0.00cp        1avio        0k   cron*
       5        0.02re        0.00cp       10avio        0k   exim*
      10        3.98re        0.00cp        2avio        0k   less
      11        0.00re        0.00cp        0avio        0k   ls
       9        3.95re        0.00cp       12avio        0k   man
       2        0.00re        0.00cp        4avio        0k   sa
      12        3.97re        0.00cp        1avio        0k   sh
...
```

From left to right, total times called, real time in minutes, sum of user and system time, in minutes, Average number of I/O operations per execution, size, command name.

The sa command can also be used to create summary files or reports based on some options, for example, here is the output when specifying a sort by CPU-time average memory usage:

```
$ sa -k
      86       30.81re        0.02cp        8avio        0k
      10        0.61re        0.00cp        7avio        0k   ***other
       2        0.00re        0.00cp        3avio        0k   atrun
       3        0.03re        0.00cp        1avio        0k   cron*
       2        0.01re        0.00cp       29avio        0k   exim
       5        0.02re        0.00cp       10avio        0k   exim*
       3        4.27re        0.01cp       45avio        0k   ispell
       4        0.00re        0.00cp        8avio        0k   lastcomm
      12        8.04re        0.00cp        2avio        0k   less
      13        0.00re        0.00cp        0avio        0k   ls
      11        8.01re        0.00cp       12avio        0k   man
       2        0.68re        0.00cp       33avio        0k   mutt
       3        0.00re        0.00cp        4avio        0k   sa
```

```
    14        8.03re        0.00cp         1avio        0k   sh
     2        1.09re        0.00cp        23avio        0k   vi
```

The sa command is very helpful on larger systems.

18.6.3 How to Put Accounting to Use

Accounting reports, as was mentioned earlier, offer a way to help predict trends, for example, on a system that has cc and make being used more and more may indicate that in a few months some changes will need to be made to keep the system running at an optimum level. Another good example is web server usage. If it begins to gradually increase, again, some sort of action may need to be taken before it becomes a problem. Luckily, with accounting tools, said actions can be reasonably predicted and planned for ahead of time.

18.7 Kernel Profiling

Profiling a kernel is normally employed when the goal is to compare the difference of new changes in the kernel to a previous one or to track down some sort of low level performance problem. Two sets of data about profiled code behavior are recorded independently: function call frequency and time spent in each function.

18.7.1 Getting Started

First, take a look at both Section 18.9 and Chapter 29. The only difference in procedure for setting up a kernel with profiling enabled is when you run config add the -p option. The build area is `../compile/<KERNEL_NAME>.PROF` , for example, a GENERIC kernel would be `../compile/GENERIC.PROF`.

Following is a quick summary of how to compile a kernel with profiling enabled on the i386 port, the assumptions are that the appropriate sources are available under `/usr/src` and the GENERIC configuration is being used, of course, that may not always be the situation:

1. **`cd /usr/src/sys/arch/i386/conf`**
2. **`config -p GENERIC`**
3. **`cd ../compile/GENERIC.PROF`**
4. **`make depend && make`**
5. **`cp /netbsd /netbsd.old`**
6. **`cp netbsd /`**
7. **`reboot`**

Once the new kernel is in place and the system has rebooted, it is time to turn on the monitoring and start looking at results.

18.7.1.1 Using kgmon

To start kgmon:

```
$ kgmon -b
kgmon: kernel profiling is running.
```

Next, send the data into the file `gmon.out`:

```
$ kgmon -p
```

Now, it is time to make the output readable:

```
$ gprof /netbsd > gprof.out
```

Since gmon is looking for `gmon.out`, it should find it in the current working directory.

By just running kgmon alone, you may not get the information you need, however, if you are comparing the differences between two different kernels, then a known good baseline should be used. Note that it is generally a good idea to stress the subsystem if you know what it is both in the baseline and with the newer (or different) kernel.

18.7.2 Interpretation of kgmon Output

Now that kgmon can run, collect and parse information, it is time to actually look at some of that information. In this particular instance, a GENERIC kernel is running with profiling enabled for about an hour with only system processes and no adverse load, in the fault insertion section, the example will be large enough that even under a minimal load detection of the problem should be easy.

18.7.2.1 Flat Profile

The flat profile is a list of functions, the number of times they were called and how long it took (in seconds). Following is sample output from the quiet system:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ns/call  ns/call  name
 99.77    163.87   163.87                             idle
  0.03    163.92     0.05      219 228310.50 228354.34  _wdc_ata_bio_start
  0.02    163.96     0.04      219 182648.40 391184.96  wdc_ata_bio_intr
  0.01    163.98     0.02     3412  5861.66  6463.02  pmap_enter
  0.01    164.00     0.02      548 36496.35 36496.35  pmap_zero_page
  0.01    164.02     0.02                             Xspllower
  0.01    164.03     0.01   481968    20.75    20.75  gettick
  0.01    164.04     0.01     6695  1493.65  1493.65  VOP_LOCK
  0.01    164.05     0.01     3251  3075.98 21013.45  syscall_plain
...
```

As expected, idle was the highest in percentage, however, there were still some things going on, for example, a little further down there is the *vn_lock* function:

```
...
  0.00    164.14     0.00     6711     0.00     0.00  VOP_UNLOCK
  0.00    164.14     0.00     6677     0.00  1493.65  vn_lock
  0.00    164.14     0.00     6441     0.00     0.00  genfs_unlock
```

This is to be expected, since locking still has to take place, regardless.

18.7.2.2 Call Graph Profile

The call graph is an augmented version of the flat profile showing subsequent calls from the listed functions. First, here is some sample output:

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.01% of 164.14 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     99.8  163.87    0.00                 idle [1]
-----------------------------------------------
                                                 <spontaneous>
[2]      0.1    0.01    0.08                 syscall1 [2]
                0.01    0.06    3251/3251        syscall_plain [7]
                0.00    0.01     414/1660        trap [9]
-----------------------------------------------
                0.00    0.09     219/219         Xintr14 [6]
[3]      0.1    0.00    0.09     219          pciide_compat_intr [3]
                0.00    0.09     219/219         wdcintr [5]
-----------------------------------------------
...
```

Now this can be a little confusing. The index number is mapped to from the trailing number on the end of the line, for example,

```
...
                0.00    0.01      85/85          dofilewrite [68]
[72]     0.0    0.00    0.01      85           soo_write [72]
                0.00    0.01      85/89          sosend [71]
...
```

Here we see that dofilewrite was called first, now we can look at the index number for 64 and see what was happening there:

```
...
                0.00    0.01     101/103         ffs_full_fsync <cycle 6> [58]
[64]     0.0    0.00    0.01     103          bawrite [64]
                0.00    0.01     103/105         VOP_BWRITE [60]
...
```

And so on, in this way, a "visual trace" can be established.

At the end of the call graph right after the terms section is an index by function name which can help map indexes as well.

18.7.3 Putting it to Use

In this example, I have modified an area of the kernel I know will create a problem that will be blatantly obvious.

Here is the top portion of the flat profile after running the system for about an hour with little interaction from users:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 93.97    139.13   139.13                             idle
  5.87    147.82     8.69       23 377826.09 377842.52  check_exec
  0.01    147.84     0.02      243    82.30    82.30  pmap_copy_page
  0.01    147.86     0.02      131   152.67   152.67  _wdc_ata_bio_start
  0.01    147.88     0.02      131   152.67   271.85  wdc_ata_bio_intr
  0.01    147.89     0.01     4428     2.26     2.66  uvn_findpage
  0.01    147.90     0.01     4145     2.41     2.41  uvm_pageactivate
  0.01    147.91     0.01     2473     4.04  3532.40  syscall_plain
  0.01    147.92     0.01     1717     5.82     5.82  i486_copyout
  0.01    147.93     0.01     1430     6.99    56.52  uvm_fault
  0.01    147.94     0.01     1309     7.64     7.64  pool_get
  0.01    147.95     0.01      673    14.86    38.43  genfs_getpages
  0.01    147.96     0.01      498    20.08    20.08  pmap_zero_page
  0.01    147.97     0.01      219    45.66    46.28  uvm_unmap_remove
  0.01    147.98     0.01      111    90.09    90.09  selscan
...
```

As is obvious, there is a large difference in performance. Right off the bat the idle time is noticeably less. The main difference here is that one particular function has a large time across the board with very few calls. That function is *check_exec*. While at first, this may not seem strange if a lot of commands had been executed, when compared to the flat profile of the first measurement, proportionally it does not seem right:

```
...
  0.00    164.14     0.00       37     0.00 62747.49  check_exec
...
```

The call in the first measurement is made 37 times and has a better performance. Obviously something in or around that function is wrong. To eliminate other functions, a look at the call graph can help, here is the first instance of *check_exec*

```
...
-----------------------------------------------
                0.00    8.69      23/23          syscall_plain [3]
[4]      5.9    0.00    8.69      23         sys_execve [4]
```

```
                8.69    0.00      23/23          check_exec [5]
                0.00    0.00      20/20          elf32_copyargs [67]
...
```

Notice how the time of 8.69 seems to affect the two previous functions. It is possible that there is something wrong with them, however, the next instance of *check_exec* seems to prove otherwise:

```
...
-----------------------------------------------
                8.69    0.00      23/23          sys_execve [4]
[5]      5.9    8.69    0.00      23         check_exec [5]
...
```

Now we can see that the problem, most likely, resides in *check_exec*. Of course, problems are not always this simple and in fact, here is the simpleton code that was inserted right after *check_exec* (the function is in `sys/kern/kern_exec.c`):

```
...
        /* A Cheap fault insertion */
        for (x = 0; x < 100000000; x++) {
                y = x;
        }
..
```

Not exactly glamorous, but enough to register a large change with profiling.

18.7.4 Summary

Kernel profiling can be enlightening for anyone and provides a much more refined method of hunting down performance problems that are not as easy to find using conventional means, it is also not nearly as hard as most people think, if you can compile a kernel, you can get profiling to work.

18.8 System Tuning

Now that monitoring and analysis tools have been addressed, it is time to look into some actual methods. In this section, tools and methods that can affect how the system performs that are applied without recompiling the kernel are addressed, the next section examines kernel tuning by recompiling.

18.8.1 Using sysctl

The sysctl utility can be used to look at and in some cases alter system parameters. There are so many parameters that can be viewed and changed they cannot all be shown here, however, for the first example here is a simple usage of sysctl to look at the system PATH environment variable:

```
$ sysctl user.cs_path
user.cs_path = /usr/bin:/bin:/usr/sbin:/sbin:/usr/pkg/bin:/usr/pkg/sbin:/usr/local/bin:/usr/
```

Fairly simple. Now something that is actually related to performance. As an example, lets say a system with many users is having file open issues, by examining and perhaps raising the kern.maxfiles parameter the problem may be fixed, but first, a look:

```
$ sysctl kern.maxfiles
kern.maxfiles = 1772
```

Now, to change it, as root with the -w option specified:

```
# sysctl -w kern.maxfiles=1972
kern.maxfiles: 1772 -> 1972
```

Note, when the system is rebooted, the old value will return, there are two cures for this, first, modify that parameter in the kernel and recompile, second (and simpler) add this line to `/etc/sysctl.conf`:

```
kern.maxfiles=1972
```

18.8.2 memfs & softdeps

An operating system can often benefit from a few configuration changes (along the same lines, it can also be of great detriment). Two particular cases where system performance can be changed are by using memory based filesystems and/or soft updates.

18.8.2.1 Using memfs

When to use and not to use the memory based filesystem can be hard on large multi user systems. In some cases, however, it makes pretty good sense, for example, on a development machine used by only one developer at a time, the obj directory might be a good place, or some of the tmp directories for builds. In a case like that, it makes sense on machines that have a fair amount of RAM on them. On the other side of the coin, if a system only has 16MB of RAM and `/var/tmp` is memfs based, there could be severe applications issues that occur.

The GENERIC kernel has memfs enabled by default. To use it on a particular directory first determine where the swap space is that you wish to use, in the example case, a quick look in `/etc/fstab` indicates that `/dev/wd0b` is the swap partition:

```
mail% cat /etc/fstab
/dev/wd0a / ffs rw 1 1
/dev/wd0b none swap sw 0 0
/kern /kern kernfs rw
```

This system is a mail server so I only want to use `/tmp` with memfs, also on this particular system I have linked `/tmp` to `/var/tmp` to save space (they are on the same drive). All I need to do is add the following entry:

```
/dev/wd0b /var/tmp mfs rw 0 0
```

Now, a word of warning, make sure said directories are empty and nothing is using them when you mount the memory file system! At this point I can either **mount -a** or reboot the system.

18.8.2.2 Using softdeps

Soft-dependencies is a mechanism that does not write meta-data to disk immediately, but it is written in an ordered fashion, which keeps the filesystem consistent. Section 5.9 describes how to enable soft-dependencies. A great deal more information about softdep capabilities can be found on the author's page (http://www.mckusick.com/softdep/index.html).

18.9 Kernel Tuning

While many system parameters can be changed with sysctl, many improvements by using enhanced system software, layout of the system and managing services (moving them in and out of inetd for example) can be achieved as well. Tuning the kernel however will provide better performance, even if it appears to be marginal.

18.9.1 Preparing to Recompile a Kernel

First, get the kernel sources for the release as described in Chapter 27, reading Chapter 29 for more information on building the kernel is recommended. Note, this document can be used for -current tuning, however, a read of the Tracking -current (http://www.NetBSD.org/Documentation/current/) documentation should be done first, much of the information there is repeated here.

18.9.2 Configuring the Kernel

Configuring a kernel in NetBSD can be daunting. This is because of multiple line dependencies within the configuration file itself, however, there is a benefit to this method and that is, all it really takes is an ASCII editor to get a new kernel configured and some dmesg output. The kernel configuration file is under `src/sys/arch/ARCH/conf` where ARCH is your architecture (for example, on a SPARC it would be under `src/sys/arch/sparc/conf`).

After you have located your kernel config file, copy it and remove (comment out) all the entries you don't need. This is where dmesg(8) becomes your friend. A clean dmesg(8)-output will show all of the devices detected by the kernel at boot time. Using dmesg(8) output, the device options really needed can be determined. For some automation, check the "adjustkernel" package.

18.9.2.1 Some example Configuration Items

In this example, an ftp server's kernel is being reconfigured to run with the bare minimum drivers and options and any other items that might make it run faster (again, not necessarily smaller, although it will be). The first thing to do is take a look at some of the main configuration items. So, in `/usr/src/sys/arch/i386/conf` the GENERIC file is copied to FTP, then the file FTP edited.

At the start of the file there are a bunch of options beginning with maxusers, which will be left alone, however, on larger multi-user systems it might be help to crank that value up a bit. Next is CPU support, looking at the dmesg output this is seen:

```
cpu0: Intel Pentium II/Celeron (Deschutes) (686-class), 400.93 MHz
```

Indicating that only the options I686_CPU options needs to be used. In the next section, all options are left alone except the PIC_DELAY which is recommended unless it is an older machine. In this case it is enabled since the 686 is “relatively new.”

Between the last section all the way down to compat options there really was no need to change anything on this particular system. In the compat section, however, there are several options that do not need to be enabled, again this is because this machine is strictly a FTP server, all compat options were turned off.

The next section is File systems, and again, for this server very few need to be on, the following were left on:

```
# File systems
file-system     FFS             # UFS
file-system     LFS             # log-structured file system
file-system     MFS             # memory file system
file-system     CD9660          # ISO 9660 + Rock Ridge file system
file-system     FDESC           # /dev/fd
file-system     KERNFS          # /kern
file-system     NULLFS          # loopback file system
file-system     PROCFS          # /proc
file-system     UMAPFS          # NULLFS + uid and gid remapping
...
options         SOFTDEP         # FFS soft updates support.
...
```

Next comes the network options section. The only options left on were:

```
options         INET            # IP + ICMP + TCP + UDP
options         INET6           # IPV6
options         IPFILTER_LOG    # ipmon(8) log support
```

IPFILTER_LOG is a nice one to have around since the server will be running ipf.

The next section is verbose messages for various subsystems, since this machine is already running and had no major problems, all of them are commented out.

18.9.2.2 Some Drivers

The configurable items in the config file are relatively few and easy to cover, however, device drivers are a different story. In the following examples, two drivers are examined and their associated “areas” in the file trimmed down. First, a small example: the cdrom, in dmesg, is the following lines:

```
...
cd0 at atapibus0 drive 0: <CD-540E, , 1.0A> type 5 cdrom removable
cd0: 32-bit data port
cd0: drive supports PIO mode 4, DMA mode 2, Ultra-DMA mode 2
pciide0: secondary channel interrupting at irq 15
cd0(pciide0:1:0): using PIO mode 4, Ultra-DMA mode 2 (using DMA data transfer
...
```

Now, it is time to track that section down in the configuration file. Notice that the "cd"-drive is on an atapibus and requires pciide support. The section that is of interest in this case is the kernel config’s "IDE and related devices" section. It is worth noting at this point, in and around the IDE section are also ISA,

PCMCIA etc., on this machine in the dmesg(8) output there are no PCMCIA devices, so it stands to reason that all PCMCIA references can be removed. But first, the "cd" drive.

At the start of the IDE section is the following:

```
...
wd*     at atabus? drive ? flags 0x0000
...
atapibus* at atapi?
...
```

Well, it is pretty obvious that those lines need to be kept. Next is this:

```
...
# ATAPI devices
# flags have the same meaning as for IDE drives.
cd*     at atapibus? drive ? flags 0x0000       # ATAPI CD-ROM drives
sd*     at atapibus? drive ? flags 0x0000       # ATAPI disk drives
st*     at atapibus? drive ? flags 0x0000       # ATAPI tape drives
uk*     at atapibus? drive ? flags 0x0000       # ATAPI unknown
...
```

The only device type that was in the dmesg(8) output was the cd, the rest can be commented out.

The next example is slightly more difficult, network interfaces. This machine has two of them:

```
...
ex0 at pci0 dev 17 function 0: 3Com 3c905B-TX 10/100 Ethernet (rev. 0x64)
ex0: interrupting at irq 10
ex0: MAC address 00:50:04:83:ff:b7
UI 0x001018 model 0x0012 rev 0 at ex0 phy 24 not configured
ex1 at pci0 dev 19 function 0: 3Com 3c905B-TX 10/100 Ethernet (rev. 0x30)
ex1: interrupting at irq 11
ex1: MAC address 00:50:da:63:91:2e
exphy0 at ex1 phy 24: 3Com internal media interface
exphy0: 10baseT, 10baseT-FDX, 100baseTX, 100baseTX-FDX, auto
...
```

At first glance it may appear that there are in fact three devices, however, a closer look at this line:

```
exphy0 at ex1 phy 24: 3Com internal media interface
```

Reveals that it is only two physical cards, not unlike the cdrom, simply removing names that are not in dmesg will do the job. In the beginning of the network interfaces section is:

```
...
# Network Interfaces

# PCI network interfaces
an*     at pci? dev ? function ?        # Aironet PC4500/PC4800 (802.11)
bge*    at pci? dev ? function ?        # Broadcom 570x gigabit Ethernet
en*     at pci? dev ? function ?        # ENI/Adaptec ATM
ep*     at pci? dev ? function ?        # 3Com 3c59x
epic*   at pci? dev ? function ?        # SMC EPIC/100 Ethernet
esh*    at pci? dev ? function ?        # Essential HIPPI card
```

```
ex*     at pci? dev ? function ?        # 3Com 90x[BC]
...
```

There is the ex device. So all of the rest under the PCI section can be removed. Additionally, every single line all the way down to this one:

```
exphy*  at mii? phy ?                   # 3Com internal PHYs
```

can be commented out as well as the remaining.

18.9.2.3 Multi Pass

When I tune a kernel, I like to do it remotely in an X windows session, in one window the dmesg output, in the other the config file. It can sometimes take a few passes to rebuild a very trimmed kernel since it is easy to accidentally remove dependencies.

18.9.3 Building the New Kernel

Now it is time to build the kernel and put it in place. In the conf directory on the ftp server, the following command prepares the build:

```
$ config FTP
```

When it is done a message reminding me to make depend will display, next:

```
$ cd ../compile/FTP
$ make depend && make
```

When it is done, I backup the old kernel and drop the new one in place:

```
# cp /netbsd /netbsd.orig
# cp netbsd /
```

Now reboot. If the kernel cannot boot, stop the boot process when prompted and type **`boot netbsd.orig`** to boot from the previous kernel.

18.9.4 Shrinking the NetBSD kernel

When building a kernel for embedded systems, it's often necessary to modify the Kernel binary to reduce space or memory footprint.

18.9.4.1 Removing ELF sections and debug information

We already know how to remove Kernel support for drivers and options that you don't need, thus saving memory and space, but you can save some KiloBytes of space by removing debugging symbols and two ELF sections if you don't need them: `.comment` and `.ident`. They are used for storing RCS strings viewable with ident(1) and a gcc(1) version string. The following examples assume you have your `TOOLDIR` under `/usr/src/tooldir.NetBSD-2.0-i386` and the target architecture is `i386`.

```
$ /usr/src/tooldir.NetBSD-2.0-i386/bin/i386--netbsdelf-objdump -h /netbsd

/netbsd:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0057a374  c0100000  c0100000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00131433  c067a380  c067a380  0057b380  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .rodata.str1.1 00035ea0  c07ab7b3  c07ab7b3  006ac7b3  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .rodata.str1.32 00059d13  c07e1660  c07e1660  006e2660  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 link_set_malloc_types 00000198  c083b374  c083b374  0073c374  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 link_set_domains 00000024  c083b50c  c083b50c  0073c50c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 link_set_pools 00000158  c083b530  c083b530  0073c530  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 link_set_sysctl_funcs 000000f0  c083b688  c083b688  0073c688  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 link_set_vfsops 00000044  c083b778  c083b778  0073c778  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 link_set_dkwedge_methods 00000004  c083b7bc  c083b7bc  0073c7bc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 link_set_bufq_strats 0000000c  c083b7c0  c083b7c0  0073c7c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 link_set_evcnts 00000030  c083b7cc  c083b7cc  0073c7cc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 12 .data         00048ae4  c083c800  c083c800  0073c800  2**5
                  CONTENTS, ALLOC, LOAD, DATA
 13 .bss          00058974  c0885300  c0885300  00785300  2**5
                  ALLOC
 14 .comment      0000cda0  00000000  00000000  00785300  2**0
                  CONTENTS, READONLY
 15 .ident        000119e4  00000000  00000000  007920a0  2**0
                  CONTENTS, READONLY
```

On the third column we can see the size of the sections in hexadecimal form. By summing `.comment` and `.ident` sizes we know how much we're going to save with their removal: around 120KB (= 52640 + 72164 = 0xcda0 + 0x119e4). To remove the sections and debugging symbols that may be present, we're going to use strip(1):

```
# cp /netbsd /netbsd.orig
# /usr/src/tooldir.NetBSD-2.0-i386/bin/i386--netbsdelf-strip -S -R .ident -R .comment /netbsd
# ls -l /netbsd /netbsd.orig
-rwxr-xr-x  1 root  wheel  8590668 Apr 30 15:56 netbsd
-rwxr-xr-x  1 root  wheel  8757547 Apr 30 15:56 netbsd.orig
```

Since we also removed debugging symbols, the total amount of disk space saved is around 160KB.

18.9.4.2 Compressing the Kernel

On some architectures, the bootloader can boot a compressed kernel. You can save several MegaBytes of disk space by using this method, but the bootloader will take longer to load the Kernel.

```
# cp /netbsd /netbsd.plain
# gzip -9 /netbsd
```

To see how much space we've saved:

```
$ ls -l /netbsd.plain /netbsd.gz
-rwxr-xr-x  1 root  wheel  8757547 Apr 29 18:05 /netbsd.plain
-rwxr-xr-x  1 root  wheel  3987769 Apr 29 18:05 /netbsd.gz
```

Note that you can only use gzip coding, by using gzip(1), bzip2 is not supported by the NetBSD bootloaders!

Chapter 19
NetBSD Veriexec subsystem

Veriexec is NetBSD's file integrity subsystem. It's kernel based, hence can provide some protection even in the case of a root compromise. This chapter applies only to NetBSD 3.0 and ownards.

19.1 How it works

Veriexec works by loading a specification file, also called the **signatures file**, to the kernel. This file contains information about files Veriexec should monitor, as well as their digital fingerprint (along with the hashing algorithm used to produce this fingerprint), and various flags that will be discussed later.

At the moment, the following hashing algorithms are supported by Veriexec: **MD5**, **SHA1**, **SHA256**, **SHA384**, **SHA512**, and **RMD160**.

19.2 Signatures file

An entry in the Veriexec signatures file looks like this:

```
/path/to/file algorithm fingerprint flags
```

Where the first element, the path, must always be an absolute path. The algorithm is one of the algorithms listed above, and fingerprint is the ASCII fingerprint.

19.3 Generating fingerprints

You can generate ASCII fingerprints for each algorithm using the following tools:

Table 19-1. Veriexec fingerprints tools

Algorithm	Tool
MD5	`/usr/bin/cksum -a md5`
SHA1	`/usr/bin/cksum -a sha1`
SHA256	`/usr/bin/cksum -a sha256`
SHA384	`/usr/bin/cksum -a sha384`
SHA512	`/usr/bin/cksum -a sha512`
RMD160	`/usr/bin/cksum -a rmd160`

For example, to generate a MD5 fingerprint for `/bin/ls`:

```
% cksum -a md5 < /bin/ls
a8b525da46e758778564308ed9b1e493
```

And to generate a SHA512 fingerprint for `/bin/ps`:

```
% cksum -a sha512 < /bin/ps
381d4ad64fd47800897446a2026eca42151e03adeae158db5a34d12c529559113d928a9fef9a7c4615d257688d1
```

Each entry may be associated with zero or more flags. Currently, these flags indicate how the file the entry is describing should be accessed. Note that this access type is enforced only in strict level 2 (IPS mode) and above.

The access types you can use are “DIRECT”, “INDIRECT”, and “FILE”.

- **DIRECT** access means that the file is executed directly, and not invoked as an interpreter for some script, or opened with an editor. Usually, most programs you use will be accessed using this mode:

  ```
  % ls /tmp
  % cp ~/foo /tmp/bar
  % rm ~/foo
  ```

- **INDIRECT** access means that the file is executed indirectly, and is invoked to interpret a script. This happens usually when scripts have a #! magic as their first line. For example, if you have a script with the following as its first line:

  ```
  #!/bin/sh
  ```

  And you run it as:

  ```
  % ./script.sh
  ```

  Then `/bin/sh` will be executed indirectly -- it will be invoked to interpret the script.

- **FILE** entries refer to everything which is not (or should not) be an executable. This includes shared libraries, configuration files, etc.

  Some examples for Veriexec signature file entries:

  ```
  /bin/ls         MD5 dc2e14dc84bdefff4bf9777958c1b20b DIRECT
  /usr/bin/perl   MD5 914aa8aa47ebd79ccd7909a09ed61f81 INDIRECT
  /etc/pf.conf    MD5 950e1dd6fcb3f27df1bf6accf7029f7d FILE
  ```

Veriexec allows you to specify more than one way to access a file in an entry. For example, even though `/usr/bin/perl` is mostly used as an interpreter, it may be desried to be able to execute it directly, too:

```
/usr/bin/perl MD5 914aa8aa47ebd79ccd7909a09ed61f81 DIRECT, INDIRECT
```

Shell scripts using #! magic to be “executable” also require two access types: We need them to be “DIRECT” so we can execute them, and we need them to be “FILE” so that the kernel can feed their contents to the interpreter they define:

```
/usr/src/build.sh MD5 e80dbb4c047ecc1d84053174c1e9264a DIRECT, FILE
```

To make it easier to create signature files, and to make the signature files themselves more readable, Veriexec allows you to use the following aliases:

Table 19-2. Veriexec access type aliases

Alias	Expansion
PROGRAM	DIRECT
INTERPRETER	INDIRECT
SCRIPT	DIRECT, FILE
LIBRARY	FILE

Sample scripts for generating fingerprints are available in `/usr/share/examples/veriexecctl`. After you've generated a signatures file, you should save it as `/etc/signatures`, and enable Veriexec in `rc.conf`:

```
veriexec=YES
```

19.4 Strict levels

Since different people might want to use Veriexec for different purposes, we also define four strict levels, ranging 0-4, and named "learning", "IDS", "IPS", and "lockdown" modes.

In **strict level 0**, learning mode, Veriexec will act passively and simply warn about any anomalies. Combined with verbose level 1, running the system in this mode can help you fine-tune the signatures file. This is also the only strict level in which you can load new entries to the kernel.

Strict level 1, or IDS mode, will deny access to files with a fingerprint mismatch. This mode suits mostly to users who simply want to prevent access to files which might've been maliciously modified by an attacker.

Strict level 2, IPS mode, takes a step towards trying to protect the integrity of monitored files. In addition to preventing access to files with a fingerprint mismatch, it will also deny write access and prevent the removal of monitored files, and enforce the way monitored files are accessed. (as the signatures file specifies).

Lockdown mode (**strict level 3**) can be used in highly critical situations such as custom made special-purpose machines, or as a last line of defense after an attacker compromised the system and we want to prevent traces from being removed, so we can perform post-mortem analysis. It will prevent the creation of new files, and deny access to files not monitored by Veriexec.

It's recommended to first run Veriexec in strict level 0 and verbose level 1 to fine-tune your signatures file, ensuring that desired applications run correctly, and only then raise the strict level (and lower the verbosity level). You can use `/etc/sysctl.conf` to auto raise the strict level to the desired level after a reboot:

```
kern.veriexec.strict=1
```

19.5 Kernel configuration

To use Veriexec, aside from creating a signatures file, you should enable it in your kernel's config file: (e.g. `/usr/src/sys/arch/i386/conf/GENERIC.local`):

```
options VERIFIED_EXEC
```

Then, you need to enable the hashing algorithms you wish to support:

```
options VERIFIED_EXEC_FP_MD5
options VERIFIED_EXEC_FP_SHA1
options VERIFIED_EXEC_FP_RMD160
options VERIFIED_EXEC_FP_SHA512
options VERIFIED_EXEC_FP_SHA384
options VERIFIED_EXEC_FP_SHA256
```

And finally, enable the Veriexec pseudo device:

```
pseudo-device veriexec 1
```

All in the same file. Once done, rebuild and reinstall your kernel.

Chapter 20
Miscellaneous operations

This chapter collects various topics, in sparse order, which didn't find a place in the previous chapters.

20.1 Creating a custom install/boot floppies for i386

Sometimes you may want to create your own boot/install floppies for i386 instead use the precompiled ones, or tailor the ones built by the NetBSD build system. This section outlines the steps to do so.

The overall idea is to have a filesystem with some tools (sysinst, ls, whatever), and embed this filesystem as some sort of ramdisk into a NetBSD kernel. The kernel needs to include the "md" pseudo device to be able to hold a ramdisk. The kernel with the ramdisk can then be put on

To perform the following steps, you need to be running a kernel with the *vnd* pseudo device enabled (this is the default for a GENERIC kernel).

1. First, you must create a valid kernel to put on your floppies, e.g. INSTALL. This kernel must include the "md" pseudo device, which allows embedding a ramdisk later. See Chapter 29 for kernel building instructions.
2. The next step is to create the ramdisk that gets embedded into the kernel. The ramdisk contains a filesystem with whatever tools are needed, usually init(8) and some tools like sysinst, ls(1), etc. To create the standard ramdisk, go to `/usr/src/distrib/i386/floppies/ramdisk-big` and run **make**.

   This will create the `ramdisk.fs` file in the directory. If you want to customize the contents of the filesystem, customize the `list` file.
3. Now, the ramdisk gets inserted into the kernel, producing a new kernel which includes the ramdisk, all in one file. To do so, change into the `/usr/src/distrib/i386/floppies/instkernel` directory and run **make**.
4. The next step is to make one or more floppy images, depending on the size of the kernel (including the ramdisk). This is done by changing into `/usr/src/distrib/i386/floppies/bootfloppy-big`, and running **make** again.

   This will create one or two (depending on the size of kernel) files named `boot1.fs` and `boot2.fs`
5. Last, transfer these files to the floppies with the commands

   ```
   # dd if=boot1.fs of=/dev/fd0a bs=36b
   # dd if=boot2.fs of=/dev/fd0a bs=36b
   ```
6. Put the first floppy in the drive and power on!

20.2 Synchronizing the system clock with NTP

It is not unusual to find that the system clock is wrong, often by several minutes: for some strange reason it seems that computer clocks are not very accurate. The problem gets worse if you administer many networked hosts: keeping the clocks in sync can easily become a nightmare. To solve this problem, the NTP protocol (version 3) comes to our aid: this protocol can be used to synchronize the clocks of a network of workstations using one or more NTP servers.

Thanks to the NTP protocol it is possible to adjust the clock of a single workstation but also to synchronize an entire network. The NTP protocol is quite complex, defining a hierarchical master-slave structure of servers divided in strata: the top of the hierarchy is occupied by stratum 1 servers, connected to an external clock (ex. a radio clock) to guarantee a high level of accuracy. Underneath, stratum 2 servers synchronize their clocks with stratum 1, and so on. The accuracy decreases as we proceed towards lower levels. This hierarchical structure avoids the congestion which could be caused by having all hosts refer to the same (few) stratum 1 servers. If, for example, you want to synchronize a network, you don't connect all the hosts to the same public stratum 1 server. Instead, you create a local server which connects to the main server and the remaining hosts synchronize their clocks with the local server.

Fortunately, to use the NTP tools you don't need to understand the details of the protocol and of its implementation (if you are interested, refer to RFC 1305) and you only need to know how to configure and start some programs. The base system of NetBSD already contains the necessary tools to utilize this protocol (and other time related protocols, as we'll see), derived from the xntp implementation. This section describes a simple method to always have a correct system time.

First, it is necessary to find the address of the public NTP servers to use as a reference; a detailed listing can be found at http://ntp.isc.org/bin/view/Servers/WebHome. As an example, for Italy the two stratum 1 servers ntp1.ien.it and ntp2.ien.it can be used.

Next, to adjust the system clock give the following command as root:

```
# ntpdate -b ntp1.ien.it ntp2.ien.it
```

(substitute the names of the servers in the example with the ones that you are actually using. Option `-b` tells **ntpdate** to set the system time with the settimeofday system call, instead of slewing it with adjtime (the default). This option is suggested when the difference between the local time and the correct time can be considerable.

As you've seen, ntpdate is not difficult to use. The next step is to start it automatically, in order to always have the correct system time. If you have a permanent connection to the Internet, you can start the program at boot with the following line of `/etc/rc.conf`:

```
ntpdate=YES       ntpdate_hosts="ntp1.ien.it"
```

The name of the NTP server to use is specified in the `ntpdate_hosts` variable; if you leave this field empty, the boot script will try to extract the name from the `/etc/ntp.conf` file.

If you don't have a permanent Internet connection (ex. you have a dial-up modem connection through an ISP) you can start ntpdate from the `ip-up` script, as explained in Chapter 22. In this case add the following line to the `ip-up` script:

```
/usr/sbin/ntpdate -s -b ntp1.ien.it
```

(the path is mandatory or the script will probably not find the executable). Option `-s` diverts logging output from the standard output (this is the default) to the system syslog(3) facility, which means that the

messages from ntpdate will usually end up in `/var/log/messages`.

Besides ntpdate there are other useful NTP commands. It is also possible to turn one of the local hosts into an NTP server for the remaining hosts of the network. The local server will synchronize its clock with a public server. For this type of configuration you must use the **ntpd** daemon and create the `/etc/ntp.conf` configuration file. For example:

```
server ntp1.ien.it
server ntp2.ien.it
```

ntpd can be started too from `rc.conf`, using the relevant option:

```
ntpd=YES
```

NTP is not your only option if you want to synchronize your network: you can also use the timed daemon or the rdate(8) command as well. timed was developed for 4.3BSD.

Timed too uses a master-slave hierarchy: when started on a host, timed asks the network time to a master and adjusts the local clock accordingly. A mixed structure, using both timed and ntpd can be used. One of the local hosts gets the correct time from a public NTP server and is the timed master for the remaining hosts of network, which become its clients and synchronize their clocks using timed. This means that the local server must run both NTP and timed; care must be taken that they don't interfere with each other (timed must be started with the `-F hostname` option so that it doesn't try to adjust the local clock).

Finally, rdate(8) can be used to synchronize once against a given host, much like ntpdate(8). The host in question must have the "time" service (port 37) enabled in `/etc/inetd.conf`.

20.3 Installing the boot manager

Sysinst, the NetBSD installation program usually installs the NetBSD boot manager on the hard disk. The boot manager can also be installed or reconfigured at a later time, if needed, with the **fdisk** command. For example:

```
# fdisk -B wd0
```

If NetBSD doesn't boot from the hard disk, you can boot it from the installation floppy and start the kernel on the hard disk. Insert the installation disk and, at the boot prompt, give the following command:

```
> boot wd0a:netbsd
```

This boots the kernel on the hard disk (use the correct device, for example sd0a for a SCSI disk).

> **Note:** Sometimes **fdisk -B** doesn't give the expected result (at least it happened to me), probably if you install/remove other operating systems like Windows 95 or Linux with LILO. In this case, try running **fdisk -i** (which is known as **fdisk /mbr** from DOS) and then run again **fdisk** from NetBSD.

20.4 Deleting the disklabel

Though this is not an operation that you need to perform frequently, it can be useful to know how to do it in case of need. Please be sure to know exactly what you are doing before performing this kind of operation. For example:

```
# dd if=/dev/zero of=/dev/rwd0c bs=8k count=1
```

The previous command deletes the disklabel (not the MBR partition table). To completely delete the disk, the `wd0d` device must be used. For example:

```
# dd if=/dev/zero of=/dev/rwd0d bs=8k
```

20.5 Speaker

I found this tip on a mailing list (I don't remember the author). To output a sound from the speaker (for example at the end of a long script) the *spkr* driver can be used in the kernel config, which is mapped on `/dev/speaker`. For example:

```
echo 'BPBPBPBPBP' > /dev/speaker
```

Note: The *spkr* device is not enabled in the generic kernel; a customized kernel is needed.

20.6 Forgot root password?

If you forget root's password, not all is lost and you can still "recover" the system with the following steps: boot single user, mount / and change root's password. In detail:

1. Boot single user: when the boot prompt appears and the five seconds countdown starts, give the following command:

   ```
   > boot -s
   ```

2. At the following prompt

   ```
   Enter pathname of shell or RETURN for sh:
   ```

   press Enter.

3. Write the following commands:

   ```
   # fsck -y /
   # mount -u /
   # fsck -y /usr
   # mount /usr
   ```

4. Change root's password:

   ```
   # passwd root
   Changing local password for root.
   ```

```
New password: (not echoed)
Retype new password: (not echoed)
#
```

5. Exit the shell to go to multiuser mode.

```
# exit
```

20.7 Adding a new hard disk

This section describes how to add a new hard disk to an already working NetBSD system. In the following example a new SCSI controller and a new hard disk, connected to the controller, will be added. If you don't need to add a new controller, skip the relevant part and go to the hard disk configuration. The installation of an IDE hard disk is identical; only the device name will be different (`wd#` instead of `sd#`).

As always, before buying new hardware, consult the hardware compatibility list of NetBSD and make sure that the new device is supported by the system.

When the SCSI controller has been physically installed in the system and the new hard disk has been connected, it's time to restart the computer and check that the device is correctly detected, using the **dmesg** command. This is the sample output for an NCR-875 controller:

```
ncr0 at pci0 dev 15 function 0: ncr 53c875 fast20 wide scsi
ncr0: interrupting at irq 10
ncr0: minsync=12, maxsync=137, maxoffs=16, 128 dwords burst, large dma fifo
ncr0: single-ended, open drain IRQ driver, using on-chip SRAM
ncr0: restart (scsi reset).
scsibus0 at ncr0: 16 targets, 8 luns per target
sd0(ncr0:2:0): 20.0 MB/s (50 ns, offset 15)
sd0: 2063MB, 8188 cyl, 3 head, 172 sec, 512 bytes/sect x 4226725 sectors
```

If the device doesn't appear in the output, check that it is supported by the kernel that you are using; if necessary, compile a customized kernel (see Chapter 29).

Now the partitions can be created using the **fdisk** command. First, check the current status of the disk:

```
# fdisk sd0
NetBSD disklabel disk geometry:
cylinders: 8188 heads: 3 sectors/track: 172 (516 sectors/cylinder)

BIOS disk geometry:
cylinders: 524 heads: 128 sectors/track: 63 (8064 sectors/cylinder)

Partition table:
0: sysid 6 (Primary 'big' DOS, 16-bit FAT (> 32MB))
    start 63, size 4225473 (2063 MB), flag 0x0
        beg: cylinder    0, head   1, sector  1
        end: cylinder  523, head 127, sector 63
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
```

In this example the hard disk already contains a DOS partition, which will be deleted and replaced with a native NetBSD partition. The command **fdisk -u sd0** allows to modify interactively the partitions. The modified data will be written on the disk only before exiting and fdisk will request a confirmation before writing, so you can work relaxedly.

Disk geometries

The geometry of the disk reported by fdisk can appear confusing. Dmesg reports 4226725 sectors with 8188/3/172 for C/H/S, but 8188*3*172 gives 4225008 and not 4226725. What happens is that most modern disks don't have a fixed geometry and the number of sectors per track changes depending on the cylinder: the only interesting parameter is the number of sectors. The disk reports the C/H/S values but it's a fictitious geometry: the value 172 is the result of the total number of sectors (4226725) divided by 8188 and then by 3.

To make things more confusing, the BIOS uses yet another "fake" geometry (C/H/S 524/128/63) which gives a total of 4225536, a value which is a better approximation to the real one than 425008. To partition the disk we will use the BIOS geometry, to maintain compatibility with other operating systems, although we will loose some sectors (4226725 - 4225536 = 1189 sectors = 594 KB).

To create the BIOS partitions the command **fdisk -u** must be used; the result is the following:

```
Partition table:
0: sysid 169 (NetBSD)
    start 63, size 4225473 (2063 MB), flag 0x0
        beg: cylinder    0, head   1, sector  1
        end: cylinder  523, head 127, sector 63
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
```

Now it's time to create the disklabel for the NetBSD partition. The correct steps to do this are:

```
# disklabel sd0 > tempfile
# vi tempfile
# disklabel -R -r sd0 tempfile
```

If you try to create the disklabel directly with

```
# disklabel -e sd0
```

you get the following message

```
disklabel: ioctl DIOCWDINFO: No disk label on disk;
use "disklabel -I" to install initial label
```

because the disklabel does not yet exist on the disk.

Now we create some disklabel partitions, editing the `tempfile` as already explained. The result is:

```
#      size   offset   fstype [fsize bsize   cpg]
a:  2048004       63   4.2BSD   1024  8192    16 # (Cyl.  0*- 3969*)
c:  4226662       63   unused      0     0       # (Cyl.  0*- 8191*)
d:  4226725        0   unused      0     0       # (Cyl.  0 - 8191*)
e:  2178658  2048067   4.2BSD   1024  8192    16 # (Cyl.  3969*- 8191*)
```

Note: When the disklabel has been created it is possible to optimize it studying the output of the command **newfs -N /dev/sd0a**, which warns about the existence of unallocated sectors at the end of a disklabel partition. The values reported by newfs can be used to adjust the sizes of the partitions with an iterative process.

The final operation is the creation of the file systems for the newly defined partitions (*a* and *e*).

```
# newfs /dev/sd0a
# newfs /dev/sd0e
```

The disk is now ready for usage, and the two partitions can be mounted. For example:

```
# mount /dev/sd0a /mnt
```

If this succeeds, you may want to put an entry for the partition into `/etc/fstab`.

20.8 Password file is busy?

If you try to modify a password and you get the mysterious message “Password file is busy”, it probably means that the file `/etc/ptmp` has not been deleted from the system. This file is a temporary copy of the `/etc/master.passwd` file: check that you are not loosing important information and then delete it:

```
# rm /etc/ptmp
```

Note: If the file `/etc/ptmp` exists you can also receive a warning message at system startup. For example:

```
root: password file may be incorrect – /etc/ptmp exists
```

20.9 How to rebuild the devices in /dev

First shutdown to single user, partitions still mounted “rw” (read-write); You can do that by just typing **shutdown now** while you are in multi user mode, or reboot with the `-s` option and make `/` and `/dev` read-writable by doing.

```
# mount -u /
# mount -u /dev
```

Then:

```
# mkdir /newdev
# cd /newdev
# cp /dev/MAKEDEV* .
# sh ./MAKEDEV all
# cd /
# mv dev olddev
# mv newdev dev
# rm -r olddev
```

Or if you fetched all the sources in `/usr/src`:

```
# mkdir /newdev
# cd /newdev
# cp /usr/src/etc/MAKEDEV.local .
# ( cd /usr/src/etc ; make MAKEDEV )
# cp /usr/src/etc/obj*/MAKEDEV .
# sh ./MAKEDEV all
# cd /
# mv dev olddev; mv newdev dev
# rm -r olddev
```

You can determine $arch by

```
# uname -m
```

or

```
# sysctl hw.machine_arch
```

IV. Networking and related issues

Chapter 21
Introduction to TCP/IP Networking

21.1 Audience

This section explains various aspects of networking. It is intended to help people with little knowledge about networks to get started. It is divided into three big parts. We start by giving a general overview of how networking works and introduce the basic concepts. Then we go into details for setting up various types of networking in the second parts, and the third part of the networking section covers a number of “advanced” topics that go beyond the basic operation as introduced in the first two sections.

The reader is assumed to know about basic system administration tasks: how to become root, edit files, change permissions, stop processes, etc. See the other chapters of this NetBSD guide and e.g. AeleenFrisch for further information on this topic. Besides that, you should know how to handle the utilities we’re going to set up here, i.e. you should know how to use telnet, FTP, ... I will not explain the basic features of those utilities, please refer to the appropriate man-pages, the references listed or of course the other parts of this document instead.

This introduction to TCP/IP Networking was written with the intention in mind to give starters a basic knowledge. If you really want to know what it’s all about, read CraigHunt. This book does not only cover the basics, but goes on and explains all the concepts, services and how to set them up in detail. It’s great, I love it! :-)

21.2 Supported Networking Protocols

There are several protocol suites supported by NetBSD, most of which were inherited from NetBSD’s predecessor, 4.4BSD, and subsequently enhanced and improved. The first and most important one today is DARPA’s Transmission Control Protocol/Internet Protocol (TCP/IP). Other protocol suites available in NetBSD include the Xerox Network System (XNS) which was only implemented at UCB to connect isolated machines to the net, Apple’s AppleTalk protocol suite and the ISO protocol suite, CCITT X.25 and ARGO TP. They are only used in some special applications these days.

Today, TCP/IP is the most widespread protocol of the ones mentioned above. It is implemented on almost every hardware and operating system, and it is also the most-used protocol in heterogenous environments. So, if you just want to connect your computer running NetBSD to some other machine at home or you want to integrate it into your company’s or university’s network, TCP/IP is the right choice. Besides the "old" IP version 4, NetBSD also supports the "new" IP version 6 (IPv6) since NetBSD 1.5, thanks to code contributed by the KAME project.

There are other protocol suites such as DECNET, Novell's IPX/SPX or Microsoft's NetBIOS, but these are not currently supported by NetBSD. These protocols differ from TCP/IP in that they are proprietary, in contrast to the others, which are well-defined in several RFCs and other open standards.

21.3 Supported Media

The TCP/IP protocol stack behaves the same regardless of the underlying media used, and NetBSD supports a wide range of these, among them are Ethernet (10/100/1000MBd), Arcnet, serial line, ATM, FDDI, Fiber Channel, USB, HIPPI, FireWire (IEEE 1394), Token Ring, and serial lines.

21.3.1 Serial Line

There are a couple of reasons for using TCP/IP over a serial line.

- If your remote host is only reachable via telephone, you can use a modem to access it.
- Many computers have a serial port, and the cable needed is rather cheap.

The disadvantage of a serial connection is that it's slower than other methods. NetBSD can use at most 115200 bit/s, making it a lot slower than e.g. Ethernet's minimum 10 Mbit/s and Arcnet's 4 Mbit/s.

There are two possible protocols to connect a host running NetBSD to another host using a serial line (possibly over a phone-line):

- Serial Line IP (SLIP)
- Point to Point Protocol (PPP)

The choice here depends on whether you use a dial-up connection through a modem or if you use a static connection (null-modem or leased line). If you dial up for your IP connection, it's wise to use PPP as it offers some possibilities to auto-negotiate IP-addresses and routes, which can be quite painful to do by hand. If you want to connect to another machine which is directly connected, use SLIP, as this is supported by about every operating system and more easy to set up with fixed addresses and routes.

PPP on a direct connection is a bit difficult to setup, as it's easy to timeout the initial handshake; with SLIP, there's no such initial handshake, i.e. you start up one side, and when the other site has its first packet, it will send it over the line.

RFC1331 and RFC1332 describe PPP and TCP/IP over PPP. SLIP is defined in RFC1055.

21.3.2 Ethernet

Ethernet is the medium commonly used to build local area networks (LANs) of interconnected machines within a limited area such as an office, company or university campus. Ethernet is based on a bus structure to which many machines can connect to, and communication always happens between two nodes at a time. When two or more nodes want to talk at the same time, both will restart communication after some timeout. The technical term for this is CSMA/CD (Carrier Sense w/ Multiple Access and Collision Detection).

Initially, Ethernet hardware consisted of a thick (yellow) cable that machines tapped into using special connectors that poked through the cable's outer shielding. The successor of this was called 10base5,

which used BNC-type connectors for tapping in special T-connectors and terminators on both ends of the bus. Today, ethernet is mostly used with twisted pair lines which are used in a collapsed bus system that are contained in switches or hubs. The twisted pair lines give this type of media its name - 10baseT for 10 Mbit/s networks, and 100baseT for 100 MBit/s ones. In switched environments there's also the distinction if communication between the node and the switch can happen in half- or in full duplex mode.

21.4 TCP/IP Address Format

TCP/IP uses 4-byte (32-bit) addresses in the current implementations (IPv4), also called IP-numbers (Internet-Protocol numbers), to address hosts.

TCP/IP allows any two machines to communicate directly. To permit this all hosts on a given network must have a unique IP address. To assure this, IP addresses are administrated by one central organisation, the InterNIC. They give certain ranges of addresses (network-addresses) directly to sites which want to participate in the internet or to internet-providers, which give the addresses to their customers.

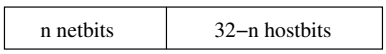
If your university or company is connected to the Internet, it has (at least) one such network-address for its own use, usually not assigned by the InterNIC directly, but rather through an Internet Service Provider (ISP).

If you just want to run your private network at home, see below on how to "build" your own IP addresses. However, if you want to connect your machine to the (real :-) Internet, you should get an IP addresses from your local network-administrator or -provider.

IP addresses are usually written in "dotted quad"-notation - the four bytes are written down in decimal (most significant byte first), separated by dots. For example, 132.199.15.99 would be a valid address. Another way to write down IP-addresses would be as one 32-bit hex-word, e.g. 0x84c70f63. This is not as convenient as the dotted-quad, but quite useful at times, too. (See below!)

Being assigned a network means nothing else but setting some of the above-mentioned 32 address-bits to certain values. These bits that are used for identifying the network are called network-bits. The remaining bits can be used to address hosts on that network, therefore they are called host-bits. Figure 21-1 illustrates the separation.

Figure 21-1. IPv4-addresses are divided into more significant network- and less significant hostbits



In the above example, the network-address is 132.199.0.0 (host-bits are set to 0 in network-addresses) and the host's address is 15.99 on that network.

How do you know that the host's address is 16 bit wide? Well, this is assigned by the provider from which you get your network-addresses. In the classless inter-domain routing (CIDR) used today, host fields are usually between as little as 2 to 16 bits wide, and the number of network-bits is written after the network address, separated by a "/", e.g. 132.199.0.0/16 tells that the network in question has 16 network-bits. When talking about the "size" of a network, it's usual to only talk about it as "/16", "/24", etc.

Before CIDR was used, there used to be four classes of networks. Each one starts with a certain bit-pattern identifying it. Here are the four classes:

- Class A starts with “0” as most significant bit. The next seven bits of a class A address identify the network, the remaining 24 bit can be used to address hosts. So, within one class A network there can be 2^{24} hosts. It’s not very likely that you (or your university, or company, or whatever) will get a whole class A address.

  The CIDR notation for a class A network with its eight network bits is an “/8”.

- Class B starts with “10” as most significant bits. The next 14 bits are used for the networks address and the remaining 16 bits can be used to address more than 65000 hosts. Class B addresses are very rarely given out today, they used to be common for companies and universities before IPv4 address space went scarce.

  The CIDR notation for a class B network with its 16 network bits is an “/16”.

  Returning to our above example, you can see that 132.199.15.99 (or 0x84c70f63, which is more appropriate here!) is on a class B network, as 0x84... = **10**00... (base 2).

  Therefore, the address 132.199.15.99 can be split into an network-address of 132.199.0.0 and a host-address of 15.99.

- Class C is identified by the MSBs being “110”, allowing only 256 (actually: only 254, see below) hosts on each of the 2^{21} possible class C networks. Class C addresses are usually found at (small) companies.

  The CIDR notation for a class C network with its 24 network bits is an “/24”.

- There are also other addresses, starting with “111”. Those are used for special purposes (e. g. multicast-addresses) and are not of interest here.

Please note that the bits which are used for identifying the network-class are part of the network-address.

When separating host-addresses from network-addresses, the “netmask” comes in handy. In this mask, all the network-bits are set to “1”, the host-bits are “0”. Thus, putting together IP-address and netmask with a logical AND-function, the network-address remains.

To continue our example, 255.255.0.0 is a possible netmask for 132.199.15.99. When applying this mask, the network-address 132.199.0.0 remains.

For addresses in CIDR notation, the number of network-bits given also says how many of the most significant bits of the address must be set to “1” to get the netmask for the corresponding network. For classful addressing, every network-class has a fixed default netmask assigned:

- Class A (/8): default-netmask: 255.0.0.0, first byte of address: 1-127
- Class B (/16): default-netmask: 255.255.0.0, first byte of address: 128-191
- Class C (/24): default-netmask: 255.255.255.0, first byte of address: 192-223

Another thing to mention here is the “broadcast-address”. When sending to this address, *all* hosts on the corresponding network will receive the message sent. The broadcast address is characterized by having all host-bits set to “1”.

Taking 132.199.15.99 with its netmask 255.255.0.0 again, the broadcast-address would result in 132.199.255.255.

You’ll ask now: But what if I want a host’s address to be all bits “0” or “1”? Well, this doesn’t work, as network- and broadcast-address must be present! Because of this, a class B (/16) network can contain at most 2^{16}-2 hosts, a class C (/24) network can hold no more than 2^{8}-2 = 254 hosts.

Besides all those categories of addresses, there's the special IP-address 127.0.0.1 which always refers to the "local" host, i.e. if you talk to 127.0.0.1 you'll talk to yourself without starting any network-activity. This is sometimes useful to use services installed on your own machine or to play around if you don't have other hosts to put on your network.

Let's put together the things we've introduced in this section:

IP-address

32 bit-address, with network- and host-bits.

Network-address

IP-address with all host bits set to "0".

Netmask

32-bit mask with "1" for network- and "0" for host-bits.

Broadcast

IP-address with all host bits set "1".

locahost's address

The local host's IP address is always 127.0.0.1.

21.5 Subnetting and Routing

After talking so much about netmasks, network-, host- and other addresses, I have to admit that this is not the whole truth.

Imagine the situation at your university, which usually has a class B (/16) address, allowing it to have up to 2^{16} ~= 65534 hosts on that net. Maybe it would be a nice thing to have all those hosts on one single network, but it's simply not possible due to limitations in the transport media commonly used today.

For example, when using thinwire ethernet, the maximum length of the cable is 185 meters. Even with repeaters in between, which refresh the signals, this is not enough to cover all the locations where machines are located. Besides that, there is a maximum number of 1024 hosts on one ethernet wire, and you'll loose quite a bit of performance if you go to this limit.

So, are you hosed now? Having an address which allows more than 60000 hosts, but being bound to media which allows far less than that limit?

Well, of course not! :-)

The idea is to divide the "big" class B net into several smaller networks, commonly called sub-networks or simply subnets. Those subnets are only allowed to have, say, 254 hosts on them (i.e. you divide one big class B network into several class C networks!).

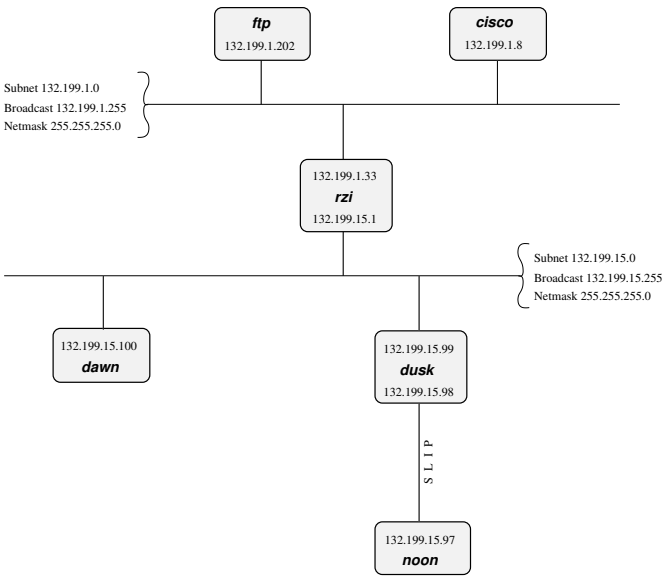
To do this, you adjust your netmask to have more network- and less host-bits on it. This is usually done on a byte-boundary, but you're not forced to do it there. So, commonly your netmask will not be 255.255.0.0 as supposed by a class B network, but it will be set to 255.255.255.0.

In CIDR notation, you now write a "/24" instead of the "/16" to show that 24 bits of the address are used for identifying the network and subnet, instead of the 16 that were used before.

This gives you one additional network-byte to assign to each (physical!) network. All the 254 hosts on that subnet can now talk directly to each other, and you can build 256 such class C nets. This should fit your needs.

To explain this better, let's continue our above example. Say our host 132.199.15.99 (I'll call him dusk from now; we'll talk about assigning hostnames later) has a netmask of 255.255.255.0 and thus is on the subnet 132.199.15.0/24. Let's furthermore introduce some more hosts so we have something to play around with, see Figure 21-2.

Figure 21-2. Our demo-network



In the above network, dusk can talk directly to dawn, as they are both on the same subnet. (There are other hosts attached to the 132.199.15.0/24-subnet but they are not of importance for us now)

But what if dusk wants to talk to a host on another subnet?

Well, the traffic will then go through one or more gateways (routers), which are attached to two subnets. Because of this, a router always has two different addresses, one for each of the subnets it is on. The router is functionally transparent, i.e. you don't have to address it to reach hosts on the "other" side. Instead, you address that host directly and the packets will be routed to it correctly.

Example. Let's say dusk wants to get some files from the local ftp-server. As dusk can't reach ftp directly (because it's on a different subnet), all its packets will be forwarded to its "defaultrouter" rzi (132.199.15.1), which knows where to forward the packets.

Dusk knows the address of its defaultrouter in its network (rzi, 132.199.15.1), and it will forward any packets to it which are not on the same subnet, i.e. it will forward all IP-packets in which the third address-byte isn't 15.

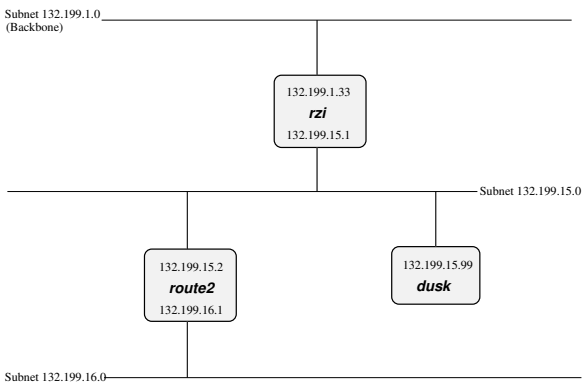
The (default)router then gives the packets to the appropriate host, as it's also on the FTP-server's network.

In this example, *all* packets are forwarded to the 132.199.1.0/24-network, simply because it's the network's backbone, the most important part of the network, which carries all the traffic that passes

between several subnets. Almost all other networks besides 132.199.15.0/24 are attached to the backbone in a similar manner.

But what if we had hooked up another subnet to 132.199.15.0/24 instead of 132.199.1.0/24? Maybe something the situation displayed in Figure 21-3.

Figure 21-3. Attaching one subnet to another one



When we now want to reach a host which is located in the 132.199.16.0/24-subnet from dusk, it won't work routing it to rzi, but you'll have to send it directly to route2 (132.199.15.2). Dusk will have to know to forward those packets to route2 and send all the others to rzi.

When configuring dusk, you tell it to forward all packets for the 132.199.16.0/24-subnet to route2, and all others to rzi. Instead of specifying this default as 132.199.1.0/24, 132.199.2.0/24, etc., 0.0.0.0 can be used to set the default-route.

Returning to Figure 21-2, there's a similar problem when dawn wants to send to noon, which is connected to dusk via a serial line running. When looking at the IP-addresses, noon seems to be attached to the 132.199.15.0-network, but it isn't really. Instead, dusk is used as gateway, and dawn will have to send its packets to dusk, which will forward them to noon then. The way dusk is forced into accepting packets that aren't destined at it but for a different host (noon) instead is called "proxy arp".

The same goes when hosts from other subnets want to send to noon. They have to send their packets to dusk (possibly routed via rzi),

21.6 Name Service Concepts

In the previous sections, when we talked about hosts, we referred to them by their IP-addresses. This was necessary to introduce the different kinds of addresses. When talking about hosts in general, it's more convenient to give them "names", as we did when talking about routing.

Most applications don't care whether you give them an IP address or a hostname. However, they'll use IP addresses internally, and there are several methods for them to map hostnames to IP addresses, each one with its own way of configuration. In this section we'll introduce the idea behind each method, in the next chapter, we'll talk about the configuration-part.

The mapping from hostnames (and domainnames) to IP-addresses is done by a piece of software called the "resolver". This is not an extra service, but some library routines which are linked to every

application using networking-calls. The resolver will then try to resolve (hence the name ;-) the hostnames you give into IP addresses. See RFC1034 and RFC1035 for details on the resolver.

Hostnames are usually up to 256 characters long, and contain letters, numbers and dashes ("-"); case is ignored.

Just as with networks and subnets, it's possible (and desirable) to group hosts into domains and subdomains. When getting your network-address, you usually also obtain a domainname by your provider. As with subnets, it's up to you to introduce subdomains. Other as with IP-addresses, (sub)domains are not directly related to (sub)nets; for example, one domain can contain hosts from several subnets.

Figure 21-2 shows this: Both subnets 132.199.1.0/24 and 132.199.15.0/24 (and others) are part of the subdomain "rz.uni-regensburg.de". The domain the University of Regensburg got from its IP-provider is "uni-regensburg.de" (".de" is for Deutschland, Germany), the subdomain "rz" is for Rechenzentrum, computing center.

Hostnames, subdomain- and domainnames are separated by dots ("."). It's also possible to use more than one stage of subdomains, although this is not very common. An example would be fox_in.socs.uts.edu.au.

A hostname which includes the (sub)domain is also called a fully qualified domain name (FQDN). For example, the IP-address 132.199.15.99 belongs to the host with the FQDN dusk.rz.uni-regensburg.de.

Further above I told you that the IP-address 127.0.0.1 always belongs to the local host, regardless what's the "real" IP-address of the host. Therefore, 127.0.0.1 is always mapped to the name "localhost".

The three different ways to translate hostnames into IP addresses are: `/etc/hosts`, the Domain Name Service (DNS) and the Network Information Service (NIS).

21.6.1 `/etc/hosts`

The first and simplest way to translate hostnames into IP-addresses is by using a table telling which IP address belongs to which hostname(s). This table is stored in the file `/etc/hosts` and has the following format:

```
IP-address        hostname [nickname [...]]
```

Lines starting with a hash mark ("#") are treated as comments. The other lines contain one IP-address and the corresponding hostname(s).

It's not possible for a hostname to belong to several IP addresses, even if I made you think so when talking about routing. rzi for example has really two distinct names for each of its two addresses: rzi and rzia (but please don't ask me which name belongs to which address!).

Giving a host several nicknames can be convenient if you want to specify your favorite host providing a special service with that name, as is commonly done with FTP-servers. The first (leftmost) name is usually the real (canonical) name of the host.

Besides giving nicknames, it's also convenient to give a host's full name (including domain) as its canonical name, and using only its hostname (without domain) as a nickname.

Important: There *must* be an entry mapping localhost to 127.0.0.1 in `/etc/hosts`!

21.6.2 Domain Name Service (DNS)

`/etc/hosts` bears an inherent problem, especially in big networks: when one host is added or one host's address changes, all the `/etc/hosts` files on all machines have to be changed! This is not only time-consuming, it's also very likely that there will be some errors and inconsistencies, leading to problems.

Another approach is to hold only one hostnames-table (-database) for a network, and make all the clients query that "nameserver". Updates will be made only on the nameserver.

This is the basic idea behind the Domain Name Service (DNS).

Usually, there's one nameserver for each domain (hence DNS), and every host (client) in that domain knows which domain it is in and which nameserver to query for its domain.

When the DNS gets a query about a host which is not in its domain, it will forward the query to a DNS which is either the DNS of the domain in question or knows which DNS to ask for the specified domain. If the DNS forwarded the query doesn't know how to handle it, it will forward that query again to a DNS one step higher. This is not ad infinitum, there are several "root"-servers, which know about any domain.

See Chapter 24 for details on DNS.

21.6.3 Network Information Service (NIS/YP)

Yellow Pages (YP) was invented by Sun Microsystems. The name has been changed into Network Information Service (NIS) because YP was already a trademark of the British telecom. So, when I'm talking about NIS you'll know what I mean. ;-)

There are quite some configuration files on a Unix-system, and often it's desired to maintain only one set of those files for a couple of hosts. Those hosts are grouped together in a NIS-domain (which has *nothing* to do with the domains built by using DNS!) and are usually contained in one workstation cluster.

Examples for the config-files shared among those hosts are `/etc/passwd`, `/etc/group` and - last but not least - `/etc/hosts`.

So, you can "abuse" NIS for getting a unique name-to-address-translation on all hosts throughout one (NIS-)domain.

There's only one drawback, which prevents NIS from actually being used for that translation: In contrast to the DNS, NIS provides no way to resolve hostnames which are not in the hosts-table. There's no hosts "one level up" which the NIS-server can query, and so the translation will fail! Suns NIS+ takes measures against that problem, but as NIS+ is only available on Solaris-systems, this is of little use for us now.

Don't get me wrong: NIS is a fine thing for managing e.g. user-information (`/etc/passwd`, ...) in workstation-clusters, it's simply not too useful for resolving hostnames.

21.6.4 Other

The name resolving methods described above are what's used commonly today to resolve hostnames into IP addresses, but they aren't the only ones. Basically, every database mechanism would do, but none is implemented in NetBSD. Let's have a quick look what you may encounter.

With NIS lacking hierarchy in data structures, NIS+ is intended to help out in that field. Tables can be setup in a way so that if a query cannot be answered by a domain's server, there can be another domain

"above" that might be able to do so. E.g. you could choose to have a domain that lists all the hosts (users, groups, ...) that are valid in the whole company, one that defines the same for each division, etc. NIS+ is not used a lot today, even Sun went back to ship back NIS by default.

Last century, the X.500 standard was designed to accommodate both simple databases like `/etc/hosts` as well as complex, hierarchical systems as can be found e.g. in DNS today. X.500 wasn't really a success, mostly due to the fact that it tried to do too much at the same time. A cut-down version is available today as the Lightweight Directory Access Protocol (LDAP), which is becoming popular in the last years to manage data like users but also hosts and others in small to medium sized organisations.

21.7 Next generation Internet protocol - IPv6

21.7.1 The Future of the Internet

According to experts, the Internet as we know it will face a serious problem in a few years. Due to its rapid growth and the limitations in its design, there will be a point at which no more free addresses are available for connecting new hosts. At that point, no more new web servers can be set up, no more users can sign up for accounts at ISPs, no more new machines can be setup to access the web or participate in online games - some people may call this a serious problem.

Several approaches have been made to solve the problem. A very popular one is to not assign a worldwide unique address to every user's machine, but rather to assign them "private" addresses, and hide several machines behind one official, globally unique address. This approach is called "Network Address Translation" (NAT, also known as IP Masquerading). It has problems, as the machines hidden behind the global address can't be addressed, and as a result of this, opening connections to them - which is used in online gaming, peer to peer networking, etc. - is not possible. For a more in-depth discussion of the drawbacks of NAT, see RFC3027.

A different approach to the problem of internet addresses getting scarce is to abandon the old Internet protocol with its limited addressing capabilities, and use a new protocol that does not have these limitations. The protocol - or actually, a set of protocols - used by machines connected to form today's Internet is know as the TCP/IP (Transmission Control Protocol, Internet Protocol) suite, and version 4 currently in use has all the problems described above. Switching to a different protocol version that does not have these problems of course requires for a 'better' version to be available, which actually is. Version 6 of the Internet Protocol (IPv6) does fulfill any possible future demands on address space, and also addresses further features such as privacy, encryption, and better support of mobile computing.

Assuming a basic understanding of how today's IPv4 works, this text is intended as an introduction to the IPv6 protocol. The changes in address formats and name resolution are covered. With the background given here, Section 26.4 will show how to use IPv6 even if your ISP doesn't offer it by using a simple yet efficient transition mechanism called 6to4. The goal is to to get online with IPv6, giving example configuration for NetBSD.

21.7.2 What good is IPv6?

When telling people to migrate from IPv4 to IPv6, the question you usually hear is "why?". There are actually a few good reasons to move to the new version:

- Bigger address space
- Support for mobile devices
- Built-in security

21.7.2.1 Bigger Address Space

The bigger address space that IPv6 offers is the most obvious enhancement it has over IPv4. While today's internet architecture is based on 32-bit wide addresses, the new version has 128 bit available for addressing. Thanks to the enlarged address space, work-arounds like NAT don't have to be used any more. This allows full, unconstrained IP connectivity for today's IP based machines as well as upcoming mobile devices like PDAs and cell phones will benefit from full IP access through GPRS and UMTS.

21.7.2.2 Mobility

When mentioning mobile devices and IP, another important point to note is that some special protocol is needed to support mobility, and implementing this protocol - called "Mobile IP" - is one of the requirements for every IPv6 stack. Thus, if you have IPv6 going, you have support for roaming between different networks, with everyone being updated when you leave one network and enter the other one. Support for roaming is possible with IPv4 too, but there are a number of hoops that need to be jumped in order to get things working. With IPv6, there's no need for this, as support for mobility was one of the design requirements for IPv6. See RFC3024 for some more information on the issues that need to be addressed with Mobile IP on IPv4.

21.7.2.3 Security

Besides support for mobility, security was another requirement for the successor to today's Internet Protocol version. As a result, IPv6 protocol stacks are required to include IPsec. IPsec allows authentication, encryption and compression of any IP traffic. Unlike application level protocols like SSL or SSH, all IP traffic between two nodes can be handled, without adjusting any applications. The benefit of this is that all applications on a machine can benefit from encryption and authentication, and that policies can be set on a per-host (or even per-network) base, not per application/service. An introduction to IPsec with a roadmap to the documentation can be found in RFC2411, the core protocol is described in RFC2401.

21.7.3 Changes to IPv4

After giving a brief overview of all the important features of IPv6, we'll go into the details of the basics of IPv6 here. A brief understanding of how IPv4 works is assumed, and the changes in IPv6 will be highlighted. Starting with IPv6 addresses and how they're split up we'll go into the various types of addresses there are, what became of broadcasts, then after discussing the IP layer go into changes for name resolving and what's new in DNS for IPv6.

21.7.3.1 Addressing

An IPv4 address is a 32 bit value, that's usually written in "dotted quad" representation, where each "quad" represents a byte value between 0 and 255, for example:

127.0.0.1

This allows a theoretical number of 2^{32} or ~4 billion hosts to be connected on the internet today. Due to grouping, not all addresses are available today.

IPv6 addresses use 128 bit, which results in 2^{128} theoretically addressable hosts. This allows for a Really Big number of machines to addressed, and it sure fits all of today's requirements plus all those nifty PDAs and cell phones with IP phones in the near future without any sweat. When writing IPv6 addresses, they are usually divided into groups of 16 bits written as four hex digits, and the groups are separated by colons. An example is:

fe80::2a0:d2ff:fea5:e9f5

This shows a special thing - a number of consecutive zeros can be abbreviated by a single "::" once in the IPv6 address. The above address is thus equivalent to fe80:0:00:000:2a0:d2ff:fea5:e9f5 - leading zeros within groups can be omitted, and only one "::" can be used in an IPv6 address.

To make addresses manageable, they are split in two parts, which are the bits identifying the network a machine is on, and the bits that identify a machine on a (sub)network. The bits are known as netbits and hostbits, and in both IPv4 and IPv6, the netbits are the "left", most significant bits of an IP address, and the host bits are the "right", least significant bits, as shown in Figure 21-4.

Figure 21-4. IPv6-addresses are divided into more significant network- and less significant hostbits, too

In IPv4, the border is drawn with the aid of the netmask, which can be used to mask all net/host bits. Typical examples are 255.255.0.0 that uses 16 bit for addressing the network, and 16 bit for the machine, or 255.255.255.0 which takes another 8 bit to allow addressing 256 subnets on e.g. a class B net.

When addressing switched from classful addressing to CIDR routing, the borders between net and host bits stopped being on 8 bit boundaries, and as a result the netmasks started looking ugly and not really manageable. As a replacement, the number of network bits is used for a given address, to denote the border, e.g.

10.0.0.0/24

is the same as a netmask of 255.255.255.0 (24 1-bits). The same scheme is used in IPv6:

2001:638:a01:2::/64

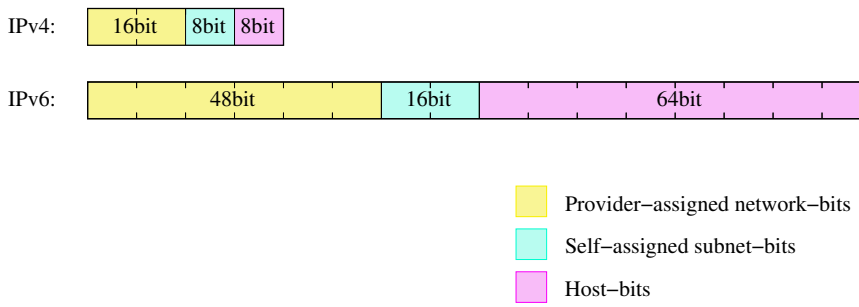
tells us that the address used here has the first (leftmost) 64 bits used as the network address, and the last (rightmost) 64 bits are used to identify the machine on the network. The network bits are commonly referred to as (network) "prefix", and the "prefixlen" here would be 64 bits.

Common addressing schemes found in IPv4 are the (old) class B and class C nets. With a class C network (/24), you get 24 bits assigned by your provider, and it leaves 8 bits to be assigned by you. If you want to add any subnetting to that, you end up with "uneven" netmasks that are a bit nifty to deal with. Easier for such cases are class B networks (/16), which only have 16 bits assigned by the provider, and that allow subnetting, i.e. splitting of the rightmost bits into two parts. One to address the on-site

subnet, and one to address the hosts on that subnet. Usually, this is done on byte (8 bit) boundaries. Using a netmask of 255.255.255.0 (or a /24 prefix) allows flexible management even of bigger networks here. Of course there is the upper limit of 254 machines per subnet, and 256 subnets.

With 128 bits available for addressing in IPv6, the scheme commonly used is the same, only the fields are wider. Providers usually assign /48 networks, which leaves 16 bits for a subnetting and 64 hostbits.

Figure 21-5. IPv6-addresses have a similar structure to class B addresses



Now while the space for network and subnets here is pretty much ok, using 64 bits for addressing hosts seems like a waste. It's unlikely that you will want to have several billion hosts on a single subnet, so what is the idea behind this?

The idea behind fixed width 64 bit wide host identifiers is that they aren't assigned manually as it's usually done for IPv4 nowadays. Instead, IPv6 host addresses are recommended (not mandatory!) to be built from so-called EUI64 addresses. EUI64 addresses are - as the name says - 64 bit wide, and derived from MAC addresses of the underlying network interface. E.g. for ethernet, the 6 byte (48 bit) MAC address is usually filled with the hex bits "fffe" in the middle and a bit is set to mark the address as unique (which is true for Ethernet), e.g. the MAC address

01:23:45:67:89:ab

results in the EUI64 address

03:23:45:ff:fe:67:89:ab

which again gives the host bits for the IPv6 address as

::0323:45ff:fe67:89ab

These host bits can now be used to automatically assign IPv6 addresses to hosts, which supports autoconfiguration of IPv6 hosts - all that's needed to get a complete IPv6 address is the first (net/subnet) bits, and IPv6 also offers a solution to assign them automatically.

When on a network of machines speaking IP, there's usually one router which acts as the gateway to outside networks. In IPv6 land, this router will send "router advertisement" information, which clients are expected to either receive during operation or to solicit upon system startup. The router advertisement information includes data on the router's address, and which address prefix it routes. With this information and the host-generated EUI64 address, an IPv6-host can calculate its IP address, and there is no need for manual address assignment. Of course routers still need some configuration.

The router advertisement information they create are part of the Neighbor Discovery Protocol (NDP, see RFC2461), which is the successor to IPv4's ARP protocol. In contrast to ARP, NDP does not only do lookup of IPv6 addresses for MAC addresses (the neighbor solicitation/advertisement part), but also does a similar service for routers and the prefixes they serve, which is used for autoconfiguration of IPv6 hosts as described in the previous paragraph.

21.7.3.2 Multiple Addresses

In IPv4, a host usually has one IP address per network interface or even per machine if the IP stack supports it. Only very rare applications like web servers result in machines having more than one IP address. In IPv6, this is different. For each interface, there is not only a globally unique IP address, but there are two other addresses that are of interest: The link local address, and the site local address. The link local address has a prefix of fe80::/64, and the host bits are built from the interface's EUI64 address. The link local address is used for contacting hosts and routers on the same network only, the addresses are not visible or reachable from different subnets. If wanted, there's the choice of either using global addresses (as assigned by a provider), or using site local addresses. Site local addresses are assigned the network address fec0::/10, and subnets and hosts can be addressed just as for provider-assigned networks. The only difference is, that the addresses will not be visible to outside machines, as these are on a different network, and their "site local" addresses are in a different physical net (if assigned at all). As with the 10/8 network in IPv4, site local addresses can be used, but don't have to. For IPv6 it's most common to have hosts assigned a link-local and a global IP address. Site local addresses are rather uncommon today, and are no substitute for globally unique addresses if global connectivity is required.

21.7.3.3 Multicasting

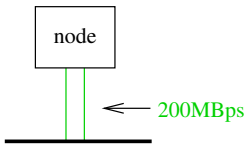
In IP land, there are three ways to talk to a host: unicast, broadcast and multicast. The most common one is by talking to it directly, using its unicast address. In IPv4, the unicast address is the "normal" IP address assigned to a single host, with all address bits assigned. The broadcast address used to address all hosts in the same IP subnet has the network bits set to the network address, and all host bits set to "1" (which can be easily done using the netmask and some bit operations). Multicast addresses are used to reach a number of hosts in the same multicast group, which can be machines spread over the whole internet. Machines must join multicast groups explicitly to participate, and there are special IPv4 addresses used for multicast addresses, allocated from the 224/8 subnet. Multicast isn't used very much in IPv4, and only few applications like the MBone audio and video broadcast utilities use it.

In IPv6, unicast addresses are used the same as in IPv4, no surprise there - all the network and host bits are assigned to identify the target network and machine. Broadcasts are no longer available in IPv6 in the way they were in IPv4, this is where multicasting comes into play. Addresses in the ff::/8 network are reserved for multicast applications, and there are two special multicast addresses that supersede the broadcast addresses from IPv4. One is the "all routers" multicast address, the others is for "all hosts". The addresses are specific to the subnet, i.e. a router connected to two different subnets can address all hosts/routers on any of the subnets it's connected to. Addresses here are:

- ff0*x*::1 for all hosts and
- ff0*x*::2 for all routers,

where "*x*" is the scope ID of the link here, identifying the network. Usually this starts from "1" for the "node local" scope, "2" for the first link, etc. Note that it's perfectly ok for two network interfaces to be attached to one link, thus resulting in double bandwidth:

Figure 21-6. Several interfaces attached to a link result in only one scope ID for the link



One use of the "all hosts" multicast is in the neighbor solicitation code of NDP, where any machine that wants to communicate with another machine sends out a request to the "all hosts" group, and the machine in question is expected to respond.

21.7.3.4 Name Resolving in IPv6

After talking a lot about addressing in IPv6, anyone still here will hope that there's a proper way to abstract all these long & ugly IPv6 addresses with some nice hostnames as one can do in IPv4, and of course there is.

Hostname to IP address resolving in IPv4 is usually done in one of three ways: using a simple table in `/etc/hosts`, by using the Network Information Service (NIS, formerly YP) or via the Domain Name System (DNS).

As of this writing, NIS/NIS+ over IPv6 is currently only available on Solaris 8, for both database contents and transport, using a RPC extension.

Having a simple address<->name map like `/etc/hosts` is supported in all IPv6 stacks. With the KAME implementation used in NetBSD, `/etc/hosts` contains IPv6 addresses as well as IPv4 addresses. A simple example is the "localhost" entry in the default NetBSD installation:

```
127.0.0.1               localhost
::1                     localhost
```

For DNS, there are no fundamentally new concepts. IPv6 name resolving is done with AAAA records that - as the name implies - point to an entity that's four times the size of an A record. The AAAA record takes a hostname on the left side, just as A does, and on the right side there's an IPv6 address, e.g.

```
noon            IN      AAAA    3ffe:400:430:2:240:95ff:fe40:4385
```

For reverse resolving, IPv4 uses the in-addr.arpa zone, and below that it writes the bytes (in decimal) in reversed order, i.e. more significant bytes are more right. For IPv6 this is similar, only that hex digits representing 4 bits are used instead of decimal numbers, and the resource records are also under a different domain, ip6.int.

So to have the reverse resolving for the above host, you would put into your `/etc/named.conf` something like:

```
zone "0.3.4.0.0.0.4.0.e.f.f.3.IP6.INT" {
      type master;
```

```
        file "db.reverse";
};
```

and in the zone file db.reverse you put (besides the usual records like SOA and NS):

```
5.8.3.4.0.4.e.f.f.f.5.9.0.4.2.0.2.0.0.0 IN   PTR   noon.ipv6.example.com.
```

The address is reversed here, and written down one hex digit after the other, starting with the least significant (rightmost) one, separating the hex digits with dots, as usual in zone files.

One thing to note when setting up DNS for IPv6 is to take care of the DNS software version in use. BIND 8.x does understand AAAA records, but it does not offer name resolving via IPv6. You need BIND 9.x for that. Beyond that, BIND 9.x supports a number of resource records that are currently being discussed but not officially introduced yet. The most noticeable one here is the A6 record which allows easier provider/prefix changing.

To sum up, this section talked about the technical differences between IPv4 and IPv6 for addressing and name resolving. Some details like IP header options, QoS and flows were deliberately left out to not make this document more complex than necessary.

Chapter 22
Setting up TCP/IP on NetBSD in practice

22.1 A walk through the kernel configuration

Before we dive into configuring various aspects of network setup, we want to walk through the necessary bits that have to or can be present in the kernel. See Chapter 29 for more details on compiling the kernel, we will concentrate on the configuration of the kernel here. We will take the i386/GENERIC config file as an example here. Config files for other platforms should contain similar information, the comments in the config files give additional hints. Besides the information given here, each kernel option is also documented in the options(4) manpage, and there is usually a manpage for each driver too, e.g. tlp(4).

The first line of each config file shows the version. It can be used to compare against other versions via CVS, or when reporting bugs.

```
options         NTP             # NTP phase/frequency locked loop
```

If you want to run the Network Time Protocol (NTP), this option can be enabled for maximum precision. If the option is not present, NTP will still work. See ntpd(8) for more information.

```
file-system     NFS             # Network File System client
```

If you want to use another machine's hard disk via the Network File System (NFS), this option is needed. Section 26.2 gives more information on NFS.

```
options         NFSSERVER       # Network File System server
```

This option includes the server side of the NFS remote file sharing protocol. Enable if you want to allow other machines to use your hard disk. Section 26.2 contains more information on NFS.

```
#options        GATEWAY         # packet forwarding
```

If you want to setup a router that forwards packets between networks or network interfaces, setting this option is needed. If doesn't only switch on packet forwarding, but also increases some buffers. See options(4) for details.

```
options         INET            # IP + ICMP + TCP + UDP
```

This enables the TCP/IP code in the kernel. Even if you don't want/use networking, you will still need this for machine-internal communication of subsystems like the X Window System. See inet(4) for more details.

```
options         INET6           # IPV6
```

If you want to use IPv6, this is your option. If you don't want IPv6, which is part of NetBSD since the 1.5 release, you can remove/comment out that option. See the inet6(4) manpage and Section 21.7 for more information on the next generation Internet protocol.

```
#options        IPSEC           # IP security
```

Includes support for the IPsec protocol, including key and policy management, authentication and compression. This option can be used without the previous option INET6, if you just want to use IPsec with IPv4, which is possible. See ipsec(4) for more information.

```
#options        IPSEC_ESP       # IP security (encryption part; define w/IPSEC)
```

This option is needed in addition to IPSEC if encryption is wanted in IPsec.

```
#options        MROUTING        # IP multicast routing
```

If multicast services like the MBone services should be routed, this option needs to be included. Note that the routing itself is controlled by the mrouted(8) daemon.

```
options         NS              # XNS
#options        NSIP            # XNS tunneling over IP
```

These options enable the Xerox Network Systems(TM) protocol family. It's not related to the TCP/IP protocol stack, and in rare use today. The ns(4) manpage has some details.

```
options         ISO,TPIP        # OSI
#options        EON             # OSI tunneling over IP
```

These options include the OSI protocol stack, which was said for a long time to be the future of networking. It's mostly history these days. :-) See the iso(4) manpage for more information.

```
options         CCITT,LLC,HDLC  # X.25
```

These options enable the X.25 protocol set for transmission of data over serial lines. It is/was used mostly in conjunction with the OSI protocols and in WAN networking.

```
options         NETATALK        # AppleTalk networking protocols
```

Include support for the AppleTalk protocol stack. Userland server programs are needed to make use of that. See pkgsrc/net/netatalk and pkgsrc/net/netatalk-asun for such packages. More information on the AppleTalk protocol and protocol stack are available in the atalk(4) manpage.

```
options         PPP_BSDCOMP     # BSD-Compress compression support for PPP
options         PPP_DEFLATE     # Deflate compression support for PPP
options         PPP_FILTER      # Active filter support for PPP (requires bpf)
```

These options tune various aspects of the Point-to-Point protocol. The first two determine the compression algorithms used and available, while the third one enables code to filter some packets.

```
options         PFIL_HOOKS      # pfil(9) packet filter hooks
options         IPFILTER_LOG    # ipmon(8) log support
```

These options enable firewalling in NetBSD, using IPfilter. See the ipf(4) and ipf(8) manpages for more information on operation of IPfilter, and Section 22.5.1 for a configuration example.

```
# Compatibility with 4.2BSD implementation of TCP/IP.  Not recommended.
#options        TCP_COMPAT_42
```

This option is only needed if you have machines on the network that still run 4.2BSD or a network stack derived from it. If you've got one or more 4.2BSD-systems on your network, you've to pay attention to set the right broadcast-address, as 4.2BSD has a bug in its networking code, concerning the broadcast address. This bug forces you to set all host-bits in the broadcast-address to "0". The TCP_COMPAT_42 option helps you ensuring this.

```
options         NFS_BOOT_DHCP,NFS_BOOT_BOOTPARAM
```

These options enable lookup of data via DHCP or the BOOTPARAM protocol if the kernel is told to use a NFS root file system. See the diskless(8) manpage for more information.

```
# Kernel root file system and dump configuration.
config          netbsd  root on ? type ?
#config         netbsd  root on sd0a type ffs
#config         netbsd  root on ? type nfs
```

These lines tell where the kernel looks for its root file system, and which filesystem type it is expected to have. If you want to make a kernel that uses a NFS root filesystem via the tlp0 interface, you can do this with "`root on tlp0 type nfs`". If a `?` is used instead of a device/type, the kernel tries to figure one out on its own.

```
# ISA serial interfaces
com0    at isa? port 0x3f8 irq 4        # Standard PC serial ports
com1    at isa? port 0x2f8 irq 3
com2    at isa? port 0x3e8 irq 5
```

If you want to use PPP or SLIP, you will need some serial (com) interfaces. Others with attachment on USB, PCMCIA or PUC will do as well.

```
# Network Interfaces
```

This rather long list contains all sorts of network drivers. Please pick the one that matches your hardware, according to the comments. For most drivers, there's also a manual page available, e.g. tlp(4), ne(4), etc.

```
# MII/PHY support
```

This section lists media independent interfaces for network cards. Pick one that matches your hardware. If in doubt, enable them all and see what the kernel picks. See the mii(4) manpage for more information.

```
# USB Ethernet adapters
aue*    at uhub? port ?         # ADMtek AN986 Pegasus based adapters
cue*    at uhub? port ?         # CATC USB-EL1201A based adapters
kue*    at uhub? port ?         # Kawasaki LSI KL5KUSB101B based adapters
```

USB-ethernet adapters only have about 2MBit/s bandwidth, but they are very convenient to use. Of course this needs other USB related options which we won't cover here, as well as the necessary hardware. See the corresponding manpages for more information.

```
# network pseudo-devices
pseudo-device   bpfilter        8       # Berkeley packet filter
```

This pseudo-device allows sniffing packets of all sorts. It's needed for tcpdump, but also rarpd and some other applications that need to know about network traffic. See bpf(4) for more information.

```
pseudo-device   ipfilter               # IP filter (firewall) and NAT
```

This one enables the IPfilter's packet filtering kernel interface used for firewalling, NAT (IP Masquerading) etc. See ipf(4) and Section 22.5.1 for more information.

```
pseudo-device   loop                    # network loopback
```

This is the "lo0" software loopback network device which is used by some programs these days, as well as for routing things. It should not be omitted. See lo(4) for more details.

```
pseudo-device   ppp             2       # Point-to-Point Protocol
```

If you want to use PPP either over a serial interface or ethernet (PPPoE), you will need this option. See ppp(4) for details on this interface.

```
pseudo-device   sl              2       # Serial Line IP
```

Serial Line IP is a simple encapsulation for IP over (well :) serial lines. It does not include negotiation of IP addresses and other options, which is the reason that it's not in widespread use today any more. See sl(4).

```
pseudo-device   strip           2       # Starmode Radio IP (Metricom)
```

If you happen to have one of the old Metricon Ricochet packet radio wireless network devices, use this pseudo-device to use it. See the strip(4) manpage for detailed information.

```
pseudo-device   tun             2       # network tunneling over tty
```

This network device can be used to tunnel network packets to a device file, `/dev/tun*`. Packets routed to the tun0 interface can be read from `/dev/tun0`, and data written to `/dev/tun0` will be sent out the tun0 network interface. This can be used to implement e.g. QoS routing in userland. See tun(4) for details.

```
pseudo-device   gre             2       # generic L3 over IP tunnel
```

The GRE encapsulation can be used to tunnel arbitrary layer 3 packets over IP, e.g. to implement VPNs. See gre(4) for more.

```
pseudo-device   ipip            2       # IP Encapsulation within IP (RFC 2003)
```

Another IP-in-IP encapsulation device, with a different encapsulation format. See the ipip(4) manpage for details.

```
pseudo-device   gif             4       # IPv[46] over IPv[46] tunnel (RFC 1933)
```

Using the GIF interface allows to tunnel e.g. IPv6 over IPv4, which can be used to get IPv6 connectivity if no IPv6-capable uplink (ISP) is available. Other mixes of operations are possible, too. See the gif(4) manpage for some examples.

```
#pseudo-device  faith           1       # IPv[46] tcp relay translation i/f
```

The faith interface captures IPv6 TCP traffic, for implementing userland IPv6-to-IPv4 TCP relays e.g. for protocol transitions. See the faith(4) manpage for more details on this device.

```
#pseudo-device  stf             1       # 6to4 IPv6 over IPv4 encapsulation
```

This adds a network device that can be used to tunnel IPv6 over IPv4 without setting up a configured tunnel before. The source address of outgoing packets contains the IPv4 address, which allows routing replies back via IPv4. See the stf(4) manpage and Section 26.4 for more details.

```
pseudo-device   vlan                    # IEEE 802.1q encapsulation
```

This interface provides support for IEEE 802.1Q Virtual LANs, which allows tagging Ethernet frames with a “vlan” ID. Using properly configured switches (that also have to support VLAN, of course), this can be used to build virtual LANs where one set of machines doesn’t see traffic from the other (broadcast and other). The vlan(4) manpage tells more about this.

22.2 Overview of the network configuration files

The following is a list of the files used to configure the network. The usage of these files, some of which have already been met the first chapters, will be described in the following sections.

`/etc/hosts`

Local hosts database file. Each line contains information regarding a known host and contains the internet address, the host’s name and the aliases. Small networks can be configured using only the hosts file, without a *name server*.

`/etc/resolv.conf`

This file specifies how the routines which provide access to the Internet Domain Name System should operate. Generally it contains the addresses of the name servers.

`/etc/ifconfig.xxx`

This file is used for the automatic configuration of the network card at boot.

`/etc/mygate`

Contains the IP address of the gateway.

`/etc/nsswitch.conf`

Name service switch configuration file. It controls how a process looks up various databases containing information regarding hosts, users, groups, etc. Specifically, this file defines the order to look up the databases. For example, the line:

```
hosts:    files dns
```

specifies that the hosts database comes from two sources, *files* (the local `/etc/hosts` file) and *DNS*, (the Internet Domain Name System) and that the local files are searched before the DNS.

It is usually not necessary to modify this file.

22.3 Connecting to the Internet with a modem

There are many types of Internet connections: this section explains how to connect to a provider using a modem over a telephone line using the PPP protocol, a very common setup. In order to have a working connection, the following steps must be done:

1. Get the necessary information from the provider.
2. Edit the file `/etc/resolv.conf` and check `/etc/nsswitch.conf`.
3. Create the directories `/etc/ppp` and `/etc/ppp/peers` if they don't exist.
4. Create the connection script, the chat file and the pppd options file.
5. Created the user-password authentication file.

Judging from the previous list it looks like a complicated procedure that requires a lot of work. Actually, the single steps are very easy: it's just a matter of modifying, creating or simply checking some small text files. In the following example it will be assumed that the modem is connected to the second serial port `/dev/tty01` (COM2 in DOS).

A few words on the difference between *com*, *COM* and *tty*. For NetBSD, "com" is the name of the serial port driver (the one that is displayed by **dmesg**) and "tty" is the name of the port. Since numbering starts at 0, com0 is the driver for the first serial port, named tty00. In the DOS world, instead, COM1 refers to the first serial port (usually located at (0x3f8), COM2 to the second, and so on. Therefore COM1 (DOS) corresponds to `/dev/tty00` (NetBSD).

Besides external modems connected to COM ports (using `/dev/tty0[012]` on i386, `/dev/tty[ab]` on sparc, ...) modems on USB (`/dev/ttyU*`) and pcmcia/cardbus (`/dev/tty0[012]`) can be used.

22.3.1 Getting the connection information

The first thing to do is ask the provider the necessary information for the connection, which means:

- The phone number of the nearest POP.
- The authentication method to be used.
- The username and password for the connection.
- The IP addresses of the name servers.

22.3.2 `resolv.conf` and `nsswitch.conf`

The `/etc/resolv.conf` file must be configured using the information supplied by the provider, especially the addresses of the DNS. In this example the two DNS will be "194.109.123.2" and "191.200.4.52".

Example 22-1. `resolv.conf`

```
nameserver 194.109.123.2
nameserver 191.200.4.52
```

And now an example of the `/etc/nsswitch.conf` file.

Example 22-2. `nsswitch.conf`

```
# /etc/nsswitch.conf
group:         compat
group_compat:  nis
hosts:         files dns
netgroup:      files [notfound=return] nis
networks:      files
passwd:        compat
passwd_compat: nis
shells:        files
```

The defaults of doing hostname lookups via `/etc/hosts` followed by the DNS works fine and there's usually no need to modify this.

22.3.3 Creating the directories for pppd

The directories `/etc/ppp` and `/etc/ppp/peers` will contain the configuration files for the PPP connection. After a fresh install of NetBSD they don't exist and must be created (chmod 700).

```
# mkdir /etc/ppp
# mkdir /etc/ppp/peers
```

22.3.4 Connection script and chat file

The connection script will be used as a parameter on the pppd command line; it is located in `/etc/ppp/peers` and has usually the name of the provider. For example, if the provider's name is BigNet and your user name for the connection to the provider is alan, an example connection script could be:

Example 22-3. Connection script

```
# /etc/ppp/peers/bignet
connect '/usr/sbin/chat -v -f /etc/ppp/peers/bignet.chat'
noauth
user alan
remotename bignet.it
```

In the previous example, the script specifies a *chat file* to be used for the connection. The options in the script are detailed in the pppd(8) man page.

Note: If you are experiencing connection problems, add the following two lines to the connection script

```
debug
kdebug 4
```

You will get a log of the operations performed when the system tries to connect. See pppd(8), syslog.conf(5).

The connection script calls the chat application to deal with the physical connection (modem initialization, dialing, ...) The parameters to chat can be specified inline in the connection script, but it is better to put them in a separate file. If, for example, the telephone number of the POP to call is 02 99999999, an example chat script could be:

Example 22-4. Chat file

```
# /etc/ppp/peers/bignet.chat
ABORT BUSY
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
" ATDT0299999999
CONNECT "
```

Note: If you have problems with the chat file, you can try connecting manually to the POP with the cu(1) program and verify the exact strings that you are receiving.

22.3.5 Authentication

During authentication each of the two systems verifies the identity of the other system, although in practice you are not supposed to authenticate the provider, but only to be verified by him, using one of the following methods:

- PAP/CHAP
- login

Most providers use a PAP/CHAP authentication.

22.3.5.1 PAP/CHAP authentication

The authentication information (speak: password) is stored in the `/etc/ppp/pap-secrets` for PAP and in `/etc/ppp/chap-secrets` for CHAP. The lines have the following format:

```
user * password
```

For example:

```
alan * pZY9o
```

For security reasons the `pap-secrets` and `chap-secrets` files should be owned by *root* and have permissions “600”.

```
# chown root /etc/ppp/pap-secrets
# chown root /etc/ppp/chap-secrets
# chmod 600 /etc/ppp/pap-secrets
# chmod 600 /etc/ppp/chap-secrets
```

22.3.5.2 Login authentication

This type of authentication is not widely used today; if the provider uses login authentication, user name and password must be supplied in the chat file instead of the PAP/CHAP files, because the chat file simulates an interactive login. In this case, set up appropriate permissions for the chat file.

The following is an example chat file with login authentication:

Example 22-5. Chat file with login

```
# /etc/ppp/peers/bignet.chat
ABORT BUSY
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
″ ATDT0299999999
CONNECT ″
TIMEOUT 50
ogin: alan
ssword: pZY9o
```

22.3.6 pppd options

The only thing left to do is the creation of the **pppd** options file, which is `/etc/ppp/options` (chmod 644).

Example 22-6. `/etc/ppp/options`

```
/dev/tty01
lock
crtscts
57600
modem
defaultroute
noipdefault
```

Check the pppd(8) man page for the meaning of the options.

22.3.7 Testing the modem

Before activating the link it is a good idea to make a quick modem test, in order to verify that the physical connection and the communication with the modem works. For the test the cu(1) program can be used, as in the following example.

1. Create the file `/etc/uucp/port` with the following lines:

   ```
   type modem
   port modem
   device /dev/tty01
   speed 115200
   ```

   (substitute the correct device in place of `/dev/tty01`).

2. Write the command **cu -p modem** to start sending commands to the modem. For example:

   ```
   # cu -p modem
   Connected.
   ATZ
   OK
   ~.

   Disconnected.
   #
   ```

   In the previous example the reset command (ATZ) was sent to the modem, which replied with OK: the communication works. To exit cu(1), write ~ (tilde) followed by . (dot), as in the example.

If the modem doesn't work, check that it is connected to the correct port (i.e. you are using the right port with cu(1). Cables are a frequent cause of trouble, too.

When you start cu(1) and a message saying "Permission denied" appears, check who is the owner of the `/dev/tty##` device, it must be "uucp". For example:

```
$ ls -l /dev/tty00
crw-------  1 uucp  wheel  8, 0 Mar 22 20:39 /dev/tty00
```

If the owner is root, the following happens:

```
$ ls -l /dev/tty00
crw-------  1 root  wheel  8, 0 Mar 22 20:39 /dev/tty00
$ cu -p modem
cu: open (/dev/tty00): Permission denied
cu: All matching ports in use
```

22.3.8 Activating the link

At last everything is ready to connect to the provider with the following command:

```
# pppd call bignet
```

where `bignet` is the name of the already described connection script. To see the connection messages of **pppd**, give the following command:

```
# tail -f /var/log/messages
```

To disconnect, do a **kill -HUP** of **pppd**.

```
 # pkill -HUP pppd
```

22.3.9 Using a script for connection and disconnection

When the connection works correctly, it's time to write a couple of scripts to avoid repeating the commands every time. These two scripts can be named, for example, `ppp-start` and `ppp-stop`.

`ppp-start` is used to connect to the provider:

Example 22-7. `ppp-start`

```
#!/bin/sh
MODEM=tty01
POP=bignet
if [ -f /var/spool/lock/LCK..$MODEM ]; then
echo ppp is already running...
else
pppd call $POP
tail -f /var/log/messages
fi
```

`ppp-stop` is used to close the connection:

Example 22-8. `ppp-stop`

```
#!/bin/sh
MODEM=tty01
if [ -f /var/spool/lock/LCK..$MODEM ]; then
echo -f killing pppd...
kill -HUP `cat /var/spool/lock/LCK..$MODEM`
echo done
else
echo ppp is not active
fi
```

The two scripts take advantage of the fact that when **pppd** is active, it creates the file `LCK..tty01` in the `/var/spool/lock` directory. This file contains the process ID (*pid*) of the **pppd** process.

The two scripts must be executable:

```
# chmod u+x ppp-start ppp-stop
```

22.3.10 Running commands after dialin

If you find yourself to always run the same set of commands each time you dial in, you can put them in a script `/etc/ppp/ip-up` which will be called by pppd(8) after successful dial-in. Likewise, before the connection is closed down, `/etc/ppp/ip-down` is executed. Both scripts are expected to be executable. See pppd(8) for more details.

22.4 Creating a small home network

Networking is one of the main strengths of Unix and NetBSD is no exception: networking is both powerful and easy to set up and inexpensive too, because there is no need to buy additional software to communicate or to build a server. Section 22.5 explains how to configure a NetBSD machine to act as a gateway for a network: with IPNAT all the hosts of the network can reach the Internet with a single connection to a provider made by the gateway machine. The only thing to be checked before creating the network is to buy network cards supported by NetBSD (check the `INSTALL.*` files for a list of supported devices).

First, the network cards must be installed and connected to a hub, switch or directly (see Figure 22-1).

Next, check that the network cards are recognized by the kernel, studying the output of the **dmesg** command. In the following example the kernel recognized correctly an NE2000 clone:

```
...
ne0 at isa0 port 0x280-0x29f irq 9
ne0: NE2000 Ethernet
ne0: Ethernet address 00:c2:dd:c1:d1:21
...
```

If the card is not recognized by the kernel, check that it is enabled in the kernel configuration file and then that the card's IRQ matches the one that the kernel expects. For example, this is the isa NE2000 line in the configuration file; the kernel expects the card to be at IRQ 9.

```
...
ne0 at isa? port 0x280 irq 9 # NE[12]000 ethernet cards
...
```

If the card's configuration is different, it will probably not be found at boot. In this case, either change the line in the kernel configuration file and compile a new kernel or change the card's setup (usually through a setup disk or, for old cards, a jumper on the card).

The following command shows the network card's current configuration:

```
# ifconfig ne0
ne0: flags=8822<BROADCAST,NOTRAILERS,SIMPLEX,MULTICAST> mtu 1500
      address: 00:50:ba:aa:a7:7f
      media: Ethernet autoselect (10baseT)
      inet6 fe80::250:baff:feaa:a77f%ne0 prefixlen 64 scopeid 0x1
```

The software configuration of the network card is very easy. The IP address "192.168.1.1" is assigned to the card.

```
# ifconfig ne0 inet 192.168.1.1 netmask 0xffffff00
```

Note that the networks 10.0.0.0/8 and 192.168.0.0/16 are reserved for private networks, which is what we're setting up here.

Repeating the previous command now gives a different result:

```
# ifconfig ne0
ne0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      address: 00:50:ba:aa:a7:7f
      media: Ethernet autoselect (10baseT)
      inet 192.168.1.1 netmask 0xffffff00 broadcast 192.168.1.255
      inet6 fe80::250:baff:feaa:a77f%ne0 prefixlen 64 scopeid 0x1
```

The output of **ifconfig** has now changed: the IP address is now printed and there are two new flags, "UP" and "RUNNING" If the interface isn't "UP", it will not be used by the system to send packets.

The host was given the IP address 192.168.1.1, which belongs to the set of addresses reserved for internal networks which are not reachable from the Internet. The configuration is finished and must now be tested; if there is another active host on the network, a *ping* can be tried. For example, if 192.168.1.2 is the address of the active host:

```
# ping 192.168.1.2
PING ape (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: icmp_seq=0 ttl=255 time=1.286 ms
64 bytes from 192.168.1.2: icmp_seq=1 ttl=255 time=0.649 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=255 time=0.681 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=255 time=0.656 ms
^C
----ape PING Statistics----
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.649/0.818/1.286/0.312 ms
```

With the current setup, at the next boot it will be necessary to repeat the configuration of the network card. In order to avoid repeating the card's configuration at each boot, add the following lines to `/etc/rc.conf`:

```
auto_ifconfig=yes
ifconfig_ne0="inet 192.168.1.1 netmask 0xffffff00"
```

In this example the variable `ifconfig_ne0` was set because the network card was recognized as *ne0* by the kernel; if you are using a different adapter, substitute the appropriate name in place of ne0.

At the next boot the network card will be configured automatically.

If you have a router that is connected to the internet, you can use it as default router, which will handle all your packets. To do so, set `defaultroute` to the router's IP address in `/etc/rc.conf`:

```
defaultroute=192.168.0.254
```

Be sure to use the default router's IP address instead of name, in case your DNS server is beyond the default router. In that case, the DNS server couldn't be reached to resolve the default router's hostname and vice versa, creating a chicken-and-egg problem.

To reach hosts on your local network, and assuming you really have very few hosts, adjust `/etc/hosts` to contain the addresses of all the hosts belonging to the internal network. For example:

Example 22-9. `/etc/hosts`

```
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
# It is used only for "ifconfig" and other operations
# before the nameserver is started.
#
#
127.0.0.1             localhost
::1                   localhost
#
# RFC 1918 specifies that these networks are "internal".
# 10.0.0.0    10.255.255.255
# 172.16.0.0  172.31.255.255
# 192.168.0.0 192.168.255.255

192.168.1.1   ape.insetti.net ape
192.168.1.2   vespa.insetti.net vespa
192.168.1.0   insetti.net
```

If you are dialed in via an Internet Service Provider, or if you have a local Domain Name Server (DNS) running, you may want to use it to resolve hostnames to IP addresses, possibly in addition to `/etc/hosts`, which would only know your own hosts. To configure a machine as DNS client, you need to edit `/etc/resolv.conf`, and enter the DNS server's address, in addition to an optional domain name that will be appended to hosts with no domain, in order to create a FQDN for resolving. Assuming your DNS server's IP address is 192.168.1.2 and it is setup to serve for "home.net", put the following into `/etc/resolv.conf`:

```
# /etc/resolv.conf
domain home.net
nameserver 192.168.1.2
```

The `/etc/nsswitch.conf` file should be checked as explained in Example 22-2.

Summing up, to configure the network the following must be done: the network adapters must be installed and physically connected. Next they must be configured (with **ifconfig**) and, finally, the file `/etc/rc.conf` must be modified to configure the interface and possibly default router, and `/etc/resolv.conf` and `/etc/nsswitch.conf`should be adjusted if DNS should be used. This type of network management is sufficient for small networks without sophisticated needs.

22.5 Setting up an Internet gateway with IPNAT

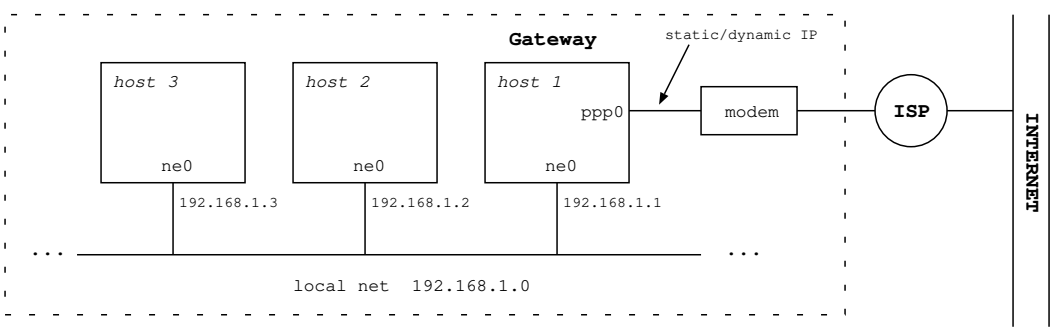
The mysterious acronym IPNAT hides the Internet Protocol Network Address Translation, which enables the routing of an internal network (e.g. your home network as described in Section 22.4) on a real network (Internet). This means that with only one "real" IP, static or dynamic, belonging to a gateway

running IPNAT, it is possible to create simultaneous connections to the Internet for all the hosts of the internal network.

Some usage examples of IPNAT can be found in the subdirectory `/usr/share/examples/ipf`: look at the files `BASIC.NAT` and `nat-setup`.

The setup for the example described in this section is detailed in Figure 22-1: *host 1* can connect to the Internet calling a provider with a modem and getting a dynamic IP address. *host 2* and *host 3* can't communicate with the Internet with a normal setup: IPNAT allows them to do it: host 1 will act as a Internet gateway for hosts 2 and 3. Using host 1 as default router, hosts 2 and 3 will be able to access the Internet.

Figure 22-1. Network with gateway



22.5.1 Configuring the gateway/firewall

To use IPNAT, the "pseudo-device ipfilter" must be compiled into the kernel, and IP packet forwarding must be enabled in the kernel. To check, run:

```
# sysctl net.inet.ip.forwarding
net.inet.ip.forwarding = 1
```

If the result is "1" as in the previous example, the option is enabled, otherwise, if the result is "0" the option is disabled. You can do two things:

1. Compile a new kernel, with the GATEWAY option enabled.
2. Enable the option in the current kernel with the following command:

   ```
   # sysctl -w net.inet.ip.forwarding=1
   ```

   You can add sysctl settings to `/etc/sysctl.conf` to have them set automatically at boot. In this case you would want to add

   ```
   net.inet.ip.forwarding=1
   ```

The rest of this section explains how to create an IPNAT configuration that is automatically started every time that a connection to the provider is activated with the PPP link. With this configuration all the host of a home network (for example) will be able to connect to the Internet through the gateway machine, even if they don't use NetBSD.

For the setup, first, create the `/etc/ipnat.conf` file containing the following rules:

```
map ppp0 192.168.1.0/24 -> 0/32 proxy port ftp ftp/tcp
map ppp0 192.168.1.0/24 -> 0/32 portmap tcp/udp 40000:60000
map ppp0 192.168.1.0/24 -> 0/32
```

192.168.1.0/24 are the network addresses that should be mapped. The first line of the configuration file is optional: it enables active FTP to work through the gateway. The second line is used to handle correctly tcp and udp packets; the portmapping is necessary because of the many to one relationship). The third line is used to enable ICMP, ping, etc.

Next, create the `/etc/ppp/ip-up` file; it will be called automatically every time that the PPP link is activated:

```
#!/bin/sh
# /etc/ppp/ip-up
/etc/rc.d/ipnat forcestart
```

Create the file `/etc/ppp/ip-down`; it will be called automatically when the PPP link is closed:

```
#!/bin/sh
# /etc/ppp/ip-down
/etc/rc.d/ipnat forcestop
```

Both `ip-up` and `ip-down` must be executable:

```
# chmod u+x ip-up ip-down
```

The gateway machine is now ready.

22.5.2 Configuring the clients

Create a `/etc/resolv.conf` file like the one on the gateway machine, to make the clients access the same DNS server as the gateway.

Next, make all clients use the gateway as their default router. Use the following command:

```
# route add default 192.168.1.1
```

192.168.1.1 is the address of the gateway machine configured in the previous section.

Of course you don't want to give this command every time, so it's better to define the "defaultroute" entry in the `/etc/rc.conf` file: the default route will be set automatically during system initialization, using the defaultroute option as an argument to the **route add default** command.

If the client machine is not using NetBSD, the configuration will be different. On Windows PC's you need to set the gateway property of the TCP/IP protocol to the IP address of the NetBSD gateway.

That's all that needs to be done on the client machines.

22.5.3 Some useful commands

The following commands can be useful for diagnosing problems:

ping

netstat -r

Displays the routing tables (similar to **route show**).

traceroute

On the client it shows the route followed by the packets to their destination.

tcpdump

Use on the gateway to monitor TCP/IP traffic.

22.6 A common LAN setup

The small home network discussed in the previous section contained many items that were configured manually. In bigger LANs that are centrally managed, one can expect Internet connectivity being available via some router, a DNS server being available, and most important, a DHCP server which hands out IP addresses to clients on request. To make a NetBSD client run in such an environment, it's usually enough to set

```
dhclient=yes
```

in `/etc/rc.conf`, and the IP address will be set automatically, `/etc/resolv.conf` will be created and routing setup to the default router.

22.7 Connecting two PCs through a serial line

If you need to transfer files between two PCs which are not networked there is a simple solution which is particularly handy when copying the files to a floppy is not practical: the two machines can be networked with a serial cable (a *null modem* cable). The following sections describe some configurations.

22.7.1 Connecting NetBSD with BSD or Linux

The easiest case is when both machines run NetBSD: making a connection with the SLIP protocol is very easy. On the first machine write the following commands:

```
# slattach /dev/tty00
# ifconfig sl0 inet 192.168.1.1 192.168.1.2
```

On the second machine write the following commands:

```
# slattach /dev/tty00
# ifconfig sl0 inet 192.168.1.2 192.168.1.1
```

Now you can test the connection with **ping**; for example, on the second PC write:

```
# ping 192.168.1.1
```

If everything worked there is now an active network connection between the two machines and ftp, telnet and other similar commands can be executed. The textual aliases of the machines can be written in the `/etc/hosts` file.

- In the previous example both PC's used the first serial port (`/dev/tty0`). Substitute the appropriate device if you are using another port.
- IP addresses like 192.168.x.x are reserved for "internal" networks. The first PC has address 192.168.1.1 and the second 192.168.1.2.
- To achieve a faster connection the `-s` *`speed`* option to **slattach** can be specified.
- **ftp** can be used to transfer files only if inetd is active and the ftpd server is enabled.

> **Linux:** If one of the two PC's runs Linux, the commands are slightly different (on the Linux machine only). If the Linux machine gets the 192.168.1.2 address, the following commands are needed:
>
> ```
> # slattach -p slip -s 115200 /dev/ttyS0 &
> # ifconfig sl0 192.168.1.2 pointopoint 192.168.1.1 up
> # route add 192.168.1.1 dev sl0
> ```
>
> Don't forget the "&" in the first command.

22.7.2 Connecting NetBSD and Windows NT

NetBSD and Windows NT can be (almost) easily networked with a serial *null modem* cable. Basically what needs to be done is to create a "Remote Access" connection under Windows NT and to start pppd on NetBSD.

Start pppd as root after having created a `.ppprc` in `/root`. Use the following example as a template.

```
connect '/usr/sbin/chat -v CLIENT CLIENTSERVER'
local
tty00
115200
crtscts
lock
noauth
nodefaultroute
:192.168.1.2
```

The meaning of the first line will be explained later in this section; 192.168.1.2 is the IP address that will be assigned by NetBSD to the Windows NT host; `tty00` is the serial port used for the connection (first serial port).

On the NT side a *null modem* device must be installed from the Control Panel (Modem icon) and a Remote Access connection using this modem must be created. The null modem driver is standard under Windows NT 4 but it's not a 100% null modem: when the link is activated, NT sends the string CLIENT

and expects to receive the answer CLIENTSERVER. This is the meaning of the first line of the `.ppprc` file: **chat** must answer to NT when the connection is activated or the connection will fail.

In the configuration of the Remote Access connection the following must be specified: use the null modem, telephone number "1" (it's not used, anyway), PPP server, enable only TCP/IP protocol, use IP address and nameservers from the server (NetBSD in this case). Select the hardware control flow and set the port to 115200 8N1.

Now everything is ready to activate the connection.

- Connect the serial ports of the two machines with the null modem cable.
- Launch **pppd** on NetBSD. To see the messages of pppd: **tail -f /var/log/messages**).
- Activate the Remote Access connection on Windows NT.

22.7.3 Connecting NetBSD and Windows 95

The setup for Windows 95 is similar to the one for Windows NT: Remote Access on Windows 95 and the PPP server on NetBSD will be used. Most (if not all) Windows 95 releases don't have the *null modem* driver, which makes things a little more complicated. The easiest solution is to find one of the available null modem drivers on the Internet (it's a small `.INF` file) and repeat the same steps as for Windows NT. The only difference is that the first line of the `.ppprc` file (the one that calls **chat**) can be removed.

If you can't find a real null modem driver for Windows 95 it's still possible to use a little trick:

- Create a Remote Access connection like the one described in Section 22.7.2 but using the "Standard Modem".
- In `.ppprc` substitute the line that calls **chat** with the following line

  ```
  connect '/usr/sbin/chat -v ATH OK AT OK ATE0V1 OK AT OK ATDT CONNECT'
  ```

- Activate the connection as described in Section 22.7.2.

In this way the **chat** program, called when the connection is activated, emulates what Windows 95 thinks is a standard modem, returning to Windows 95 the same answers that a standard modem would return. Whenever Windows 95 sends a modem command string, **chat** returns OK.

Chapter 23
The Internet Super Server inetd

The "internet super server", or inetd(8), is available on all Unix(like) systems, providing many of the basic network services available. This chapter describes the relationship between the daemon and several of the config files in the `/etc/` directory.

23.1 Overview

In this document we will look at a simple definition of inetd(8), how several files that relate to inetd(8) work (not that these files are not related to other software), how to add a service to inetd(8) and some considerations both to use inetd(8) for a particular service and times when a service might be better off running outside of inetd(8).

23.2 What is inetd?

In traditional Unix scenarios, one server (daemon) process watches for connections on a particular port, and handles incoming requests. Now if a machine offers many services, many daemon processes would be needed, mostly running idle but still wasting resources like memory. The internet super server, inetd, is an approach to this problem. It listens on a number of ports, and when it receives a request it then determines which program to run to handle the request and starts an instance of that program.

Following is a very simple diagram to illustrate inetd(8):

```
 pop3  ------  |
               |
 ftpd -------  | INETD | ---- Internet / DMZ / Switch / Whatever . . .
               |
cvsupserver -  |
```

In the above diagram you can see the general idea. The inetd(8) process receives a request and then starts the appropriate server process. What inetd(8) is doing is software multiplexing. An important note here, regarding security: On many other UNIX-like systems, a package called tcpwrappers is used as a security enhancement for inetd(8). On NetBSD the tcpwrapper functionality is built into inetd(8) using libwrap.

23.3 Configuring inetd - `/etc/inetd.conf`

The operation of inetd(8) is controlled by its own config file, surprisingly named `/etc/inetd.conf`, see inetd.conf(5). The `inetd.conf` file basically provides enabling and mapping of services the systems administrator would like to have multiplexed through inetd(8), indicating which program should be started for incoming requests on which port.

inetd.conf(5) is an ascii file containing one service per line, and several fields per line. The basic field layout is:

```
service-name socket-type protocol wait/nowait user:group server-program arguments
```

service-name:

> The service name indicates the port inetd(8) should listen on. It is either a decimal number, or a name matching a service name given in `/etc/services`.

socket-type:

> The communications socket type, the different types are "stream" for a TCP stream, "dgram" for an UDP service, "raw" for a raw socket, "rdm" for reliably delivered message and "seqpacket" for a sequenced packet socket. The most common socket types are "stream" and "dgram".

protocol

> The protocol used, mostly "tcp", "tcp6", "udp" and "udp6" for stream-oriented services via the Transmission Control Protocol, or datagram-oriented services via the User Datagram Protocol. It is worth noting that "tcp" and "udp" mean they use the default (currently IPv4), "tcp4" specifically means communication via IPv4 only, and "tcp6" and "udp6" are IPv6-only. In addition to those, protocols based on Remote Procedure Calls (RPC) can be specified as either "rpc/tcp" or "rpc/udp".

wait/nowait

> This field tells inetd(8) if it should wait for a server program to return or to continue processing new connections immediately. Many connections to server processes require answers after data transfers are complete, where other types can keep transmitting on a connection continuously, the latter is a "nowait" and the former "wait". In most cases, this entry corresponds to the socket-type, for example a streaming connection would (most of the time) have a "nowait" value in this field.

user[:group]

> This field gives the user name and optionally a group name that the server process which inetd(8) starts up runs as.

server-program

> This field is the full path of the program that gets started.

program-arguments

> This field contains the argument vector argv[] of the program started, including the program name and additional arguments the systems administrator may need to specify for the server program that is started.

That is all a lot to digest and there are other things the systems administrator can do with some of the fields. Here is a sample line from an `inetd.conf` file:

```
ftp       stream  tcp    nowait  root   /usr/libexec/ftpd    ftpd -ll
```

From the left, the service-name is "ftp", socket-type is "stream", protocol is "tcp", inetd(8) won't wait for the server process to terminate ("nowait"), the process runs as user "root", path is `/usr/libexec/ftpd`

and program name and arguments are "ftpd -ll". Notice in the last field, the program name is different from the service-name.

23.4 Services - `/etc/services`

The next file to consider is the service name data base that can be found in `/etc/services`. This file basically contains information mapping a service name to a port number. The format of the `/etc/services` file is:

```
service-name port-number/protocol-name [aliases]
```

"service-name" is the name of the service, "port-number" is the port number assigned to the service, "protocol-name" is either "tcp" or "udp", and if alias names for a port are needed, they can be added as "aliases", separated by white spaces. Comments may be added after a hash mark (#).

Let's take a look at the "ssh" entries as an example:

```
ssh             22/tcp           # Secure Shell
ssh             22/udp
```

As we can see, from the left, the service name is "ssh", the port number is "22", the protocols are both "tcp" and "udp". Notice that there is a separate entry for every protocol a service can use (even on the same port).

23.5 Protocols - `/etc/protocols`

Another file read by inetd(8) is `/etc/protocols`. This file has the information pertaining to DARPA Internet protocols. The format of the protocols name data base is:

```
protocol-name number [aliases]
```

where "protocol-name" describes the payload of an IP packet, e.g. "tcp" or "udp". "number" is the official protocol number assigned by IANA, and optional alias names can be added after that.

Let's look at the seventh entry in the `/etc/protocols` db as an example:

```
tcp     6       TCP             # transmission control protocol
```

Starting from the left, we see that the protocol name is "tcp", the number is "6" and the only aliases listed is "TCP", belonging to the Transmission Control Protocol as indicated by the comment in that line.

23.6 Remote Procedure Calls (RPC) - `/etc/rpc`

The rpc program number data base used by services with the "rpc" protocol type in inetd.conf(5) is kept in `/etc/rpc` and contains name mappings to rpc program numbers. The format of the file is:

```
server-name program-number aliases
```

For example, here is the nfs entry:

```
nfs             100003  nfsprog
```

23.7 Allowing and denying hosts - `/etc/hosts.{allow,deny}`

As mentioned above, NetBSD's inetd(8) has the tcpwrapper package built in via the libwrap library. As such, inetd(8) can allow or deny access to each service on a more fine-grained base than just allowing a service to everyone, or not enabling it at all. The access control is defined in the files `/etc/hosts.allow` and `/etc/hosts.deny`, see the hosts_access(5) manpage.

Each of the two files contains several lines that describe access restrictions for a certain server. Access is allowed if permission is given in `/etc/hosts.allow`. If the service is not listened in `/etc/hosts.allow` but in `/etc/hosts.deny`, it is denied. If a service is listed in neither file, it is allowed, giving standard inetd(8) behaviour.

Each line in `/etc/hosts.allow` and `/etc/hosts.deny` contains a service either by name (as given in the field for argv[0] in `/etc/inetd.conf`, e.g. "ftpd" instead of "ftp"), or the special service "ALL" which obviously applies to all services. Following the service name is - separated by a colon - a number of access restrictions, which can be hostnames, domains, single IP addresses, whole IP subnets or some other restrictions, please check hosts_access(5) for all the details.

An example configuration that is mostly open but denies access to services to a certain host and all machines from a certain domain would look like this:

```
# /etc/hostname.deny:
ALL: some.host.name, .some.domain
```

Another example that would be mostly closed, denying access to all but very few machines would need entries in both `/etc/hosts.allow` and `/etc/hosts.deny`. The entry for `/etc/hosts.deny would be:`

```
# /etc/hosts.deny
ALL: ALL
```

The entry to allow a few hosts would be put into `/etc/hosts.allow`:

```
# /etc/hosts.allow
ALL: friend.host.domain otherfriend.otherhost.otherdomain
```

23.8 Adding a Service

Many times a systems administrator will find that they need to add a service to their system that is not already in inetd(8) or they may wish to move a service to it because it does not get very much traffic. This is usually pretty simple, so as an example we will look at adding a version of POP3 on a NetBSD system.

In this case we have retrieved and installed the "cucipop" package, which can be found in `pkgsrc/mail/cucipop`. This server is pretty simple to use, the only oddities are different path locations. Since it is POP3 we know it is a stream oriented connection with "nowait". Running as "root" will be fine, the only item that is different is the location of the program and the name of the program itself.

So the first half of the new entry in /etc/inetd.conflooks like this:

```
pop3   stream  tcp     nowait  root
```

After installation, pkgsrc deposited cucipop in /usr/pkg/sbin/cucipop. So with the next field we have:

```
pop3   stream  tcp     nowait  root /usr/pkg/sbin/cucipop
```

Last, we want to use the Berkeley mailbox format, so our server program must be called with the -Y option. This leaves the entire entry looking like so:

```
pop3   stream  tcp     nowait  root /usr/pkg/sbin/cucipop cucipop -Y
```

We have added the service named "pop3" to /etc/inetd.conf. Next item to check is that the system can map the service name to a port number in /etc/services:

```
# grep ^pop3 /etc/services
pop3            110/tcp         # POP version 3
pop3            110/udp
pop3s           995/tcp                 # pop3 protocol over TLS/SSL (was spop3)
pop3s           995/udp                 # pop3 protocol over TLS/SSL (was spop3)
```

The "pop3" entries here are of interest, i.e. they are already contained in the /etc/services file shipped with NetBSD.

Now, to have inetd(8) use the new entry, we simply restart it using the rc script:

```
# sh /etc/rc.d/inetd restart
```

All done, in most cases, the software you are using has documentation that will specify the entry, in the off case it does not, sometimes it helps to try and find something similar to the server program you will be adding. A classic example of this is a MUD server which has built-in telnet. You can pretty much borrow the telnet entry and change parts where needed.

23.9 When to use or not to use inetd

The decision to add or move a service into or out of inetd(8) is usually based on server load. As an example, on most systems the telnet daemon does not require as many new connections as say a mail server. Most of the time the administrator has to feel out if a service should be moved.

A good example I have seen is mail services such as smtp and pop. I had setup a mail server in which pop3 was in inetd(8) and exim was running in standalone, I mistakenly assumed it would run fine since there was a low amount of users, namely myself and a diagnostic account. The server was also setup to act as a backup MX and relay in case another heavily used one went down. When I ran some tests I discovered a huge time lag for pop connections remotely. This was because of my steady fetching of mail and the diagnostic user constantly mailing diagnostics back and forth. In the end I had to move the pop3 service out of inetd(8).

The reason for moving the service is actually quite interesting. When a particular service becomes heavily used, of course, it causes a load on the system. In the case of a service that runs within the inetd(8) meta daemon the effects of a heavily loaded service can also harm other services that use

inetd(8). If the multiplexor is getting too many requests for one particular service, it will begin to affect the performance of other services that use inetd(8). The fix, in a situation like that, is to make the offending service run outside of inetd(8) so the response time of both the service and inetd(8) will increase.

23.10 Other Resources

Following is some additional reading and information about topics covered in this document.

NetBSD manual pages:

- inetd(8) (http://netbsd.gw.com/cgi-bin/man-cgi/man?inetd+8+NetBSD-current)
- protocols(5) (http://netbsd.gw.com/cgi-bin/man-cgi/man?protocols+5+NetBSD-current)
- rpc(5) (http://netbsd.gw.com/cgi-bin/man-cgi/man?rpc+5+NetBSD-current)
- services(5) (http://netbsd.gw.com/cgi-bin/man-cgi/man?services+5+NetBSD-current)
- hosts_access(5) (http://netbsd.gw.com/cgi-bin/man-cgi/man?hosts_access+5+NetBSD-current)

Miscellaneous links:

- IANA: Protocol Numbers and Assignment Services (http://www.iana.org/numbers.htm)
- RFC1700: Assigned Numbers (http://www.isi.edu/in-notes/rfc1700.txt)

Chapter 24
The Domain Name System

Use of the Domain Name System has been discussed in previous chapters, without going into detail on the setup of the server providing the service. This chapter describes setting up a simple, small domain with one Domain Name System (DNS) nameserver on a NetBSD system. It includes a brief explanation and overview of the DNS; further information can be obtained from the DNS Resources Directory (DNSRD) at http://www.dns.net/dnsrd/.

24.1 DNS Background and Concepts

The DNS is a widely used *naming service* on the Internet and other TCP/IP networks. The network protocols, data and file formats, and other aspects of the DNS are Internet Standards, specified in a number of RFC documents, and described by a number of other reference and tutorial works. The DNS has a distributed, client-server architecture. There are reference implementations for the server and client, but these are not part of the standard. There are a number of additional implementations available for many platforms.

24.1.1 Naming Services

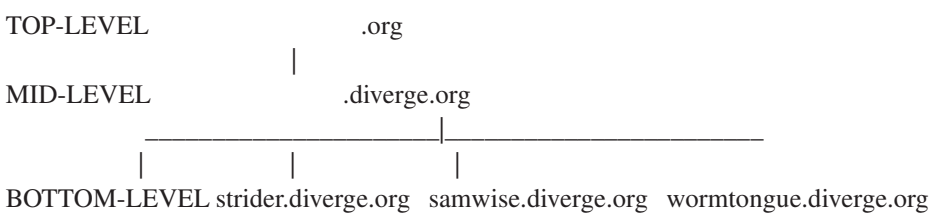
Naming services are used to provide a mapping between textual names and configuration data of some form. A *nameserver* maintains this mapping, and clients request the nameserver to *resolve* a name into its attached data.

The reader should have a good understanding of basic hosts to IP address mapping and IP address class specifications, see Section 21.6.

In the case of the DNS, the configuration data bound to a name is in the form of standard *Resource Records* (RR's). These textual names conform to certain structural conventions.

24.1.2 The DNS namespace

The DNS presents a hierarchical name space, much like a UNIX filesystem, pictured as an inverted tree with the *root* at the top.



The system can also be logically divided even further if one wishes at different points. The example shown above shows three nodes on the diverge.org domain, but we could even divide diverge.org into subdomains such as "strider.net1.diverge.org", "samwise.net2.diverge.org" and "wormtongue.net2.diverge.org"; in this case, 2 nodes reside in "net2.diverge.org" and one in "net1.diverge.org".

There are directories of names, some of which may be sub-directories of further names. These directories are sometimes called *zones*. There is provision for symbolic links, redirecting requests for information on one name to the records bound to another name. Each name recognised by the DNS is called a *Domain Name*, whether it represents information about a specific host, or a directory of subordinate Domain Names (or both, or something else).

Unlike most filesystem naming schemes, however, Domain Names are written with the innermost name on the left, and progressively higher-level domains to the right, all the way up to the root directory if necessary. The separator used when writing Domain Names is a period, ".".

Like filesystem pathnames, Domain Names can be written in an absolute or relative manner, though there are some differences in detail. For instance, there is no way to indirectly refer to the parent domain like with the UNIX `..` directory. Many (but not all) resolvers offer a search path facility, so that partially-specified names can be resolved relative to additional listed sub-domains other than the client's own domain. Names that are completely specified all the way to the root are called *Fully Qualified Domain Names* or *FQDN*s. A defining characteristic of an FQDN is that it is written with a terminating period. The same name, without the terminating period, may be considered relative to some other sub-domain. It is rare for this to occur without malicious intent, but in part because of this possibility, FQDNs are required as configuration parameters in some circumstances.

On the Internet, there are some established conventions for the names of the first few levels of the tree, at which point the hierarchy reaches the level of an individual organisation. This organisation is responsible for establishing and maintaining conventions further down the tree, within its own domain.

24.1.3 Resource Records

Resource Records for a domain are stored in a standardised format in an ASCII text file, often called a *zone file*. The following Resource Records are commonly used (a number of others are defined but not often used, or no longer used). In some cases, there may be multiple RR types associated with a name, and even multiple records of the same type.

Common DNS Resource Records

A: Address

> This record contains the numerical IP address associated with the name.

CNAME: Canonical Name

> This record contains the Canonical Name (an FQDN with an associated A record) of the host name to which this record is bound. This record type is used to provide name aliasing, by providing a link to another name with which other appropriate RR's are associated. If a name has a CNAME record bound to it, it is an alias, and no other RR's are permitted to be bound to the same name.

It is common for these records to be used to point to hosts providing a particular service, such as an FTP or HTTP server. If the service must be moved to another host, the alias can be changed, and the same name will reach the new host.

PTR: Pointer

This record contains a textual name. These records are bound to names built in a special way from numerical IP addresses, and are used to provide a reverse mapping from an IP address to a textual name. This is described in more detail in Section 24.1.8.

NS: Name Server

This record type is used to *delegate* a sub-tree of the Domain Name space to another nameserver. The record contains the FQDN of a DNS nameserver with information on the sub-domain, and is bound to the name of the sub-domain. In this manner, the hierarchical structure of the DNS is established. Delegation is described in more detail in Section 24.1.4.

MX: Mail eXchange

This record contains the FQDN for a host that will accept SMTP electronic mail for the named domain, together with a priority value used to select an MX host when relaying mail. It is used to indicate other servers that are willing to receive and spool mail for the domain if the primary MX is unreachable for a time. It is also used to direct email to a central server, if desired, rather than to each and every individual workstation.

HINFO: Host Information

Contains two strings, intended for use to describe the host hardware and operating system platform. There are defined strings to use for some systems, but their use is not enforced. Some sites, because of security considerations, do not publicise this information.

TXT: Text

A free-form text field, sometimes used as a comment field, sometimes overlaid with site-specific additional meaning to be interpreted by local conventions.

SOA: Start of Authority

This record is required to appear for each zone file. It lists the primary nameserver and the email address of the person responsible for the domain, together with default values for a number of fields associated with maintaining consistency across multiple servers and caching of the results of DNS queries.

24.1.4 Delegation

Using NS records, authority for portions of the DNS namespace below a certain point in the tree can be delegated, and further sub-parts below that delegated again. It is at this point that the distinction between a domain and a zone becomes important. Any name in the DNS is called a domain, and the term applies to that name and to any subordinate names below that one in the tree. The boundaries of a zone are narrower, and are defined by delegations. A zone starts with a delegation (or at the root), and encompasses all names in the domain below that point, excluding names below any subsequent delegations.

This distinction is important for implementation - a zone is a single administrative entity (with a single SOA record), and all data for the zone is referred to by a single file, called a *zone file*. A zone file may contain more than one period-separated level of the namespace tree, if desired, by including periods in the names in that zone file. In order to simplify administration and prevent overly-large zone files, it is quite legal for a DNS server to delegate to itself, splitting the domain into several zones kept on the same server.

24.1.5 Delegation to multiple servers

For redundancy, it is common (and often administratively required) that there be more than one nameserver providing information on a zone. It is also common that at least one of these servers be located at some distance (in terms of network topology) from the others, so that knowledge of that zone does not become unavailable in case of connectivity failure. Each nameserver will be listed in an NS record bound to the name of the zone, stored in the parent zone on the server responsible for the parent domain. In this way, those searching the name hierarchy from the top down can contact any one of the servers to continue narrowing their search. This is occasionally called *walking the tree*.

There are a number of nameservers on the Internet which are called *root nameservers*. These servers provide information on the very top levels of the domain namespace tree. These servers are special in that their addresses must be pre-configured into nameservers as a place to start finding other servers. Isolated networks that cannot access these servers may need to provide their own root nameservers.

24.1.6 Secondaries, Caching, and the SOA record

In order to maintain consistency between these servers, one is usually configured as the *primary* server, and all administrative changes are made on this server. The other servers are configured as *secondaries*, and transfer the contents of the zone from the primary. This operational model is not required, and if external considerations require it, multiple primaries can be used instead, but consistency must then be maintained by other means. DNS servers that store Resource Records for a zone, whether they be primary or secondary servers, are said to be *authoritative* for the zone. A DNS server can be authoritative for several zones.

When nameservers receive responses to queries, they can *cache* the results. This has a significant beneficial impact on the speed of queries, the query load on high-level nameservers, and network utilisation. It is also a major contributor to the memory usage of the nameserver process.

There are a number of parameters that are important to maintaining consistency amongst the secondaries and caches. The values for these parameters for a particular domain zone file are stored in the SOA record. These fields are:

Fields of the SOA Record

Serial

> A serial number for the zone file. This should be incremented any time the data in the domain is changed. When a secondary wants to check if its data is up-to-date, it checks the serial number on the primary's SOA record.

Refresh

A time, in seconds, specifying how often the secondary should check the serial number on the primary, and start a new transfer if the primary has newer data.

Retry

If a secondary fails to connect to the primary when the refresh time has elapsed (for example, if the host is down), this value specifies, in seconds, how often the connection should be retried.

Expire

If the retries fail to reach the primary within this number of seconds, the secondary destroys its copies of the zone data file(s), and stops answering requests for the domain. This stops very old and potentially inaccurate data from remaining in circulation.

TTL

This field specifies a time, in seconds, that the resource records in this zone should remain valid in the caches of other nameservers. If the data is volatile, this value should be short. TTL is a commonly-used acronym, that stands for "Time To Live".

24.1.7 Name Resolution

DNS clients are configured with the addresses of DNS servers. Usually, these are servers which are authoritative for the domain of which they are a member. All requests for name resolution start with a request to one of these local servers. DNS queries can be of two forms:

- A *recursive* query asks the nameserver to resolve a name completely, and return the result. If the request cannot be satisfied directly, the nameserver looks in its configuration and caches for a server higher up the domain tree which may have more information. In the worst case, this will be a list of pre-configured servers for the root domain. These addresses are returned in a response called a *referral*. The local nameserver must then send its request to one of these servers.
- Normally, this will be an *iterative* query, which asks the second nameserver to either respond with an authoritative reply, or with the addresses of nameservers (NS records) listed in its tables or caches as authoritative for the relevant zone. The local nameserver then makes iterative queries, walking the tree downwards until an authoritative answer is found (either positive or negative) and returned to the client.

In some configurations, such as when firewalls prevent direct IP communications between DNS clients and external nameservers, or when a site is connected to the rest of the world via a slow link, a nameserver can be configured with information about a *forwarder*. This is an external nameserver to which the local nameserver should make requests as a client would, asking the external nameserver to perform the full recursive name lookup, and return the result in a single query (which can then be cached), rather than reply with referrals.

24.1.8 Reverse Resolution

The DNS provides resolution from a textual name to a resource record, such as an A record with an IP address. It does not provide a means, other than exhaustive search, to match in the opposite direction;

there is no mechanism to ask which name is bound to a particular RR.

For many RR types, this is of no real consequence, however it is often useful to identify by name the host which owns a particular IP address. Rather than complicate the design and implementation of the DNS database engine by providing matching functions in both directions, the DNS utilises the existing mechanisms and creates a special namespace, populated with PTR records, for IP address to name resolution. Resolving in this manner is often called *reverse resolution*, despite the inaccurate implications of the term.

The manner in which this is achieved is as follows:

- A normal domain name is reserved and defined to be for the purpose of mapping IP addresses. The domain name used is "in-addr.arpa." which shows the historical origins of the Internet in the US Government's Defence Advanced Research Projects Agency's funding program.
- This domain is then subdivided and delegated according to the structure of IP addresses. IP addresses are often written in *decimal dotted quad notation*, where each octet of the 4-octet long address is written in decimal, separated by dots. IP address ranges are usually delegated with more and more of the left-most parts of the address in common as the delegation gets smaller. Thus, to allow delegation of the reverse lookup domain to be done easily, this is turned around when used with the hierarchical DNS namespace, which places higher level domains on the right of the name.
- Each byte of the IP address is written, as an ASCII text representation of the number expressed in decimal, with the octets in reverse order, separated by dots and appended with the in-addr.arpa. domain name. For example, to determine the hostname of a network device with IP address 11.22.33.44, this algorithm would produce the string "44.33.22.11.in-addr.arpa." which is a legal, structured Domain Name. A normal nameservice query would then be sent to the nameserver asking for a PTR record bound to the generated name.
- The PTR record, if found, will contain the FQDN of a host.

One consequence of this is that it is possible for mismatch to occur. Resolving a name into an A record, and then resolving the name built from the address in that A record to a PTR record, may not result in a PTR record which contains the original name. There is no restriction within the DNS that the "reverse" mapping must coincide with the "forward" mapping. This is a useful feature in some circumstances, particularly when it is required that more than one name has an A record bound to it which contains the same IP address.

While there is no such restriction within the DNS, some application server programs or network libraries will reject connections from hosts that do not satisfy the following test:

- the state information included with an incoming connection includes the IP address of the source of the request.
- a PTR lookup is done to obtain an FQDN of the host making the connection
- an A lookup is then done on the returned name, and the connection rejected if the source IP address is not listed amongst the A records that get returned.

This is done as a security precaution, to help detect and prevent malicious sites impersonating other sites by configuring their own PTR records to return the names of hosts belonging to another organisation.

24.2 The DNS Files

Now let's look at actually setting up a small DNS enabled network. We will continue to use the examples mentioned in Chapter 22, i.e. we assume that:

- Our IP networking is working correctly
- We have IPNAT working correctly
- Currently all hosts use the ISP for DNS

Our Name Server will be the "strider" host which also runs IPNAT, and our two clients use "strider" as a gateway. It is not really relevant as to what type of interface is on "strider", but for argument's sake we will say a 56k dial up connection.

So, before going any further, let's look at our `/etc/hosts` file on "strider" before we have made the alterations to use DNS.

Example 24-1. strider's `/etc/hosts` file

```
127.0.0.1       localhost
192.168.1.1     strider
192.168.1.2     samwise sam
192.168.1.3     wormtongue worm
```

This is not exactly a huge network, but it is worth noting that the same rules apply for larger networks as we discuss in the context of this section.

The other assumption we want to make is that the domain we want to set up is `diverge.org`, and that the domain is only known on our internal network, and not worldwide. Proper registration of the nameserver's IP address as primary would be needed in addition to a static IP. These are mostly administrative issues which are left out here.

The NetBSD operating system provides a set of config files for you to use for setting up DNS. They are stored in the `/etc/namedb` directory, I strongly suggest making a backup copy of this directory for reference purposes.

The default directory contains the following files:

- `named.conf`
- `localhost`
- `127`
- `loopback.v6`
- `root.cache`

You will see modified versions of these files in this section.

> **Note:** The examples in this chapter refer to BIND major version 8, however, it should be noted that format of the name database and other config files are almost 100% compatible between version. The only difference I noticed was that the "$TTL" information was not required.

24.2.1 /etc/namedb/named.conf

The first file we want to look at is /etc/namedb/named.conf. This file is the config file for bind (hence the catchy name). Setting up system like the one we are doing is relatively simple. First, here is what mine looks like:

```
options {
        directory "/etc/namedb";
        allow-transfer { 192.168.1.0/24; };
        allow-query { 192.168.1.0/24; };
        listen-on port 53 { 192.168.1.1; };
};

zone "localhost" {
   type master;
   notify no;
   file "localhost";
};

zone "127.IN-ADDR.ARPA" {
   type master;
   notify no;
   file "127";
};

zone "0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.int" {
   type master;
   file "loopback.v6";
};

zone "diverge.org" {
   type master;
   notify no;
   file "diverge.org";
};

zone "1.168.192.in-addr.arpa" {
   type master;
   notify no;
   file "1.168.192";
};

zone "." in {
   type hint;
   file "root.cache";
};
```

Note that in my named.conf the root (".") section is last, that is because there is another domain called diverge.org on the internet (I happen to own it) so I want the resolver to look out on the internet last. This is not normally the case on most systems.

Another very important thing to remember here is that if you have an internal setup, in other words no live internet connection and/or no need to do root server lookups, comment out the root (".") zone. It may

cause lookup problems if a particular client decides it wants to reference a domain on the internet, which our server couldn't resolve itself.

Looks like a pretty big mess, upon closer examination it is revealed that many of the lines in each section are somewhat redundant. So we should only have to explain them a few times.

Lets go through the sections of `named.conf`:

24.2.1.1 options

This section defines some global parameters, most noticeable is the location of the DNS tables, on this particular system, they will be put in `/etc/namedb` as indicated by the "directory" option.

Following are the rest of the params:

allow-transfer

> This option lists which remote DNS servers acting as secondaries are allowed to do zone transfers, i.e. are allowed to read all DNS data at once. For privacy reasons, this should be restricted to secondary DNS servers only.

allow-query

> This option defines hosts from what network may query this name server at all. Restricting queries only to the local network (192.168.1.0/24) prevents queries arriving on the DNS server's external interface, and prevent possible privacy issues.

listen-on port

> This option defined the port and associated IP addresses this server will run named(8) on. Again, the "external" interface is not listened here, to prevent queries getting received from "outside".

The rest of the `named.conf` file consists of "zone"s. A zone is an area that can have items to resolve attached, e.g. a domain can have hostnames attached to resolve into IP addresses, and a reverse-zone can have IP addresses attached that get resolved back into hostnames. Each zone has a file associated with it, and a table within that file for resolving that particular zone. As is readily apparent, their format in `named.conf` is strikingly similar, so I will highlight just one of their records:

24.2.1.2 zone "diverge.org"

type

> The type of a zone is usually of type "master" in all cases except for the root zone "." and for zones that a secondary (backup) service is provided - the type obviously is "secondary" in the latter case.

notify

> Do you want to send out notifications to secondaries when your zone changes? Obviously not in this setup, so this is set to "no".

file

> This option sets the filename in our `/etc/namedb` directory where records about this particular zone may be found. For the "diverge.org" zone, the file `/etc/namedb/diverge.org` is used.

24.2.2 `/etc/namedb/localhost`

For the most part, the zone files look quite similar, however, each one does have some unique properties. Here is what the `localhost` file looks like:

Example 24-2. `localhost`

```
 1|$TTL    3600
 2|@              IN SOA  strider.diverge.org. root.diverge.org. (
 3|                        1       ; Serial
 4|                        8H      ; Refresh
 5|                        2H      ; Retry
 6|                        1W      ; Expire
 7|                        1D)     ; Minimum TTL
 8|                IN NS   localhost.
 9|localhost.      IN      A       127.0.0.1
10|                IN      AAAA    ::1
```

Line by line:

Line 1:

> This is the Time To Live for lookups, which defines how long other DNS servers will cache that value before discarding it. This value is generally the same in all the files.

Line 2:

> This line is generally the same in all zone files except `root.cache`. It defines a so-called "Start Of Authority" (SOA) header, which contains some basic information about a zone. Of specific interest on this line are "strider.diverge.org." and "root.diverge.org." (note the trailing dots!). Obviously one is the name of this server and the other is the contact for this DNS server, in most cases root seems a little ambiguous, it is preferred that a regular email account be used for the contact information, with the "@" replaced by a "." (for example, mine would be "jrf.diverge.org.").

Line 3:

> This line is the serial number identifying the "version" of the zone's data set (file). The serial number should be incremented each time there is a change to the file, the usual forat is to either start with a value of "1" and increase it for every change, or use a value of "YYYYMMDDNN" to encode year (YYYY), month (MM), day (DD) and change within one day (NN) in the serial number.

Line 4:

> This is the refresh rate of the server, in this file it is set to once every 8 hours.

Line 5:

> The retry rate.

Line 6:

> Lookup expiry.

Line 7:

The minimum Time To Live.

Line 8:

This is the Nameserver line, which uses a "NS" ressource record to show that "localhost" is the only DNS server handing out data for this zone (which is "@", which indicates the zone name used in the `named.conf` file, i.e. "diverge.org") is, well, "localhost".

Line 9:

This is the localhost entry, which uses an "A" ressource record to indicate that the name "localhost" should be resolved into the IP-address 127.0.0.1 for IPv4 queries (which specifically ask for the "A" record).

Line 10:

This line is the IPv6 entry, which returns ::1 when someone asks for an IPv6-address (by specifically asking for the AAAA record) of "localhost.".

24.2.3 `/etc/namedb/zone.127.0.0`

This is the reverse lookup file (or zone) to resolve the special IP address 127.0.0.1 back to "localhost":

```
1| $TTL    3600
2| @               IN SOA  strider.diverge.org. root.diverge.org. (
3|                         1       ; Serial
4|                         8H      ; Refresh
5|                         2H      ; Retry
6|                         1W      ; Expire
7|                         1D)     ; Minimum TTL
8|                  IN NS   localhost.
9| 1.0.0            IN PTR  localhost.
```

In this file, all of the lines are the same as the localhost zonefile with exception of line 9, this is the reverse lookup (PTR) record. The zone used here is "@" again, which got set to the value given in `named.conf`, i.e. "127.in-addr.arpa". This is a special "domain" which is used to do reverse-lookup of IP addresses back into hostnames. For it to work, the four bytes of the IPv4 address are reserved, and the domain "in-addr.arpa" attached, so to resolve the IP address "127.0.0.1", the PTR record of "1.0.0.127.in-addr.arpa" is queried, which is what is defined in that line.

24.2.4 `/etc/namedb/diverge.org`

This zone file is populated by records for all of our hosts. Here is what it looks like:

```
1| $TTL    3600
2| @               IN SOA  strider.diverge.org. root.diverge.org. (
3|                          1       ; serial
4|                          8H      ; refresh
5|                          2H      ; retry
6|                          1W      ; expire
```

```
 7|                         1D )    ; minimum seconds
 8|                 IN NS   strider.diverge.org.
 9|                 IN MX   10 strider.diverge.org.   ; primary mail server
10|                 IN MX   20 samwise.diverge.org.   ; secondary mail server
11| strider         IN A     192.168.1.1
12| samwise         IN A     192.168.1.2
13| www             IN CNAME samwise.diverge.org.
14| worm            IN A     192.168.1.3
```

There is a lot of new stuff here, so lets just look over each line that is new here:

Line 9

This line shows our mail exchanger (MX), in this case it is "strider". The number that precedes "strider.diverge.org." is the priority number, the lower the number their higher the priority. The way we are setup here is if "strider" cannot handle the mail, then "samwise" will.

Line 11

CNAME stands for canonical name, or an alias for an existing hostname, which must have an A record. So we have aliased the following:

www.diverge.org to samwise.diverge.org

The rest of the records are simply mappings of IP address to a full name (A records).

24.2.5 `/etc/namedb/1.168.192`

This zone file is the reverse file for all of the host records, to map their IP numbers we use on our private network back into hostnames. The format is similar to that of the "localhost" version with the obvious exception being the addresses are different via the different zone given in the `named.conf` file, i.e. "0.168.192.in-addr.arpa" here:

```
 1|$TTL    3600
 2|@               IN SOA  strider.diverge.org. root.diverge.org. (
 3|                       1       ; serial
 4|                       8H      ; refresh
 5|                       2H      ; retry
 6|                       1W      ; expire
 7|                       1D )    ; minimum seconds
 8|                IN NS   strider.diverge.org.
 9|1               IN PTR  strider.diverge.org.
10|2               IN PTR  samwise.diverge.org.
11|3               IN PTR  worm.diverge.org.
```

24.2.6 `/etc/namedb/root.cache`

This file contains a list of root name servers for your server to query when it gets requests outside of its own domain that it cannot answer itself. Here are first few lines of a root zone file:

```
;
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/db.cache
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:    Jan 29, 2004
;       related version of root zone:   2004012900
;
;
; formerly NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
;
; formerly NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     192.228.79.201
;
; formerly C.PSI.NET
;
.                        3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
;
...
```

This file can be obtained from ISC at http://www.isc.org/ and usually comes with a distribution of BIND. A `root.cache` file is included in the NetBSD operating system's "etc" set.

This section has described the most important files and settings for a DNS server. Please see the BIND documentation in `/usr/src/dist/bind/doc/bog` and named.conf(5) for more information.

24.3 Using DNS

In this section we will look at how to get DNS going and setup "strider" to use its own DNS services.

Setting up named to start automatically is quite simple. In `/etc/rc.conf` simply set `named=yes`. Additional options can be specified in `named_flags`, for example, I like to use `-g nogroup -u nobody`, so a non-root account runs the "named" process.

In addition to being able to startup "named" at boot time, it can also be controlled with the **ndc** command. In a nutshell the **ndc** command can stop, start or restart the named server process. It can also do a great many other things. Before use, it has to be setup to communicate with the "named" process,

see the ndc(8) and named.conf(5) man pages for more details on setting up communication channels between "ndc" and the "named" process.

Next we want to point "strider" to itself for lookups. We have two simple steps, first, decide on our resolution order. On a network this small, it is likely that each host has a copy of the hosts table, so we can get away with using `/etc/hosts` first, and then DNS. However, on larger networks it is much easier to use DNS. Either way, the file where order of name services used for resolution is determined is `/etc/nsswitch.conf` (see Example 22-2). Here is part of a typical `nsswitch.conf`:

```
. . .
group_compat:   nis
hosts:          files dns
netgroup:       files [notfound=return] nis
. . .
```

The line we are interested in is the "hosts" line. "files" means the system uses the `/etc/hosts` file first to determine ip to name translation, and if it can't find an entry, it will try DNS.

The next file to look at is `/etc/resolv.conf`, which is used to configure DNS lookups ("resolution") on the client side. The format is pretty self explanatory but we will go over it anyway:

```
domain diverge.org
search diverge.org
nameserver 192.168.1.1
```

In a nutshell this file is telling the resolver that this machine belongs to the "diverge.org" domain, which means that lookups that contain only a hostname without a "." gets this domain appended to build a FQDN. If that lookup doesn't succeed, the domains in the "search" line are tried next. Finally, the "nameserver" line gives the IP addresses of one or more DNS servers that should be used to resolve DNS queries.

To test our nameserver we can use several commands, for example:

```
# host sam
sam.diverge.org has address 192.168.1.2
```

As can be seen, the domain was appended automatically here, using the value from `/etc/resolv.conf`. Here is another example, the output of running **host www.yahoo.com**:

```
$ host www.yahoo.com
www.yahoo.com is an alias for www.yahoo.akadns.net.
www.yahoo.akadns.net has address 68.142.226.38
www.yahoo.akadns.net has address 68.142.226.39
www.yahoo.akadns.net has address 68.142.226.46
www.yahoo.akadns.net has address 68.142.226.50
www.yahoo.akadns.net has address 68.142.226.51
www.yahoo.akadns.net has address 68.142.226.54
www.yahoo.akadns.net has address 68.142.226.55
www.yahoo.akadns.net has address 68.142.226.32
```

Other commands for debugging DNS besides host(1) are nslookup(8) and dig(1). Note that ping(8) is *not* useful for debugging DNS, as it will use whatever is configured in `/etc/nsswitch.conf` to do the name-lookup.

At this point the server is configured properly. The procedure for setting up the client hosts are are easier, you only need to setup `/etc/nsswitch.conf` and `/etc/resolv.conf` to the same values as on the server.

24.4 Setting up a caching only name server

A caching only name server has no local zones; all the queries it receives are forwarded to the root servers and the replies are accumulated in the local cache. The next time the query is performed the answer will be faster because the data is already in the server's cache. Since this type of server doesn't handle local zones, to resolve the names of the local hosts it will still be necessary to use the already known `/etc/hosts` file.

Since NetBSD supplies defaults for all the files needed by a caching only server, it only needs to be enabled and started and is immediately ready for use! To enable named, put `named=yes` into `/etc/rc.conf`, and tell the system to use it adding the following line to the `/etc/resolv.conf` file:

```
# cat /etc/resolv.conf
nameserver 127.0.0.1
```

Now we can start named:

```
# sh /etc/rc.d/named restart
```

24.4.1 Testing the server

Now that the server is running we can test it using the nslookup(8) program:

```
$ nslookup
Default server: localhost
Address: 127.0.0.1

>
```

Let's try to resolve a host name, for example "www.NetBSD.org":

```
> www.NetBSD.org
Server:  localhost
Address:  127.0.0.1

Name:    www.NetBSD.org
Address:  204.152.190.12
```

If you repeat the query a second time, the result is slightly different:

```
> www.NetBSD.org
Server:  localhost
Address:  127.0.0.1

Non-authoritative answer:
Name:    www.NetBSD.org
Address:  204.152.190.12
```

As you've probably noticed, the address is the same, but the message "Non-authoritative answer" has appeared. This message indicates that the answer is not coming from an authoritative server for the domain NetBSD.org but from the cache of our own server.

The results of this first test confirm that the server is working correctly.

We can also try the host(1) and dig(1) commands, which give the following result.

```
$ host www.NetBSD.org
www.NetBSD.org has address 204.152.190.12
$
$ dig www.NetBSD.org

; <<>> DiG 8.3 <<>> www.NetBSD.org
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19409
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 5, ADDITIONAL: 0
;; QUERY SECTION:
;;      www.NetBSD.org, type = A, class = IN

;; ANSWER SECTION:
www.NetBSD.org.         23h32m54s IN A  204.152.190.12

;; AUTHORITY SECTION:
NetBSD.org.             23h32m54s IN NS  uucp-gw-1.pa.dec.com.
NetBSD.org.             23h32m54s IN NS  uucp-gw-2.pa.dec.com.
NetBSD.org.             23h32m54s IN NS  ns.NetBSD.org.
NetBSD.org.             23h32m54s IN NS  adns1.berkeley.edu.
NetBSD.org.             23h32m54s IN NS  adns2.berkeley.edu.

;; Total query time: 14 msec
;; FROM: miyu to SERVER: 127.0.0.1
;; WHEN: Thu Nov 25 22:59:36 2004
;; MSG SIZE  sent: 32  rcvd: 175
```

As you can see dig(1) gives quite a bit of output, the expected answer can be found in the "ANSWER SECTION". The other data given may be of interest when debugging DNS problems.

Chapter 25
Mail and news

This chapter explains how to set up NetBSD to use mail and news. Only a simple but very common setup is described: the configuration of a host connected to the Internet with a modem through a provider. You can think of this chapter as the continuation of Chapter 22, assuming a similar network configuration. Even this "simple" setup proves to be difficult if you don't know where to start or if you've only read introductory or technical documentation; in fact you will notice that some details are really challenging (for example, mapping your internal network names to "real" names requires a good knowledge of sendmail). A general description of mail and news configuration is beyond the scope of this guide; please read a good Unix Administration book (some very good ones are listed on the NetBSD site). The problem is in fact very complex because of the myriad of possible configurations and connections and because it's not enough to configure a single program - you need to correctly match the configuration of several cooperating components.

This chapter also briefly describes the configuration (but not the usage) of two popular applications, mutt for mail and tin for news. The usage is not described because they are easy to use and well documented. Obviously, both mutt and tin are not mandatory choices: many other similar applications exist but I think that they are a good starting point because they are widely used, simple, work well and don't use too much disk space and memory. Both are console mode programs; if you prefer graphics applications there are also many choices for X.

In short, the programs required for the configuration described in this chapter are:

- sendmail
- fetchmail
- m4
- mutt
- tin

Of these, only sendmail and m4 are installed with the base system; you can install the other programs from the NetBSD package collection, pkgsrc - see Part VI in *The NetBSD & pkgsrc Guide*.

Before continuing, remember that none of the programs presented in this chapter is mandatory: there are other applications performing similar tasks and many users prefer them. You'll find different opinions reading the mailing lists. You can also use different strategies for sending and receiving mail: the one explained here is only a starting point; once you understand how it works you'll probably want to modify it to suit your needs or to adopt a different method altogether.

At the opposite extreme of the example presented here, there is the usage of an application like Mozilla, which does everything and frees you from the need of configuring many components: with Mozilla you can browse the Internet, send and receive mail and read news. Besides, the setup is very simple. There is a price to pay, though: Mozilla is a "closed" program that will not cooperate easily with other standard Unix utilities.

Another possibility is to use emacs to read mail and news. Emacs needs no introduction to Unix users but, in case you don't know, it is an extensible editor (although calling emacs an editor is somewhat reductive) which becomes a complete work environment, and can be used to read mail, news and to perform many operations. For many people emacs is the only environment that they need and they use it for all their work. The configuration of emacs for mail and news is described in the emacs manual.

In the rest of this chapter we will deal with a host connected to the Internet through a PPP connection via serial modem to a provider.

- the local host's name is "ape" and the internal network is "insetti.net", which means that the FQDN (Fully Qualified Domain Name) is "ape.insetti.net".
- the user's login name on ape is "carlo".
- the provider's name is BigNet.
- the provider's mail server is "mail.bignet.it".
- the provider's news server is "news.bignet.it".
- the user's ("carlo") account at the provider is "alan" with the password "pZY9o".

First some basic terminology:

MUA (mail user agent)

: a program to read and write mail. For example: mutt, elm and pine but also the simple mail application installed with the base system.

MTA (mail transfer agent)

: a program that transfers mail between two host but also locally (on the same host). The MTA decides the path that the mail will follow to get to the destination. On BSD systems (but not only) the standard MTA is sendmail, other examples are postfix, qmail, exim and Microsoft Exchange.

MDA (mail delivery agent)

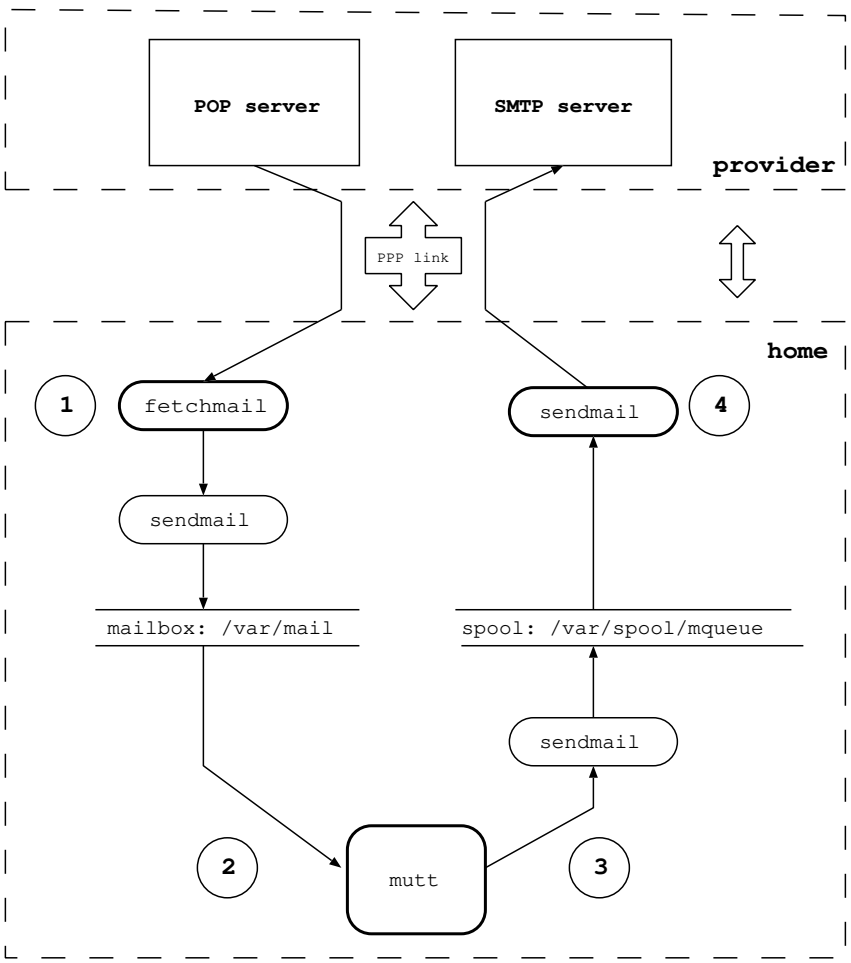
: a program, usually used by the MTA, that delivers the mail; for example, it physically puts the messages in the recipient's mailbox. For example, sendmail uses one or more MDA to deliver mail, and procmail is another well-known MDA.

Figure 25-1 depicts the mail system that we want to set up. Between the local network (or the single host) and the provider there is a modem PPP connection. The "bubbles" with the thick border (sendmail, fetchmail, mutt) are the programs launched manually by the user; the remaining bubbles are the programs that are launched automatically. The circled numbers refere to the logical steps of the mail cycle:

1. In step (1) mail is downloaded from the provider's POP server using fetchmail, which hands messages off to sendmail to put the messages in the user's mailbox.
2. In step (2) the user launches mutt (or another MUA) to read mail, reply and write new messages.
3. In step (3) the user "sends" the mail from within mutt. Messages are accumulated in the spool area.
4. In step (4) the user calls sendmail to transfer the messages to the provider's SMTP server, that will deliver them to the final destination (possibly through other mail servers). The provider's SMTP server acts as a *relay* for our mail.

The connection with the provider must be up only during steps (1) and (4); for the remaining steps it is not needed.

Figure 25-1. Structure of the mail system



25.1 sendmail

When an MTA must deliver a local message, it is delivered directly. If the message is intended for a different domain, the MTA must find out the address of the mail server for that domain. Sendmail uses the DNS service (described in Chapter 24) to find a mail exchanger handling mail for the given domain, and delivers the message to that mail server then.

The most used *MTA* in the world is probably sendmail. Sendmail is controlled by a set of configuration files and databases, of which `/etc/mail/sendmail.cf` is the most important. In general, if you are not an expert it is better not to modify the `/etc/mail/sendmail.cf` file directly; instead, use a set of predefined macros and the m4 preprocessor, which greatly (well, almost) simplify the configuration.

Note: Prior to version 1.5 of NetBSD, the mail configuration files were in `/etc` instead of `/etc/mail`.

Even using the macros, the configuration of sendmail is not for the faint of heart, and the next sections only describe an example which can be modified to suit different needs and different configurations. If you connect to the Internet with a modem, the example configuration file will probably fit all your needs: just replace the fictitious data with yours.

The first problem to be solved is that the local network we are dealing with is an internal network, i.e. not directly accessible from the Internet. This means that the names used internally have no meaning on the Internet; in short, "ape.insetti.net" cannot be reached by an external host: no one will be able to reply to a mail sent with this return address (many mail systems will even reject the message as spam prevention as it comes from an unknown host). The true address, the one visible from everybody, is assigned by the provider and, therefore, it is necessary to convert the local address "carlo@ape.insetti.net" to the real address "alan@bignet.it". Sendmail, if correctly configured, will take care of this when it transfers the messages.

You'll probably also want to configure sendmail in order to send the e-mails to the provider's mail server, using it as a *relay*. In the configuration described in this chapter, sendmail does not directly contact the recipient's mail server (as previously described) but relays all its mail to the provider's mail server.

> **Note:** The provider's mail server acts as a *relay*, which means that it delivers mail which is not destined to its own domain, to another mail server. It acts as an intermediary between two servers.

Since the connection with the provider is not always active, it is not necessary to start sendmail as a daemon in `/etc/rc.conf`: you can disable it with the line "`sendmail=NO`". As a consequence it will be necessary to launch sendmail manually when you want to transfer mail to the provider. Local mail is delivered correctly even if sendmail is not active as a daemon.

Let's start configuring sendmail.

25.1.1 Configuration with genericstable

This type of configuration uses the file `/etc/mail/genericstable` which contains the mapping used by sendmail to rewrite the internal hostnames.

The first step is therefore to write the `genericstable` file. For example:

```
carlo:          alan@bignet.it
root:           alan@bignet.it
news:           alan@bignet.it
```

These entries will map the mail sent from the users given on the left side into the globally valid email addresses given on the right, making it appear as if the mail was really sent from that address.

For the sake of efficiency, `genericstable` must be transformed into a binary file with the following command:

```
# /usr/sbin/sendmail -bi -oA/etc/mail/genericstable
```

Now it's time to create the prototype configuration file which we'll use to create the sendmail configuration file.

```
# cd /usr/share/sendmail/cf
```

The new sendmail configuration file, which we'll call `mycf.mc`, contains:

```
divert(-1)dnl
include('../m4/cf.m4')dnl
VERSIONID('mycf.mc created by carlo@ape.insetti.net May 18 2001')dnl
OSTYPE(bsd4.4)dnl

dnl # Settings for masquerading.  Addresses of the following types
dnl # are rewritten
dnl #     carlo@ape.insetti.net
dnl #     carlo@ape
GENERICS_DOMAIN(ape.insetti.net ape)dnl
FEATURE(genericstable)dnl
FEATURE(masquerade_envelope)dnl

define('SMART_HOST','mail.bignet.it')dnl

FEATURE(redirect)dnl
FEATURE(nocanonify)dnl

dnl # The following feature is useful if sendmail is called by
dnl # fetchmail (which is usually the case).  If sendmail cannot
dnl # resolve the name of the sender, the mail will not be delivered.
dnl # For example:
dnl # MAIL FROM:<www-owner@NetBSD.org> SIZE=2718
dnl # 501 <www-owner@NetBSD.org>... Sender domain must exist
FEATURE('accept_unresolvable_domains')dnl

dnl # accept_unqualified_senders is useful with some MUA, which send
dnl # mail as, for example:
dnl # MAIL FROM:<carlo>
FEATURE('accept_unqualified_senders')dnl

dnl # Mail for 'smtp' mailer is marked 'expensive' ('e' flag):
dnl # instead of connecting with the relay, sendmail puts it in
dnl # a queue for delayed processing.
dnl # Sendmail starts complaining about undelivered messages after
dnl # 16 hours.
define('SMTP_MAILER_FLAGS','e')dnl
define('confCON_EXPENSIVE','True')dnl
define('confTO_QUEUEWARN', '16h')dnl

dnl # For european users
define('confDEF_CHAR_SET','ISO-8859-1')dnl

dnl # Enable the following three lines to use procmail as a local
dnl # delivery agent.  The third line is optional, only the first
dnl # two are required.
dnl # define('PROCMAIL_MAILER_PATH', /usr/pkg/bin/procmail)dnl
dnl # FEATURE(local_procmail)dnl
dnl # MAILER(procmail)dnl
```

```
dnl # The following two mailers must always be defined
MAILER(local)dnl
MAILER(smtp)dnl
```

> **Note:** In the previous example, everything after a “dnl” is considered a comment and will be discarded by the m4 preprocessor.

This configuration tells sendmail to rewrite the addresses of type “ape.insetti.net” using the real names found in the `/etc/mail/genericstable` file. It also says that mail should be sent to the “mail.bignet.it” server. The meaning of the options is described in detail in the file `/usr/share/sendmail/README`.

In order to create your own version of the example configuration file, you must change only two lines, substituting your real data:

```
GENERICS_DOMAIN(ape.insetti.net ape)dnl
define(‘SMART_HOST’,‘mail.bignet.it’)dnl
```

Finally, the new configuration file must be generated, after having saved the previous version:

```
# cp /etc/mail/sendmail.cf /etc/mail/sendmail.cf.bak
# m4 mycf.mc > /etc/mail/sendmail.cf
```

Note that there is afile `netbsd-proto.mc` in the `/usr/share/sendmail/cf` directory which is used to create the default `/etc/mail/sendmail.cf` shipped with NetBSD. With the **make** command it can be rebuilt, if needed.

Now everything is ready to start sending mail.

25.1.2 Testing the configuration

Sendmail is finally configured and ready to work, but before sending real mail it is better to do some simple tests. First let’s try sending a local e-mail with the following command:

```
$ sendmail -v carlo
Subject: test

Hello world
.
carlo... Connecting to local...
carlo... Sent
```

Please follow exactly the example above: leave a blank line after Subject: and end the message with a line containing only one dot. Now you should be able to read the message with your mail client and verify that the From: field has been correctly rewritten.

```
From: alan@bignet.it
```

Next you can verify the address rewriting rules directly, using sendmail in address test mode with the `-bt` option. This mode shows the parsing performed by sendmail for an address and how it gets rewritten according to the rules in the configuration file. It is also possible to perform other tests and view some information.

```
$ sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>
```

You can display the help with the "**?**" command.

First, let's verify that the genericstable map file works correctly:

```
/map generics carlo
map_lookup: generics (carlo) returns alan@bignet.it
```

Everything's ok here; sendmail found the local name and its real counterpart in the map.

Now we can test the rewriting of the envelope's sender address with the following commands:

```
/tryflags ES
/try smtp carlo@ape.insetti.net
```

The result should be similar to the following:

```
Trying envelope sender address carlo@ape.insetti.net for mailer smtp
rewrite: ruleset   3   input: carlo @ ape . insetti . net
rewrite: ruleset  96   input: carlo < @ ape . insetti . net >
rewrite: ruleset  96 returns: carlo < @ ape . insetti . net . >
rewrite: ruleset   3 returns: carlo < @ ape . insetti . net . >
rewrite: ruleset   1   input: carlo < @ ape . insetti . net . >
rewrite: ruleset   1 returns: carlo < @ ape . insetti . net . >
rewrite: ruleset  11   input: carlo < @ ape . insetti . net . >
rewrite: ruleset  51   input: carlo < @ ape . insetti . net . >
rewrite: ruleset  51 returns: carlo < @ ape . insetti . net . >
rewrite: ruleset  61   input: carlo < @ ape . insetti . net . >
rewrite: ruleset  61 returns: carlo < @ ape . insetti . net . >
rewrite: ruleset  94   input: carlo < @ ape . insetti . net . >
rewrite: ruleset  93   input: carlo < @ ape . insetti . net . >
rewrite: ruleset   3   input: alan @ bignet . it
rewrite: ruleset  96   input: alan < @ bignet . it >
rewrite: ruleset  96 returns: alan < @ bignet . it >
rewrite: ruleset   3 returns: alan < @ bignet . it >
rewrite: ruleset  93 returns: alan < @ bignet . it >
rewrite: ruleset  94 returns: alan < @ bignet . it >
rewrite: ruleset  11 returns: alan < @ bignet . it >
rewrite: ruleset   4   input: alan < @ bignet . it >
rewrite: ruleset   4 returns: alan @ bignet . it
Rcode = 0, addr = alan@bignet.it
>
```

As you can see, the local address has been rewritten to the real address, which will appear in your e-mails when they leave your system.

You can achieve a similar result with the following command:

```
/try smtp carlo
```

We can also verify the rewriting of the header's sender with the following commands:

```
/tryflags HS
/try smtp carlo@ape.insetti.net
```

25.1.3 Using an alternative MTA

Starting from version 1.4 of NetBSD sendmail is not called directly:

```
$ ls -l /usr/sbin/sendmail
lrwxr-xr-x  1 root  wheel  21 Nov  1 01:14 /usr/sbin/sendmail@ -> /usr/sbin/mailwrapper
```

The purpose of mailwrapper is to allow the usage of an alternative MTA instead of sendmail (for example, postfix). If you plan to use a different mailer I suggest that you read the mailwrapper(8) and the mailer.conf(5) manpages, which are very clear.

25.2 fetchmail

If someone sends me mail, it is received and stored by the provider, and not automatically transfered to the local hosts; therefore it is necessary to download it. Fetchmail is a very popular program that downloads mail from a remote mail server (using e.g. the Post Office Protocol, POP) and forwards it to the local system for delivery (usually using sendmail). It is powerful yet easy to use and configure: after installation, the file `~/.fetchmailrc` must be created and the program is ready to run (`~/.fetchmailrc` contains a password so appropriate permissions on the file are required).

This is an example `.fetchmailrc`:

```
poll mail.bignet.it
protocol POP3
username alan there with password pZY9o is carlo here
flush
mda "/usr/sbin/sendmail -oem %T"
```

The last line ("mda ...") is used only if sendmail is not active as daemon on the system. Please note that the POP-mail server indicated in this file (mail.bignet.it) is only used to retrieve mails, and that it is not necessary the same as the mail relay used by sendmail to send out mails.

After setting up the `.fetchmailrc` file, the following command can be used to download and deliver mail to the local system:

```
$ fetchmail
```

The messages can now be read with mutt.

25.3 Reading and writing mail with mutt

Mutt is one of the most popular mail programs: it is “lightweight”, easy to use and has lots of features. The man page mutt is very bare bones; the real documentation is in `/usr/pkg/share/doc/mutt/`, in particular `manual.txt`.

Mutt’s configuration is defined by the `~/.muttrc` file. The easiest way to create it is to copy mutt’s example muttrc file (usually `/usr/pkg/share/examples/mutt/sample.muttrc`) to the home directory and modify it. The following example shows how to achieve some results:

- Save a copy of sent mail.
- Define a directory and two files for incoming and outgoing mail saved by mutt (in this example the directory is `~/Mail` and the files are `incoming` and `outgoing`).
- Define some colors.
- Define an alias.

```
set copy=yes
set edit_headers
set folder="~/Mail"
unset force_name
set mbox="~/Mail/incoming"
set record="~/Mail/outgoing"
unset save_name

bind pager <up> previous-page
bind pager <down> next-page

color normal white black
color hdrdefault blue black
color indicator white blue
color markers red black
color quoted cyan black
color status white blue
color error red white
color underline yellow black

mono quoted standout
mono hdrdefault underline
mono indicator underline
mono status bold

alias pippo Pippo Verdi <pippo.verdi@pluto.net>
```

To start mutt:

```
$ mutt
```

Please note that mutt supports color, but this depends on the terminal settings. Under X you can use "xterm-color", for example:

```
$ env TERM=xterm-color mutt
```

25.4 Strategy for receiving mail

This section describes a simple method for receiving and reading mail. The connection to the provider is activated only for the time required to dowload the messages; mail is then read offline.

1. Activate the connection to the provider.
2. Run **fetchmail**.
3. Deactivate the connection.
4. Read mail with mutt.

25.5 Strategy for sending mail

When mail has been written and “sent” with mutt, the messages must be transferred to the provider with sendmail. Mail is sent from mutt with the **y** command, but this does not really send it; the messages are enqueued in the spool area; if sendmail is not active as a daemon it is necessary to start it manually or the messages will remain on the hard disk. The necessary steps are:

1. Write mail with mutt, send it and exit mutt. You can check if and what messages are in the sendmail mail queue using the mailq(1) program.
2. Activate the connection with the provider.
3. If your provider requires you to do "SMTP-after-POP", i.e. it first wants to make sure to know who you are before you are allowed to send mail (and no spam), you need to run **fetchmail** again first.
4. Write the command **/usr/sbin/sendmail -q -v** to transfer the queued messages to the provider.
5. Deactivate the connection.

25.6 Advanced mail tools

When you start using mail, you won’t probably have very sophisticated requirements and the already described standard configuration will satisfy all your needs. But for many users the number of daily messages will increase with time and a more rational organization of the mail storage will become necessary, for example subdividing mail in different mail boxes organized by topic. If, for example, you subscribe to a mailing list, you will likely receive many messages every day and you will want to keep them separate from the rest of your mail. You will soon find that you are spending too much time every day repeating the same manual operations to organize your mail boxes.

Why repeat the same operations manually when you can have a program perform them automatically for you? There are numerous tools that you can add to your mail system to increase its flexibility and automatically process your messages. Amongst the most known and used there are:

- procmail, an advanced mail delivery agent and general purpose mail filter for local mail, which automatically processes incoming mail using user defined rulesets. It integrates smoothly with sendmail.
- spamassassin or spamprobe, to help fight spam.

- metamail, a tool to process attachments.
- formail, a mail formatter.

In the remaining part of this section a sample configuration for procmail will be presented for a very common case: delivering automatically to a user defined mailbox all the messages coming from a mailing list. The configuration of sendmail will be modified in order to call procmail directly (procmail will be the *local mailer* used by sendmail). and a custom configuration file for procmail will be created.

First, procmail must be installed using the package system (`mail/procmail`) or **pkg_add**.

Next, the configuration of sendmail must be changed, in order to use procmail as local mailer. Uncomment the following three lines from the `mycf.mc` sendmail M4 prototype file and recreate the sendmail configuration file as described above:

```
define(‘PROCMAIL_MAILER_PATH’, /usr/pkg/bin/procmail)dnl
FEATURE(local_procmail)dnl
MAILER(procmail)dnl
```

The first line defines the path of the procmail program (you can see where procmail is installed with the command **which procmail**). The second line tells sendmail to use procmail for local mail delivery and the third adds procmail to the list of sendmail’s mailers. The third line adds procmail to the list of sendmail’s mailers (this line is optional).

The last step is the creation of the procmail configuration file, containing the recipes for mail delivery.

Let’s say that, for example, you subscribed to a mailing list on roses whose address is “roses@flowers.org” and that every message from the list contains the following line in the header:

```
Delivered-To: roses@flowers.org
```

Assuming you want to automatically sort all mails going over that list into the local mail folder "roses_list", the procmail configuration file (`.procmailrc`) looks like this:

```
PATH=/bin:/usr/bin:/usr/pkg/bin
MAILDIR=$HOME/Mail
LOGFILE=$MAILDIR/from

:0
* ^Delivered-To: roses@flowers.org
roses_list
```

The previous file contains only one rule, beginning with the line containing “:0”. The following line identifies all messages containing the string “Delivered-To: roses@flowers.org” and the last line says that the selected messages must go to the `roses_list` mailbox (which you should have created in $MAILDIR). The remaining messages will be delivered to the default mailbox. Note that $MAILDIR is the same directory that you have configured with mutt:

```
set folder="~/Mail"
```

Of course the mailing list is only an example; procmail is a very versatile tool which can be used to filter mail based on many criteria. As usual, refer to the man pages for more details: procmail(1), procmailrc(5), and procmailex(5) (this last one contains many examples of configuration files).

You can check that procmail is used as local mailer by sendmail if you run the latter in test mode:

```
$ /usr/sbin/sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>
```

The following command displays the list of mailers known to sendmail:

```
> =M
```

You should find a line like the following one:

```
mailer 3 (local): P=/usr/pkg/bin/procmail S=EnvFromL/HdrFromL ...
```

25.7 News with tin

The word *news* indicates the set of messages posted to the USENET newsgroups, a service available on the Internet. Each newsgroup contains articles related to a specific topic. Reading a newsgroup is different than reading a mailing list: when you subscribe to a mailing list you receive the articles by mail and you read them with a standard mail program like mutt, which you use also to send replies. News, instead, are read directly from a news server with a dedicated program called *newsreader* like, for example, tin. With tin you can subscribe to the newsgroups that you're interested in and follow the *threads*. A thread is a sequence of articles which all derive from an article that we could call "original". In short, a message is sent to the group, someone answers, other people answer to those who answered in the first place and so on, creating a tree like structure of messages and replies: without a newsreader it is impossible to understand the correct sequence of messages.

After the installation of tin (from the package collection as usual) the only thing left to do is to write the name of the NNTP server in the file `/usr/pkg/etc/nntp/server`, which you may need to create first. For example:

```
news.bignet.it
```

Once this has been done, the program can be started with the command **tin**. On the screen something similar to the following example will be displayed:

```
$ tin
Connecting to news.bignet.it...
news.bignet.it InterNetNews NNRP server INN 1.7.2 08-Dec-1997 ready (posting ok).
Reading groups from active file...
Checking for new groups...
Reading attributes file...
Reading newsgroups file...
Creating newsrc file...
Autosubscribing groups...
Reading newsrc file...
```

Be patient when you connect for the first time, because tin downloads an immense list of newsgroups to which you can subscribe and this takes several minutes. When the donwload is finished, the program's main screen is displayed; usually no groups are displayed; to see the list of groups press **y**. To subscribe to a group, move on the group's name and press **y**.

Once that you have subscribed to some newsgroups you can start tin more quickly with the command **tin -Q**. The search for new groups is disabled (`-q`), only active groups are searched (`-n`) and newsgroup description are not loaded (`-d`): it will not be possible to use the **y** (yank) command in tin. When tin is started with this option it can't tell if a newsgroup is moderated or not.

Note that if you are connecting from an internal network (like in our example), when you send ("post") a message the address at the beginning of the message will be wrong (because it is the internal address). To solve the problem, use the option "mail_address" in the tin configuration file (`~/.tin/tinrc`) or set the REPLYTO environment variable.

Chapter 26
Miscellaneous networking topics

26.1 Bridge

A bridge can be used to combine different physical networks into one logical network, i.e. connect them at layer 2 of the ISO-OSI model, not at layer 3, which is what a router would do. The NetBSD "bridge" driver provides bridge functionality on NetBSD systems.

26.1.1 Bridge example

In this example two physical networks are going to be combined in one logical network, 192.168.1.0, using a NetBSD bridge. The NetBSD machine which is going to act as bridge has two interfaces, ne0 and ne1, which are both connected to one physical network.

The first step is to make sure support for the "bridge" is compiled in the running kernel. Support is included in the GENERIC kernel.

When the system is ready the bridge can be created, this can be done using the **brconfig**(8) command. First of a bridge interface has to be created. With the following **ifconfig** command the "bridge0" interface will be created:

```
$ ifconfig bridge0 create
```

Please make sure that at this point both the ne0 and ne1 interfaces are up. The next step is to add the ne0 and ne1 interfaces to the bridge.

```
$ brconfig bridge0 add ne0 add ne1 up
```

This configuration can be automatically set up by creating an `/etc/ifconfig.interface` file, in this case `/etc/ifconfig.bridge0`, with the following contents:

```
create
!brconfig $int add ne0 add ne1 up
```

After setting up the bridge the bridge configuration can be displayed using the **brconfig -a** command. Remember that if you want to give the bridge machine an IP address you can only allocate an IP address to one of the interfaces which are part of the bridge.

26.2 Network File System (NFS)

Now that the network is working it is possible to share files and directories over the network using the Network File System (NFS). From the point of view of file sharing, the computer which gives access to

its files and directories is called the *server*, and the computer using these files and directories is the *client*. A computer can be client and server at the same time.

- A kernel must be compiled with the appropriate options for the client and the server (the options are easy to find in the kernel configuration file. See Section 22.1 for more information on NFS related kernel options.
- The server must enable the rpcbind, mountd lockd statd and `nfs_server` daemons in `/etc/rc.conf`:

  ```
  rpcbind=yes
  mountd=yes
  nfs_server=yes
  lockd=yes
  statd=yes
  ```

- The client must enable the rpcbind, lockd statd and `nfs_client` daemons in `/etc/rc.conf`:

  ```
  rpcbind=yes
  nfs_client=yes
  lockd=yes
  statd=yes
  ```

- The server must list the exported directories in `/etc/exports` and then run the command **kill -HUP 'cat /var/run/mountd.pid** (**hup mountd** may work too!).

A client host can access a remote directory through NFS if:

- The server host exports the directory to the client. The list of filesystems a NFS server exports can be checked with the **showmount -e** command, see showmount(8):

  ```
  # showmount -e 192.168.1.2
  Exports list on 192.168.1.2:
  /home                                host1 host2 host3
  ```

- The client host mounts the remote directory with the command **mount 192.168.1.2:/home /home**

The **mount** command has a rich set of options for remote directories which are not very intuitive (to say the least).

26.2.1 NFS setup example

The scenario described here is the following: five client machines (cli1, ..., cli5) share some directories on a server (buzz.toys.org). Some of the directories exported by the server are reserved for a specific client, the other directories are common for all client machines. All the clients boot from the server and must mount the directories.

The directories exported from the server are:

`/export/cli?/root`

: the five root directories for the five client machines. Each client has its own root directory.

`/export/cli?/swap`

: Five swap directories for the five swap machines.

`/export/common/usr`

`/usr` directory; common for all client hosts.

`/usr/src`

Common `/usr/src` directory for all client machines.

The following file systems exist on the server

```
/dev/ra0a on /
/dev/ra0f on /usr
/dev/ra1a on /usr/src
/dev/ra2a on /export
```

Each client needs the following file systems

```
buzz:/export/cli?/root   on /
buzz:/export/common/usr  on /usr
buzz:/usr/src            on /usr/src
```

The server configuration is the following:

```
# /etc/exports
/usr/src  -network 192.168.1.0 -mask 255.255.255.0
/export   -alldirs -maproot=root -network 192.168.1.0 -mask 255.255.255.0
```

On the client machines `/etc/fstab` contains:

```
buzz:/export/cliX/root  /        nfs rw
buzz:/export/common/usr /usr     nfs ro,nodev,nosuid
buzz:/usr/src           /usr/src nfs rw,nodev,nosuid
```

Each client machine has its number substituted to the "*X*" character in the first line of the previous example.

26.3 Setting up NFS automounting for `/net` with amd(8)

26.3.1 Introduction

The problem with NFS (and other) mounts is, that you usually have to be root to make them, which can be rather inconvenient for users. Using amd(8) you can set up a certain directory (Commonly `/net`), under which one can make any NFS-mount as a normal user, as long as the filesystem about to be accessed is actually exported by the NFS server.

To check if a certain server exports a filesystem, and which ones, use the **showmount**-command with the `-e` (export) switch:

```
$ showmount -e wuarchive.wustl.edu
Exports list on wuarchive.wustl.edu:
/export/home                       onc.wustl.edu
/export/local                      onc.wustl.edu
```

```
/export/adm/log                    onc.wustl.edu
/usr                               onc.wustl.edu
/                                  onc.wustl.edu
/archive                           Everyone
```

If you then want to mount a directory to access anything below it (for example `/archive/systems/unix/NetBSD`), just change into that directory:

```
$ cd /net/wuarchive.wustl.edu/archive/systems/unix/NetBSD
```

The filesystem will be mounted (by **amd**), and you can a access any files just as if the directory was mounted by the superuser of your system.

26.3.2 Actual setup

You can set up such a `/net` directory with the following steps (including basic **amd** configuration):

1. in `/etc/rc.conf`, set the following variable:

   ```
   amd=yes
   ```

2. **mkdir /amd**
3. **mkdir /net**
4. Taking `/usr/share/examples/amd/amd.conf`, put the following into `/etc/amd.conf`:

   ```
   [ /net ]
   map_name =              /etc/amd/net
   map_type =              file
   ```

5. Taking `/usr/share/examples/amd/net` as example, put the following into `/etc/amd/net`:

   ```
   /defaults       type:=host;rhost:=${key};fs:=${autodir}/${rhost}/root
   *             host==${key};type:=link;fs:=/                          \
                 host!=${key};opts:=ro,soft,intr,nodev,nosuid,noconn
   ```

6. Reboot, or (re)start **amd** by hand:

   ```
   # sh /etc/rc.d/amd restart
   ```

26.4 IPv6 Connectivity & Transition via 6to4

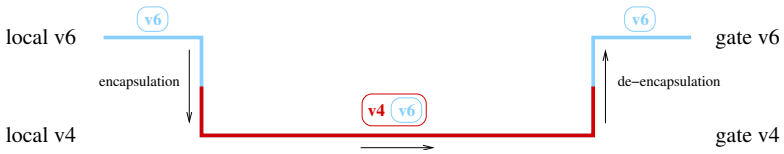
This section will concentrate on how to get network connectivity for IPv6 and - as that is rarely available directly - talk at length about the alternatives to native IPv6 connectivity as a transitional method until native IPv6 peers are available.

Finding an ISP that offers IPv6 natively needs quite some luck. What you need next is a router that will be able to handle the traffic. To date, not all router manufacturers offer IPv6 or hardware accelerated IPv6 features, and gateway NAT boxes only rarely support IPv6 and also block IPv6 tunnels. An alternative approach involves configuring a standard PC running NetBSD to act as a router. The base NetBSD system contains a complete IPv6 routing solution, and for special routing needs software like Zebra can provide additional routing protocols. This solution is rather common for sites that want IPv6

connectivity today. The drawbacks are that you need an ISP that supports IPv6 and that you may need a dedicated uplink only for IPv6.

IPv6 to-the-door may be rare, but you can still get IPv6 connectivity by using tunnels. Instead of talking IPv6 on the wire, the IPv6 packets are encapsulated in IPv4 packets, as shown in Figure 26-1. Using the existing IPv4 infrastructure, the encapsulated packets are sent to a IPv6-capable uplink that will then remove the encapsulation, and forward the IPv6 packets.

Figure 26-1. A frequently used method for transition is tunneling IPv6 in IPv4 packets



When using tunnels, there are two possibilities. One is to use a so-called “configured” tunnel, the other is called an “automatic” tunnel. A “configured” tunnel is one that required preparation from both ends of the tunnel, usually connected with some kind of registration to exchange setup information. An example for such a configured tunnel is the IPv6-over-IPv4 encapsulation described in RFC1933, and that’s implemented e.g. by the gif(4) device found in NetBSD.

An “automatic” tunnel consists of a public server that has some kind of IPv6 connectivity, e.g. via 6Bone. That server has made its connectivity data public, and also runs a tunneling protocol that does not require an explicit registration of the sites using it as uplink. A well-used example of such a protocol is the 6to4 mechanism described in RFC3056, and that is implemented in the stf(4) device found in NetBSD’s. Another mechanism that does not require registration of IPv6-information is the 6over4 mechanism, which implements transporting of IPv6 over a multicast-enabled IPv4 network, instead of e.g. ethernet or FDDI. 6over4 is documented in RFC2529. It’s main drawback is that you do need existing multicast infrastructure. If you don’t have that, setting it up is about as much effort as setting up a configured IPv6 tunnel directly, so it’s usually not worth bothering in that case.

26.4.1 Getting 6to4 IPv6 up & running

6to4 is an easy way to get IPv6 connectivity for hosts that only have an IPv4 uplink, especially if you have the background given in Section 21.7. It can be used with static as well as dynamically assigned IPv4 addresses, e.g. as found in modem dialup scenarios today. When using dynamic IPv4 addresses, a change of IP addresses will be a problem for incoming traffic, i.e. you can’t run persistent servers.

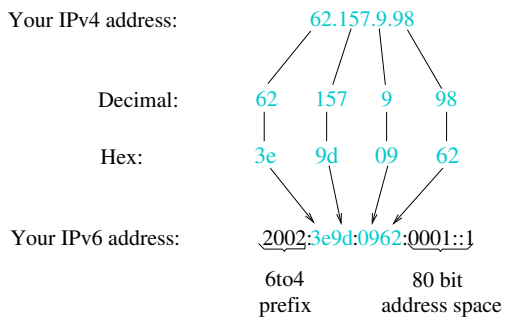
Example configurations given in this section is for NetBSD 1.5.2.

26.4.2 Obtaining IPv6 Address Space for 6to4

The 6to4 IPv6 setup on your side doesn’t consist of a single IPv6 address; Instead, you get a whole /48 network! The IPv6 addresses are derived from your (single) IPv4 address. The address prefix “2002:” is reserved for 6to4 based addresses (i.e. IPv6 addresses derived from IPv4 addresses). The next 32 bits are your IPv4 address. This results in a /48 network that you can use for your very own purpose. It leaves 16 bits space for 2^{16} IPv6 subnets, which can take up to 2^{64} nodes each. Figure 26-2 illustrates the building of your IPv6 address (range) from your IPv4 address.

Thanks to the 6to4 prefix and your worldwide unique IPv4 address, this address block is unique, and it's mapped to your machine carrying the IPv4 address in question.

Figure 26-2. 6to4 derives an IPv6 from an IPv4 address

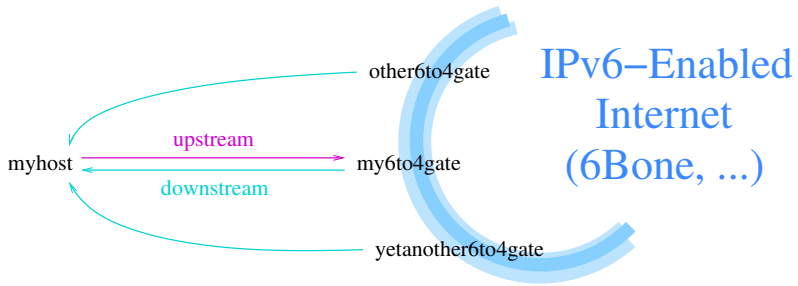


26.4.3 How to get connected

In contrast to the configured "IPv6-over-IPv4 tunnel" setup, you do not have to register at a 6bone-gateway, which would only then forward your IPv6 traffic encapsulated in IPv4. Instead, as your IPv6 address is derived from your IPv4 address, inbound traffic can be sent through the nearest 6to4 relay router. De-encapsulation of the packet is done via a 6to4-capable network interface, which then forwards the resulting IPv6 packet according to your routing setup (in case you have more than one machine connected on your 6to4 assigned network).

To transmit IPv6 packets, the 6to4 router will encapsulate them inside IPv4 packets; a system performing these functions is called a 6to4 border router. These packets have a default route to the *6to4 relay anycast prefix*. This anycast prefix will route the tunnel to a *6to4 relay router*. Figure 26-3 illustrates this.

Figure 26-3. Request and reply can be routed via different gateways in 6to4



26.4.4 Security Considerations

In contrast to the "configured tunnel" setup, you usually can't setup packet filters to block 6to4-packets from unauthorized sources, as this is exactly how (and why) 6to4 works at all. As such, malicious users can send packets with invalid/hazardous IPv6 payload. If you don't already filter on your border

gateways anyways, packets with the following characteristics should not be allowed as valid 6to4 packets, and some firewalling seems to be justified for them:

- unspecified IPv4 source/destination address: 0.0.0.0/8
- loopback address in outer (v4) source/destination: 127.0.0.0/8
- IPv4 multicast in source/destination: 224.0.0.0/4
- limited broadcasts: 255.0.0.0/8
- subnet broadcast address as source/destination: depends on your IPv4 setup

The NetBSD stf(4) manual page documents some common configuration mistakes intercepted by default by the KAME stack as well as some further advice on filtering, but keep in mind that because of the requirement of these filters, 6to4 is not perfectly secure. Still, if forged 6to4 packets become a problem, you can use IPsec authentication to ensure the IPv6 packets are not modified.

26.4.5 Data Needed for 6to4 Setup

In order to setup and configure IPv6 over 6to4, a few bits of configuration data must be known in advance. These are:

- Your local IPv4 address. It can be determined using either the '**ifconfig -a**' or '**netstat -i**' commands on most Unix systems. If you use a NATing gateway or something, be sure to use the official, outside-visible address, not your private (10/8 or 192.168/16) one.

  We will use 62.224.57.114 as the local IPv4 address in our example.
- Your local IPv6 address, as derived from the IPv4 address. See Figure 26-2 on how to do that.

  For our example, this is 2002:3ee0:3972:0001::1 (62.224.57.114 == 0x3ee03972, 0001::1 arbitrarily chosen).
- The *6to4 IPv6 relay anycast address*. which is 2002:c058:6301::, or the IPv6 address of a specific 6to4 relay router you want to use. The IPv6 address will do, as it also contains the IPv4 address in the usual 6to4 translation.

26.4.6 Kernel Preparation

To process 6to4 packets, the operating system kernel needs to know about them. For that a driver has to be compiled in that knows about 6to4, and how to handle it.

For a NetBSD kernel, put the following into your kernel config file to prepare it for using IPv6 and 6to4, e.g. on NetBSD use:

```
options INET6                   # IPv6
pseudo-device stf               # 6to4 IPv6 over IPv4 encapsulation
```

Note that the stf(4) device is not enabled by default. Rebuild your kernel, then reboot your system to use the new kernel. Please consult Chapter 29 for further information on configuring, building and installing a new kernel!

26.4.7 6to4 Setup

This section describes the commands to setup 6to4. In short, the steps performed here are:

1. Configure interface
2. Set default route
3. Setup Router Advertisement, if wanted

The first step in setting up 6to4 is assigning an IPv6 address to the 6to4 interface. This is achieved with the ifconfig(8) command. Assuming the example configuration above, the command for NetBSD is:

```
# ifconfig stf0 inet6 2002:3ee0:3972:1::1 prefixlen 16 alias
```

After configuring the 6to4 device with these commands, routing needs to be setup, to forward all tunneled IPv6 traffic to the 6to4 relay router. The best way to do this is by setting a default route, the command to do so is, for NetBSD:

```
# route add -inet6 default 2002:c058:6301::
```

Note that NetBSD's stf(4) device determines the IPv4 address of the 6to4 uplink from the routing table. Using this feature, it is easy to setup your own 6to4 (uplink) gateway if you have an IPv6 uplink, e.g. via 6Bone.

After these commands, you are connected to the IPv6-enabled world - Congratulations! Assuming name resolution is still done via IPv4, you can now ping an IPv6-site like www.kame.net or www6.NetBSD.org:

```
# /sbin/ping6 www.kame.net
```

As a final step in setting up IPv6 via 6to4, you will want to setup Router Advertisement if you have several hosts on your network. While it is possible to setup 6to4 on each node, doing so will result in very expensive routing from one node to the other - packets will be sent to the remote 6to4 gateway, which will then route the packets back to the neighbor node. Instead, setting up 6to4 on one machine and talking native IPv6 on-wire is the preferred method of handling things.

The first step to do so is to assign an IPv6-address to your ethernet. In the following example we will assume subnet "2" of the IPv6-net is used for the local ethernet and the MAC address of the ethernet interface is 12:34:56:78:9a:bc, i.e. your local gateway's ethernet interface's IP address will be 2002:3ee0:3972:2:1234:56ff:fe78:9abc. Assign this address to your ethernet interface, e.g.

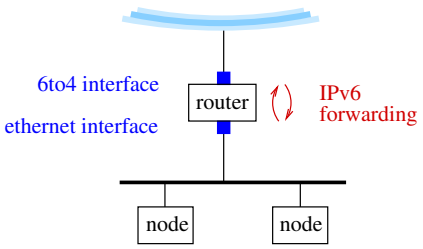
```
# ifconfig ne0 inet6 alias 2002:3ee0:3972:2:1234:56ff:fe78:9abc
```

Here, "ne0" is an example for your ethernet card interface. This will most likely be different for your setup, depending on what kind of card is used.

Next thing that needs to be ensured for setting up the router is that it will actually forward packets from the local 6to4 device to the ethernet device and back. To enable IPv6 packet forwarding, set "ip6mode=router" in NetBSD's `/etc/rc.conf`, which will result in the "net.inet6.ip6.forwarding" sysctl being set to "1":

```
# sysctl -w net.inet6.ip6.forwarding=1
```

Figure 26-4. Enabling packet forwarding is needed for a 6to4 router



To setup router advertisement on BSD, the file `/etc/rtadvd.conf` needs to be checked. It allows configuration of many things, but usually the default config of not containing any data is ok. With that default, IPv6 addresses found on all of the router's network interfaces will be advertised.

After checking the router advertisement configuration is correct and IPv6 forwarding is turned on, the daemon handling it can be started. Under NetBSD, it is called '**rtadvd**'. Start it up either manually (for testing it the first time) or via the system's startup scripts, and see all your local nodes automagically configure the advertised subnet address in addition to their already-existing link local address.

```
# rtadvd
```

26.4.8 Quickstart using pkgsrc/net/hf6to4

So far, we have described how 6to4 works and how to set it up manually. For an automated way to make everything happen e.g. when going online, the 'hf6to4' package is convenient. It will determine your IPv6 address from the IPv4 address you got assigned by your provider, then set things up that you are connected.

Steps to setup the pkgsrc/net/hf6to4 package are:

1. Install the package either by compiling it from pkgsrc, or by **pkg_add**'ing the 6to4-1.2 package.

   ```
   # cd /usr/pkgsrc/net/hf6to4
   # make install
   ```

2. Make sure you have the stf(4) pseudo-device in your kernel, see above.

3. Configure the 'hf6to4' package. First, copy `/usr/pkg/share/examples/hf6to4/hf6to4.conf` to `/usr/pkg/etc/hf6to4.conf`, then adjust the variables. Note that the file is in /bin/sh syntax.

   ```
   # cd /usr/pkg/etc
   # cp ../share/examples/hf6to4/hf6to4.conf hf6to4.conf
   # vi hf6to4.conf
   ```

   Please see the hf6to4(8) manpage for an explanation of all the variables you can set in `hf6to4.conf`. If you have dialup IP via PPP, and don't want to run Router Advertizing for other IPv6 machines on your home or office network, you don't need to configure anything. If you want to setup Router Advertising, you need to set the `in_if` to the internal (ethernet) interface, e.g.

   ```
   $in_if="rtk0";            # Inside (ethernet) interface
   ```

4. Now dial up, then start the 6to4 command manually:

   ```
   # /usr/pkg/sbin/hf6to4 start
   ```

5. After that, you should be connected, use ping6(8): to see if everything works:

```
# ping6 www.NetBSD.org
PING6(56=40+8+8 bytes) 2002:d954:110b:1::1 --> 2001:4f8:4:7:2e0:81ff:fe52:9a6b
16 bytes from 2001:4f8:4:7:2e0:81ff:fe52:9a6b, icmp_seq=0 hlim=60 time=250.234 ms
16 bytes from 2001:4f8:4:7:2e0:81ff:fe52:9a6b, icmp_seq=1 hlim=60 time=255.652 ms
16 bytes from 2001:4f8:4:7:2e0:81ff:fe52:9a6b, icmp_seq=2 hlim=60 time=251.237 ms
^C
--- www.NetBSD.org ping6 statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 250.234/252.374/255.652/2.354 ms

# traceroute6 www.NetBSD.org
traceroute6 to www.NetBSD.org (2001:4f8:4:7:2e0:81ff:fe52:9a6b)
from 2002:d954:110b:1::1, 64 hops max, 12 byte packets
 1  2002:c25f:6cbf:1::1  66.31 ms  66.382 ms  69.062 ms
 2  nr-erl1.6win.dfn.de  76.134 ms *  76.87 ms
 3  nr-fra1.6win.dfn.de  76.371 ms  80.709 ms  79.482 ms
 4  dfn.de6.de.6net.org  92.763 ms  90.863 ms  94.322 ms
 5  de.nl6.nl.6net.org  116.115 ms  93.463 ms  96.331 ms
 6  nl.uk6.uk.6net.org  103.347 ms  99.334 ms  100.803 ms
 7  uk1.uk61.uk.6net.org  99.481 ms  100.421 ms  100.119 ms
 8  2001:798:28:300::2  89.711 ms  90.435 ms  90.035 ms
 9  ge-1-0-0-2.r20.londen03.uk.bb.verio.net  179.671 ms  185.141 ms  185.86 ms
10  p16-0-0-0.r81.nycmny01.us.bb.verio.net  177.067 ms  179.086 ms  178.05 ms
11  p16-1-1-3.r20.nycmny01.us.bb.verio.net  178.04 ms  179.727 ms  184.165 ms
12  p16-0-1-1.r20.mlpsca01.us.bb.verio.net  249.856 ms  247.476 ms  249.012 ms
13  p64-0-0-0.r21.snjsca04.us.bb.verio.net  239.691 ms  241.404 ms  240.998 ms
14  p64-0-0-0.r21.plalca01.us.bb.verio.net  247.541 ms  246.661 ms  246.359 ms
15  xe-0-2-0.r20.plalca01.us.bb.verio.net  240.987 ms 239.056 ms  241.251 ms
16  ge-6-1.a01.snfcca05.us.ra.verio.net  240.868 ms  241.29 ms  242.337 ms
17  fa-5-2.a01.snfcca05.us.ce.verio.net  249.477 ms  250.4 ms  256.035 ms
18  2001:4f8:4:7:2e0:81ff:fe52:9a6b  268.164 ms  252.97 ms  252.366 ms
```

   Please note that **traceroute6** shows the v6 hops only, any underlying tunnels are invisible and thus not displayed.

6. If this works, you can put the following lines into your `/etc/ppp/ip-up` script to run the command each time you go online:

```
logger -p user.info -t ip-up Configuring 6to4 IPv6
/usr/pkg/sbin/hf6to4 stop
/usr/pkg/sbin/hf6to4 start
```

7. If you want to route IPv6 for your LAN, you can instruct **6to4.pl** to setup Router Advertising for you too:

```
# /usr/pkg/sbin/hf6to4 rtadvd-start
```

   You can put that command into `/etc/ppp/ip-up` as well to make it permanent.

8. If you have changed `/etc/ppp/ip-up` to setup 6to4 automatically, you will most likely want to change `/etc/ppp/ip-down` too, to shut it down when you go offline. Here's what to put into `/etc/ppp/ip-down`:

```
logger -p user.info -t ip-down Shutting down 6to4 IPv6
/usr/pkg/sbin/hf6to4 rtadvd-stop
```

```
/usr/pkg/sbin/hf6to4 stop
```

26.4.9 Known 6to4 Relay Routers

It is normally not necessary to pick a specific 6to4 relay router, but if necessary, you may find a list of known working routers at http://www.kfu.com/~nsayer/6to4/. In tests, only 6to4.kfu.com and 6to4.ipv6.microsoft.com were found working. Cisco has one that requires registration, see http://www.cisco.com/ipv6/.

There's also an experimental 6to4 server located in Germany, 6to4.ipv6.fh-regensburg.de. This server runs under NetBSD 1.6 and was setup using the configuration steps described above. The whole configuration of the machine can be seen at http://www.feyrer.de/IPv6/netstart.local.

26.4.10 Tunneling 6to4 through an IPFilter firewall

The 6to4 protocol encapsulates IPv6 packets in IPv4, and gives them their own IP type, which most firewalls block as unknown, as their payload type is directly "TCP", "UDP" or "ICMP". Usually, you want to setup your 6to4 gateway on the same machine that is directly connected to the (IPv4) internet, and which usually runs the firewall. For the case that you want to run your 6to4 gateway behind a firewall, you need to drill a hole into the firewall, to let 6to4 packets through. Here is how to do this!

The example assumes that you use the "ppp0" interface on your firewall to connect to the Internet.

Put the following lines into `/etc/ipf.conf` to allow your IPfilter firewall let all 6to4 packets pass (lines broken with \ due to space restrictions; please put them lines continued that way all in one line):

```
# Handle traffic by different rulesets
block in  quick on ppp0 all head 1
block out quick on ppp0 all head 2

### Incoming packets:
# allow some IPv4:
pass  in  log quick on ppp0 proto tcp from any to any \
 port = www flags S keep state keep frags group 1
pass  in      quick on ppp0 proto tcp from any to any \
 port = ssh keep state     group 1
pass  in      quick on ppp0 proto tcp from any to any \
 port = mail keep state    group 1
pass  in  log quick on ppp0 proto tcp from any to any \
 port = ftp keep state    group 1
pass  in  log quick on ppp0 proto tcp from any to any \
 port = ftp-data keep state   group 1
pass  in  log quick on ppp0 proto icmp from any to any       group 1
# allow all IPv6:
pass in       quick on ppp0 proto ipv6       from any to any  group 1
pass in  log  quick on ppp0 proto ipv6-route from any to any  group 1
pass in  log  quick on ppp0 proto ipv6-frag  from any to any  group 1
pass in  log  quick on ppp0 proto ipv6-icmp  from any to any  group 1
pass in  log  quick on ppp0 proto ipv6-nonxt from any to any  group 1
pass in  log  quick on ppp0 proto ipv6-opts  from any to any  group 1
# block rest:
```

```
blockin  log  quick on ppp0 all                                group 1

### Outgoing packets:
# allow usual stuff:
pass  out     quick on ppp0 proto  tcp from any to any flags S \
 keep state keep frags group 2
pass  out     quick on ppp0 proto  udp from any to any         \
 keep state keep frags group 2
pass  out     quick on ppp0 proto icmp from any to any         \
 keep state            group 2
# allow all IPv6:
pass out      quick on ppp0 proto ipv6       from any to any  group 2
pass out log  quick on ppp0 proto ipv6-route from any to any  group 2
pass out log  quick on ppp0 proto ipv6-frag  from any to any  group 2
pass out log  quick on ppp0 proto ipv6-icmp  from any to any  group 2
pass out log  quick on ppp0 proto ipv6-nonxt from any to any  group 2
pass out log  quick on ppp0 proto ipv6-opts  from any to any  group 2
# block rest:
block out log quick on ppp0 all          group 2
```

Now any host on your network can send (the "out" rules") and receive (the "in" rules) v4-encapsulated IPv6 packets, allowing setup of any of them as a 6to4 gateway. Of course you only want to do this on one host and use native IPv6 between your hosts, and you may also want to enforce this with more restrictive rulesets, please see ipf.conf(5) for more information on IPFilter rules.

After your firewall lets pass encapsulated IPv6 packets, you may want to set up your 6to4 gateway to monitor the IPv6 traffic, or even restrict it. To do so, you need to setup IPfilter on your 6to4 gateway as well. For basic monitoring, enable "ipfilter=yes" in `/etc/rc.conf` and put the following into `/etc/ipf6.conf`:

```
pass in  log quick on stf0 from any to any
pass out log quick on stf0 from any to any
```

This logs all (IPv6) traffic going in and out of your "stf0" tunneling interface. You can add filter rules as well if needed.

If you are more interested in traffic stats than a general overview of your network traffic, using MRTG in conjunction with the "net-snmp" package is recommended instead of analyzing IPfilter log files.

26.4.11 Conclusion & Further Reading

Compared to where IPv4 is today, IPv6 is still in its early steps. It is working, there are all sort of services and clients available, only the userbase is missing. It is hoped the information provided here helps people better understand what IPv6 is, and to start playing with it.

A few links should be mentioned here for interested parties:

- An example script to setup 6to4 on BSD based machines is available at http://www.NetBSD.org/packages/net/hf6to4/. The script determines your IPv6 address and sets up 6to4 and (if wanted) router advertising. It was designed to work in dialup setups with changing IPv4 addresses.

- Given that there isn't a standard for IPv6 in Linux land today, there are different setup instructions for most distributions. The setup of IPv6 on Debian GNU/Linux can be found at http://people.debian.org/~csmall/ipv6/setup.html.
- The BSD Unix implementations have their own IPv6 documentation each, interesting URLs are http://www.NetBSD.org/Documentation/network/ipv6/ for NetBSD, http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/network-ipv6.html for FreeBSD.
- Projects working on implementing IPv6 protocol stacks for free Unix like operating systems are KAME for BSD and USAGI for Linux. Their web sites can be found at http://www.kame.net/ and http://www.linux-ipv6.org/. A list of host and router implementations can be found at http://playground.sun.com/pub/ipng/html/ipng-implementations.html.
- Besides the official RFC archive at ftp://ftp.isi.edu/in-notes, information on IPv6 can be found at several web sites. First and foremost, the 6Bone's web page at http://www.6bone.net/ must be mentioned. 6Bone was started as the testbed for IPv6, and is now an important part of the IPv6-connected world. Other web pages that contain IPv6-related contents include http://www.ipv6.org/, http://playground.sun.com/pub/ipng/html/ and http://www.ipv6forum.com/. Most of these sites carry further links - be sure to have a look!

V. Building the system

Chapter 27
Obtaining the sources

To read the NetBSD sources from your local disk or to build the system or parts of it, you need to download the NetBSD sources. This chapter explains how to get the NetBSD source using a number of different ways, although the prefered one is to get the tarballs and then update via cvs(1).

27.1 Preparing directories

Kernel and userland sources are usually placed in `/usr/src`. This directory is not present by default in the NetBSD installation and you will need to create it first. As it is in a system directory, you will need root access to create the directory and make sure your normal user account can write to it. For demonstration purposes, it is assumed that your non-root login is “carlo”:

```
% su
Password: *****
# mkdir /usr/src
# chown carlo /usr/src
```

If you want to use pkgsrc (which is most likely), you can prepare `/usr/pkgsrc` at the same time:

```
# mkdir /usr/pkgsrc
# chown carlo /usr/pkgsrc
```

Also, if you want X11R6 sources, you can prepare `/usr/xsrc`:

```
# mkdir /usr/xsrc
# chown carlo /usr/xsrc
```

Please note that for the subsequent steps, root access is neither needed nor recommended, so this preparation step should be done first. All CVS operations can (and should) be done as normal user and you don’t need root privileges any more:

```
# exit
%
```

27.2 Terminology

Before starting to fetch or download the required files, you may want to know the definitions of “Formal releases”, “Maintenance branches” and other related terms. That information is available under the NetBSD release glossary and graphs (http://www.NetBSD.org/Releases/release-map.html).

27.3 Downloading tarballs

It's sometimes faster to download a tarball and then continue updating with cvs(1). You can download tarballs (see tar(1)) from ftp.NetBSD.org (or any other mirror) for a number of releases or branches.

The only drawback is that the tarballs are updated less often. Normally, every three days.

Also, please note that these tarballs include the `CVS` directories, so you can download them and then update your source tree using cvs(1), as explained in the CVS section.

27.3.1 Downloading a NetBSD release

The tarball files for a specific release are available under `/pub/NetBSD/NetBSD-`*`RELEASE-NUMBER`*`/source/sets/` on ftp.NetBSD.org (or a mirror), where *`RELEASE-NUMBER`* is the release you want to fetch (for example, 2.0.2).

To fetch a NetBSD release using tarballs (make sure you have enough hard disk space under `/tmp`), simply do:

```
% cd /tmp
% ftp -i ftp://ftp.NetBSD.org/pub/NetBSD/NetBSD-2.0.2/source/sets/
Trying 2001:4f8:4:7:2e0:81ff:fe21:6563...
Connected to ftp.NetBSD.org.
220 ftp.NetBSD.org FTP server (NetBSD-ftpd 20040809) ready.
331 Guest login ok, type your name as password.
[...]
250 CWD command successful.
250 CWD command successful.
250 CWD command successful.
ftp> mget *.tgz
local: gnusrc.tgz remote: gnusrc.tgz
229 Entering Extended Passive Mode (|||58302|)
150 Opening BINARY mode data connection for 'gnusrc.tgz' (79233899 bytes).
[...]
ftp> quit
221-
    Data traffic for this session was 232797304 bytes in 5 files.
    Total traffic for this session was 232803039 bytes in 6 transfers.
221 Thank you for using the FTP service on ftp.NetBSD.org.
```

You should now have 5 files:

```
% ls *.tgz
gnusrc.tgz      sharesrc.tgz    src.tgz         syssrc.tgz      xsrc.tgz
```

You now must extract them all to `/`:

```
% foreach file (*.tgz)
?   tar -xzf $file -C /
? end
```

27.3.2 Downloading snapshots from a NetBSD stable branch

```
% ftp -i ftp://ftp.NetBSD.org/pub/NetBSD/NetBSD-release-2-0/tar_files/src/
Trying 2001:4f8:4:7:2e0:81ff:fe21:6563...
Connected to ftp.NetBSD.org.
220 ftp.NetBSD.org FTP server (NetBSD-ftpd 20040809) ready.
331 Guest login ok, type your name as password.
[...]
250 CWD command successful.
250 CWD command successful.
250 CWD command successful.
250 CWD command successful.
ftp> mget *.tar.gz
local: bin.tar.gz remote: bin.tar.gz
229 Entering Extended Passive Mode (|||56011|)
150 Opening BINARY mode data connection for 'bin.tar.gz' (914202 bytes).
[...]
ftp> quit
221-
    Data traffic for this session was 149221420 bytes in 22 files.
    Total traffic for this session was 149231539 bytes in 23 transfers.
221 Thank you for using the FTP service on ftp.NetBSD.org.
```

You should now have 22 files:

```
% ls *.tar.gz
bin.tar.gz          doc.tar.gz          libexec.tar.gz      tools.tar.gz
config.tar.gz       etc.tar.gz          regress.tar.gz      top-level.tar.gz
contrib.tar.gz      games.tar.gz        rescue.tar.gz       usr.bin.tar.gz
crypto.tar.gz       gnu.tar.gz          sbin.tar.gz         usr.sbin.tar.gz
dist.tar.gz         include.tar.gz      share.tar.gz
distrib.tar.gz      lib.tar.gz          sys.tar.gz
```

You now must extract them all to `/usr`:

```
% foreach file (*.tar.gz)
?   tar -xzf $file -C /usr
? end
```

27.3.3 Downloading the NetBSD-current development branch

To download the NetBSD-current tarballs, located under `/pub/NetBSD/NetBSD-current/tar_files/src`, just follow the same steps as in the previous section, but now on a different directory.

You may also want to fetch the X11R6 source, available under: `/pub/NetBSD/NetBSD-current/tar_files/xsrc`.

27.3.4 Downloading a pkgsrc-200xQy stable branch

Pkgsrc ("package source") is the NetBSD packages collection which can be used to easily install and deinstall software on your NetBSD system. See Part VI in *The NetBSD & pkgsrc Guide* for more information.

You should follow the stable branch, unless you're developing new package or adding a new feature.

To download a pkgsrc stable tarball, run:

```
% cd /tmp
% ftp ftp://ftp.NetBSD.org/pub/pkgsrc/200xQy/pkgsrc-200xQy.tar.gz
```

Where <*pkgsrc-200xQy*> is the stable branch to be downloaded, for example, "pkgsrc-2004Q1" or "pkgsrc-2005Q2".

Then, extract it with:

```
% tar -xzf pkgsrc-200xQy.tar.gz -C /usr
```

This will create the directory `pkgsrc` in your `/usr` (if not already created) and all the package source will be stored under `/usr/pkgsrc`.

27.3.5 Downloading the pkgsrc-current development branch

To download pkgsrc-current, run:

```
% ftp ftp://ftp.NetBSD.org/pub/pkgsrc/current/pkgsrc.tar.gz
```

Then extract it with:

```
% tar -xzf pkgsrc.tar.gz -C /usr
```

27.4 Fetching by CVS

CVS (Concurrent Versions System) can be used to fetch the NetBSD source tree or to keep the NetBSD source tree up to date with respect to changes made to the NetBSD sources. There are three trees maintained for which you can use CVS to obtain them or keep them up to date:

The list of currently maintained branches is available under `src/doc/BRANCHES` (see the "Status" entry on "Release branches" section).

CVS also supports several connection methods. If you want to use the "external" method (only available with ssh(1)) you should do:

```
% setenv CVS_RSH ssh
% setenv CVSROOT :ext:anoncvs@anoncvs.NetBSD.org:/cvsroot
```

Otherwise, if you want to use the "pserver" method:

```
% unsetenv CVS_RSH
% setenv CVSROOT :pserver:anoncvs@anoncvs.NetBSD.org:/cvsroot
```

We'll also use the -P option in the examples below since it's used to prune empty directories.

27.4.1 Getting CVS

The CVS program (http://www.CVShome.org/) is part of NetBSD. If you have an old release which didn't come with CVS, you can install it either by using **pkg_add**, or building from pkgsrc.

To install via **pkg_add**, just do:

```
% pkg_add ftp://ftp.NetBSD.org/pub/NetBSD/packages/OS Ver/arch/All/cvs
```

where *OS Ver* and *arch* can be obtained by running.

```
% sysctl kern.osrelease hw.machine_arch
```

Otherwise, if you want to build cvs(1) from pkgsrc, (assuming you already fetched a stable pkgsrc branch or pkgsrc-current) you would do:

```
% cd /usr/pkgsrc/devel/cvs
% make install
```

27.4.2 Fetching a NetBSD release

A release is a set of particular versions of source files, and once released does not change over time.

To get the NetBSD (kernel and userland) sources from a specific release, run the following command after the preparations done above:

```
% cd /usr
% cvs checkout -r <BRANCH> -P src
```

Where <*BRANCH*> is the release branch to be checked out, for example, "netbsd-2-0-RELEASE", "netbsd-2-1-RELEASE" or "netbsd-3-0-RELEASE". If you want to fetch a different patchlevel, you would use "netbsd-2-0-1-RELEASE" or "netbsd-2-0-2-RELEASE".

For example, in order to fetch "netbsd-2-0-RELEASE" you would use:

```
% cvs checkout -r netbsd-2-0-RELEASE -P src
```

To fetch the X11R6 source, just "checkout" the "xsrc" module. For example:

```
% cvs checkout -r netbsd-2-0-RELEASE -P xsrc
```

27.4.3 Fetching a NetBSD stable branch

NetBSD stable branches are also called "Maintenance branches". Please consult the Section 27.2.

If you want to follow a stable branch, just pass the branch name to the cvs(1) -r option.

For example, if you want to fetch the most recent version of "netbsd-2", you just need to use that tag:

```
% cd /usr
```

```
% cvs checkout -r netbsd-2 -P src
```

And for the "xsrc" module:

```
% cvs checkout -r netbsd-2 -P xsrc
```

If you have checked out sources from a stable branch in `/usr/src` and want to update them to get the latest security-fixes and bug-fixes, run:

```
% cd /usr/src
% cvs update -Pd
```

The same applies to the "xsrc" module, but in that case you'll have to change your working directory to `/usr/xsrc` first.

27.4.4 Fetching the NetBSD-current development branch

To obtain the NetBSD-current source just omit "`-r` *`BRANCH`*" and replace it by "`-A`":

```
% cd /usr
% cvs checkout -A -P src
```

The "xsrc" is also available:

```
% cd /usr
% cvs checkout -A -P xsrc
```

To update your NetBSD-current source tree, add the `-A` flag:

```
% cd /usr/src
% cvs update -A -Pd
```

27.4.5 Fetching a pkgsrc-200xQy stable branch

Pkgsrc ("package source") is the NetBSD packages collection which can be used to easily install and deinstall software on your NetBSD system. See Part VI in *The NetBSD & pkgsrc Guide* for more information.

Stable pkgsrc branches are created every quarter, hence the naming "<year>-Q<quarter>". You should follow the stable branch, unless you're developing new package or adding a new feature.

To know the pkgsrc branches for the current year, run:

```
% cd /usr/pkgsrc
% cvs log Makefile | grep pkgsrc-`date +%Y`Q.:
        pkgsrc-2005Q2: 1.73.0.2
        pkgsrc-2005Q1: 1.67.0.2
```

Given this example, you should choose "pkgsrc-2005Q2" since older branches are no longer maintained.

To fetch a specific pkgsrc stable branch from scratch into `/usr/pkgsrc`, run:

```
% cd /usr
% cvs checkout -r <pkgsrc-200xQy> -P pkgsrc
```

Where <*pkgsrc-200xQy*> is the stable branch to be checked out, for example, “pkgsrc-2004Q1” or “pkgsrc-2005Q2”.

This will create the directory `pkgsrc` in your `/usr` (if not already created by preparation steps above) and all the package source will be stored under `/usr/pkgsrc`.

To update pkgsrc just do:

```
% cd /usr/pkgsrc
% cvs update -Pd
```

27.4.6 Fetching the pkgsrc-current development branch

To obtain pkgsrc-current, run:

```
% cd /usr
% cvs checkout -A -P pkgsrc
```

To update pkgsrc-current just replace `-r..` with `-A`:

```
% cd /usr/pkgsrc
% cvs update -A -Pd
```

27.4.7 Saving some cvs(1) options

If you find yourself typing some options to cvs over and over again, you can as well put them into a file `.cvsrc` in your home directory. The following example is taken from my `$HOME/.cvsrc`, it is useful for just typing **cvs update** on a directory with a branch checked out to update it (adding `-A` would revert the branch to the -current branch, which is not what I usually want!), I prefer unified diffs, transfers should do some compression and “cvs update” should be mostly quiet:

Example 27-1. `.cvsrc`

```
#update -dPA
update  -dP
rdiff   -u
diff    -u
cvs     -q
```

27.5 Sources on CD (ISO)

If you prefer to download (and maybe burn) a CD-ROM image with the NetBSD source, just fetch `sourcecd.iso` from ftp.NetBSD.org or any other mirror.

The `sourcecd.iso` file is located under /pub/NetBSD/iso/*RELEASE*, where *RELEASE* is a NetBSD release available under ISO9660 format (for example, 1.6.2, 2.0 or 2.0.2)

The next step is to burn the ISO image or mount it with the help of vnconfig(8). Please see Chapter 13 as it explains in detail how to do it.

Assuming you have mounted the CD under `/mnt`, `/mnt/source/sets` should have everything you need to extract:

```
% ls /mnt/source/sets
BSDSUM          MD5             gnusrc.tgz      src.tgz         xsrc.tgz
CKSUM           SYSVSUM         sharesrc.tgz    syssrc.tgz
```

All tarballs should be extracted to the root file system (`/`). The following command will do it:

```
% foreach file (*.tgz)
?   tar -xzf $file -C /
? end
```

After that, you should have `/usr/src` and `/usr/xsrc` populated with the NetBSD sources.

Chapter 28
Crosscompiling NetBSD with `build.sh`

When targeting a product for an embedded platform, it's not feasible to have all the development tools available on that same platform. Instead, some method of crosscompiling is usually used today. NetBSD 1.6 and forward comes with a framework to build both the operating system's kernel and the whole userland for either the same platform that the compiler runs on, or for a different platform, using crosscompiling. Crosscompiling requires assembler, linker, compiler etc. to be available and built for the target platform. The new build scheme will take care of creating these tools for a given platform, and make them available ready to use to do development work.

In this chapter, we will show how to use `build.sh` to first create a crosscompiling toolchain, including cross-compiler, cross-assembler, cross-linker and so on. While native kernel builds are covered in Chapter 29, these tools are then used to manually configure and crosscompile a kernel for a different platform, and then show how to use `build.sh` as a convenient alternative. After that works, the whole NetBSD userland will be compiled and packed up in the format of a NetBSD release. In the examples, we will use the Sun UltraSPARC ("sparc64") 64-bit platform as target platform, any other platform supported by NetBSD can be targetted as well specifying its name (see `/usr/src/sys/arch`).

Before starting, take note that it is assumed that the NetBSD sources from the "netbsd-2-0" branch are available in `/usr/src` as described in Chapter 27.

A more detailed description of the `build.sh` framework can be found in Luke Mewburn and Matthew Green's paper (http://www.mewburn.net/luke/papers/build.sh.pdf) and their presentation (http://www.mewburn.net/luke/talks/bsdcon-2003/index.html) from BSDCon 2003 as well as in `/usr/src/BUILDING`.

28.1 Building the crosscompiler

The first step to do cross-development is to get all the necessary tools available. In NetBSD terminology, this is called the "toolchain", and it includes BSD-compatible make(1), C/C++ compilers, linker, assembler, config(8), as well as a fair number of tools that are only required when crosscompiling a full NetBSD release, which we won't cover here.

The command to create the crosscompiler is quite simple, using NetBSD's new `src/build.sh` script. Please note that all the commands here can be run as normal (non-root) user:

```
% cd /usr/src
% ./build.sh -m sparc64 tools
```

If the tools have been built previously and they only need updated, then the update option "-u" can be used to only rebuild tools that have changed:

```
% ./build.sh -u -m sparc64 tools
```

When the tools are built, information about them and several environment variables is printed out:

```
...
===> build.sh started: Thu Dec  2 22:18:11 CET 2004
===> build.sh ended:   Thu Dec  2 22:28:22 CET 2004
===> Summary of results:
         build.sh command: ./build.sh -m sparc64 tools
         build.sh started: Thu Dec  2 22:18:11 CET 2004
         No nonexistent/bin/nbmake, needs building.
         Bootstrapping nbmake
         MACHINE:          sparc64
         MACHINE_ARCH:     sparc64
         TOOLDIR path:     /usr/src/tooldir.NetBSD-2.0-i386
         DESTDIR path:     /usr/src/destdir.sparc64
         RELEASEDIR path:  /usr/src/releasedir
         Created /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake
         makewrapper:      /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64
         Updated /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64
         Tools built to /usr/src/tooldir.NetBSD-2.0-i386
         build.sh started: Thu Dec  2 22:18:11 CET 2004
         build.sh ended:   Thu Dec  2 22:28:22 CET 2004
===> .
```

During the build, object directories are used consistently, i.e. special directories are kept that keep the platform-specific object files and compile results. In our example, they will be kept in directories named "obj.sparc64" as we build for UltraSPARC as target platform.

The toolchain itself is part of this, but as it's hosted and compiled for a i386 system, it will get placed in its own directory indicating where to cross-build from. Here's where our crosscompiler tools are located:

```
% pwd
/usr/src
% ls -d tooldir.*
tooldir.NetBSD-2.0-i386
```

So the general rule of thumb is for a given "host" and "target" system combination, the crosscompiler will be placed in the "src/tooldir.host" directory by default. A full list of all tools created for crosscompiling the whole NetBSD operating system includes:

```
% ls tooldir.NetBSD-2.0-i386/bin/
nbasn1_compile          nbmakefs                nbzic
nbcap_mkdb              nbmakeinfo              sparc64--netbsd-addr2li
nbcat                   nbmakewhatis            sparc64--netbsd-ar
nbcksum                 nbmenuc                 sparc64--netbsd-as
nbcompile_et            nbmkcsmapper            sparc64--netbsd-c++
nbconfig                nbmkdep                 sparc64--netbsd-c++filt
nbcrunchgen             nbmkesdb                sparc64--netbsd-cpp
nbctags                 nbmklocale              sparc64--netbsd-dbsym
nbdb                    nbmknod                 sparc64--netbsd-g++
nbeqn                   nbmktemp                sparc64--netbsd-g77
nbfgen                  nbmsgc                  sparc64--netbsd-gcc
```

```
nbfile                  nbmtree                 sparc64--netbsd-gcc-3.3
nbgencat                nbnroff                 sparc64--netbsd-gccbug
nbgroff                 nbpax                   sparc64--netbsd-gcov
nbhexdump               nbpic                   sparc64--netbsd-ld
nbhost-mkdep            nbpwd_mkdb              sparc64--netbsd-lint
nbindxbib               nbrefer                 sparc64--netbsd-mdsetim
nbinfo                  nbrpcgen                sparc64--netbsd-nm
nbinfokey               nbsoelim                sparc64--netbsd-objcopy
nbinstall               nbstat                  sparc64--netbsd-objdump
nbinstall-info          nbsunlabel              sparc64--netbsd-ranlib
nbinstallboot           nbtbl                   sparc64--netbsd-readelf
nblex                   nbtexi2dvi              sparc64--netbsd-size
nblorder                nbtexindex              sparc64--netbsd-strings
nbm4                    nbtsort                 sparc64--netbsd-strip
nbmake                  nbuudecode
nbmake-sparc64          nbyacc
```

As you can see, most of the tools that are available native on NetBSD are present with some program prefix to identify the target platform for tools that are specific to a certain target platform.

One important tool that should be pointed out here is "nbmake-sparc64". This is a shell wrapper for a BSD compatible make(1) command that's setup to use all the right commands from the crosscompiler toolchain. Using this wrapper instead of /usr/bin/make allows crosscompiling programs that were written using the NetBSD Makefile infrastructure (see src/share/mk). We will use this make(1) wrapper in a second to cross compile the kernel!

28.2 Configuring the kernel manually

Now that we have a working crosscompiler available, the "usual" steps for building a kernel are needed - create a kernel config file, run config(8), then build. As the config(8) program used to create header files and Makefile for a kernel build is platform specific, we need to use the "nbconfig" program that's part of our new toolchain. That aside, the procedure is just as like compiling a "native" NetBSD kernel. Commands involved here are:

```
% cd /usr/src/sys/arch/sparc64/conf
% cp GENERIC MYKERNEL
% vi MYKERNEL
% /usr/src/tooldir.NetBSD-2.0-i386/bin/nbconfig MYKERNEL
```

That's all. This command has created a directory `../compile/`*`MYKERNEL`* with a number of header files defining information about devices to compile into the kernel, a Makefile that is setup to build all the needed files for the kernel, and link them together.

28.3 Crosscompiling the kernel manually

We have all the files and tools available to crosscompile our UltraSPARC-based kernel from our Intel-based host system, so let's get to it! After changing in the directory created in the previous step, we need to use the crosscompiler toolchain's `nbmake-sparc64` shell wrapper, which just calls make(1) with all the necessary settings for crosscompiling for a sparc64 platform:

```
% cd ../compile/MYKERNEL/
% /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64 depend
% /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64
```

This will churn away a bit, then spit out a kernel:

```
...
   text    data     bss     dec     hex filename
5016899  163728  628752 5809379  58a4e3 netbsd
% ls -l netbsd
-rwxr-xr-x  1 feyrer  666  5874663 Dec  2 23:17 netbsd
% file netbsd
netbsd: ELF 64-bit MSB executable, SPARC V9, version 1 (SYSV), statically linked, not stripp
```

Now the kernel in the file `netbsd` can either be transferred to a UltraSPARC machine (via NFS, FTP, scp, etc.) and booted from a possible harddisk, or directly from our cross-development machine using NFS.

After configuring and crosscompiling the kernel, the next logical step is to crosscompile the whole system, and bring it into a distribution-ready format. Before doing so, an alternative approach to crosscompiling a kernel will be shown in the next section, using the `build.sh` script to do configuration and crosscompilation of the kernel in one step.

28.4 Crosscompiling the kernel with `build.sh`

A cross compiled kernel can be done manually as described in the previous sections, or by the easier method of using `build.sh`, which will be shown here.

Preparation of the kernel config file is the same as described above:

```
% cd /usr/src/sys/arch/sparc64/conf
% cp GENERIC MYKERNEL
% vi MYKERNEL
```

Then edit *MYKERNEL* and once finished, all that needs to be done is to use `build.sh` to build the kernel (it will also configure it, running the steps shown above):

```
% cd /usr/src
% ./build.sh -u -m sparc64 kernel=MYKERNEL
```

Notice that update ("-u") was specified, the tools are already built, there is no reason to rebuild all of the tools. Once the kernel is built, `build.sh` will print out the location of it along with other information:

```
...
===> Summary of results:
         build.sh command: ./build.sh -u -m sparc64 kernel=MYKERNEL
         build.sh started: Thu Dec  2 23:30:02 CET 2004
         No nonexistent/bin/nbmake, needs building.
         Bootstrapping nbmake
         MACHINE:          sparc64
         MACHINE_ARCH:     sparc64
         TOOLDIR path:     /usr/src/tooldir.NetBSD-2.0-i386
```

```
        DESTDIR path:     /usr/src/destdir.sparc64
        RELEASEDIR path:  /usr/src/releasedir
        Created /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake
        makewrapper:      /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64
        Updated /usr/src/tooldir.NetBSD-2.0-i386/bin/nbmake-sparc64
        Building kernel without building new tools
        Building kernel:  MYKERNEL
        Build directory:  /usr/src/sys/arch/sparc64/compile/obj.sparc64/GENERIC
        Kernels built from MYKERNEL:
         /usr/src/sys/arch/sparc64/compile/obj.sparc64/MYKERNEL/netbsd
        build.sh started: Thu Dec  2 23:30:02 CET 2004
        build.sh ended:   Thu Dec  2 23:38:22 CET 2004
===> .
```

The path to the kernel built is of interest here: `/usr/src/sys/arch/sparc64/compile/obj.sparc64/`*`MYKERNEL`*`/netbsd`, it can be used the same way as described above.

28.5 Crosscompiling the userland

By now it is probably becoming clear that the toolchain actually works in stages. First the crosscompiler is built, then a kernel. Since `build.sh` will attempt to rebuild the tools at every invocation, using "update" saves time. It is probably also clear that outside of a few options, the `build.sh` semantics are basically `build.sh` *`command`*. So, it stands to reason that building the whole userland and/or a release is a matter of using the right commands.

It should be no surprise that building and creating a release would look like the following:

```
% ./build.sh -U -u -m sparc64 build
% ./build.sh -U -u -m sparc64 release
```

These commands will compile the full NetBSD userland and put it into a destination directory, and then build a release from it in a release directory. The "-U" switch is added here for an "unprivileged" build, i.e. one that's running as normal user and not as root. As no further switches to `build.sh` were given nor any environment variables were set, the defaults of `DESTDIR=/usr/src/destdir.sparc64` and `RELEASEDIR=/usr/src/releasedir` are used, as shown in the `build.sh`-output above.

28.6 Crosscompiling the X Window System

The NetBSD project has its own copy of the X Window System's source which is currently based on XFree86 version 4, and which contains changes to make X going on as many of the platforms supported by NetBSD as possible. Due to this, it is desirable to use the X Window System version available from and for NetBSD, which can also be crosscompiled much like the kernel and base system. To do so, the "xsrc" sources must be checked out from CVS into `/usr/xsrc` just as "src" and "pkgsrc" were as described in Chapter 27.

After this, X can be crosscompiled for the target platform by adding the "-x" switch to build.sh, e.g. when creating a full release:

```
% ./build.sh -U -x -u -m sparc64 release
```

The `-U` flag for doing unprivileged (non-root) builds and the `-u` flag for not removing old files before building as well as the `-m` *`arch`* option to define the target architecture have already been introduced, and the `-x` option to also (cross)compile "xsrc" is another option.

28.7 Changing build behaviour

Similar to the old, manual building method, the new toolchain has a lot of variables that can be used to direct things like where certain files go, what (if any) tools are used and so on. A look in `src/BUILDING` covers most of them. In this section some examples of changing default settings are given, each following its own ways.

28.7.1 Changing the Destination Directory

Many people like to track NetBSD-current and perform cross compiles of architectures that they use. The logic for this is simple, sometimes a new feature or device becomes available and someone may wish to use it. By keeping track of changes and building every now and again, one can be assured that these architectures can build their own release.

It is reasonable to assume that if one is tracking and building for more than one architecture, they might want to keep the builds in a different location than the default. There are two ways to go about this, either use a script to set the new DESTDIR, or simply do so interactively. In any case, it can be set the same way as any other variable (depending on your shell of course).

For bash, the Bourne or Korn shell, this is:

```
% export DESTDIR=/usr/builds/sparc64
```

For tcsh and the C shell, the command is:

```
% setenv DESTDIR /usr/builds/sparc64
```

Simple enough. When the build is run, the binaries and files will be sent to `/usr/builds`.

28.7.2 Static Builds

The NetBSD toolchain builds and links against shared libraries by default. Many users still prefer to be able to link statically. Sometimes a small system can be created without having shared libraries, which is a good example of doing a full static build. If a particular build machine will always need one environment variable set in a particular way, then it is easiest to simply add the changed setting to `/etc/mk.conf`.

To make sure a build box always builds statically, simply add the following line to `/etc/mk.conf`:

```
LDSTATIC=-static
```

28.7.3 Using `build.sh` options

Besides variables in environment and `/etc/mk.conf`, the build process can be influenced by a number of switches to the `build.sh` script itself, as we have already seen when forcing unprivileged (non-root) builds, selecting the target architecture or preventing deletion of old files before the build. All these options can be listed by running **build.sh -h**:

```
% cd /usr/src
% build.sh -h
Usage: build.sh [-EnorUux] [-a arch] [-B buildid] [-D dest] [-j njob]
  [-M obj] [-m mach] [-N noisy] [-O obj] [-R release] [-T tools]
  [-V var=[value]] [-w wrapper] [-X x11src] [-Z var]
  operation [...]

 Build operations (all imply "obj" and "tools"):
    build               Run "make build".
    distribution        Run "make distribution" (includes DESTDIR/etc/ files).
    release             Run "make release" (includes kernels & distrib media).

 Other operations:
    help                Show this message and exit.
    makewrapper         Create nbmake-${MACHINE} wrapper and nbmake.
                        Always performed.
    obj                 Run "make obj".  [Default unless -o is used]
    tools               Build and install tools.
    install=idir        Run "make installworld" to 'idir' to install all sets
   except 'etc'.  Useful after "distribution" or "release"
    kernel=conf         Build kernel with config file 'conf'
    releasekernel=conf  Install kernel built by kernel=conf to RELEASEDIR.
    sets                Create binary sets in RELEASEDIR/MACHINE/binary/sets.
   DESTDIR should be populated beforehand.
    sourcesets          Create source sets in RELEASEDIR/source/sets.
    params              Display various make(1) parameters.

 Options:
    -a arch     Set MACHINE_ARCH to arch.  [Default: deduced from MACHINE]
    -B buildId  Set BUILDID to buildId.
    -D dest     Set DESTDIR to dest.  [Default: destdir.MACHINE]
    -E          Set "expert" mode; disables various safety checks.
                Should not be used without expert knowledge of the build system.
    -j njob     Run up to njob jobs in parallel; see make(1) -j.
    -M obj      Set obj root directory to obj; sets MAKEOBJDIRPREFIX.
                Unsets MAKEOBJDIR.
    -m mach     Set MACHINE to mach; not required if NetBSD native.
    -N noisy Set the noisyness (MAKEVERBOSE) level of the build:
      0 Quiet
      1 Operations are described, commands are suppressed
      2 Full output
  [Default: 2]
    -n          Show commands that would be executed, but do not execute them.
    -O obj      Set obj root directory to obj; sets a MAKEOBJDIR pattern.
                Unsets MAKEOBJDIRPREFIX.
    -o          Set MKOBJDIRS=no; do not create objdirs at start of build.
```

```
  -R release  Set RELEASEDIR to release.  [Default: releasedir]
  -r          Remove contents of TOOLDIR and DESTDIR before building.
  -T tools    Set TOOLDIR to tools.  If unset, and TOOLDIR is not set in
              the environment, nbmake will be (re)built unconditionally.
  -U          Set MKUNPRIVED=yes; build without requiring root privileges,
    install from an UNPRIVED build with proper file permissions.
  -u          Set MKUPDATE=yes; do not run "make clean" first.
Without this, everything is rebuilt, including the tools.
  -V v=[val]  Set variable 'v' to 'val'.
  -w wrapper  Create nbmake script as wrapper.
              [Default: ${TOOLDIR}/bin/nbmake-${MACHINE}]
  -X x11src   Set X11SRCDIR to x11src.  [Default: /usr/xsrc]
  -x          Set MKX11=yes; build X11R6 from X11SRCDIR
  -Z v        Unset ("zap") variable 'v'.
```

As can be seen, a number of switches can be set to change the standard build behaviour. A number of them has already been introduced, others can be set as appropriate.

28.7.4 make(1) variables used during build

Several variables control the behaviour of NetBSD builds. Unless otherwise specified, these variables may be set in either the process environment or in the make(1) configuration file specified by `MAKECONF`. For a definitive list of these options, see `BUILDING` in the toplevel source directory.

BUILDID

Identifier for the build. The identifier will be appended to object directory names, and can be consulted in the make(1) configuration file in order to set additional build parameters, such as compiler flags.

DESTDIR

Directory to contain the built NetBSD system. If set, special options are passed to the compilation tools to prevent their default use of the host system's `/usr/include`, `/usr/lib`, and so forth. This pathname should not end with a slash (/) character (For installation into the system's root directory, set `DESTDIR` to an empty string). The directory must reside on a filesystem which supports long filenames and hard links.

Defaults to an empty string if `USETOOLS` is "yes"; unset otherwise. Note: `build.sh` will provide a default (destdir.MACHINE in the top-level `.OBJDIR`) unless run in "expert" mode.

MAKEVERBOSE

The verbosity of build messages. Supported values:

0	No descriptive messages are shown.
1	Descriptive messages are shown.

2	Descriptive messages are shown (prefixed with a '#') and command output is not suppressed.

Default: 2

MKCATPAGES

Can be set to "yes" or "no". Indicates whether preformatted plaintext manual pages will be created during a build.

Default: "yes"

MKCRYPTO

Can be set to "yes" or "no". Indicates whether cryptographic code will be included in a build; provided for the benefit of countries that do not allow strong cryptography. Will not affect the standard low-security password encryption system, crypt(3).

Default: "yes"

MKDOC

Can be set to "yes" or "no". Indicates whether system documentation destined for `DESTDIR/usr/share/doc` will be installed during a build.

Default: "yes"

MKHOSTOBJ

Can be set to "yes" or "no". If set to "yes", then for programs intended to be run on the compile host, the name, release and architecture of the host operating system will be suffixed to the name of the object directory created by "make obj". This allows for multiple host stystems to compile NetBSD for a single target. If set to "no", then programs built to be run on the compile host will use the same object directory names as programs built to be run on the target.

Default: "no"

MKINFO

Can be set to "yes" or "no". Indicates whether GNU info files, used for the documentation of most of the compilation tools, will be created and installed during a build.

Default: "yes"

MKLINT

Can be set to "yes" or "no". Indicates whether lint(1) will be run against portions of the NetBSD source code during the build, and whether lint libraries will be installed into `DESTDIR/usr/libdata/lint`

Default: "yes"

MKMAN

Can be set to “yes” or “no”. Indicates whether manual pages will be installed during a build.

Default: “yes”

MKNLS

Can be set to “yes” or “no”. Indicates whether Native Language System locale zone files will be compiled and installed during a build.

Default: “yes”

MKOBJ

Can be set to “yes” or “no”. Indicates whether object directories will be created when running “make obj”. If set to “no”, then all built files will be located inside the regular source tree.

Default: “yes”

MKPIC

Can be set to “yes” or “no”. Indicates whether shared objects and libraries will be created and installed during a build. If set to “no”, the entire build will be statically linked.

Default: Platform dependent. As of this writing, all platforms except sh3 default to “yes”

MKPICINSTALL

Can be set to “yes” or “no”. Indicates whether the ar(1) format libraries (`lib*_pic.a`), used to generate shared libraries, are installed during a build.

Default: “yes”

MKPROFILE

Can be set to “yes” or “no”. Indicates whether profiled libraries (`lib*_p.a`) will be built and installed during a build.

Default: “yes”; however, some platforms turn off `MKPROFILE` by default at times due to toolchain problems with profiled code.

MKSHARE

Can be set to “yes” or “no”. Indicates whether files destined to reside in `DESTDIR/usr/share` will be built and installed during a build. If set to “no”, then all of `MKCATPAGES`, `MKDOC`, `MKINFO`, `MKMAN` and `MKNLS` will be set to “no” unconditionally.

Default: “yes”

MKTTINTERP

Can be set to “yes” or “no”. For X builds, decides if the TrueType bytecode interpreter is turned on. See freetype.org (http://freetype.org/patents.html) for details.

Default: “no”

MKUNPRIVED

Can be set to “yes” or “no”. Indicates whether an unprivileged install will occur. The user, group, permissions and file flags will not be set on the installed items; instead the information will be appended to a file called `METALOG` in `DESTDIR`. The contents of `METALOG` are used during the generation of the distribution tar files to ensure that the appropriate file ownership is stored.

Default: “no”

MKUPDATE

Can be set to “yes” or “no”. Indicates whether all install operations intended to write to `DESTDIR` will compare file timestamps before installing, and skip the install phase if the destination files are up-to-date. This also has implications on full builds (See below).

Default: “no”

MKX11

Can be set to “yes” or “no”. Indicates whether X11R6 is built from `X11SRCDIR`.

Default: “yes”

TOOLDIR

Directory to hold the host tools, once built. This directory should be unique to a given host system and NetBSD source tree. (However, multiple targets may share the same `TOOLDIR`; the target-dependent files have unique names). If unset, a default based on the uname(1) information of the host platform will be created in the `.OBJDIR` of `src`.

Default: Unset.

USETOOLS

Indicates whether the tools specified by `TOOLDIR` should be used as part of a build in progress. Must be set to “yes” if cross-compiling.

yes	Use the tools from `TOOLDIR`.
no	Do not use the tools from `TOOLNAME`, but refuse to build native compilation tool components that are version-specific for that tool.

never	Do not use the tools from `TOOLNAME`, even when building native tool components. This is similar to the traditional NetBSD build method, but does not verify that the compilation tools in use are up-to-date enough in order to build the tree succesfully. This may cause build or runtime problems when building the whole NetBSD source tree.

Default: “yes” if building all or part of a whole NetBSD source tree (detected automatically); “no” otherwise (to preserve traditional semantics of the `bsd.*.mk` make(1) include files).

X11SRCDIR

Directory containing the X11R6 source. The main X11R6 source is found in `X11SRCDIR/xfree/xc`.

Default: “usr/xsrc”

The following variables only affect the top level `Makefile` and do not affect manually building subtrees of the NetBSD source code.

INSTALLWORLDDIR

Location for the “make installworld” target to install to.

Default: “/”

MKOBJDIRS

Can be set to “yes” or “no”. Indicates whether object directories will be created automatically (via a “make obj” pass) at the start of a build.

Default: “no”

MKUPDATE

Can be set to “yes” or “no”. If set, then addition to the effects described for `MKUPDATE=yes` above, this implies the effect of `NOCLEANDIR` (i.e., “make cleandir” is avoided).

Default: “no”

NOCLEANDIR

If set, avoids the “make cleandir” phase of a full build. This has the effect of allowing only changed files in a source tree to recompiled. This can speed up builds when updating only a few files in the tree.

Default: Unset

NODISTRIBDIRS

If set, avoids the “make distrib-dirs” of a full build. This skips running mtree(8) on `DESTDIR`, useful on systems where building as an unprivileged user, or where it is known that the system wide mtree files have not changed.

Default: Unset

NOINCLUDES

If set, avoids the “make includes” phase of a full build. This has the effect of preventing make(1) from thinking that some programs are out-of-date simply because system include files have changed. However, this option should not be trusted when updating the entire NetBSD source tree arbitrarily; it is suggested to use `MKUPDATE=yes` in that case.

Default: Unset

RELEASEDIR

If set, specifies the directory to which a release(7) layout will be written at the end of a “make release”.

Default: Unset

Chapter 29
Compiling the kernel

Most NetBSD users will sooner or later want to recompile their kernel, or compile a customized kernel. This might be for several reasons:

- you can install bug-fixes, security updates, or new functionality by rebuilding the kernel from updated sources.
- by removing unused device drivers and kernel sub-systems from your configuration, you can dramatically reduce kernel size and, therefore, memory usage.
- by enabling optimisations more specific to your hardware, or tuning the system to match your specific sizing and workload, you can improve performance.
- you can access additional features by enabling kernel options or sub-systems, some of which are experimental or disabled by default.
- you can solve problems of detection/conflicts of peripherals.
- you can customize some options (for example keyboard layout, BIOS clock offset, ...)
- you can get a deeper knowledge of the system.

29.1 Requirements and procedure

To recompile the kernel you must have installed the compiler set (`comp.tgz`).

The basic steps to an updated or customised kernel then are:

1. Install or update the kernel sources
2. Create or modify the kernel configuration file
3. Building the kernel from the configuration file, either manually or using **build.sh**
4. Install the kernel

29.2 Installing the kernel sources

> You can get the kernel sources from AnonCVS (see Chapter 27), or from the `syssrc.tgz` tarball that is located in the `source/sets/` directory of the release that you are using.

If you chose to use AnonCVS to fetch the entire source tree, be patient, the operation can last many minutes, because the repository contains thousands of files.

If you have a source tarball, you can extract it as root:

```
# cd /
# tar zxf /path/to/syssrc.tgz
```

Even if you used the tarball from the release, you may wish to use AnonCVS to update the sources with changes that have been applied since the release. This might be especially relevant if you are updating the kernel to include the fix for a specific bug, including a vulnerability described in a NetBSD Security Advisory. You might want to get the latest sources on the relevant release or critical updates branch for your version, or Security Advisories will usually contain information on the dates or revisions of the files containing the specific fixes concerned. See Section 27.4 for more details on the CVS commands used to update sources from these branches.

Once you have the sources available, you can create a custom kernel: this is not as difficult as you might think. In fact, a new kernel can be created in a few steps which will be described in the following sections.

29.3 Creating the kernel configuration file

The directories described in this section are i386 specific. Users of other architectures must substitute the appropriate directories, see the subdirectories of `src/sys/arch` for a list.

The kernel configuration file defines the type, the number and the characteristics of the devices supported by the kernel as well as several kernel configuration options. For the i386 port, kernel configuration files are located in the `/usr/src/sys/arch/i386/conf` directory.

Please note that the names of the kernel configuration files are historically in all uppercase, so they are easy to distinguish from other files in that directory:

```
$ cd /usr/src/sys/arch/i386/conf/
$ ls
CARDBUS                 GENERIC_PS2TINY         NET4501
CVS                     GENERIC_TINY            SWINGER
DELPHI                  GENERIC_VERIEXEC        SWINGER.MP
DISKLESS                INSTALL                 VIRTUALPC
GENERIC                 INSTALL.MP              files.i386
GENERIC.FAST_IPSEC      INSTALL_LAPTOP          kern.ldscript
GENERIC.MP              INSTALL_PS2             kern.ldscript.4MB
GENERIC.MPDEBUG         INSTALL_SMALL           largepages.inc
GENERIC.local           INSTALL_TINY            majors.i386
GENERIC_DIAGNOSTIC      IOPENER                 std.i386
GENERIC_ISDN            LAMB
GENERIC_LAPTOP          Makefile.i386
```

The easiest way to create a new file is to copy an existing one and modify it. Usually the best choice on most platforms is the GENERIC configuration, as it contains most drivers and options. In the configuration file there are comments describing the options; a more detailed description is found in the options(4) man page. So, the usual procedure is:

```
$ cp GENERIC MYKERNEL
$ vi MYKERNEL
```

The modification of a kernel configuration file basically involves three operations:

1. support for hardware devices is included/excluded in the kernel (for example, SCSI support can be removed if it is not needed.)
2. support for kernel features is enabled/disabled (for example, enable NFS client support, enable Linux compatibility, ...)
3. tuning kernel parameters.

Lines beginning with "#" are comments; lines are disabled by commenting them and enabled by removing the comment character. It is better to comment lines instead of deleting them; it is always possible uncomment them later.

The output of the dmesg(8) command can be used to determine which lines can be disabled. For each line of the type:

```
XXX at YYY
```

both *XXX* and *YYY* must be active in the kernel configuration file. You'll probably have to experiment a bit before achieving a minimal configuration but on a desktop system without SCSI and PCMCIA you can halve the kernel size.

You should also examine the options in the configuration file and disable the ones that you don't need. Each option has a short comment describing it, which is normally sufficient to understand what the option does. Many options have a longer and more detailed description in the options(4) man page. While you are at it you should set correctly the options for local time on the CMOS clock. For example:

```
options RTC_OFFSET=-60
```

The **adjustkernel** Perl script, which is available through pkgsrc, analyzes the output of dmesg(8) and automatically generates a minimal configuration file. The installation of packages is described extensively in in the Part VI in *The NetBSD & pkgsrc Guide*, but installing **adjustkernel** basically boils down to:

```
$ cd /usr/pkgsrc/sysutils/adjustkernel
$ make install
```

You can now run the script with:

```
$ cd /usr/src/sys/arch/i386/conf
$ adjustkernel GENERIC > MYKERNEL
```

This script usually works very well, saving a lot of manual editing. But be aware that the script only configures the available devices: you must still configure the other options manually.

29.4 Building the kernel manually

Based on your kernel configuration file, either one of the standard configurations or your customised configuration, a new kernel must be built.

These steps can either be performed manually, or using the **build.sh** command that was introduced in section Chapter 28. This section will give instructions on how to build a native kernel using manual steps, the following section Section 29.5 describes how to use **build.sh** to do the same.

- Configure the kernel
- Generate dependencies
- Compile the kernel

29.4.1 Configuring the kernel manually

When you've finished modifying the kernel configuration file (which we'll call `MYKERNEL`), you should issue the following command:

```
$ config MYKERNEL
```

If `MYKERNEL` contains no errors, the config(8) program will create the necessary files for the compilation of the kernel, otherwise it will be necessary to correct the errors before running config(8) again.

29.4.2 Generating dependencies and recompiling manually

Dependencies generation and kernel compilation is performed by the following commands:

```
$ cd ../compile/MYKERNEL
$ make depend
$ make
```

It can happen that the compilation stops with errors; there can be a variety of reasons but the most common cause is an error in the configuration file which didn't get caught by config(8). Sometimes the failure is caused by a hardware problem (often faulty RAM chips): the compilation puts a higher stress on the system than most applications do. Another typical error is the following: option B, active, requires option A which is not active. A full compilation of the kernel can last from some minutes to several hours, depending on the hardware.

The result of a successful make command is the `netbsd` file in the compile directory, ready to be installed.

29.5 Building the kernel using `build.sh`

After creating and possibly editing the kernel config file, the manual steps of configuring the kernel, generating dependencies and recompiling can also be done using the `src/build.sh` script, all in one go:

```
$ cd /usr/src
$ ./build.sh kernel=MYKERNEL
```

This will perform the same steps as above, with one small difference: before compiling, all old object files will be removed, to start with a fresh build. This is usually overkill, and it's fine to keep the old file and only rebuild the ones whose dependencies have changed. To do this, add the `-u` option to `build.sh`:

```
$ cd /usr/src
$ ./build.sh -u kernel=MYKERNEL
```

At the end of its job, `build.sh` will print out the location where the new compiled kernel can be found. It can then be installed.

29.6 Installing the new kernel

Whichever method was used to produce the new kernel file, it must now be installed. The new kernel file should be copied to the root directory, after saving the previous version.

```
# mv /netbsd /netbsd.old
# mv netbsd /
```

Customization can considerably reduce the kernel's size. In the following example `netbsd.old` is the install kernel and `netbsd` is the new kernel.

```
-rwxr-xr-x  3 root  wheel  3523098 Dec 10 00:13 /netbsd
-rwxr-xr-x  3 root  wheel  7566271 Dec 10 00:13 /netbsd.old
```

The new kernel is activated after rebooting:

```
# shutdown -r now
```

29.7 If something went wrong

When the computer is restarted it can happen that the new kernel doesn't work as expected or even doesn't boot at all. Don't worry: if this happens, just reboot with the previously saved kernel and remove the new one (it is better to reboot "single user"):

- Reboot the machine
- Press the space bar at the boot prompt during the 5 seconds countdown

  ```
  boot:
  ```

- Type

  ```
  > boot netbsd.old -s
  ```

- Now issue the following commands to restore the previous version of the kernel:

  ```
  # fsck /
  # mount /
  # mv netbsd.old netbsd
  # reboot
  ```

This will give you back the working system you started with, and you can revise your custom kernel config file to resolve the problem. In general, it's wise to start with a GENERIC kernel first, and then make gradual changes.

Chapter 30
Console drivers

In NetBSD versions before 1.4 the user could choose between two different drivers for screen and keyboard, pccons (specific for i386) and pcvt. In NetBSD 1.4 the new wscons multiplatform driver appeared, which has substituted the previous drivers, of which pccons is still supported as it uses less system ressources and is used for install floppies due to that.

30.1 wscons

Wscons is NetBSD's standard console driver. It offers virtual terminals on i386, support for international keyboards, mouse handling, etc. The capabilities of wscons can vary depending on the port, the i386 version is very feature rich.

30.1.1 Virtual consoles

The number of pre-allocated virtual console is controlled by the following option

```
options     WSDISPLAY_DEFAULTSCREENS=4
```

Other consoles can be added by enabling the relevant lines in the `/etc/wscons.conf` file: the comment mark (#) must be removed from the lines beginning with "screen x". In the following example a fifth console is added to the four pre-allocated ones:

```
# screens to create
#       idx     screen  emul
#screen 0       -       vt100
screen 1        -       vt100
screen 2        -       vt100
screen 3        -       vt100
screen  4       -       -
#screen 4       80x25bf vt100
#screen 5       80x50   vt100
```

The `rc.wscons` script transforms each of the non commented lines in a call to the **wsconscfg** command: the columns become the parameters of the call. The *idx* column becomes the `index` parameter, the *screen* column becomes the `-t type` parameter (which defines the type of screen: rows and columns, number of colors, ...) and the *emul* column becomes the `-e emul` parameter, which defines the emulation. For example:

```
screen 3       -       vt100
```

becomes a call to:

```
wsconscfg -e vt100 3
```

Please note that it is possible to have a (harmless) conflict between the consoles pre-allocated by the kernel and the consoles allocated at boot time through `/etc/wscons.conf`. If during boot the system tries to allocate an already allocated screen, the following message will be displayed:

```
wsconscfg: WSDISPLAYIO_ADDSCREEN: Device busy
```

The soulution is to comment out the offending lines in `/etc/wscons.conf`.

The virtual console must also be active in `/etc/ttys`, so that NetBSD runs the getty(8) program to ask for login. For example:

```
console "/usr/libexec/getty Pc"         pc3     off secure
ttyE0   "/usr/libexec/getty Pc"         vt220   on secure
ttyE1   "/usr/libexec/getty Pc"         vt220   on secure
ttyE2   "/usr/libexec/getty Pc"         vt220   on secure
ttyE3   "/usr/libexec/getty Pc"         vt220   off secure
...
```

When starting up the X server, it will look for a virtual console with no getty(8) program running, e.g. one console should left as "off" in `/etc/ttys`. The line

```
ttyE3   "/usr/libexec/getty Pc"         vt220   off secure
```

of `/etc/ttys` is used by the X server for this purpose. To use a screen different from number 4, a parameter of the form vt*n* must be passed to the X server, where *n* is the number of the function key used to activate the screen for X.

For example, “screen 7” could be enabled in `/etc/wscons.conf` and X could be started with “vt8”. If you use xdm you must edit `/usr/X11R6/lib/X11/xdm/Xserver`. For example:

```
:0 local /usr/X11R6/bin/X +kb dpms -bpp 16 dpms vt8
```

For xdm3d the path is different: `/usr/X11R6/share/xdm3d/Xservers`.

30.1.2 50 lines text mode with wscons

A text mode with 50 lines can be used starting with version 1.4.1 of NetBSD. This mode is activated in the `/etc/wscons.conf`. The following line must be uncommented:

```
font ibm  -  8  ibm  /usr/share/pcvt/fonts/vt220l.808
```

Then the following lines must be modified:

```
#screen 0       80x50   vt100
screen  1       80x50   vt100
screen  2       80x50   vt100
screen  3       80x50   vt100
screen  4       80x50   vt100
screen  5       80x50   vt100
screen  6       80x50   vt100
screen  7       80x50   vt100
```

This configuration enables eight screens, which can be accessed with the key combination Ctrl-Alt-F*n* (where *n* varies from 1 to 8); the corresponding devices are ttyE0..ttyE7. To enable them and get a login prompt, `/etc/ttys` must be modified:

```
ttyE0   "/usr/libexec/getty Pc"         vt220   on secure
ttyE1   "/usr/libexec/getty Pc"         vt220   on secure
ttyE2   "/usr/libexec/getty Pc"         vt220   on secure
ttyE3   "/usr/libexec/getty Pc"         vt220   on secure
ttyE4   "/usr/libexec/getty Pc"         vt220   on secure
ttyE5   "/usr/libexec/getty Pc"         vt220   on secure
ttyE6   "/usr/libexec/getty Pc"         vt220   on secure
ttyE7   "/usr/libexec/getty Pc"         vt220   on secure
```

It is not possible to modify the 80x25 setting of screen 0, probably to guarantee that even in case of problems there is always a working screen.

30.1.3 Keyboard mappings

Wscons also allows setting the keymap to map the keys on various national keyboards to the right characters. E.g. to set the keymap for an italian keymap, run:

```
# wsconsctl -k -w encoding=it
encoding -> it
```

This setting will last until the next reboot. To make it permanent, add a “encoding”line to `/etc/wscons.conf`: it will be executed automatically the next time you reboot.

```
# cp /etc/wscons.conf /etc/wscons.conf.orig
# echo encoding it >>/etc/wscons.conf
```

Please be careful and type two “>” characters. If you type only one “>”, you will overwrite the file instead of adding a line. But that’s why we always make backup files before touching critical files!

A full list of keyboard mappings can be found in `/usr/src/sys/dev/wscons/wsksymdef.h`:

- be - Belgian
- de - German
- dk - Danish
- es - Spanish
- fr - French
- hu - Hungarian
- it - Italian
- jp - Japanese
- no - Norwegian
- pl - Polish
- pt - Portuguese

- ru - Russian
- sf - Swiss French
- sg - Swiss German
- sv - Swedish
- ua - Ukrainian
- uk - UK-English
- us - US-English

30.1.4 Cut&paste on the console with wsmoused

It is possible to use the mouse on the wscons console to mark (cut) text with one mouse button, and insert (paste) it again with another button. To enable cut&paste in wscons, you first need to build support for it into the kernel, which is disabled by default. To so so, enable the following option:

```
options WSDISPLAY_CHARFUNCS             # mouse console support
```

Beware that this option won't work in conjunction with the `VGA_RASTERCONSOLE` option. See Chapter 29 for more information on building the kernel.

After the kernel is built, enable "wsmoused" in `/etc/rc.conf`, and start it:

```
# echo wsmoused=yes >>/etc/rc.conf
# sh /etc/rc.d/wsmoused start
```

After that you can use the mouse to mark text with the left mouse button, and paste it with the right one. To tune the behaviour of wsmoused(8) see its manpage, which also describes the format of the wsmoused.conf(5) config file, an example of which can be found in `/usr/share/examples/wsmoused`.

30.1.5 Enable scrollback on the console

You can enable scrolling back on wscons consoles by compiling the `WSDISPLAY_SCROLLSUPPORT` option into your kernel. Make sure you don't have option `VGA_RASTERCONSOLE` enabled at the same time though! See Chapter 29 for instructions on building a kernel.

When you have a kernel with options `WSDISPLAY_SCROLLSUPPORT` running, you can scroll up on the console by pressing LEFT SHIFT plus PAGE UP/DOWN. Please note that this may not work on your system console (ttyE0)!

30.2 pccons

This console driver doesn't offer virtual consoles and utility programs for configuration but takes up very little space. Due to this, it can be found on the i386 install floppy.

To enable it, put the following line in your kernel config file:

```
pc0 at isa? port 0x60 irq 1 # pccons generic PC console driver
```

You can also set one of several options to compile in a non-english keymap:

```
# Keyboard layout configuration for pccons
#options        FRENCH_KBD
#options        FINNISH_KBD
#options        GERMAN_KBD
#options        NORWEGIAN_KBD
```

Remove the comment character in front of one of this to enable the corresponding keymap, then follow the instructions in Chapter 29 to rebuild your kernel.

In general, you shouldn't need pccons though, and wscons should fit all your needs.

VI. The pkgsrc user’s guide

Chapter 31
Where to get pkgsrc

There are three ways to get pkgsrc. Either as a tar file, via SUP, or via CVS. All three ways are described here.

31.1 As tar file

To get pkgsrc going, you need to get the pkgsrc.tar.gz file from ftp.NetBSD.org (ftp://ftp.NetBSD.org/pub/NetBSD/NetBSD-current/tar_files/pkgsrc.tar.gz) and unpack it into `/usr/pkgsrc`.

31.2 Via SUP

As an alternative to the tar file, you can get pkgsrc via the Software Update Protocol, SUP. To do so, make sure your supfile has a line

```
release=pkgsrc
```

in it, see the examples in `/usr/share/examples/supfiles`, and that the `/usr/pkgsrc` directory exists. Then, simply run **sup -v *`/path/to/your/supfile`***.

31.3 Via CVS

To get pkgsrc via CVS, make sure you have "cvs" installed. To do an initial (full) checkout of pkgsrc, do the following steps:

```
% setenv CVSROOT anoncvs@anoncvs.NetBSD.org:/cvsroot
% setenv CVS_RSH ssh
% cd /usr
% cvs checkout -P pkgsrc
```

This will create the `pkgsrc` directory in your `/usr`, and all the package source will be stored under `/usr/pkgsrc`. To update pkgsrc after the initial checkout, make sure you have `CVS_RSH` set as above, then do:

```
% cd /usr/pkgsrc
% cvs -q update -dP
```

Please also note that it is possible to have multiple copies of the pkgsrc hierarchy in use at any one time - all work is done relatively within the pkgsrc tree.

Chapter 32
Using pkgsrc on systems other than NetBSD

32.1 Bootstrapping pkgsrc

For operating systems other than NetBSD, we provide a bootstrap kit to build the required tools to use pkgsrc on your platform. Besides support for native NetBSD, pkgsrc and the bootstrap kit have support for the following operating systems:

- Darwin (Mac OS X)
- DragonFlyBSD
- FreeBSD
- Interix (Windows 2000, XP, 2003)
- IRIX
- Linux
- OpenBSD
- Solaris
- Tru64 (Digital UNIX/OSF1)

Support for other platforms is under development.

Installing the bootstrap kit should be as simple as:

```
# env CVS_RSH=ssh cvs -d anoncvs@anoncvs.NetBSD.org:/cvsroot checkout pkgsrc
# cd pkgsrc/bootstrap
# ./bootstrap
```

See Chapter 31 for other ways to get pkgsrc before bootstrapping. The given **bootstrap** command will use the defaults of `/usr/pkg` for the *prefix* where programs will be installed in, and `/var/db/pkg` for the package database directory where pkgsrc will do its internal bookkeeping. However, these can also be set using command-line arguments.

Binary packages for the pkgsrc tools and an initial set of packages is available for supported platforms. An up-to-date list of these can be found on www.pkgsrc.org (http://www.pkgsrc.org/).

> **Note:** The bootstrap installs a **bmake** tool. Use this **bmake** when building via pkgsrc. For examples in this guide, use **bmake** instead of “make”.

32.2 Platform-specific notes

Here are some platform-specific notes you should be aware of.

32.2.1 Darwin (Mac OS X)

Darwin 5.x and 6.x are supported. There are two methods of using pkgsrc on Mac OS X, by using a disk image, or a UFS partition.

Before you start, you will need to download and install the Mac OS X Developer Tools from Apple's Developer Connection. See http://developer.apple.com/macosx/ for details. Also, make sure you install X11 for Mac OS X and the X11 SDK from http://www.apple.com/macosx/x11/download/ if you intend to build packages that use the X11 Window System.

If you already have a UFS partition, or have a spare partition that you can format as UFS, it is recommended to use that instead of the disk image. It'll be somewhat faster and will mount automatically at boot time, where you must manually mount a disk image.

> **Note:** You cannot use a HFS+ file system for pkgsrc, because pkgsrc currently requires the file system to be case-sensitive, and HFS+ is not.

32.2.1.1 Using a disk image

Create the disk image:

```
# cd pkgsrc/bootstrap
# ./ufsdiskimage create ~/Documents/NetBSD 512 # megabytes - season to taste
# ./ufsdiskimage mount ~/Documents/NetBSD
# sudo chown `id -u`:`id -g` /Volumes/NetBSD
```

That's it!

32.2.1.2 Using a UFS partition

By default, `/usr` will be on your root file system, normally HFS+. It is possible to use the default *prefix* of `/usr/pkg` by symlinking `/usr/pkg` to a directory on a UFS file system. Obviously, another symlink is required if you want to place the package database directory outside the *prefix*. e.g.

```
# ./bootstrap --pkgdbdir /usr/pkg/pkgdb --pkgsrcdir /Volumes/ufs/pkgsrc
```

If you created your partitions at the time of installing Mac OS X and formatted the target partition as UFS, it should automatically mount on `/Volumes/<volume name>` when the machine boots. If you are (re)formatting a partition as UFS, you need to ensure that the partition map correctly reflects "Apple_UFS" and not "Apple_HFS".

The problem is that none of the disk tools will let you touch a disk that is booted from. You can unmount the partition, but even if you newfs it, the partition type will be incorrect and the automounter won't mount it. It can be mounted manually, but it won't appear in Finder.

You'll need to boot off of the OS X Installation (User) CD. When the Installation program starts, go up to the menu and select Disk Utility. Now, you will be able to select the partition you want to be UFS, and Format it Apple UFS. Quit the Disk Utility, quit the installer which will reboot your machine. The new UFS file system will appear in Finder.

Be aware that the permissions on the new file system will be writable by root only.

This note is as of 10.2 (Jaguar) and applies to earlier versions. Hopefully Apple will fix Disk Utility in 10.3 (Panther).

32.2.2 FreeBSD

FreeBSD 4.7 and 5.0 have been tested and are supported, other versions may work.

Care should be taken so that the tools that this kit installs do not conflict with the FreeBSD userland tools. There are several steps:

1. FreeBSD stores its ports pkg database in `/var/db/pkg`. It is therefore recommended that you choose a different location (e.g. `/usr/pkgdb`) by using the --pkgdbdir option to the bootstrap script.
2. If you do not intend to use the FreeBSD ports tools, it's probably a good idea to move them out of the way to avoid confusion, e.g.

```
# cd /usr/sbin
# mv pkg_add pkg_add.orig
# mv pkg_create pkg_create.orig
# mv pkg_delete pkg_delete.orig
# mv pkg_info pkg_info.orig
```

3. An example `/etc/mk.conf` file will be placed in `/etc/mk.conf.example` file when you use the bootstrap script.

32.2.3 Interix

Interix is a POSIX-compatible subsystem for the Windows NT kernel, providing a Unix-like environment with a tighter kernel integration than available with Cygwin. It is part of the Windows Services for Unix package, available for free for any licensed copy of Windows 2000, XP (not including XP Home), or 2003. SFU can be downloaded from http://www.microsoft.com/windows/sfu/.

Services for Unix 3.5, current as of this writing, has been tested. 3.0 or 3.1 may work, but are not officially supported. (The main difference in 3.0/3.1 is lack of pthreads.)

32.2.3.1 When installing Interix/SFU

At an absolute minimum, the following packages must be installed from the Windows Services for Unix 3.5 distribution in order to use pkgsrc:

- Utilities -> Base Utilities
- Interix GNU Components -> (all)
- Remote Connectivity

- Interix SDK

When using pkgsrc on Interix, DO NOT install the Utilities subcomponent "UNIX Perl". That is Perl 5.6 without shared module support, installed to /usr/local, and will only cause confusion. Instead, install Perl 5.8 from pkgsrc (or from a binary package).

The Remote Connectivity subcomponent "Windows Remote Shell Service" does not need to be installed, but Remote Connectivity itself should be installed in order to have a working inetd.

Finally, during installation you may be asked whether to enable setuid behavior for Interix programs, and whether to make pathnames default to case-sensitive. Setuid should be enabled, and case-sensitivity MUST be enabled. (Without case-sensitivity, a large number of packages including perl will not build.)

32.2.3.2 What to do if Interix/SFU is already installed

If SFU is already installed and you wish to alter these settings to work with pkgsrc, note the following things.

- To uninstall UNIX Perl, use Add/Remove Programs, select Microsoft Windows Services for UNIX, then click Change. In the installer, choose Add or Remove, then uncheck Utilities->UNIX Perl.
- To enable case-sensitivity for the file system, run REGEDIT.EXE, and change the following registry key:

  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\kernel

  Set the DWORD value "obcaseinsensitive" to 0; then reboot.
- To enable setuid binaries (optional), run REGEDIT.EXE, and change the following registry key:

  HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Services for UNIX

  Set the DWORD value "EnableSetuidBinaries" to 1; then reboot.

32.2.3.3 Important notes for using pkgsrc

The package imanager (either the pkgsrc "su" user, or the user running "pkg_add") must be a member of the local Administrators group. Such a user must also be used to run the bootstrap. This is slightly relaxed from the normal pkgsrc requirement of "root".

The package manager should use a umask of 002. "make install" will automatically complain if this is not the case. This ensures that directories written in /var/db/pkg are Administrators-group writeable.

The popular Interix binary packages from http://www.interopsystems.com/ use an older version of pkgsrc's pkg_* tools. Ideally, these should NOT be used in conjunction with pkgsrc. If you choose to use them at the same time as the pkgsrc packages, ensure that you use the proper pkg_* tools for each type of binary package.

The TERM setting used for DOS-type console windows (including those invoked by the csh and ksh startup shortcuts) is "interix". Most systems don't have a termcap/terminfo entry for it, but the following .termcap entry provides adequate emulation in most cases:

```
interix:kP=\E[S:kN=\E[T:kH=\E[U:dc@:DC@:tc=pcansi:
```

32.2.4 IRIX

You will need a working C compiler, either gcc or SGI's MIPS and MIPSpro compiler (cc/c89). Please set the CC environment variable according to your preference. If you do not have a license for the MIPSpro compiler suite, you can download a gcc tardist file from http://freeware.sgi.com/.

Please note that you will need IRIX 6.5.17 or higher, as this is the earliest version of IRIX providing support for if_indextoname(3), if_nametoindex(3), etc.

At this point in time, pkgsrc only supports one ABI at a time. That is, you can not switch between the old 32-bit ABI, the new 32-bit ABI and the 64-bit ABI. If you start out using "abi=n32", that's what all your packages will be built with.

Therefore, please make sure that you have no conflicting CFLAGS in your environment or the /etc/mk.conf. Particularly, make sure that you do not try to link n32 object files with lib64 or vice versa. Check your /etc/compiler.defaults!

If you have the actual pkgsrc tree mounted via NFS from a different host, please make sure to set WRKOBJDIR to a local directory, as it appears that IRIX linker occasionally runs into issues when trying to link over a network-mounted file system.

The bootstrapping process should set all the right options for programs such as imake(1), but you may want to set some options depending on your local setup. Please see pkgsrc/mk/defaults/mk.conf and, of course, your compiler's man pages for details.

If you are using SGI's MIPSPro compiler, please set

```
PKGSRC_COMPILER=mipspro
```

in /etc/mk.conf. Otherwise, pkgsrc will assume you are using gcc and may end up passing invalid flags to the compiler. Note that bootstrap should create an appropriate mk.conf.example by default.

If you have both the MIPSPro compiler chain installed as well as gcc, but want to make sure that MIPRPro is used, please set your PATH to *not* include the location of gcc (often /usr/freeware/bin), and (important) pass the '--preserve-path' flag.

32.2.5 Linux

Some versions of Linux (for example Debian GNU/Linux) need either libtermcap or libcurses (libncurses). Installing the distributions libncurses-dev package (or equivalent) should fix the problem.

pkgsrc supports both gcc (GNU Compiler Collection) and icc (Intel C++ Compiler). gcc is the default. icc 8.0 and 8.1 on i386 have been tested.

To bootstrap using icc, assuming the default icc installation directory:

```
env CC=/opt/intel_cc_80/bin/icc LDFLAGS=-static-libcxa \
                ac_cv___attribute__=yes ./bootstrap
```

Note: icc 8.1 needs the '-i-static' argument instead of -static-libcxa.

icc supports __attribute__, but the GNU configure test uses a nested function, which icc does not support. #undef'ing __attribute__ has the unfortunate side-effect of breaking many of the Linux header files, which cannot be compiled properly without __attribute__. The test must be overridden so that __attribute__ is assumed supported by the compiler.

After bootstrapping, you should set `PKGSRC_COMPILER` in `/etc/mk.conf`:

```
PKGSRC_COMPILER=icc
```

The default installation directory for icc is `/opt/intel_cc_80`, which is also the pkgsrc default. If you have installed it into a different directory, set `ICCBASE` in `/etc/mk.conf`:

```
ICCBASE=/opt/icc
```

pkgsrc uses the static linking method of the runtime libraries provided by icc, so binaries can be run on other systems which do not have the shared libraries installed.

Libtool, however, extracts a list of libraries from the ld(1) command run when linking a C++ shared library and records it, throwing away the -Bstatic and -Bdynamic options interspersed between the libraries. This means that libtool-linked C++ shared libraries will have a runtime dependency on the icc libraries until this is fixed in libtool.

32.2.6 OpenBSD

OpenBSD 3.0 and 3.2 are tested and supported.

Care should be taken so that the tools that this kit installs do not conflict with the OpenBSD userland tools. There are several steps:

1. OpenBSD stores its ports pkg database in `/var/db/pkg`. It is therefore recommended that you choose a different location (e.g. `/usr/pkgdb`) by using the --pkgdbdir option to the bootstrap script.
2. If you do not intend to use the OpenBSD ports tools, it's probably a good idea to move them out of the way to avoid confusion, e.g.

   ```
   # cd /usr/sbin
   # mv pkg_add pkg_add.orig
   # mv pkg_create pkg_create.orig
   # mv pkg_delete pkg_delete.orig
   # mv pkg_info pkg_info.orig
   ```

3. An example `/etc/mk.conf` file will be placed in `/etc/mk.conf.example` file when you use the bootstrap script. OpenBSD's make program uses `/etc/mk.conf` as well. You can work around this by enclosing all the pkgsrc-specific parts of the file with:

   ```
   .ifdef BSD_PKG_MK
   # pkgsrc stuff, e.g. insert defaults/mk.conf or similar here
   .else
   # OpenBSD stuff
   .endif
   ```

32.2.7 Solaris

Solaris 2.6 through 9 are supported on both x86 and sparc. You will need a working C compiler. Both gcc 2.95.3 and Sun WorkShop 5 have been tested.

The following packages are required on Solaris 8 for the bootstrap process and to build packages.

- SUNWsprot
- SUNWarc
- SUNWbtool
- SUNWtoo
- SUNWlibm

Please note the use of GNU binutils on Solaris is *not* supported.

32.2.7.1 If you are using gcc

It makes life much simpler if you only use the same gcc consistently for building all packages.

It is recommended that an external gcc be used only for bootstrapping, then either build gcc from `lang/gcc` or install a binary gcc package, then remove gcc used during bootstrapping.

Binary packages of gcc can be found through
http://www.sun.com/bigadmin/common/freewareSearch.html.

32.2.7.2 If you are using Sun WorkShop

You will need at least the following packages installed (from WorkShop 5.0)

- SPROcc - Sun WorkShop Compiler C 5.0
- SPROcpl - Sun WorkShop Compiler C++ 5.0
- SPROild - Sun WorkShop Incremental Linker
- SPROlang - Sun WorkShop Compilers common components

You should set `CC`, `CXX` and optionally, `CPP` in `/etc/mk.conf`, e.g.:

```
CC=     cc
CXX=    CC
CPP=    /usr/ccs/lib/cpp
```

You may also want to build 64-bit binaries, e.g.:

```
CFLAGS= -xtarget=ultra -xarch=v9
```

Whichever compiler you use, please ensure the compiler tools and your $prefix are in your `PATH`. This includes `/usr/ccs/{bin,lib}` and e.g. `/usr/pkg/{bin,sbin}`.

Chapter 33
Using pkgsrc

33.1 Working with binary packages

This section describes how to find, retrieve and install a precompiled binary package that someone else already prepared for your type of machine.

33.1.1 Where to get binary packages

Precompiled packages are stored on ftp.NetBSD.org and its mirrors in the directory `/pub/NetBSD/packages/<OSVERSION>/<ARCH>/` for anonymous FTP access. `OSVERSION` is the NetBSD version (**uname -r**), `ARCH` is the architecture (**uname -p**). In that directory, there is a subdirectory for each category plus a subdirectory `All` which includes the actual binaries in `.tgz` files. The category subdirectories use symbolic links to those files (this is the same directory layout as in `/usr/pkgsrc/packages`).

This same directory layout applies for CD-ROM distributions, only that the directory may be rooted somewhere else, probably somewhere below `/cdrom`. Please consult your CD-ROMs documentation for the exact location.

33.1.2 How to use binary packages

If you have the files on a CD-ROM or downloaded them to your hard disk, you can install them with the following command (be sure to **su** to root first):

```
# pkg_add /path/to/package.tgz
```

If you have FTP access and you don't want to download the packages via FTP prior to installation, you can do this automatically by giving **pkg_add** an FTP URL:

```
# pkg_add ftp://ftp.NetBSD.org/pub/NetBSD/packages/<OSVERSION>/<ARCH>/All/package.tgz
```

Note that any prerequisite packages needed to run the package in question will be installed, too, assuming they are present where you install from.

To save some typing, you can set the `PKG_PATH` environment variable to a semicolon-separated list of paths (including remote URLs); trailing slashes are not allowed.

Additionally to the `All` directory there exists a `vulnerable` directory to which binary packages with known vulnerabilities are moved, since removing them could cause missing dependencies. To use these packages, add the `vulnerable` directory to your `PKG_PATH`. However, you should run `security/audit-packages` regularly, especially after installing new packages, and verify that the vulnerabilities are acceptable for your configuration. An example `PKG_PATH` would be:

`ftp://ftp.NetBSD.org/pub/NetBSD/packages/<OSVERSION>/<ARCH>/All;ftp://ftp.NetBSD.org/pub/Net`
Please note that semicolon (';') is a shell meta-character, so you'll probably have to quote it.

After you've installed packages, be sure to have `/usr/pkg/bin` and `/usr/pkg/sbin` in your `PATH` so you can actually start the just installed program.

33.1.3 A word of warning

Please pay very careful attention to the warnings expressed in the pkg_add(1) manual page about the inherent dangers of installing binary packages which you did not create yourself, and the security holes that can be introduced onto your system by indiscriminate adding of such files.

33.2 Building packages from source

This assumes that the package is already in pkgsrc. If it is not, see
Part VII in *The NetBSD & pkgsrc Guide* for instructions how to create your own packages.

33.2.1 Requirements

To build packages from source on a NetBSD system the "comp" and the "text" distribution sets must be installed. If you want to build X11-related packages the "xbase" and "xcomp" distribution sets are required, too.

33.2.2 Fetching distfiles

The first step for building a package is downloading the distfiles (i.e. the unmodified source). If they have not yet been downloaded, pkgsrc will fetch them automatically.

You can overwrite some of the major distribution sites to fit to sites that are close to your own. Have a look at `pkgsrc/mk/defaults/mk.conf` to find some examples — in particular, look for the `MASTER_SORT`, `MASTER_SORT_REGEX` and `INET_COUNTRY` definitions. This may save some of your bandwidth and time.

You can change these settings either in your shell's environment, or, if you want to keep the settings, by editing the `/etc/mk.conf` file, and adding the definitions there.

If you don't have a permanent Internet connection and you want to know which files to download, **make fetch-list** will tell you what you'll need. Put these distfiles into `/usr/pkgsrc/distfiles`.

33.2.3 How to build and install

Assuming that the distfile has been fetched (see previous section), become root and change into the relevant directory and run **make**.

> **Note:** If using bootstrap or pkgsrc on a non-NetBSD system, use the pkgsrc **bmake** command instead of "make" in the examples in this guide.

For example, type

```
% cd misc/figlet
% make
```

at the shell prompt to build the various components of the package, and

```
# make install
```

to install the various components into the correct places on your system. Installing the package on your system requires you to be root. However, pkgsrc has a *just-in-time-su* feature, which allows you to only become root for the actual installation step

Taking the figlet utility as an example, we can install it on our system by building as shown in Appendix C.

The program is installed under the default root of the packages tree - `/usr/pkg`. Should this not conform to your tastes, set the `LOCALBASE` variable in your environment, and it will use that value as the root of your packages tree. So, to use `/usr/local`, set `LOCALBASE=/usr/local` in your environment. Please note that you should use a directory which is dedicated to packages and not shared with other programs (i.e., do not try and use `LOCALBASE=/usr`). Also, you should not try to add any of your own files or directories (such as `src/`, `obj/`, or `pkgsrc/`) below the `LOCALBASE` tree. This is to prevent possible conflicts between programs and other files installed by the package system and whatever else may have been installed there.

Some packages look in `/etc/mk.conf` to alter some configuration options at build time. Have a look at `pkgsrc/mk/defaults/mk.conf` to get an overview of what will be set there by default. Environment variables such as `LOCALBASE` can be set in `/etc/mk.conf` to save having to remember to set them each time you want to use pkgsrc.

Occasionally, people want to “look under the covers” to see what is going on when a package is building or being installed. This may be for debugging purposes, or out of simple curiosity. A number of utility values have been added to help with this.

1. If you invoke the make(1) command with `PKG_DEBUG_LEVEL=2`, then a huge amount of information will be displayed. For example,

   ```
   make patch PKG_DEBUG_LEVEL=2
   ```

   will show all the commands that are invoked, up to and including the “patch” stage.

2. If you want to know the value of a certain make(1) definition, then the `VARNAME` definition should be used, in conjunction with the show-var target. e.g. to show the expansion of the make(1) variable `LOCALBASE`:

   ```
   % make show-var VARNAME=LOCALBASE
   /usr/pkg
   %
   ```

If you want to install a binary package that you’ve either created yourself (see next section), that you put into pkgsrc/packages manually or that is located on a remote FTP server, you can use the "bin-install" target. This target will install a binary package - if available - via pkg_add(1), else do a **make package**.

The list of remote FTP sites searched is kept in the variable BINPKG_SITES, which defaults to ftp.NetBSD.org. Any flags that should be added to pkg_add(1) can be put into BIN_INSTALL_FLAGS. See pkgsrc/mk/defaults/mk.conf for more details.

A final word of warning: If you set up a system that has a non-standard setting for LOCALBASE, be sure to set that before any packages are installed, as you can not use several directories for the same purpose. Doing so will result in pkgsrc not being able to properly detect your installed packages, and fail miserably. Note also that precompiled binary packages are usually built with the default LOCALBASE of /usr/pkg, and that you should *not* install any if you use a non-standard LOCALBASE.

33.2.4 Selecting the compiler

By default, pkgsrc will use GCC to build packages. This may be overridden by setting the following variables in /etc/mk.conf:

PKGSRC_COMPILER:

This is a list of values specifying the chain of compilers to invoke when building packages. Valid values are:

- distcc: distributed C/C++ (chainable)
- ccache: compiler cache (chainable)
- gcc: GNU C/C++ Compiler
- mipspro: Silicon Graphics, Inc. MIPSpro (n32/n64)
- mipspro: Silicon Graphics, Inc. MIPSpro (o32)
- sunpro: Microsystems, Inc. WorkShip/Forte/Sun ONE Studio

The default is "gcc". You can use ccache and/or distcc with an appropriate PKGSRC_COMPILER setting, e.g. "ccache gcc". This variable should always be terminated with a value for a real compiler.

GCC_REQD:

This specifies the minimum version of GCC to use when building packages. If the system GCC doesn't satisfy this requirement, then pkgsrc will build and install one of the GCC packages to use instead.

Chapter 34
Configuring pkgsrc

34.1 General configuration

In this section, you can find some variables that apply to all pkgsrc packages. The preferred method of setting these variables is by setting them in `/etc/mk.conf`.

- `LOCALBASE`: Where packages will be installed. The default is `/usr/pkg`. Do not mix binary packages with different `LOCALBASE`s!
- `CROSSBASE`: Where "cross" category packages will be installed. The default is `${LOCALBASE}/cross`.
- `X11BASE`: Where X11 is installed on the system. The default is `/usr/X11R6`.
- `DISTDIR`: Where to store the downloaded copies of the original source distributions used for building pkgsrc packages. The default is `${PKGSRCDIR}/distfiles`.
- `MASTER_SITE_OVERRIDE`: If set, override the packages' `MASTER_SITES` with this value.
- `MASTER_SITE_BACKUP`: Backup location(s) for distribution files and patch files if not found locally or in `${MASTER_SITES}` or `${PATCH_SITES}` respectively. The defaults are `ftp://ftp.NetBSD.org/pub/NetBSD/packages/distfiles/${DIST_SUBDIR}/` and `ftp://ftp.freebsd.org/pub/FreeBSD/distfiles/${DIST_SUBDIR}/`.
- `BINPKG_SITES`: List of sites carrying binary pkgs.

34.2 Variables affecting the build process

XXX

- `PACKAGES`: The top level directory for the binary packages. The default is `${PKGSRCDIR}/packages`.
- `WRKOBJDIR`: The top level directory where, if defined, the separate working directories will get created, and symbolically linked to from `${WRKDIR}` (see below). This is useful for building packages on several architectures, then `${PKGSRCDIR}` can be NFS-mounted while `${WRKOBJDIR}` is local to every architecture. (It should be noted that `PKGSRCDIR` should not be set by the user — it is an internal definition which refers to the root of the pkgsrc tree. It is possible to have many pkgsrc tree instances.)
- `LOCALPATCHES`: Directory for local patches that aren't part of pkgsrc. See Section 37.3 for more information. *`rel`* and *`arch`* are replaced with OS release ("2.0", etc.) and architecture ("mipsel", etc.).

- PKGMAKECONF: Location of the mk.conf file used by a package's BSD-style Makefile. If this is not set, MAKECONF is set to /dev/null to avoid picking up settings used by builds in /usr/src.

34.3 Developer/advanced settings

XXX

- PKG_DEVELOPER: Run some sanity checks that package developers want:
  - make sure patches apply with zero fuzz
  - run check-shlibs to see that all binaries will find their shared libs.
- PKG_DEBUG_LEVEL: The level of debugging output which is displayed whilst making and installing the package. The default value for this is 0, which will not display the commands as they are executed (normal, default, quiet operation); the value 1 will display all shell commands before their invocation, and the value 2 will display both the shell commands before their invocation, and their actual execution progress with **set -x** will be displayed.

34.4 Selecting Build Options

Some packages have build time options, usually to select between different dependencies, enable optional support for big dependencies or enable experimental features.

To see which options, if any, a package supports, and which options are mutually exclusive, run **make show-options**, for example:

```
The following options are supported by this package:
ssl      Enable SSL support.
Exactly one of the following gecko options is required:
firefox  Use firefox as gecko rendering engine.
mozilla  Use mozilla as gecko rendering engine.
At most one of the following database options may be selected:
mysql    Enable support for MySQL database.
pgsql    Enable support for PostgreSQL database.

These options are enabled by default: firefox
These options are currently enabled: mozilla ssl
```

The following variables can be defined in /etc/mk.conf to select which options to enable for a package: PKG_DEFAULT_OPTIONS, which can be used to select or disable options for all packages that support them, and PKG_OPTIONS.*pkgbase*, which can be used to select or disable options specifically for package *pkgbase*. Options listed in these variables are selected, options preceded by "-" are disabled.

The following settings are consulted in the order given, and the last setting that selects or disables an option is used:

1. the default options as suggested by the package maintainer
2. the options implied by the settings of legacy variables (see below)
3. `PKG_DEFAULT_OPTIONS`
4. `PKG_OPTIONS.`*`pkgbase`*

For groups of mutually exclusive options, the last option selected is used, all others are automatically disabled. If an option of the group is explicitly disabled, the previously selected option, if any, is used. It is an error if no option from a required group of options is selected, and building the package will fail.

Before the options framework was introduced, build options were selected by setting a variable in `/etc/mk.conf` for each option. To ease transition to the options framework for the user, these legacy variables are converted to the appropriate options setting automatically. A warning is issued to prompt the user to update `/etc/mk.conf` to use the options framework directly. Support for these legacy variables will be removed eventually.

Chapter 35
Creating binary packages

35.1 Building a single binary package

Once you have built and installed a package, you can create a *binary package* which can be installed on another system with pkg_add(1). This saves having to build the same package on a group of hosts and wasting CPU time. It also provides a simple means for others to install your package, should you distribute it.

To create a binary package, change into the appropriate directory in pkgsrc, and run **make package**:

```
# cd misc/figlet
# make package
```

This will build and install your package (if not already done), and then build a binary package from what was installed. You can then use the **pkg_*** tools to manipulate it. Binary packages are created by default in `/usr/pkgsrc/packages`, in the form of a gzipped tar file. See Section C.2 for a continuation of the above `misc/figlet` example.

See Chapter 46 for information on how to submit such a binary package.

35.2 Settings for creation of binary packages

See Section 43.3.

35.3 Doing a bulk build of all packages

If you want to get a full set of precompiled binary packages, this section describes how to get them. Beware that the bulk build will remove all currently installed packages from your system! Having an FTP server configured either on the machine doing the bulk builds or on a nearby NFS server can help to make the packages available to everyone. See ftpd(8) for more information. If you use a remote NFS server's storage, be sure to not actually compile on NFS storage, as this slows things down a lot.

35.3.1 Configuration

35.3.1.1 /etc/mk.conf

You may want to set variables in `/etc/mk.conf`. Look at `pkgsrc/mk/defaults/mk.conf` for details of the default settings. You will want to ensure that `ACCEPTABLE_LICENSES` meet your local policy. As used in this example, `_ACCEPTABLE=yes` accepts *all* licenses.

```
PACKAGES?=      ${_PKGSRCDIR}/packages/${MACHINE_ARCH}
WRKOBJDIR?=     /usr/tmp/pkgsrc   # build here instead of in pkgsrc
BSDSRCDIR=      /usr/src
BSDXSRCDIR=     /usr/xsrc         # for x11/xservers
OBJHOSTNAME?=   yes               # use work.`hostname`
FAILOVER_FETCH= yes               # insist on the correct checksum
PKG_DEVELOPER?= yes
_ACCEPTABLE=    yes
```

35.3.1.2 `build.conf`

In `pkgsrc/mk/bulk`, copy `build.conf-example` to `build.conf` and edit it, following the comments in that file. This is the config file that determines where log files are generated after the build, where to mail the build report to, where your pkgsrc tree is located and the user to which user to su(8) to do a **cvs update**.

35.3.1.3 `pre-build.local`

It is possible to configure the bulk build to perform certain site-specific tasks at the end of the pre-build stage. If the file `pre-build.local` exists in `/usr/pkgsrc/mk/bulk`, it will be executed (as a sh(1) script) at the end of the usual pre-build stage. An example use of `pre-build.local` is to have the line:

```
# echo "I do not have enough disk space to build this pig." \
    > pkgsrc/misc/openoffice/$BROKENF
```

to prevent the system from trying to build a particular package which requires nearly 3 GB of disk space.

35.3.2 Other environmental considerations

As `/usr/pkg` will be completely deleted at the start of bulk builds, make sure your login shell is placed somewhere else. Either drop it into `/usr/local/bin` (and adjust your login shell in the passwd file), or (re-)install it via pkg_add(1) from `/etc/rc.local`, so you can login after a reboot (remember that your current process won't die if the package is removed, you just can't start any new instances of the shell any more). Also, if you use NetBSD earlier than 1.5, or you still want to use the pkgsrc version of ssh for some reason, be sure to install ssh before starting it from `rc.local`:

```
( cd /usr/pkgsrc/security/ssh ; make bulk-install )
if [ -f /usr/pkg/etc/rc.d/sshd ]; then
    /usr/pkg/etc/rc.d/sshd
fi
```

Not doing so will result in you being not able to log in via ssh after the bulk build is finished or if the machine gets rebooted or crashes. You have been warned! :)

35.3.3 Operation

Make sure you don't need any of the packages still installed.

> **Warning**
>
> During the bulk build, *all packages will be removed!*

Be sure to remove all other things that might interfere with builds, like some libs installed in `/usr/local`, etc. then become root and type:

```
# cd /usr/pkgsrc
# sh mk/bulk/build
```

If for some reason your last build didn't complete (power failure, system panic, ...), you can continue it by running:

```
# sh mk/bulk/build restart
```

At the end of the bulk build, you will get a summary via mail, and find build logs in the directory specified by `FTP` in the `build.conf` file.

35.3.4 What it does

The bulk builds consist of three steps:

1. pre-build

   The script updates your pkgsrc tree via (anon)cvs, then cleans out any broken distfiles, and removes all packages installed.

2. the bulk build

   This is basically "make bulk-package" with an optimised order in which packages will be built. Packages that don't require other packages will be built first, and packages with many dependencies will be built later.

3. post-build

   Generates a report that's placed in the directory specified in the `build.conf` file named `broken.html`, a short version of that report will also be mailed to the build's admin.

During the build, a list of broken packages will be compiled in `/usr/pkgsrc/.broken` (or `.../.broken.${MACHINE}` if `OBJMACHINE` is set), individual build logs of broken builds can be found in the package's directory. These files are used by the bulk-targets to mark broken builds to not waste time trying to rebuild them, and they can be used to debug these broken package builds later.

35.3.5 Disk space requirements

Currently, roughly the following requirements are valid for NetBSD 2.0/i386:

- 10 GB - distfiles (NFS ok)
- 8 GB - full set of all binaries (NFS ok)
- 5 GB - temp space for compiling (local disk recommended)

Note that all pkgs will be de-installed as soon as they are turned into a binary package, and that sources are removed, so there is no excessively huge demand to disk space. Afterwards, if the package is needed again, it will be installed via pkg_add(1) instead of building again, so there are no cycles wasted by recompiling.

35.3.6 Setting up a sandbox for chrooted builds

If you don't want all the packages nuked from a machine (rendering it useless for anything but pkg compiling), there is the possibility of doing the package bulk build inside a chroot environment.

The first step is to set up a chroot sandbox, e.g. `/usr/sandbox`. This can be done by using null mounts, or manually.

There is a shell script called `pkgsrc/mk/bulk/mksandbox` which will set up the sandbox environment using null mounts. It will also create a script called `sandbox` in the root of the sandbox environment, which will allow the null mounts to be activated using the **sandbox mount** command and deactivated using the **sandbox umount** command.

To set up a sandbox environment by hand, after extracting all the sets from a NetBSD installation or doing a **make distribution DESTDIR=/usr/sandbox** in `/usr/src/etc`, be sure the following items are present and properly configured:

1. Kernel

   ```
   # cp /netbsd /usr/sandbox
   ```

2. `/dev/*`

   ```
   # cd /usr/sandbox/dev ; sh MAKEDEV all
   ```

3. `/etc/resolv.conf` (for `security/smtpd` and mail):

   ```
   # cp /etc/resolv.conf /usr/sandbox/etc
   ```

4. Working(!) mail config (hostname, sendmail.cf):

   ```
   # cp /etc/mail/sendmail.cf /usr/sandbox/etc/mail
   ```

5. `/etc/localtime` (for `security/smtpd`):

   ```
   # ln -sf /usr/share/zoneinfo/UTC /usr/sandbox/etc/localtime
   ```

6. `/usr/src` (system sources, for `sysutils/aperture`, `net/ppp-mppe`):

   ```
   # ln -s ../disk1/cvs .
   # ln -s cvs/src-2.0 src
   ```

7. Create `/var/db/pkg` (not part of default install):

   ```
   # mkdir /usr/sandbox/var/db/pkg
   ```

8. Create `/usr/pkg` (not part of default install):

   ```
   # mkdir /usr/sandbox/usr/pkg
   ```

9. Checkout pkgsrc via cvs into /usr/sandbox/usr/pkgsrc:

   ```
   # cd /usr/sandbox/usr
   # cvs -d anoncvs@anoncvs.NetBSD.org:/cvsroot checkout -d -P pkgsrc
   ```

   Do not mount/link this to the copy of your pkgsrc tree you do development in, as this will likely cause problems!

10. Make /usr/sandbox/usr/pkgsrc/packages and .../distfiles point somewhere appropriate. NFS- and/or nullfs-mounts may come in handy!

11. Edit /etc/mk.conf, see Section 35.3.1.1.

12. Adjust mk/bulk/build.conf to suit your needs.

13. If you have set CVS_USER in build.conf, make sure that account exists and can do a **cvs ${CVS_FLAGS} update** properly!

When the chroot sandbox is set up, you can start the build with the following steps:

```
# cd /usr/sandbox/usr/pkgsrc
# sh mk/bulk/do-sandbox-build
```

This will just jump inside the sandbox and start building. At the end of the build, mail will be sent with the results of the build. Created binary pkgs will be in /usr/sandbox/usr/pkgsrc/packages (wherever that points/mounts to/from).

35.3.7 Building a partial set of packages

In addition to building a complete set of all packages in pkgsrc, the pkgsrc/mk/bulk/build script may be used to build a subset of the packages contained in pkgsrc. By setting SPECIFIC_PKGS in /etc/mk.conf, the variables

- SITE_SPECIFIC_PKGS
- HOST_SPECIFIC_PKGS
- GROUP_SPECIFIC_PKGS
- USER_SPECIFIC_PKGS

will define the set of packages which should be built. The bulk build code will also include any packages which are needed as dependencies for the explicitly listed packages.

One use of this is to do a bulk build with SPECIFIC_PKGS in a chroot sandbox periodically to have a complete set of the binary packages needed for your site available without the overhead of building extra packages that are not needed.

35.3.8 Uploading results of a bulk build

This section describes how pkgsrc developers can upload binary pkgs built by bulk builds to ftp.NetBSD.org.

If you would like to automatically create checksum files for the binary packages you intend to upload, remember to set MKSUMS=yes in your mk/bulk/build.conf.

If you would like to PGP sign the checksum files (highly recommended!), remember to set `SIGN_AS=username@NetBSD.org` in your `mk/bulk/build.conf`. This will prompt you for your GPG password to sign the files before uploading everything.

Then, make sure that you have `RSYNC_DST` set properly in your `mk/bulk/build.conf` file, i.e. adjust it to something like one of the following:

```
RSYNC_DST=$CVS_USER@ftp.NetBSD.org:/pub/NetBSD/packages/pkgsrc-200xQy/NetBSD-a.b.c/arch/uplo
```

Please use appropriate values for "pkgsrc-200xQy", "NetBSD-a.b.c" and "arch" here. If your login on ftp.NetBSD.org is different from `CVS_USER`, write your login directly into the variable, e.g. my local account is "feyrer", but for my login "hubertf", I use:

```
RSYNC_DST=hubertf@ftp.NetBSD.org:/pub/NetBSD/packages/pkgsrc-200xQy/NetBSD-a.b.c/arch/upload
```

A separate `upload` directory is used here to allow "closing" the directory during upload. To do so, run the following command on ftp.NetBSD.org next:

```
nbftp% mkdir -p -m 750 /pub/NetBSD/packages/pkgsrc-200xQy/NetBSD-a.b.c/arch/upload
```

Please note that `/pub/NetBSD/packages` is only appropriate for packages for the NetBSD operating system. Binary packages for other operating systems should go into `/pub/pkgsrc`.

Before uploading the binary pkgs, ssh authentication needs to be set up. This example shows how to set up temporary keys for the root account *inside the sandbox* (assuming that no keys should be present there usually):

```
# chroot /usr/sandbox
chroot-# rm $HOME/.ssh/id-dsa*
chroot-# ssh-keygen -t dsa
chroot-# cat $HOME/.ssh/id-dsa.pub
```

Now take the output of `id-dsa.pub` and append it to your `~/.ssh/authorized_keys` file on ftp.NetBSD.org. You can remove the key after the upload is done!

Next, test if your ssh connection really works:

```
chroot-# ssh ftp.NetBSD.org date
```

Use "-l yourNetBSDlogin" here as appropriate!

Now after all this works, you can exit the sandbox and start the upload:

```
chroot-# exit
# cd /usr/sandbox/usr/pkgsrc
# sh mk/bulk/do-sandbox-upload
```

The upload process may take quite some time. Use ls(1) or du(1) on the FTP server to monitor progress of the upload. The upload script will take care of not uploading restricted packages and putting vulnerable packages into the `vulnerable` subdirectory.

After the upload has ended, first thing is to revoke ssh access:

```
nbftp% vi ~/.ssh/authorized_keys
Gdd:x!
```

Use whatever is needed to remove the key you've entered before! Last, move the uploaded packages out of the `upload` directory to have them accessible to everyone:

```
nbftp% cd /pub/NetBSD/packages/pkgsrc-200xQy/NetBSD-a.b.c/arch
nbftp% mv upload/* .
nbftp% rmdir upload
nbftp% chmod 755 .
```

35.4 Creating a multiple CD-ROM packages collection

After your pkgsrc bulk-build has completed, you may wish to create a CD-ROM set of the resulting binary packages to assist in installing packages on other machines. The `pkgtools/cdpack` package provides a simple tool for creating the ISO 9660 images. **cdpack** arranges the packages on the CD-ROMs in a way that keeps all the dependencies for a given package on the same CD as that package.

35.4.1 Example of cdpack

Complete documentation for cdpack is found in the cdpack(1) man page. The following short example assumes that the binary packages are left in `/usr/pkgsrc/packages/All` and that sufficient disk space exists in `/u2` to hold the ISO 9660 images.

```
# mkdir /u2/images
# pkg_add /usr/pkgsrc/packages/All/cdpack
# cdpack /usr/pkgsrc/packages/All /u2/images
```

If you wish to include a common set of files (`COPYRIGHT`, `README`, etc.) on each CD in the collection, then you need to create a directory which contains these files. e.g.

```
# mkdir /tmp/common
# echo "This is a README" > /tmp/common/README
# echo "Another file" > /tmp/common/COPYING
# mkdir /tmp/common/bin
# echo "#!/bin/sh" > /tmp/common/bin/myscript
# echo "echo Hello world" >> /tmp/common/bin/myscript
# chmod 755 /tmp/common/bin/myscript
```

Now create the images:

```
# cdpack -x /tmp/common /usr/pkgsrc/packages/All /u2/images
```

Each image will contain `README`, `COPYING`, and `bin/myscript` in their root directories.

Chapter 36
Frequently Asked Questions

This section contains hints, tips & tricks on special things in pkgsrc that we didn't find a better place for in the previous chapters, and it contains items for both pkgsrc users and developers.

36.1 Are there any mailing lists for pkg-related discussion?

The following mailing lists may be of interest to pkgsrc users:

- pkgsrc-bugs (http://www.NetBSD.org/MailingLists/index.html#pkgsrc-bugs): A list where problem reports related to pkgsrc are sent and discussed.
- pkgsrc-bulk (http://www.NetBSD.org/MailingLists/index.html#pkgsrc-bulk): A list where the results of pkgsrc bulk builds are sent and discussed.
- pkgsrc-changes (http://www.NetBSD.org/MailingLists/index.html#pkgsrc-changes): A list where all commit messages to pkgsrc are sent.
- tech-pkg (http://www.NetBSD.org/MailingLists/index.html#tech-pkg): A general discussion list for all things related to pkgsrc.

To subscribe, do:

```
% echo subscribe listname | mail majordomo@NetBSD.org
```

Archives for all these mailing lists are available from http://mail-index.NetBSD.org/.

36.2 Where's the pkgviews documentation?

Pkgviews is tightly integrated with buildlink. You can find a pkgviews User's guide in `pkgsrc/mk/buildlink3/PKGVIEWS_UG`.

36.3 Utilities for package management (pkgtools)

The `pkgsrc/pkgtools` directory pkgtools contains a number of useful utilities for both users and developers of pkgsrc. This section attempts only to make the reader aware of the utilities and when they might be useful, and not to duplicate the documentation that comes with each package.

Utilities used by pkgsrc (automatically installed when needed):

- `pkgtools/x11-links`: Symlinks for use by buildlink.

OS tool augmentation (automatically installed when needed):

- pkgtools/digest: Calculates various kinds of checksums (including SHA1).
- pkgtools/libnbcompat: Compatibility library for pkgsrc tools.
- pkgtools/mtree: Installed on non-BSD systems due to lack of native mtree.
- pkgtools/pkg_install: Up-to-date replacement for /usr/sbin/pkg_install, or for use on operating systems where pkg_install is not present.

Utilities used by pkgsrc (not automatically installed):

- pkgtools/pkg_tarup: Create a binary package from an already-installed package. Used by **make replace** to save the old package.
- pkgtools/dfdisk: Adds extra functionality to pkgsrc, allowing it to fetch distfiles from multiple locations. It currently supports the following methods: multiple CD-ROMs and network FTP/HTTP connections.
- pkgtools/xpkgwedge: Put X11 packages someplace else (enabled by default).
- devel/cpuflags: Determine the best compiler flags to optimise code for your current CPU and compiler.

Utilities for keeping track of installed packages, being up to date, etc:

- pkgtools/pkg_chk: Reports on packages whose installed versions do not match the latest pkgsrc entries.
- pkgtools/pkgdep: Makes dependency graphs of packages, to aid in choosing a strategy for updating.
- pkgtools/pkgdepgraph: Makes graphs from the output of pkgtools/pkgdep (uses graphviz).
- pkgtools/pkglint: The pkglint(1) program checks a pkgsrc entry for errors, lintpkgsrc(1) does various checks on the complete pkgsrc system.
- pkgtools/pkgsurvey: Report what packages you have installed.

Utilities for people maintaining or creating individual packages:

- pkgtools/pkgdiff: Automate making and maintaining patches for a package (includes pkgdiff, pkgvi, mkpatches, etc.).
- pkgtools/rpm2pkg, pkgtools/url2pkg: Aids in converting to pkgsrc.
- pkgtools/gensolpkg: Convert pkgsrc to a Solaris package.

Utilities for people maintaining pkgsrc (or: more obscure pkg utilities)

- pkgtools/pkg_comp: Build packages in a chrooted area.
- pkgtools/libkver: Spoof kernel version for chrooted cross builds.

36.4 How to use pkgsrc as non-root

If you want to use pkgsrc as non-root user, you can set some variables to make pkgsrc work under these conditions. At the very least, you need to set UNPRIVILEGED to "yes"; this will turn on unprivileged mode and set multiple related variables to allow installation of packages as non-root.

In case the defaults are not enough, you may want to tune some other variables used. For example, if the automatic user/group detection leads to incorrect values (or not the ones you would like to use), you can change them by setting UNPRIVILEGED_USER and UNPRIVILEGED_GROUP respectively.

As regards bootstrapping, please note that the **bootstrap** script will ease non-root configuration when given the "--ignore-user-check" flag, as it will choose and use multiple default directories under ~/pkg as the installation targets. These directories can be overriden by the "--prefix" flag provided by the script, as well as some others that allow finer tuning of the tree layout.

36.5 How to resume transfers when fetching distfiles?

By default, resuming transfers in pkgsrc is disabled, but you can enable this feature by adding the option PKG_RESUME_TRANSFERS=YES into /etc/mk.conf. If, during a fetch step, an incomplete distfile is found, pkgsrc will try to resume it.

You can also use a different program than the default ftp(1) by changing the FETCH_CMD variable. Don't forget to set FETCH_RESUME_ARGS and FETCH_OUTPUT_ARGS if you are not using default values.

For example, if you want to use wget to resume downloads, you'll have to use something like:

```
FETCH_CMD=wget
FETCH_BEFORE_ARGS=--passive-ftp
FETCH_RESUME_ARGS=-c
FETCH_OUTPUT_ARGS=-O
```

36.6 How can I install/use XFree86 from pkgsrc?

If you want to use XFree86 from pkgsrc instead of your system's own X11 (/usr/X11R6, /usr/openwin, ...), you will have to add the following line into /etc/mk.conf:

```
X11_TYPE=XFree86
```

36.7 How can I install/use X.org from pkgsrc?

If you want to use X.org from pkgsrc instead of your system's own X11 (/usr/X11R6, /usr/openwin, ...) you will have to add the following line into /etc/mk.conf:

```
X11_TYPE=xorg
```

36.8 How to fetch files from behind a firewall

If you are sitting behind a firewall which does not allow direct connections to Internet hosts (i.e. non-NAT), you may specify the relevant proxy hosts. This is done using an environment variable in the form of a URL, e.g. in Amdahl, the machine "orpheus.amdahl.com" is one of the firewalls, and it uses port 80 as the proxy port number. So the proxy environment variables are:

```
ftp_proxy=ftp://orpheus.amdahl.com:80/
http_proxy=http://orpheus.amdahl.com:80/
```

36.9 How do I tell make fetch to do passive FTP?

This depends on which utility is used to retrieve distfiles. From `bsd.pkg.mk`, `FETCH_CMD` is assigned the first available command from the following list:

- `${LOCALBASE}/bin/ftp`
- `/usr/bin/ftp`

On a default NetBSD installation, this will be `/usr/bin/ftp`, which automatically tries passive connections first, and falls back to active connections if the server refuses to do passive. For the other tools, add the following to your `/etc/mk.conf` file: `PASSIVE_FETCH=1`.

Having that option present will prevent `/usr/bin/ftp` from falling back to active transfers.

36.10 How to fetch all distfiles at once

You would like to download all the distfiles in a single batch from work or university, where you can't run a **make fetch**. There is an archive of distfiles on ftp.NetBSD.org (ftp://ftp.NetBSD.org/pub/NetBSD/packages/distfiles/), but downloading the entire directory may not be appropriate.

The answer here is to do a **make fetch-list** in `/usr/pkgsrc` or one of its subdirectories, carry the resulting list to your machine at work/school and use it there. If you don't have a NetBSD-compatible ftp(1) (like lukemftp) at work, don't forget to set `FETCH_CMD` to something that fetches a URL:

At home:

```
% cd /usr/pkgsrc
% make fetch-list FETCH_CMD=wget DISTDIR=/tmp/distfiles >/tmp/fetch.sh
% scp /tmp/fetch.sh work:/tmp
```

At work:

```
% sh /tmp/fetch.sh
```

then tar up `/tmp/distfiles` and take it home.

If you have a machine running NetBSD, and you want to get *all* distfiles (even ones that aren't for your machine architecture), you can do so by using the above-mentioned **make fetch-list** approach, or fetch the distfiles directly by running:

```
% make mirror-distfiles
```

If you even decide to ignore NO_{SRC,BIN}_ON_{FTP,CDROM}, then you can get everything by running:

```
% make fetch NO_SKIP=yes
```

36.11 What does “Don’t know how to make /usr/share/tmac/tmac.andoc” mean?

When compiling the pkgtools/pkg_install package, you get the error from make that it doesn’t know how to make /usr/share/tmac/tmac.andoc? This indicates that you don’t have installed the “text” set (nroff, ...) from the NetBSD base distribution on your machine. It is recommended to do that to format man pages.

In the case of the pkgtools/pkg_install package, you can get away with setting NOMAN=YES either in the environment or in /etc/mk.conf.

36.12 What does “Could not find bsd.own.mk” mean?

You didn’t install the compiler set, comp.tgz, when you installed your NetBSD machine. Please get and install it, by extracting it in /:

```
# cd /
# tar --unlink -zxvpf .../comp.tgz
```

comp.tgz is part of every NetBSD release. Get the one that corresponds to your release (determine via **uname -r**).

36.13 Using ’sudo’ with pkgsrc

When installing packages as non-root user and using the just-in-time su(1) feature of pkgsrc, it can become annoying to type in the root password for each required package installed. To avoid this, the sudo package can be used, which does password caching over a limited time. To use it, install sudo (either as binary package or from security/sudo) and then put the following into your /etc/mk.conf:

```
.if exists(${LOCALBASE}/bin/sudo)
SU_CMD=${LOCALBASE}/bin/sudo /bin/sh -c
.endif
```

36.14 How do I change the location of configuration files?

As the system administrator, you can choose where configuration files are installed. The default settings make all these files go into ${PREFIX}/etc or some of its subdirectories; this may be suboptimal

depending on your expectations (e.g., a read-only, NFS-exported PREFIX with a need of per-machine configuration of the provided packages).

In order to change the defaults, you can modify the PKG_SYSCONFBASE variable (in /etc/mk.conf) to point to your preferred configuration directory; some common examples include /etc or /etc/pkg.

Furthermore, you can change this value on a per-package basis by setting the PKG_SYSCONFDIR.${PKG_SYSCONFVAR} variable. PKG_SYSCONFVAR's value usually matches the name of the package you would like to modify, that is, the contents of PKGBASE.

Note that after changing these settings, you must rebuild and reinstall any affected packages.

36.15 Automated security checks

Please be aware that there can often be bugs in third-party software, and some of these bugs can leave a machine vulnerable to exploitation by attackers. In an effort to lessen the exposure, the NetBSD packages team maintains a database of known-exploits to packages which have at one time been included in pkgsrc. The database can be downloaded automatically, and a security audit of all packages installed on a system can take place. To do this, install the security/audit-packages package. It has two components:

1. **download-vulnerability-list**, an easy way to download a list of the security vulnerabilities information. This list is kept up to date by the NetBSD security officer and the NetBSD packages team, and is distributed from the NetBSD ftp server:

   ftp://ftp.NetBSD.org/pub/NetBSD/packages/distfiles/pkg-vulnerabilities

2. **audit-packages**, an easy way to audit the current machine, checking each vulnerability which is known. If a vulnerable package is installed, it will be shown by output to stdout, including a description of the type of vulnerability, and a URL containing more information.

Use of the security/audit-packages package is strongly recommended! After "audit-packages" is installed, please read the package's message, which you can get by running **pkg_info -D audit-package**.

VII. The pkgsrc developer’s guide

Chapter 37
Package components - files, directories and contents

Whenever you're preparing a package, there are a number of files involved which are described in the following sections.

37.1 `Makefile`

Building, installation and creation of a binary package are all controlled by the package's `Makefile`. The `Makefile` describes various things about a package, for example from where to get it, how to configure, build, and install it.

A package `Makefile` contains several sections that describe the package.

In the first section there are the following variables, which should appear exactly in the order given here.

- `DISTNAME` is the basename of the distribution file to be downloaded from the package's website.
- `PKGNAME` is the name of the package, as used by pkgsrc. You only need to provide it if it differs from DISTNAME. Usually it is the directory name together with the version number. It must match the regular expression `^[A-Za-z0-9][A-Za-z0-9_.+]*$`, that is, it starts with a letter or digit, and contains only letters, digits, dashes, underscores, dots and plus signs.
- `CATEGORIES` is a list of categories which the package fits in. You can choose any of the top-level directories of pkgsrc for it.

  Currently the following values are available for `CATEGORIES`. If more than one is used, they need to be separated by spaces:

  ```
  archivers     cross         geography     meta-pkgs     security
  audio         databases     graphics      misc          shells
  benchmarks    devel         ham           multimedia    sysutils
  biology       editors       inputmethod   net           textproc
  cad           emulators     lang          news          time
  chat          finance       mail          parallel      wm
  comms         fonts         math          pkgtools      www
  converters    games         mbone         print         x11
  ```

- `MASTER_SITES` is a list of URLs where the distribution files can be downloaded. Each URL must end with a slash.

  The `MASTER_SITES` may make use of the following predefined sites:

  ```
  ${MASTER_SITE_APACHE}
  ${MASTER_SITE_BACKUP}
  ${MASTER_SITE_CYGWIN}
  ```

```
${MASTER_SITE_DEBIAN}
${MASTER_SITE_FREEBSD}
${MASTER_SITE_FREEBSD_LOCAL}
${MASTER_SITE_GNOME}
${MASTER_SITE_GNU}
${MASTER_SITE_GNUSTEP}
${MASTER_SITE_IFARCHIVE}
${MASTER_SITE_MOZILLA}
${MASTER_SITE_OPENOFFICE}
${MASTER_SITE_PERL_CPAN}
${MASTER_SITE_R_CRAN}
${MASTER_SITE_SOURCEFORGE}
${MASTER_SITE_SUNSITE}
${MASTER_SITE_SUSE}
${MASTER_SITE_TEX_CTAN}
${MASTER_SITE_XCONTRIB}
${MASTER_SITE_XEMACS}
```

If one of these predefined sites is chosen, you may want to specify a subdirectory of that site. Since these macros may expand to more than one actual site, you *must* use the following construct to specify a subdirectory:

```
${MASTER_SITE_GNU:=subdirectory/name/}
${MASTER_SITE_SOURCEFORGE:=project_name/}
```

Note the trailing slash after the subdirectory name.

If the package has multiple DISTFILES or multiple PATCHFILES from different sites, set SITES_foo to a list of URIs where file "foo" may be found. "foo" includes the suffix, e.g.:

```
DISTFILES=      ${DISTNAME}${EXTRACT_SUFX}
DISTFILES+=     foo-file.tar.gz
SITES_foo-file.tar.gz=http://www.somewhere.com/somehow/ \
        http://www.somewhereelse.com/mirror/somehow/
```

- DISTFILES: Name(s) of archive file(s) containing distribution. The default is ${DISTNAME}${EXTRACT_SUFX}. Should only be set if you have more than one distfile.

  Note that the normal default setting of DISTFILES must be made explicit if you want to add to it (rather than replace it), as you usually would.

- EXTRACT_SUFX: Suffix of the distribution file, will be appended to DISTNAME. Defaults to .tar.gz.

The second section contains information about separately downloaded patches, if any.

- PATCHFILES: Name(s) of additional files that contain distribution patches. There is no default. pkgsrc will look for them at PATCH_SITES. They will automatically be uncompressed before patching if the names end with .gz or .Z.
- PATCH_SITES: Primary location(s) for distribution patch files (see PATCHFILES below) if not found locally.

The third section contains the following variables.

- MAINTAINER is the email address of the contact person for this package. The person who feels responsible for this package, and who is most likely to look at problems or questions regarding this package which have been reported with send-pr(1). The right person to contact before making major changes to the package. When packaging a new program, set MAINTAINER to yourself. If you really can't maintain the package for future updates, set it to <tech-pkg@NetBSD.org>.
- HOMEPAGE is a URL where users can find more information about the package.
- COMMENT is a one-line description of the package (should not include the package name).

Other variables that affect the build:

- WRKSRC: The subdirectory of ${WRKDIR} to which the distribution actually unpacks. The default is ${WRKDIR}/${DISTNAME}. The value of WRKSRC should be set explicitly if the package does not follow standard conventions and include the package's name as a subdirectory.

Please pay attention to the following gotchas:

- Add MANCOMPRESSED if man pages are installed in compressed form by the package; see comment in bsd.pkg.mk.
- Replace /usr/local with "${PREFIX}" in all files (see patches, below).
- If the package installs any info files, see Section 44.5.7.

37.2 distinfo

Most important, the mandatory message digest, or checksum, of all the distfiles needed for the package to compile, confirming they match the original file distributed by the author. This ensures that the distfile retrieved from the Internet has not been corrupted during transfer or altered by a malign force to introduce a security hole. It is generated using the **make makesum** command. The digest algorithm used was, at one stage, md5, but that was felt lacking compared to sha1, and so sha1 is now the default algorithm. The distfile size is also generated and stored in new distinfo files. The pkgtools/digest utility calculates all of the digests in the distinfo file, and it provides various different algorithms. At the current time, the algorithms provided are: *md5*, *rmd160*, *sha1*, *sha256*, *sha384* and *sha512*.

Some packages have different sets of distfiles on a per architecture basis, for example www/navigator). These are kept in the same distinfo file and care should be taken when upgrading such a package to ensure distfile information is not lost.

The message digest/checksum for all the official patches found in the patches/ directory (see Section 37.3) for the package is also stored in the distinfo file. This is a message digest/checksum of all lines in the patch file except the NetBSD RCS Id. This file is generated by invoking **make makepatchsum** (or **make mps** if you're in a hurry).

37.3 patches/*

This directory contains files that are used by the patch(1) command to modify the sources as distributed in the distribution file into a form that will compile and run perfectly on NetBSD. The files are applied successively in alphabetic order (as returned by a shell "patches/patch-*" glob expansion), so `patch-aa` is applied before `patch-ab`, etc.

The `patch-*` files should be in **diff -bu** format, and apply without a fuzz to avoid problems. (To force patches to apply with fuzz you can set `PATCH_FUZZ_FACTOR=-F2`). Furthermore, do not put changes for more than one file into a single patch file, as this will make future modifications more difficult.

Similar, a file should be patched at most once, not several times by several different patches. If a file needs several patches, they should be combined into one file.

One important thing to mention is to pay attention that no RCS IDs get stored in the patch files, as these will cause problems when later checked into the NetBSD CVS tree. Use the **pkgdiff** from the `pkgtools/pkgdiff` package to avoid these problems.

For even more automation, we recommend using **mkpatches** from the same package to make a whole set of patches. You just have to backup files before you edit them to `filename.orig`, e.g. with **cp -p filename filename.orig** or, easier, by using **pkgvi** again from the same package. If you upgrade a package this way, you can easily compare the new set of patches with the previously existing one with **patchdiff**.

When you have finished a package, remember to generate the checksums for the patch files by using the **make makepatchsum** command, see Section 37.2.

When adding a patch that corrects a problem in the distfile (rather than e.g. enforcing pkgsrc's view of where man pages should go), send the patch as a bug report to the maintainer. This benefits non-pkgsrc users of the package, and usually enables removing the patch in future version.

Patch files that are distributed by the author or other maintainers can be listed in `$PATCHFILES`.

If it is desired to store any patches that should not be committed into pkgsrc, they can be kept outside the pkgsrc tree in the `$LOCALPATCHES` directory. The directory tree there is expected to have the same "category/package" structure as pkgsrc, and patches are expected to be stored inside these dirs (also known as `$LOCALPATCHES/$PKGPATH`). For example, if you want to keep a private patch for `pkgsrc/graphics/png`, keep it in `$LOCALPATCHES/graphics/png/mypatch`. All files in the named directory are expected to be patch files, and *they are applied after pkgsrc patches are applied*.

37.4 Other mandatory files

`DESCR`

A multi-line description of the piece of software. This should include any credits where they are due. Please bear in mind that others do not share your sense of humour (or spelling idiosyncrasies), and that others will read everything that you write here.

`PLIST`

This file governs the files that are installed on your system: all the binaries, manual pages, etc. There are other directives which may be entered in this file, to control the creation and deletion of directories, and the location of inserted files. See Chapter 39 for more information.

37.5 Optional files

INSTALL

This shell script is invoked twice by pkg_add(1). First time after package extraction and before files are moved in place, the second time after the files to install are moved in place. This can be used to do any custom procedures not possible with @exec commands in PLIST. See pkg_add(1) and pkg_create(1) for more information.

DEINSTALL

This script is executed before and after any files are removed. It is this script's responsibility to clean up any additional messy details around the package's installation, since all pkg_delete knows is how to delete the files created in the original distribution. See pkg_delete(1) and pkg_create(1) for more information.

MESSAGE

Display this file after installation of the package. Useful for things like legal notices on almost-free software and hints for updating config files after installing modules for apache, PHP etc. Please note that you can modify variables in it easily by using MESSAGE_SUBST in the package's Makefile:

```
MESSAGE_SUBST+=  SOMEVAR="somevalue"
```

replaces "${SOMEVAR}" with "somevalue" in MESSAGE.

37.6 work*

When you type **make**, the distribution files are unpacked into the directory denoted by WRKDIR. It can be removed by running **make clean**. Besides the sources, this directory is also used to keep various timestamp files. The directory gets *removed completely* on clean. The default is ${.CURDIR}/work or ${.CURDIR}/work.${MACHINE_ARCH} if OBJMACHINE is set.

If a package doesn't create a subdirectory for itself (like GNU software does, for instance), but extracts itself in the current directory, you should set WRKSRC accordingly, e.g. editors/sam again, but the quick answer is:

```
WRKSRC=              ${WRKDIR}
```

If a package doesn't create a subdir with the name of DISTNAME but some different name, set WRKSRC to point to the proper name in ${WRKDIR}. See lang/tcl and x11/tk for examples, and here is another one:

```
WRKSRC=          ${WRKDIR}/${DISTNAME}/unix
```

The name of the working directory created by pkgsrc is work by default. If the same pkgsrc tree should be used on several different platforms, the variable OBJMACHINE can be set in /etc/mk.conf to attach the platform to the directory name, e.g. work.i386 or work.sparc.

37.7 `files/*`

If you have any files that you wish to be placed in the package prior to configuration or building, you could place these files here and use a "${CP}" command in the "pre-configure" target to achieve this. Alternatively, you could simply diff the file against `/dev/null` and use the patch mechanism to manage the creation of this file.

Chapter 38
Programming in `Makefiles`

Pkgsrc consists of many `Makefile` fragments, each of which forms a well-defined part of the pkgsrc system. Using the make(1) system as a programming language for a big system like pkgsrc requires some discipline to keep the code correct and understandable.

The basic ingredients for `Makefile` programming are variables (which are actually macros) and shell commands. Among these shell commands may even be more complex ones like awk(1) programs. To make sure that every shell command runs as intended it is necessary to quote all variables correctly when they are used.

This chapter describes some patterns, that appear quite often in `Makefile`s, including the pitfalls that come along with them.

38.1 `Makefile` variables

`Makefile` variables contain strings that can be processed using the five operators "=", "+=", "?=", ":=", and "!=", which are described in the make(1) man page.

When a variable's value is parsed from a `Makefile`, the hash character "#" and the backslash character "\" are handled specially. If a backslash is followed by a newline, any whitespace immediately in front of the backslash, the backslash, the newline, and any whitespace immediately behind the newline are replaced with a single space. A backspace character and an immediately following hash character are replaced with a single hash character. Otherwise, the backslash is passed as is. In a variable assignment, any hash character that is not preceded by a backslash starts a comment that continues upto the end of the logical line.

Note: Because of this parsing algorithm the only way to create a variable consisting of a single backslash is using the "!=" operator, for example: `BACKSLASH!=echo "\\"`.

So far for defining variables. The other thing you can do with variables is evaluating them. A variable is evaluated when it is part of the right side of the ":=" or the "!=" operator, or directly before executing a shell command which the variable is part of. In all other cases, make(1) performs lazy evaluation, that is, variables are not evaluated until there's no other way. The "modifiers" mentioned in the man page also evaluate the variable.

Some of the modifiers split the string into words and then operate on the words, others operate on the string as a whole. When a string is split into words, it is split as you would expect it from sh(1).

No rule without exception—the **.for** loop does not follow the shell quoting rules but splits at sequences of whitespace.

There are several types of variables that should be handled differently. Strings and two types of lists.

- *Strings* can contain arbitrary characters. Nevertheless, you should restrict yourself to only using printable characters. Examples are `PREFIX` and `COMMENT`.

- *Internal lists* are lists that are never exported to any shell command. Their elements are separated by whitespace. Therefore, the elements themselves cannot have embedded whitespace. Any other characters are allowed. Internal lists can be used in **.for** loops. Examples are DEPENDS and BUILD_DEPENDS.
- *External lists* are lists that may be exported to a shell command. Their elements can contain any characters, including whitespace. That's why they cannot be used in **.for** loops. Examples are DISTFILES and MASTER_SITES.

38.1.1 Naming conventions

- All variable names starting with an underscore are reserved for use by the pkgsrc infrastructure. They shall not be used by package Makefiles.
- In **.for** loops you should use lowercase variable names for the iteration variables.
- All list variables should have a "plural" name, e.g. PKG_OPTIONS or DISTFILES.

38.2 Code snippets

This section presents you with some code snippets you should use in your own code. If you don't find anything appropriate here, you should test your code and add it here.

38.2.1 Adding things to a list

```
STRING=                 foo * bar `date`
INT_LIST=               # empty
ANOTHER_INT_LIST=       apache-[0-9]*:../../www/apache
EXT_LIST=               # empty
ANOTHER_EXT_LIST=       a=b c=d

INT_LIST+=              ${STRING}               # 1
INT_LIST+=              ${ANOTHER_INT_LIST}     # 2
EXT_LIST+=              ${STRING:Q}             # 3
EXT_LIST+=              ${ANOTHER_EXT_LIST}     # 4
```

When you add a string to an external list (example 3), it must be quoted. In all other cases, you must not add a quoting level. You must not merge internal and external lists, unless you are sure that all entries are correctly interpreted in both lists.

38.2.2 Converting an internal list into an external list

```
EXT_LIST=       # empty
.for i in ${INT_LIST}
EXT_LIST+=      ${i:Q}""
.endfor
```

This code converts the internal list INT_LIST into the external list EXT_LIST. As the elements of an internal list are unquoted they must be quoted here. The reason for appending "" is explained below.

38.2.3 Passing variables to a shell command

```
STRING=         foo bar <    > * `date` $$HOME ' "
EXT_LIST=       string=${STRING:Q} x=second\ item

all:
        echo ${STRING}                  # 1
        echo "${STRING}"                # 2
        echo "${STRING:Q}"              # 3
        echo ${STRING:Q}                # 4
        echo x${STRING:Q} | sed 1s,.,,  # 5
        env ${EXT_LIST} /bin/sh -c 'echo "$$string"; echo "$$x"'
```

Example 1 leads to a syntax error in the shell, as the characters are just copied.

Example 2 leads to a syntax error too, and if you leave out the last " character from ${STRING}, date(1) will be executed. The $HOME shell variable would be evaluated, too.

Example 3 outputs each space character preceded by a backslash (or not), depending on the implementation of the echo(1) command.

Example 4 handles correctly every string that does not start with a dash. In that case, the result depends on the implementation of the echo(1) command. As long as you can guarantee that your input does not start with a dash, this form is appropriate.

Example 5 handles even the case of a leading dash correctly.

The EXT_LIST does not need to be quoted because the quoting has already been done when adding elements to the list.

As internal lists shall not be passed to the shell, there is no example for it.

38.2.4 Quoting guideline

There are many possible sources of wrongly quoted variables. This section lists some of the commonly known ones.

- Whenever you use the value of a list, think about what happens to leading or trailing whitespace. If the list is a well-formed shell expression, you can apply the :M* modifier to strip leading and trailing whitespace from each word. The :M operator first splits its argument according to the rules of the shell, and then creates a new list consisting of all words that match the shell glob expression *, that is: all. One class of situations where this is needed is when adding a variable like CPPFLAGS to CONFIGURE_ARGS. If the configure script invokes other configure scripts, it strips the leading and trailing whitespace from the variable and then passes it to the other configure scripts. But these

configure scripts expect the (child) CPPFLAGS variable to be the same as the parent CPPFLAGS. That's why we better pass the CPPFLAGS value properly trimmed. And here is how we do it:

```
CPPFLAGS=               # empty
CPPFLAGS+=              -Wundef -DPREFIX=\"${PREFIX:Q}\"
CPPFLAGS+=              ${MY_CPPFLAGS}

CONFIGURE_ARGS+=        CPPFLAGS=${CPPFLAGS:M*:Q}

all:
        echo x${CPPFLAGS:Q}x            # leading and trailing whitespace
        echo x${CONFIGURE_ARGS}x        # properly trimmed
```

- The example above contains one bug: The ${PREFIX} is a properly quoted shell expression, but there is the C compiler after it, which also expects a properly quoted string (this time in C syntax). The version above is therefore only correct if ${PREFIX} does not have embedded backslashes or double quotes. If you want to allow these, you have to add another layer of quoting to each variable that is used as a C string literal. You cannot use the :Q operator for it, as this operator only works for the shell.
- Whenever a variable can be empty, the :Q operator can have surprising results. Here are two completely different cases which can be solved with the same trick.

```
EMPTY=                  # empty
empty_test:
        for i in a ${EMPTY:Q} c; do \
                echo "$$i"; \
        done

for_test:
.for i in a:\ a:\test.txt
        echo ${i:Q}
        echo "foo"
.endfor
```

The first example will only print two of the three lines we might have expected. This is because ${EMPTY:Q} expands to the empty string, which the shell cannot see. The workaround is to write ${EMPTY:Q}"". This pattern can be often found as ${TEST} -z ${VAR:Q} or as ${TEST} -f ${FNAME:Q} (both of these are wrong).

The second example will only print three lines instead of four. The first line looks like a:\ echo foo. This is because the backslash of the value a:\ is interpreted as a line-continuation by make(1), which makes the second line the arguments of the echo(1) command from the first line. To avoid this, write ${i:Q}"".

38.2.5 Workaround for a bug in BSD Make

The pkgsrc bmake program does not handle the following assignment correctly. In case _othervar_ contains a "-" character, one of the closing braces is included in ${VAR} after this code executes.

```
VAR:= ${VAR:N${_othervar_:C/-//}}
```

For a more complex code snippet and a workaround, see the package `regress/make-quoting`, testcase `bug1`.

Chapter 39
PLIST issues

The PLIST file contains a package's "packing list", i.e. a list of files that belong to the package (relative to the ${PREFIX} directory it's been installed in) plus some additional statements - see the pkg_create(1) man page for a full list. This chapter addresses some issues that need attention when dealing with the PLIST file (or files, see below!).

39.1 RCS ID

Be sure to add a RCS ID line as the first thing in any PLIST file you write:

```
@comment $NetBSD$
```

39.2 Semi-automatic PLIST generation

You can use the **make print-PLIST** command to output a PLIST that matches any new files since the package was extracted. See Section 43.3 for more information on this target.

39.3 Tweaking output of make print-PLIST

If you have used any of the *-dirs packages, as explained in Section 39.8, you may have noticed that **make print-PLIST** outputs a set of @comments instead of real @dirrm lines. You can also do this for specific directories and files, so that the results of that command are very close to reality. This helps *a lot* during the update of packages.

The PRINT_PLIST_AWK variable takes a set of AWK patterns and actions that are used to filter the output of print-PLIST. You can *append* any chunk of AWK scripting you like to it, but be careful with quoting.

For example, to get all files inside the libdata/foo directory removed from the resulting PLIST:

```
PRINT_PLIST_AWK+= /^libdata\/foo/ { next; }
```

And to get all the @dirrm lines referring to a specific (shared) directory converted to @comments:

```
PRINT_PLIST_AWK+= /^@dirrm share\/specific/ { print "@comment " $$0; next; }
```

39.4 Variable substitution in PLIST

A number of variables are substituted automatically in PLISTs when a package is installed on a system. This includes the following variables:

`${MACHINE_ARCH}`, `${MACHINE_GNU_ARCH}`

Some packages like emacs and perl embed information about which architecture they were built on into the pathnames where they install their files. To handle this case, PLIST will be preprocessed before actually used, and the symbol "`${MACHINE_ARCH}`" will be replaced by what **uname -p** gives. The same is done if the string `${MACHINE_GNU_ARCH}` is embedded in PLIST somewhere - use this on packages that have GNU autoconf-created configure scripts.

> **Legacy note:** There used to be a symbol "`$ARCH`" that was replaced by the output of **uname -m**, but that's no longer supported and has been removed.

`${OPSYS}`, `${LOWER_OPSYS}`, `${OS_VERSION}`

Some packages want to embed the OS name and version into some paths. To do this, use these variables in the `PLIST`:

- `${OPSYS}` - output of "**uname -s**"
- `${LOWER_OPSYS}` - lowercase common name (eg. "solaris")
- `${OS_VERSION}` - "**uname -r**"

`${PKGLOCALEDIR}`

Packages that install locale files should list them in the PLIST as "${PKGLOCALEDIR}/locale/de/LC_MESSAGES/..." instead of "share/locale/de/LC_MESSAGES/...". This properly handles the fact that different operating systems expect locale files to be either in `share` or `lib` by default.

For a complete list of values which are replaced by default, please look in `bsd.pkg.mk` (and search for *PLIST_SUBST*).

If you want to change other variables not listed above, you can add variables and their expansions to this variable in the following way, similar to `MESSAGE_SUBST` (see Section 37.5):

```
PLIST_SUBST+=    SOMEVAR="somevalue"
```

This replaces all occurrences of "${SOMEVAR}" in the PLIST with "somevalue".

39.5 Man page compression

Man pages should be installed in compressed form if `MANZ` is set (in `bsd.own.mk`), and uncompressed otherwise. To handle this in the `PLIST` file, the suffix ".gz" is appended/removed automatically for man

pages according to MANZ and MANCOMPRESSED being set or not, see above for details. This modification of the PLIST file is done on a copy of it, not PLIST itself.

39.6 Changing PLIST source with PLIST_SRC

To use one or more files as source for the PLIST used in generating the binary package, set the variable PLIST_SRC to the names of that file(s). The files are later concatenated using cat(1), and order of things is important.

39.7 Platform-specific and differing PLISTs

Some packages decide to install a different set of files based on the operating system being used. These differences can be automatically handled by using the following files:

- PLIST.common
- PLIST.${OPSYS}
- PLIST.${MACHINE_ARCH}
- PLIST.${OPSYS}-${MACHINE_ARCH}
- PLIST.common_end

39.8 Sharing directories between packages

A “shared directory” is a directory where multiple (and unrelated) packages install files. These directories are problematic because you have to add special tricks in the PLIST to conditionally remove them, or have some centralized package handle them.

Within pkgsrc, you’ll find both approaches. If a directory is shared by a few unrelated packages, it’s often not worth to add an extra package to remove it. Therefore, one simply does:

```
@unexec ${RMDIR} %D/path/to/shared/directory 2>/dev/null || ${TRUE}
```

in the PLISTs of all affected packages, instead of the regular "@dirrm" line.

However, if the directory is shared across many packages, two different solutions are available:

1. If the packages have a common dependency, the directory can be removed in that. For example, see textproc/scrollkeeper, which removes the shared directory share/omf.
2. If the packages using the directory are not related at all (they have no common dependencies), a *-dirs package is used.

From now on, we’ll discuss the second solution. To get an idea of the *-dirs packages available, issue:

```
% cd .../pkgsrc
% ls -d */*-dirs
```

Their use from other packages is very simple. The USE_DIRS variable takes a list of package names (without the "-dirs" part) together with the required version number (always pick the latest one when writing new packages).

For example, if a package installs files under share/applications, it should have the following line in it:

```
USE_DIRS+= xdg-1.1
```

After regenerating the PLIST using **make print-PLIST**, you should get the right (commented out) lines.

Note that even if your package is using $X11BASE, it must not depend on the *-x11-dirs packages. Just specify the name without that part and pkgsrc (in particular, mk/dirs.mk) will take care of it.

Chapter 40
Buildlink methodology

Buildlink is a framework in pkgsrc that controls what headers and libraries are seen by a package's configure and build processes. This is implemented in a two step process:

1. Symlink headers and libraries for dependencies into `BUILDLINK_DIR`, which by default is a subdirectory of `WRKDIR`.
2. Create wrapper scripts that are used in place of the normal compiler tools that translate `-I${LOCALBASE}/include` and `-L${LOCALBASE}/lib` into references to `BUILDLINK_DIR`. The wrapper scripts also make native compiler on some operating systems look like GCC, so that packages that expect GCC won't require modifications to build with those native compilers.

This normalizes the environment in which a package is built so that the package may be built consistently despite what other software may be installed. Please note that the normal system header and library paths, e.g. `/usr/include`, `/usr/lib`, etc., are always searched -- buildlink3 is designed to insulate the package build from non-system-supplied software.

40.1 Converting packages to use buildlink3

The process of converting packages to use the buildlink3 framework ("bl3ifying") is fairly straightforward. The things to keep in mind are:

1. Ensure that the build always calls the wrapper scripts instead of the actual toolchain. Some packages are tricky, and the only way to know for sure is the check `${WRKDIR}/.work.log` to see if the wrappers are being invoked.
2. Don't override `PREFIX` from within the package Makefile, e.g. Java VMs, standalone shells, etc., because the code to symlink files into `${BUILDLINK_DIR}` looks for files relative to "pkg_info -qp *`pkgname`*".
3. Remember that *only* the `buildlink3.mk` files that you list in a package's Makefile are added as dependencies for that package.

If a dependency on a particular package is required for its libraries and headers, then we replace:

```
DEPENDS+=   foo>=1.1.0:../../category/foo
```

with

```
.include "../../category/foo/buildlink3.mk"
```

The buildlink3.mk files usually define the required dependencies. If you need a newer version of the dependency when using buildlink3.mk files, then you can define it in your Makefile; for example:

```
BUILDLINK_DEPENDS.foo+=   foo>=1.1.0
.include "../../category/foo/buildlink3.mk"
```

There are several buildlink3.mk files in pkgsrc/mk that handle special package issues:

- bdb.buildlink3.mk chooses either the native or a pkgsrc Berkeley DB implementation based on the values of BDB_ACCEPTED and BDB_DEFAULT.
- curses.buildlink3.mk: If the system comes with neither Curses nor NCurses, this will take care to install the devel/ncurses package.
- krb5.buildlink3.mk uses the value of KRB5_ACCEPTED to choose between adding a dependency on Heimdal or MIT-krb5 for packages that require a Kerberos 5 implementation.
- motif.buildlink3.mk checks for a system-provided Motif installation or adds a dependency on x11/lesstif or x11/openmotif.
- ossaudio.buildlink3.mk defines several variables that may be used by packages that use the Open Sound System (OSS) API.
- pgsql.buildlink3.mk will accept either Postgres 7.3 or 7.4, whichever is found installed. See the file for more information.
- pthread.buildlink3.mk uses the value of PTHREAD_OPTS and checks for native pthreads or adds a dependency on devel/pth as needed.
- xaw.buildlink3.mk uses the value of XAW_TYPE to choose a particular Athena widgets library.

The comments in those buildlink3.mk files provide a more complete description of how to use them properly.

40.2 Writing buildlink3.mk files

A package's buildlink3.mk file is included by Makefiles to indicate the need to compile and link against header files and libraries provided by the package. A buildlink3.mk file should always provide enough information to add the correct type of dependency relationship and include any other buildlink3.mk files that it needs to find headers and libraries that it needs in turn.

To generate an initial buildlink3.mk file for further editing, Rene Hexel's pkgtools/createbuildlink package is highly recommended. For most packages, the following command will generate a good starting point for buildlink3.mk files:

```
% cd pkgsrc/category/pkgdir
% createbuildlink -3 >buildlink3.mk
```

40.2.1 Anatomy of a buildlink3.mk file

The following real-life example buildlink3.mk is taken from pkgsrc/graphics/tiff:

```
# $NetBSD: buildlink3.mk,v 1.7 2004/03/18 09:12:12 jlam Exp $

BUILDLINK_DEPTH:=       ${BUILDLINK_DEPTH}+
TIFF_BUILDLINK3_MK:=    ${TIFF_BUILDLINK3_MK}+
```

```
.if !empty(BUILDLINK_DEPTH:M+)
BUILDLINK_DEPENDS+=     tiff
.endif

BUILDLINK_PACKAGES:=    ${BUILDLINK_PACKAGES:Ntiff}
BUILDLINK_PACKAGES+=    tiff

.if !empty(TIFF_BUILDLINK3_MK:M+)
BUILDLINK_DEPENDS.tiff+=        tiff>=3.6.1
BUILDLINK_PKGSRCDIR.tiff?=      ../../graphics/tiff
.endif  # TIFF_BUILDLINK3_MK

.include "../../devel/zlib/buildlink3.mk"
.include "../../graphics/jpeg/buildlink3.mk"

BUILDLINK_DEPTH:=       ${BUILDLINK_DEPTH:S/+$//}
```

The header and footer manipulate `BUILDLINK_DEPTH`, which is common across all `buildlink3.mk` files and is used to track at what depth we are including `buildlink3.mk` files.

The first section controls if the dependency on *`pkg`* is added. `BUILDLINK_DEPENDS` is the global list of packages for which dependencies are added by buildlink3.

The second section advises pkgsrc that the `buildlink3.mk` file for *`pkg`* has been included at some point. `BUILDLINK_PACKAGES` is the global list of packages for which `buildlink3.mk` files have been included. It must *always* be appended to within a `buildlink3.mk` file.

The third section is protected from multiple inclusion and controls how the dependency on *`pkg`* is added. Several important variables are set in the section:

- `BUILDLINK_DEPENDS.`*`pkg`* is the actual dependency recorded in the installed package; this should always be set using **+=** to ensure that we're appending to any pre-existing list of values. This variable should be set to the first version of the package that had the last change in the major number of a shared library or that had a major API change.
- `BUILDLINK_PKGSRCDIR.`*`pkg`* is the location of the *`pkg`* pkgsrc directory.
- `BUILDLINK_DEPMETHOD.`*`pkg`* (not shown above) controls whether we use `BUILD_DEPENDS` or `DEPENDS` to add the dependency on *`pkg`*. The build dependency is selected by setting `BUILDLINK_DEPMETHOD.`*`pkg`* to "build". By default, the full dependency is used.
- `BUILDLINK_INCDIRS.`*`pkg`* and `BUILDLINK_LIBDIRS.`*`pkg`* (not shown above) are lists of subdirectories of `${BUILDLINK_PREFIX.`*`pkg`*`}` to add to the header and library search paths. These default to "include" and "lib" respectively.
- `BUILDLINK_CPPFLAGS.`*`pkg`* (not shown above) is the list of preprocessor flags to add to `CPPFLAGS`, which are passed on to the configure and build phases. The "-I" option should be avoided and instead be handled using `BUILDLINK_INCDIRS.`*`pkg`* as above.

The following variables are all optionally defined within this second section (protected against multiple inclusion) and control which package files are symlinked into `${BUILDLINK_DIR}` and how their names are transformed during the symlinking:

- BUILDLINK_FILES.*pkg* (not shown above) is a shell glob pattern relative to ${BUILDLINK_PREFIX.*pkg*} to be symlinked into ${BUILDLINK_DIR}, e.g. include/*.h.
- BUILDLINK_FILES_CMD.*pkg* (not shown above) is a shell pipeline that outputs to stdout a list of files relative to ${BUILDLINK_PREFIX.*pkg*}. The resulting files are to be symlinked into ${BUILDLINK_DIR}. By default, this takes the +CONTENTS of a *pkg* and filters it through ${BUILDLINK_CONTENTS_FILTER.*pkg*}.
- BUILDLINK_CONTENTS_FILTER.*pkg* (not shown above) is a filter command that filters +CONTENTS input into a list of files relative to ${BUILDLINK_PREFIX.*pkg*} on stdout. By default for overwrite packages, BUILDLINK_CONTENTS_FILTER.*pkg* outputs the contents of the include and lib directories in the package +CONTENTS, and for pkgviews packages, it outputs any libtool archives in lib directories.
- BUILDLINK_TRANSFORM.*pkg* (not shown above) is a list of sed arguments used to transform the name of the source filename into a destination filename, e.g. **-e "s|/curses.h|/ncurses.h|g"**.

The last section includes any buildlink3.mk needed for *pkg*'s library dependencies. Including these buildlink3.mk files means that the headers and libraries for these dependencies are also symlinked into ${BUILDLINK_DIR} whenever the *pkg* buildlink3.mk file is included.

40.2.2 Updating BUILDLINK_DEPENDS.*pkg* in buildlink3.mk files

There are two situations that require increasing the dependency listed in BUILDLINK_DEPENDS.*pkg* after a package update:

1. if the sonames (major number of the library version) of any installed shared libraries change.
2. if the API or interface to the header files change.

In these cases, BUILDLINK_DEPENDS.*pkg* should be adjusted to require at least the new package version. In some cases, the packages that depend on this new version may need their PKGREVISIONs increased and, if they have buildlink3.mk files, their BUILDLINK_DEPENDS.*pkg* adjusted, too. This is needed so that binary packages made using it will require the correct package dependency and not settle for an older one which will not contain the necessary shared libraries.

Please take careful consideration before adjusting BUILDLINK_DEPENDS.*pkg* as we don't want to cause unneeded package deletions and rebuilds. In many cases, new versions of packages work just fine with older dependencies. See Section 44.1.4 for more information about dependencies on other packages, including the BUILDLINK_RECOMMENDED and RECOMMENDED definitions.

40.3 Writing builtin.mk files

Some packages in pkgsrc install headers and libraries that coincide with headers and libraries present in the base system. Aside from a buildlink3.mk file, these packages should also include a builtin.mk file that includes the necessary checks to decide whether using the built-in software or the pkgsrc software is appropriate.

The only requirements of a builtin.mk file for *pkg* are:

1. It should set USE_BUILTIN.*pkg* to either “yes” or “no” after it is included.
2. It should *not* override any USE_BUILTIN.*pkg* which is already set before the builtin.mk file is included.
3. It should be written to allow multiple inclusion. This is *very* important and takes careful attention to Makefile coding.

40.3.1 Anatomy of a builtin.mk file

The following is the recommended template for builtin.mk files:

```
.if !defined(IS_BUILTIN.foo)
#
# IS_BUILTIN.foo is set to "yes" or "no" depending on whether "foo"
# genuinely exists in the system or not.
#
IS_BUILTIN.foo?=        no

# BUILTIN_PKG.foo should be set here if "foo" is built-in and its package
# version can be determined.
#
.  if !empty(IS_BUILTIN.foo:M[yY][eE][sS])
BUILTIN_PKG.foo?=       foo-1.0
.  endif
.endif  # IS_BUILTIN.foo

.if !defined(USE_BUILTIN.foo)
USE_BUILTIN.foo?=       ${IS_BUILTIN.foo}
.  if defined(BUILTIN_PKG.foo)
.    for _depend_ in ${BUILDLINK_DEPENDS.foo}
.      if !empty(USE_BUILTIN.foo:M[yY][eE][sS])
USE_BUILTIN.foo!=                                                       \
      if ${PKG_ADMIN} pmatch '${_depend_}' ${BUILTIN_PKG.foo}; then     \
              ${ECHO} "yes";                                            \
      else                                                              \
              ${ECHO} "no";                                             \
      fi
.      endif
.    endfor
.  endif
.endif  # USE_BUILTIN.foo

CHECK_BUILTIN.foo?=     no
.if !empty(CHECK_BUILTIN.foo:M[nN][oO])
#
# Here we place code that depends on whether USE_BUILTIN.foo is set to
# "yes" or "no".
#
.endif  # CHECK_BUILTIN.foo
```

The first section sets IS_BUILTIN.*pkg* depending on if *pkg* really exists in the base system. This should not be a base system software with similar functionality to *pkg*; it should only be "yes" if the actual package is included as part of the base system. This variable is only used internally within the builtin.mk file.

The second section sets BUILTIN_PKG.*pkg* to the version of *pkg* in the base system if it exists (if IS_BUILTIN.*pkg* is "yes"). This variable is only used internally within the builtin.mk file.

The third section sets USE_BUILTIN.*pkg* and is *required* in all builtin.mk files. The code in this section must make the determination whether the built-in software is adequate to satisfy the dependencies listed in BUILDLINK_DEPENDS.*pkg*. This is typically done by comparing BUILTIN_PKG.*pkg* against each of the dependencies in BUILDLINK_DEPENDS.*pkg*. USE_BUILTIN.*pkg* *must* be set to the correct value by the end of the builtin.mk file. Note that USE_BUILTIN.*pkg* may be "yes" even if IS_BUILTIN.*pkg* is "no" because we may make the determination that the built-in version of the software is similar enough to be used as a replacement.

The last section is guarded by CHECK_BUILTIN.*pkg*, and includes code that uses the value of USE_BUILTIN.*pkg* set in the previous section. This typically includes, e.g., adding additional dependency restrictions and listing additional files to symlink into ${BUILDLINK_DIR} (via BUILDLINK_FILES.*pkg*).

40.3.2 Global preferences for native or pkgsrc software

When building packages, it's possible to choose whether to set a global preference for using either the built-in (native) version or the pkgsrc version of software to satisfy a dependency. This is controlled by setting PREFER_PKGSRC and PREFER_NATIVE. These variables take values of either "yes", "no", or a list of packages. PREFER_PKGSRC tells pkgsrc to use the pkgsrc versions of software, while PREFER_NATIVE tells pkgsrc to use the built-in versions. Preferences are determined by the most specific instance of the package in either PREFER_PKGSRC or PREFER_NATIVE. If a package is specified in neither or in both variables, then PREFER_PKGSRC has precedence over PREFER_NATIVE. For example, to require using pkgsrc versions of software for all but the most basic bits on a NetBSD system, you can set:

```
PREFER_PKGSRC=  yes
PREFER_NATIVE=  getopt skey tcp_wrappers
```

A package *must* have a builtin.mk file to be listed in PREFER_NATIVE, otherwise it is simply ignored in that list.

Chapter 41
The pkginstall framework

This chapter describes the framework known as `pkginstall`, whose key features are:

- Generic installation and manipulation of directories and files outside the pkgsrc-handled tree, `LOCALBASE`.
- Automatic handling of configuration files during installation, provided that packages are correctly designed.
- Generation and installation of system startup scripts.
- Registration of system users and groups.
- Registration of system shells.

The following sections inspect each of the above points in detail. Note that in order to use any of the described functionalities, you must add the following to your package's `Makefile`:

```
USE_PKGINSTALL=YES
```

You may be thinking that many of the things described here could be easily done with simple code in the package's post-installation target (`post-install`). *This is incorrect*, as the code in them is only executed when building from source. Machines using binary packages could not benefit from it at all (as the code itself could be unavailable). Therefore, the only way to achieve any of the items described above is by means of the installation scripts, which are automatically generated by pkginstall.

41.1 Files and directories outside the installation prefix

As you already know, the `PLIST` file holds a list of files and directories that belong to a package. The names used in it are relative to the installation prefix (`${PREFIX}`), which means that it cannot register files outside this directory (absolute path names are not allowed). Despite this restriction, some packages need to install files outside this location; e.g., under `${VARBASE}` or `${PKG_SYSCONFDIR}`.

The only way to achieve this is to create such files during installation time by using the installation scripts. These scripts can run arbitrary commands, so they have the potential to create and manage files anywhere in the file system. Here is where pkginstall comes into play: it provides generic scripts to abstract the manipulation of such files and directories based on variables set in the package's `Makefile`. The rest of this section describes these variables.

41.1.1 Directory manipulation

The following variables can be set to request the creation of directories anywhere in the file system:

- MAKE_DIRS and OWN_DIRS contain a list of directories that should be created and should attempt to be destroyed by the installation scripts. The difference between the two is that the latter prompts the administrator to remove any directories that may be left after deinstallation (because they were not empty), while the former does not.
- MAKE_DIRS_PERMS and OWN_DIRS_PERMS contain a list of tuples describing which directories should be created and should attempt to be destroyed by the installation scripts. Each tuple holds the following values, separated by spaces: the directory name, its owner, its group and its numerical mode. For example:

  ```
  MAKE_DIRS_PERMS+= ${VARBASE}/foo/private ${ROOT_USER} ${ROOT_GROUP} 0700
  ```

  The difference between the two is exactly the same as their non-PERMS counterparts.

41.1.2 File manipulation

Creating non-empty files outside the installation prefix is tricky because the PLIST forces all files to be inside it. To overcome this problem, the only solution is to extract the file in the known place (i.e., inside the installation prefix) and copy it to the appropriate location during installation (done by the installation scripts generated by pkginstall). We will call the former the *master file* in the following paragraphs, which describe the variables that can be used to automatically and consistently handle files outside the installation prefix:

- CONF_FILES and SUPPORT_FILES are pairs of master and target files. During installation time, the master file is copied to the target one if and only if the latter does not exist. Upon deinstallation, the target file is removed provided that it was not modified by the installation.

  The difference between the two is that the latter prompts the administrator to remove any files that may be left after deinstallation (because they were not empty), while the former does not.
- CONF_FILES_PERMS and SUPPORT_FILES_PERMS contain tuples describing master files as well as their target locations. For each of them, it also specifies their owner, their group and their numeric permissions, in this order. For example:

  ```
  SUPPORT_FILES_PERMS+= ${PREFIX}/share/somefile ${VARBASE}/somefile ${ROOT_USER} ${ROOT_GRO
  ```

  The difference between the two is exactly the same as their non-PERMS counterparts.

41.2 Configuration files

Configuration files are special in the sense that they are installed in their own specific directory, PKG_SYSCONFDIR, and need special treatment during installation (most of which is automated by pkginstall). The main concept you must bear in mind is that files marked as configuration files are automatically copied to the right place (somewhere inside PKG_SYSCONFDIR) during installation *if and only if* they didn't exist before. Similarly, they will not be removed if they have local modifications. This ensures that administrators never lose any custom changes they may have made.

41.2.1 How PKG_SYSCONFDIR is set

As said before, the PKG_SYSCONFDIR variable specifies where configuration files shall be installed. Its contents are set based upon the following variables:

- PKG_SYSCONFBASE: The configuration's root directory. Defaults to ${PREFIX}/etc although it may be overridden by the user to point to his preferred location (e.g., /etc, /etc/pkg, etc.). Packages must not use it directly.
- PKG_SYSCONFSUBDIR: A subdirectory of PKG_SYSCONFBASE under which the configuration files for the package being built shall be installed. The definition of this variable only makes sense in the package's Makefile (i.e., it is not user-customizable).

  As an example, consider the Apache package, www/apache2, which places its configuration files under the httpd/ subdirectory of PKG_SYSCONFBASE. This should be set in the package Makefile.
- PKG_SYSCONFVAR: Specifies the name of the variable that holds this package's configuration directory (if different from PKG_SYSCONFBASE). It defaults to PKGBASE's value, and is always prefixed with PKG_SYSCONFDIR.
- PKG_SYSCONFDIR.${PKG_SYSCONFVAR}: Holds the directory where the configuration files for the package identified by PKG_SYSCONFVAR's shall be placed.

Based on the above variables, pkginstall determines the value of PKG_SYSCONFDIR, which is the *only* variable that can be used within a package to refer to its configuration directory. The algorithm used to set its value is basically the following:

1. If PKG_SYSCONFDIR.${PKG_SYSCONFVAR} is set, its value is used.
2. If the previous variable is not defined but PKG_SYSCONFSUBDIR is set in the package's Makefile, the resulting value is ${PKG_SYSCONFBASE}/${PKG_SYSCONFSUBDIR}.
3. Otherwise, it is set to ${PKG_SYSCONFBASE}.

It is worth mentioning that ${PKG_SYSCONFDIR} is automatically added to OWN_DIRS. See Section 41.1.1 what this means.

41.2.2 Telling the software where configuration files are

Given that pkgsrc (and users!) expect configuration files to be in a known place, you need to teach each package where it shall install its files. In some cases you will have to patch the package Makefiles to achieve it. If you are lucky, though, it may be as easy as passing an extra flag to the configuration script; this is the case of GNU Autoconf- generated files:

```
CONFIGURE_ARGS+= --sysconfdir=${PKG_SYSCONFDIR}
```

Note that this specifies where the package has to *look for* its configuration files, not where they will be originally installed (although the difference is never explicit, unfortunately).

41.2.3 Patching installations

As said before, pkginstall automatically handles configuration files. This means that **the packages themselves must not touch the contents of `${PKG_SYSCONFDIR}` directly**. Bad news is that many software installation scripts will, out of the box, mess with the contents of that directory. So what is the correct procedure to fix this issue?

You must teach the package (usually by manually patching it) to install any configuration files under the examples hierarchy, `share/examples/${PKGBASE}/`. This way, the `PLIST` registers them and the administrator always has the original copies available.

Once the required configuration files are in place (i.e., under the examples hierarchy), the pkginstall framework can use them as master copies during the package installation to update what is in `${PKG_SYSCONFDIR}`. To achieve this, the variables `CONF_FILES` and `CONF_FILES_PERMS` are used. Check out Section 41.1.2 for information about their syntax and their purpose. Here is an example, taken from the `mail/mutt` package:

```
EGDIR=          ${PREFIX}/share/doc/mutt/samples
CONF_FILES=     ${EGDIR}/Muttrc ${PKG_SYSCONFDIR}/Muttrc
```

Note that the `EGDIR` variable is specific to that package and has no meaning outside it.

41.2.4 Disabling handling of configuration files

The automatic copying of config files can be toggled by setting the environment variable `PKG_CONFIG` prior to package installation.

41.3 System startup scripts

System startup scripts are special files because they must be installed in a place known by the underlying OS, usually outside the installation prefix. Therefore, the same rules described in Section 41.1 apply, and the same solutions can be used. However, pkginstall provides a special mechanism to handle these files.

In order to provide system startup scripts, the package has to:

1. Store the script inside `${FILESDIR}`, with the `.sh` suffix appended. Considering the `print/cups` package as an example, it has a `cupsd.sh` in its files directory.
2. Tell pkginstall to handle it, appending the name of the script, without its extension, to the `RCD_SCRIPTS` variable. Continuing the previous example:

   ```
   RCD_SCRIPTS+= cupsd
   ```

Once this is done, pkginstall will do the following steps for each script in an automated fashion:

1. Process the file found in the files directory applying all the substitutions described in the `FILES_SUBST` variable.
2. Copy the script from the files directory to the examples hierarchy, `${PREFIX}/share/examples/rc.d/`. Note that this master file must be explicitly registered in the `PLIST`.

3. Add code to the installation scripts to copy the startup script from the examples hierarchy into the system-wide startup scripts directory.

41.3.1 Disabling handling of system startup scripts

The automatic copying of config files can be toggled by setting the environment variable `PKG_RCD_SCRIPTS` prior to package installation. Note that the scripts will be always copied inside the examples hierarchy, `${PREFIX}/share/examples/rc.d/`, no matter what the value of this variable is.

41.4 System users and groups

If a package needs to create special users and/or groups during installation, it can do so by using the pkginstall framework.

Users can be created by adding entries to the `PKG_USERS` variable. Each entry has the following syntax, which mimics `/etc/passwd`:

```
user:group[:[userid][:[descr][:[home][:shell]]]]
```

Only the user and group are required; everything else is optional, but the colons must be in the right places when specifying optional bits. By default, a new user will have home directory `/nonexistent`, and login shell `/sbin/nologin` unless they are specified as part of the user element. Note that if the description contains spaces, then spaces should be double backslash-escaped, as in:

```
foo:foogrp::The\\ Foomister
```

Similarly, groups can be created using the `PKG_GROUPS` variable, whose syntax is:

```
group[:groupid]
```

As before, only the group name is required; the numeric identifier is optional.

41.5 System shells

Packages that install system shells should register them in the shell database, `/etc/shells`, to make things easier to the administrator. This must be done from the installation scripts to keep binary packages working on any system. pkginstall provides an easy way to accomplish this task.

When a package provides a shell interpreter, it has to set the `PKG_SHELL` variable to its absolute file name. This will add some hooks to the installation scripts to handle it. Consider the following example, taken from `shells/zsh`:

```
USE_PKGINSTALL= YES
PKG_SHELL=      ${PREFIX}/bin/zsh
```

41.5.1 Disabling handling of configuration files

The automatic registration of shell interpreters can be disabled by the administrator by setting the `PKG_REGISTER_SHELLS` environment variable to `NO`.

Chapter 42
Options handling

Many packages have the ability to be built to support different sets of features. `bsd.options.mk` is a framework in pkgsrc that provides generic handling of those options that determine different ways in which the packages can be built. It's possible for the user to specify exactly which sets of options will be built into a package or to allow a set of global default options apply.

42.1 Global default options

Global default options are listed in `PKG_DEFAULT_OPTIONS`, which is a list of the options that should be built into every package if that option is supported. This variable should be set in `/etc/mk.conf`.

42.2 Converting packages to use `bsd.options.mk`

The following example shows how `bsd.options.mk` should be used by the hypothetical "wibble" package, either in the package `Makefile`, or in a file, e.g. `options.mk`, that is included by the main package `Makefile`.

```
PKG_OPTIONS_VAR=  PKG_OPTIONS.wibble
PKG_SUPPORTED_OPTIONS=  wibble-foo ldap
PKG_OPTIONS_OPTIONAL_GROUPS= database
PKG_OPTIONS_GROUP.database= mysql pgsql
PKG_SUGGESTED_OPTIONS=  wibble-foo
PKG_OPTIONS_LEGACY_VARS+= WIBBLE_USE_OPENLDAP:ldap
PKG_OPTIONS_LEGACY_OPTS+= foo:wibble-foo

.include "../../mk/bsd.prefs.mk"

# this package was previously named wibble2
.if defined(PKG_OPTIONS.wibble2)
PKG_LEGACY_OPTIONS+=  ${PKG_OPTIONS.wibble2}
PKG_OPTIONS_DEPRECATED_WARNINGS+="Deprecated variable PKG_OPTIONS.wibble2 used, use "${PKG_O
.endif

.include "../../mk/bsd.options.mk"

# Package-specific option-handling

###
### FOO support
###
.if !empty(PKG_OPTIONS:Mwibble-foo)
CONFIGURE_ARGS+= --enable-foo
```

```
.endif

###
### LDAP support
###
.if !empty(PKG_OPTIONS:Mldap)
.  include "../../databases/openldap/buildlink3.mk"
CONFIGURE_ARGS+= --enable-ldap=${BUILDLINK_PREFIX.openldap}
.endif

###
### database support
###
.if !empty(PKG_OPTIONS:Mmysql)
.  include "../../mk/mysql.buildlink3.mk"
.endif
.if !empty(PKG_OPTIONS:Mpgsql)
.  include "../../mk/pgsql.buildlink3.mk"
.endif
```

The first section contains the information about which build options are supported by the package, and any default options settings if needed.

1. PKG_OPTIONS_VAR is the name of the make(1) variable that the user can set to override the default options. It should be set to "PKG_OPTIONS.*pkgbase*".
2. PKG_SUPPORTED_OPTIONS is a list of build options supported by the package.
3. PKG_OPTIONS_OPTIONAL_GROUPS is a list of names of groups of mutually exclusive options. The options in each group are listed in PKG_OPTIONS_GROUP.*groupname*. The most specific setting of any option from the group takes precedence over all other options in the group. Options from the groups will be automatically added to PKG_SUPPORTED_OPTIONS.
4. PKG_OPTIONS_REQUIRED_GROUPS is like PKG_OPTIONS_OPTIONAL_GROUPS, but building the packages will fail if no option from the group is selected.
5. PKG_OPTIONS_NONEMPTY_SETS is a list of names of sets of options. At least one option from each set must be selected. The options in each set are listed in PKG_OPTIONS_SET.*setname*. Options from the sets will be automatically added to PKG_SUPPORTED_OPTIONS. Building the package will fail if no option from the set is selected.
6. PKG_SUGGESTED_OPTIONS is a list of build options which are enabled by default.
7. PKG_OPTIONS_LEGACY_VARS is a list of "*USE_VARIABLE*:*option*" pairs that map legacy /etc/mk.conf variables to their option counterparts. Pairs should be added with "+=" to keep the listing of global legacy variables. A warning will be issued if the user uses a legacy variable.
8. PKG_OPTIONS_LEGACY_OPTS is a list of "*old-option*:*new-option*" pairs that map options that have been renamed to their new counterparts. Pairs should be added with "+=" to keep the listing of global legacy options. A warning will be issued if the user uses a legacy option.
9. PKG_LEGACY_OPTIONS is a list of options implied by deprecated variables used. This can be used for cases that neither PKG_OPTIONS_LEGACY_VARS nor PKG_OPTIONS_LEGACY_OPTS can handle, e. g. when PKG_OPTIONS_VAR is renamed.

10. PKG_OPTIONS_DEPRECATED_WARNINGS is a list of warnings about deprecated variables or options used, and what to use instead.

A package should never modify PKG_DEFAULT_OPTIONS or the variable named in PKG_OPTIONS_VAR. These are strictly user-settable. To suggest a default set of options, use PKG_SUGGESTED_OPTIONS.

PKG_OPTIONS_VAR must be defined before including bsd.options.mk. If none of PKG_SUPPORTED_OPTIONS, PKG_OPTIONS_OPTIONAL_GROUPS, and PKG_OPTIONS_REQUIRED_GROUPS are defined (as can happen with platform-specific options if none of them is supported on the current platform), PKG_OPTIONS is set to the empty list and the package is otherwise treated as not using the options framework.

After the inclusion of bsd.options.mk, the variable PKG_OPTIONS contains the list of selected build options, properly filtered to remove unsupported and duplicate options.

The remaining sections contain the logic that is specific to each option. The correct way to check for an option is to check whether it is listed in PKG_OPTIONS:

```
.if !empty(PKG_OPTIONS:M option)
```

42.3 Option Names

Options that enable similar features in different packages (like optional support for a library) should use a common name in all packages that support it (like the name of the library). If another package already has an option with the same meaning, use the same name.

Options that enable features specific to one package, where it's unlikely that another (unrelated) package has the same (or a similar) optional feature, should use a name prefixed with *pkgname*-.

If a group of related packages share an optional feature specific to that group, prefix it with the name of the "main" package (e. g. djbware-errno-hack).

For new options, add a line to mk/defaults/options.description. Lines have two fields, separated by tab. The first field is the option name, the second its description. The description should be a whole sentence (starting with an uppercase letter and ending with a period) that describes what enabling the option does. E. g. "Enable ispell support." The file is sorted by option names.

Chapter 43
The build process

The basic steps for building a program are always the same. First the program's source (*distfile*) must be brought to the local system and then extracted. After any patches to compile properly on NetBSD are applied, the software can be configured, then built (usually by compiling), and finally the generated binaries, etc. can be put into place on the system. These are exactly the steps performed by the NetBSD package system, which is implemented as a series of targets in a central Makefile, `pkgsrc/mk/bsd.pkg.mk`.

43.1 Program location

Before outlining the process performed by the NetBSD package system in the next section, here's a brief discussion on where programs are installed, and which variables influence this.

The automatic variable `PREFIX` indicates where all files of the final program shall be installed. It is usually set to `LOCALBASE` (`/usr/pkg`), or `CROSSBASE` for pkgs in the "cross" category. The value of `PREFIX` needs to be put into the various places in the program's source where paths to these files are encoded. See Section 37.3 and Section 44.3.1 for more details.

When choosing which of these variables to use, follow the following rules:

- `PREFIX` always points to the location where the current pkg will be installed. When referring to a pkg's own installation path, use "${PREFIX}".
- `LOCALBASE` is where all non-X11 pkgs are installed. If you need to construct a -I or -L argument to the compiler to find includes and libraries installed by another non-X11 pkg, use "${LOCALBASE}".
- `X11BASE` is where the actual X11 distribution (from xsrc, etc.) is installed. When looking for *standard* X11 includes (not those installed by a pkg), use "${X11BASE}".
- X11-based packages are special in that they may be installed in either `X11BASE` or `LOCALBASE`.

  Usually, X11 packages should be installed under `LOCALBASE` whenever possible. Note that you will need to include `../../mk/x11.buildlink3.mk` in them to request the presence of X11 and to get the right compilation flags.

  Even though, there are some packages that cannot be installed under `LOCALBASE`: those that come with app-defaults files. These packages are special and they must be placed under `X11BASE`. To accomplish this, set either `USE_X11BASE` or `USE_IMAKE` in your package.

  Some notes: If you need to find includes or libraries installed by a pkg that has `USE_IMAKE` or `USE_X11BASE` in its pkg `Makefile`, you need to look in *both* `${X11BASE}` and `${LOCALBASE}`. To force installation of all X11 packages in `LOCALBASE`, the `pkgtools/xpkgwedge` package is enabled by default.

- X11PREFIX should be used to refer to the installed location of an X11 package. X11PREFIX will be set to X11BASE if xpkgwedge is not installed, and to LOCALBASE if xpkgwedge is installed.
- If xpkgwedge is installed, it is possible to have some packages installed in X11BASE and some in LOCALBASE. To determine the prefix of an installed package, the EVAL_PREFIX definition can be used. It takes pairs in the format "DIRNAME=<package>", and the make(1) variable DIRNAME will be set to the prefix of the installed package <package>, or "${X11PREFIX}" if the package is not installed.

  This is best illustrated by example.

  The following lines are taken from pkgsrc/wm/scwm/Makefile:

```
EVAL_PREFIX+=           GTKDIR=gtk+
CONFIGURE_ARGS+=        --with-guile-prefix=${LOCALBASE}        \
                        --with-gtk-prefix="${GTKDIR}"           \
                        --enable-multibyte
```

  Specific defaults can be defined for the packages evaluated using EVAL_PREFIX, by using a definition of the form:

```
GTKDIR_DEFAULT= ${LOCALBASE}
```

  where GTKDIR corresponds to the first definition in the EVAL_PREFIX pair.
- Within ${PREFIX}, packages should install files according to hier(7), with the exception that manual pages go into ${PREFIX}/man, not ${PREFIX}/share/man.

43.2 Main targets

The main targets used during the build process defined in bsd.pkg.mk are:

fetch

This will check if the file(s) given in the variables DISTFILES and PATCHFILES (as defined in the package's Makefile) are present on the local system in /usr/pkgsrc/distfiles. If they are not present, an attempt will be made to fetch them using commands of the form:

```
${FETCH_CMD} ${FETCH_BEFORE_ARGS} ${site}${file} ${FETCH_AFTER_ARGS}
```

where ${site} varies through several possibilities in turn: first, MASTER_SITE_OVERRIDE is tried, then the sites specified in either SITES_file if defined, else MASTER_SITES or PATCH_SITES, as applies, then finally the value of MASTER_SITE_BACKUP. The order of all except the first can be optionally sorted by the user, via setting either MASTER_SORT_AWK or MASTER_SORT_REGEX.

checksum

After the distfile(s) are fetched, their checksum is generated and compared with the checksums stored in the distinfo file. If the checksums don't match, the build is aborted. This is to ensure the same distfile is used for building, and that the distfile wasn't changed, e.g. by some malign force, deliberately changed distfiles on the master distribution site or network lossage.

extract

When the distfiles are present on the local system, they need to be extracted, as they are usually in the form of some compressed archive format, most commonly .tar.gz.

If only some of the distfiles need to be uncompressed, the files to be uncompressed should be put into EXTRACT_ONLY.

If the distfiles are not in .tar.gz format, they can be extracted by setting either EXTRACT_SUFX, or EXTRACT_CMD, EXTRACT_BEFORE_ARGS and EXTRACT_AFTER_ARGS. In the former case, pkgsrc knows how to extract a number of suffixes (.tar.gz, .tgz, .tar.gz2, .tbz, .tar.Z, .tar, .shar.gz, .shar.bz2, .shar.Z, .shar, .Z, .bz2 and .gz; see the definition of the various DECOMPRESS_CMD variables in bsd.pkg.extract.mk for a complete list). Here's an example on how to use the other variables for a program that comes with a compressed shell archive whose name ends in .msg.gz:

```
EXTRACT_SUFX=    .msg.gz
EXTRACT_CMD=             zcat
EXTRACT_BEFORE_ARGS=
EXTRACT_AFTER_ARGS=      |sh
```

patch

After extraction, all the patches named by the PATCHFILES, those present in the patches subdirectory of the package as well as in $LOCALPATCHES/$PKGPATH (e.g. /usr/local/patches/graphics/png) are applied. Patchfiles ending in .Z or .gz are uncompressed before they are applied, files ending in .orig or .rej are ignored. Any special options to patch(1) can be handed in PATCH_DIST_ARGS. See Section 37.3 for more details.

By default patch(1) is given special args to make it fail if the patches apply with some lines of fuzz. Please fix (regen) the patches so that they apply cleanly. The rationale behind this is that patches that don't apply cleanly may end up being applied in the wrong place, and cause severe harm there.

configure

Most pieces of software need information on the header files, system calls, and library routines which are available in NetBSD. This is the process known as configuration, and is usually automated. In most cases, a script is supplied with the source, and its invocation results in generation of header files, Makefiles, etc.

If the program's distfile contains its own configure script, this can be invoked by setting HAS_CONFIGURE. If the configure script is a GNU autoconf script, GNU_CONFIGURE should be specified instead. In either case, any arguments to the configure script can be specified in the CONFIGURE_ARGS variable, and the configure script's name can be set in CONFIGURE_SCRIPT if it differs from the default "configure". Here's an example from the sysutils/top package:

```
HAS_CONFIGURE=          yes
CONFIGURE_SCRIPT=       Configure
CONFIGURE_ARGS+=        netbsd13
```

If the program uses an Imakefile for configuration, the appropriate steps can be invoked by setting USE_IMAKE to "YES". (If you only want the package installed in $X11PREFIX but xmkmf not being run, set USE_X11BASE instead!)

build

Once configuration has taken place, the software will be built by invoking $MAKE_PROGRAM on $MAKEFILE with $BUILD_TARGET as the target to build. The default MAKE_PROGRAM is "gmake" if USE_TOOLS contains "gmake", "make" otherwise. MAKEFILE is set to "Makefile" by default, and

BUILD_TARGET defaults to “all”. Any of these variables can be set in the package’s Makefile to change the default build process.

install

Once the build stage has completed, the final step is to install the software in public directories, so users can access the programs and files. As in the build-target, $MAKE_PROGRAM is invoked on $MAKEFILE here, but with the $INSTALL_TARGET instead, the latter defaulting to “install” (plus “install.man”, if USE_IMAKE is set).

If no target is specified, the default is “build”. If a subsequent stage is requested, all prior stages are made: e.g. **make build** will also perform the equivalent of:

```
make fetch
make checksum
make extract
make patch
make configure
make build
```

43.3 Other helpful targets

pre/post-*

For any of the main targets described in the previous section, two auxiliary targets exist with “pre-” and “post-” used as a prefix for the main target’s name. These targets are invoked before and after the main target is called, allowing extra configuration or installation steps be performed from a package’s Makefile, for example, which a program’s configure script or install target omitted.

do-*

Should one of the main targets do the wrong thing, and should there be no variable to fix this, you can redefine it with the do-* target. (Note that redefining the target itself instead of the do-* target is a bad idea, as the pre-* and post-* targets won’t be called anymore, etc.) You will not usually need to do this.

reinstall

If you did a **make install** and you noticed some file was not installed properly, you can repeat the installation with this target, which will ignore the “already installed” flag.

deinstall

This target does a pkg_delete(1) in the current directory, effectively de-installing the package. The following variables can be used to tune the behaviour:

PKG_VERBOSE

Add a "-v" to the pkg_delete(1) command.

DEINSTALLDEPENDS

Remove all packages that require (depend on) the given package. This can be used to remove any packages that may have been pulled in by a given package, e.g. if **make deinstall DEINSTALLDEPENDS=1** is done in pkgsrc/x11/kde, this is likely to remove whole KDE. Works by adding "-R" to the pkg_delete(1) command line.

update

This target causes the current package to be updated to the latest version. The package and all depending packages first get de-installed, then current versions of the corresponding packages get compiled and installed. This is similar to manually noting which packages are currently installed, then performing a series of **make deinstall** and **make install** (or whatever UPDATE_TARGET is set to) for these packages.

You can use the "update" target to resume package updating in case a previous **make update** was interrupted for some reason. However, in this case, make sure you don't call **make clean** or otherwise remove the list of dependent packages in WRKDIR. Otherwise, you lose the ability to automatically update the current package along with the dependent packages you have installed.

Resuming an interrupted **make update** will only work as long as the package tree remains unchanged. If the source code for one of the packages to be updated has been changed, resuming **make update** will most certainly fail!

The following variables can be used either on the command line or in /etc/mk.conf to alter the behaviour of **make update**:

UPDATE_TARGET

Install target to recursively use for the updated package and the dependent packages. Defaults to DEPENDS_TARGET if set, "install" otherwise for **make update**. e.g. **make update UPDATE_TARGET=package**

NOCLEAN

Don't clean up after updating. Useful if you want to leave the work sources of the updated packages around for inspection or other purposes. Be sure you eventually clean up the source tree (see the "clean-update" target below) or you may run into troubles with old source code still lying around on your next **make** or **make update**.

REINSTALL

Deinstall each package before installing (making DEPENDS_TARGET). This may be necessary if the "clean-update" target (see below) was called after interrupting a running **make update**.

DEPENDS_TARGET

Allows you to disable recursion and hardcode the target for packages. The default is "update" for the update target, facilitating a recursive update of prerequisite packages. Only set DEPENDS_TARGET if you want to disable recursive updates. Use UPDATE_TARGET instead to just set a specific target for each package to be installed during **make update** (see above).

clean-update

Clean the source tree for all packages that would get updated if **make update** was called from the current directory. This target should not be used if the current package (or any of its depending packages) have already been de-installed (e.g., after calling **make update**) or you may lose some packages you intended to update. As a rule of thumb: only use this target *before* the first time you run **make update** and only if you have a dirty package tree (e.g., if you used NOCLEAN).

If you are unsure about whether your tree is clean, you can either perform a **make clean** at the top of the tree, or use the following sequence of commands from the directory of the package you want to update (*before* running **make update** for the first time, otherwise you lose all the packages you wanted to update!):

```
# make clean-update
# make clean CLEANDEPENDS=YES
# make update
```

The following variables can be used either on the command line or in /etc/mk.conf to alter the behaviour of **make clean-update**:

CLEAR_DIRLIST

After **make clean**, do not reconstruct the list of directories to update for this package. Only use this if **make update** successfully installed all packages you wanted to update. Normally, this is done automatically on **make update**, but may have been suppressed by the NOCLEAN variable (see above).

info

This target invokes pkg_info(1) for the current package. You can use this to check which version of a package is installed.

readme

This target generates a README.html file, which can be viewed using a browser such as www/mozilla or www/links. The generated files contain references to any packages which are in the PACKAGES directory on the local host. The generated files can be made to refer to URLs based on FTP_PKG_URL_HOST and FTP_PKG_URL_DIR. For example, if I wanted to generate README.html files which pointed to binary packages on the local machine, in the directory /usr/packages, set FTP_PKG_URL_HOST=file://localhost and FTP_PKG_URL_DIR=/usr/packages. The ${PACKAGES} directory and its subdirectories will be searched for all the binary packages.

readme-all

Use this target to create a file README-all.html which contains a list of all packages currently available in the NetBSD Packages Collection, together with the category they belong to and a short description. This file is compiled from the pkgsrc/*/README.html files, so be sure to run this *after* a **make readme**.

cdrom-readme

This is very much the same as the "readme" target (see above), but is to be used when generating a pkgsrc tree to be written to a CD-ROM. This target also produces `README.html` files, and can be made to refer to URLs based on `CDROM_PKG_URL_HOST` and `CDROM_PKG_URL_DIR`.

show-distfiles

This target shows which distfiles and patchfiles are needed to build the package. (`DISTFILES` and `PATCHFILES`, but not `patches/*`)

show-downlevel

This target shows nothing if the package is not installed. If a version of this package is installed, but is not the version provided in this version of pkgsrc, then a warning message is displayed. This target can be used to show which of your installed packages are downlevel, and so the old versions can be deleted, and the current ones added.

show-pkgsrc-dir

This target shows the directory in the pkgsrc hierarchy from which the package can be built and installed. This may not be the same directory as the one from which the package was installed. This target is intended to be used by people who may wish to upgrade many packages on a single host, and can be invoked from the top-level pkgsrc Makefile by using the "show-host-specific-pkgs" target.

show-installed-depends

This target shows which installed packages match the current package's `DEPENDS`. Useful if out of date dependencies are causing build problems.

check-shlibs

After a package is installed, check all its binaries and (on ELF platforms) shared libraries to see if they find the shared libs they need. Run by default if `PKG_DEVELOPER` is set in `/etc/mk.conf`.

print-PLIST

After a "make install" from a new or upgraded pkg, this prints out an attempt to generate a new `PLIST` from a **find -newer work/.extract_done**. An attempt is made to care for shared libs etc., but it is *strongly* recommended to review the result before putting it into `PLIST`. On upgrades, it's useful to diff the output of this command against an already existing `PLIST` file.

If the package installs files via tar(1) or other methods that don't update file access times, be sure to add these files manually to your `PLIST`, as the "find -newer" command used by this target won't catch them!

See Section 39.3 for more information on this target.

bulk-package

Used to do bulk builds. If an appropriate binary package already exists, no action is taken. If not, this target will compile, install and package it (and its depends, if `PKG_DEPENDS` is set properly. See Section 35.3.1). After creating the binary package, the sources, the just-installed package and its required packages are removed, preserving free disk space.

Beware that this target may deinstall all packages installed on a system!

bulk-install

Used during bulk-installs to install required packages. If an up-to-date binary package is available, it will be installed via pkg_add(1). If not, **make bulk-package** will be executed, but the installed binary won't be removed.

A binary package is considered "up-to-date" to be installed via pkg_add(1) if:

- None of the package's files (`Makefile`, ...) were modified since it was built.
- None of the package's required (binary) packages were modified since it was built.

Beware that this target may deinstall all packages installed on a system!

Chapter 44
Notes on fixes for packages

44.1 General operation

44.1.1 How to pull in variables from /etc/mk.conf

The problem with package-defined variables that can be overridden via MAKECONF or /etc/mk.conf is that make(1) expands a variable as it is used, but evaluates preprocessor-like statements (.if, .ifdef and .ifndef) as they are read. So, to use any variable (which may be set in /etc/mk.conf) in one of the .if* statements, the file /etc/mk.conf must be included before that .if* statement.

Rather than having a number of ad-hoc ways of including /etc/mk.conf, should it exist, or MAKECONF, should it exist, include the pkgsrc/mk/bsd.prefs.mk file in the package Makefile before any preprocessor-like .if, .ifdef, or .ifndef statements:

```
.include "../../mk/bsd.prefs.mk"

    .if defined(USE_MENUS)
      ...
    .endif
```

If you wish to set the CFLAGS variable in /etc/mk.conf, please make sure to use:

```
CFLAGS+=  -your -flags
```

Using CFLAGS= (i.e. without the "+") may lead to problems with packages that need to add their own flags. Also, you may want to take a look at the devel/cpuflags package if you're interested in optimization for the current CPU.

44.1.2 Where to install documentation

Documentation should be installed into ${PREFIX}/share/doc/${PKGBASE} or ${PREFIX}/share/doc/${PKGNAME} (the latter includes the version number of the package).

44.1.3 Restricted packages

Some licenses restrict how software may be re-distributed. In order to satisfy these restrictions, the package system defines five make variables that can be set to note these restrictions:

- RESTRICTED

This variable should be set whenever a restriction exists (regardless of its kind). Set this variable to a string containing the reason for the restriction.

- NO_BIN_ON_CDROM

  Binaries may not be placed on CD-ROM. Set this variable to ${RESTRICTED} whenever a binary package may not be included on a CD-ROM.

- NO_BIN_ON_FTP

  Binaries may not be placed on an FTP server. Set this variable to ${RESTRICTED} whenever a binary package may not not be made available on the Internet.

- NO_SRC_ON_CDROM

  Distfiles may not be placed on CD-ROM. Set this variable to ${RESTRICTED} if re-distribution of the source code or other distfile(s) is not allowed on CD-ROMs.

- NO_SRC_ON_FTP

  Distfiles may not be placed on FTP. Set this variable to ${RESTRICTED} if re-distribution of the source code or other distfile(s) via the Internet is not allowed.

Please note that the use of NO_PACKAGE, IGNORE, NO_CDROM, or other generic make variables to denote restrictions is deprecated, because they unconditionally prevent users from generating binary packages!

44.1.4 Handling dependencies

Your package may depend on some other package being present - and there are various ways of expressing this dependency. pkgsrc supports the BUILD_DEPENDS and DEPENDS definitions, the USE_TOOLS definition, as well as dependencies via buildlink3.mk, which is the preferred way to handle dependencies, and which uses the variables named above. See Chapter 40 for more information.

The basic difference between the two variables is as follows: The DEPENDS definition registers that pre-requisite in the binary package so it will be pulled in when the binary package is later installed, whilst the BUILD_DEPENDS definition does not, marking a dependency that is only needed for building the package.

This means that if you only need a package present whilst you are building, it should be noted as a BUILD_DEPENDS.

The format for a BUILD_DEPENDS and a DEPENDS definition is:

```
<pre-req-package-name>:../../<category>/<pre-req-package>
```

Please note that the “pre-req-package-name” may include any of the wildcard version numbers recognized by pkg_info(1).

1. If your package needs another package’s binaries or libraries to build or run, and if that package has a buildlink3.mk file available, use it:

   ```
   .include "../../graphics/jpeg/buildlink3.mk"
   ```

2. If your package needs to use another package to build itself and there is no buildlink3.mk file available, use the BUILD_DEPENDS definition:

```
BUILD_DEPENDS+=  autoconf-2.13:../../devel/autoconf
```

3. If your package needs a library with which to link and again there is no `buildlink3.mk` file available, this is specified using the `DEPENDS` definition. An example of this is the `print/lyx` package, which uses the xpm library, version 3.4j to build:

   ```
   DEPENDS+=        xpm-3.4j:../../graphics/xpm
   ```

   You can also use wildcards in package dependences:

   ```
   DEPENDS+=        xpm-[0-9]*:../../graphics/xpm
   ```

   Note that such wildcard dependencies are retained when creating binary packages. The dependency is checked when installing the binary package and any package which matches the pattern will be used. Wildcard dependencies should be used with care.

   The "-[0-9]*" should be used instead of "-*" to avoid potentially ambiguous matches such as "tk-postgresql" matching a "tk-*" `DEPENDS`.

   Wildcards can also be used to specify that a package will only build against a certain minimum version of a pre-requisite:

   ```
   DEPENDS+=        tiff>=3.5.4:../../graphics/tiff
   ```

   This means that the package will build against version 3.5.4 of the tiff library or newer. Such a dependency may be warranted if, for example, the API of the library has changed with version 3.5.4 and a package would not compile against an earlier version of tiff.

   Please note that such dependencies should only be updated if a package requires a newer pre-requisite, but not to denote recommendations such as security updates or ABI changes that do not prevent a package from building correctly. Such recommendations can be expressed using `RECOMMENDED`:

   ```
   RECOMMENDED+=   tiff>=3.6.1:../../graphics/tiff
   ```

   In addition to the above `DEPENDS` line, this denotes that while a package will build against tiff>=3.5.4, at least version 3.6.1 is recommended. `RECOMMENDED` entries will be turned into dependencies unless explicitly ignored (in which case a warning will be printed).

   To ignore these dependency recommendations and just use the required `DEPENDS`, set `IGNORE_RECOMMENDED=YES`. This may make it easier and faster to update packages built using pkgsrc, since older compatible dependencies can continue to be used. This is useful for people who watch their rebuilds very carefully; it is not very good as a general-purpose hammer. If you use it, you need to be mindful of possible ABI changes, including those from the underlying OS.

   Packages that are built with recommendations ignored may not be uploaded to ftp.NetBSD.org by developers and should not be used across different systems that may have different versions of binary packages installed.

   For security fixes, please update the package vulnerabilities file as well as setting `RECOMMENDED`, see Section 44.1.8 for more information.

4. If your package needs some executable to be able to run correctly and if there's no `buildlink3.mk` file, this is specified using the `DEPENDS` variable. The `print/lyx` package needs to be able to execute the latex binary from the teTeX package when it runs, and that is specified:

   ```
   DEPENDS+=         teTeX-[0-9]*:../../print/teTeX
   ```

   The comment about wildcard dependencies from previous paragraph applies here, too.

If your package needs files from another package to build, see the first part of the "do-configure" target print/ghostscript5 package (it relies on the jpeg sources being present in source form during the build):

```
if [ ! -e ${_PKGSRCDIR}/graphics/jpeg/${WRKDIR:T}/jpeg-6b ]; then \
          cd ${_PKGSRCDIR}/../../graphics/jpeg && ${MAKE} extract;              \
    fi
```

If you build any other packages that way, please make sure the working files are deleted too when this package's working files are cleaned up. The easiest way to do so is by adding a pre-clean target:

```
pre-clean:
          cd ${_PKGSRCDIR}/../../graphics/jpeg && ${MAKE} clean
```

Please also note the BUILD_USES_MSGFMT and BUILD_USES_GETTEXT_M4 definitions, which are provided as convenience definitions. The former works out whether msgfmt(1) is part of the base system, and, if it isn't, installs the devel/gettext package. The latter adds a build dependency on either an installed version of an older gettext package, or if it isn't, installs the devel/gettext-m4 package.

44.1.5 Handling conflicts with other packages

Your package may conflict with other packages a user might already have installed on his system, e.g. if your package installs the same set of files like another package in our pkgsrc tree.

In this case you can set CONFLICTS to a space-separated list of packages (including version string) your package conflicts with.

For example, x11/Xaw3d and x11/Xaw-Xpm install the same shared library, thus you set in pkgsrc/x11/Xaw3d/Makefile:

```
CONFLICTS=      Xaw-Xpm-[0-9]*
```

and in pkgsrc/x11/Xaw-Xpm/Makefile:

```
CONFLICTS=      Xaw3d-[0-9]*
```

Packages will automatically conflict with other packages with the name prefix and a different version string. "Xaw3d-1.5" e.g. will automatically conflict with the older version "Xaw3d-1.3".

44.1.6 Packages that cannot or should not be built

There are several reasons why a package might be instructed to not build under certain circumstances. If the package builds and runs on most platforms, the exceptions should be noted with NOT_FOR_PLATFORM. If the package builds and runs on a small handful of platforms, set ONLY_FOR_PLATFORM instead. Both ONLY_FOR_PLATFORM and NOT_FOR_PLATFORM are OS triples (OS-version-platform) that can use glob-style wildcards.

If the package should be skipped (for example, because it provides functionality already provided by the system), set PKG_SKIP_REASON to a descriptive message. If the package should fail because some preconditions are not met, set PKG_FAIL_REASON to a descriptive message.

44.1.7 Packages which should not be deleted, once installed

To ensure that a package may not be deleted, once it has been installed, the PKG_PRESERVE definition should be set in the package Makefile. This will be carried into any binary package that is made from this pkgsrc entry. A "preserved" package will not be deleted using pkg_delete(1) unless the "-f" option is used.

44.1.8 Handling packages with security problems

When a vulnerability is found, this should be noted in `localsrc/security/advisories/pkg-vulnerabilities`, and after committing that file, use **make upload** in the same directory to update the file on ftp.NetBSD.org.

After fixing the vulnerability by a patch, its PKGREVISION should be increased (this is of course not necessary if the problem is fixed by using a newer release of the software). In addition, if a `buildlink3.mk` file exists for an affected package, a corresponding BUILDLINK_RECOMMENDED.*pkg* entry should be added or updated in it.

Also, if the fix should be applied to the stable pkgsrc branch, be sure to submit a pullup request!

Binary packages already on ftp.NetBSD.org will be handled semi-automatically by a weekly cron job.

44.1.9 How to handle compiler bugs

Some source files trigger bugs in the compiler, based on combinations of compiler version and architecture and almost always relation to optimisation being enabled. Common symptoms are gcc internal errors or never finishing compiling a file.

Typically, a workaround involves testing the MACHINE_ARCH and compiler version, disabling optimisation for that file/MACHINE_ARCH/compiler combination, and documenting it in `pkgsrc/doc/HACKS`. See that file for a number of examples!

44.1.10 How to handle incrementing versions when fixing an existing package

When making fixes to an existing package it can be useful to change the version number in PKGNAME. To avoid conflicting with future versions by the original author, a "nb1", "nb2", ... suffix can be used on package versions by setting PKGREVISION=1 (2, ...). The "nb" is treated like a "." by the pkg tools. e.g.

```
DISTNAME= foo-17.42
  PKGREVISION=  9
```

will result in a PKGNAME of "foo-17.42nb9".

When a new release of the package is released, the PKGREVISION should be removed, e.g. on a new minor release of the above package, things should be like:

```
DISTNAME=   foo-17.43
```

44.1.11 Portability of packages

One appealing feature of pkgsrc is that it runs on many different platforms. As a result, it is important to ensure, where possible, that packages in pkgsrc are portable. There are some particular details you should pay attention to while working on pkgsrc.

44.1.11.1 ${INSTALL}, ${INSTALL_DATA_DIR}, ...

The BSD-compatible **install** supplied with some operating systems will not perform more than one operation at a time. As such, you should call "${INSTALL}", etc. like this:

```
${INSTALL_DATA_DIR} ${PREFIX}/dir1
${INSTALL_DATA_DIR} ${PREFIX}/dir2
```

44.2 Possible downloading issues

44.2.1 Packages whose distfiles aren't available for plain downloading

If you need to download from a dynamic URL you can set `DYNAMIC_MASTER_SITES` and a **make fetch** will call `files/getsite.sh` with the name of each file to download as an argument, expecting it to output the URL of the directory from which to download it. `graphics/ns-cult3d` is an example of this usage.

If the download can't be automated, because the user must submit personal information to apply for a password, or must pay for the source, or whatever, you can set `_FETCH_MESSAGE` to a macro which displays a message explaining the situation. `_FETCH_MESSAGE` must be executable shell commands, not just a message. (Generally, it executes `${ECHO}`). As of this writing, the following packages use this: `cad/simian`, `devel/ipv6socket`, `emulators/vmware-module`, `fonts/acroread-jpnfont`, `multimedia/realplayer`, `sysutils/storage-manager`, `www/ap-aolserver`, `www/openacs`. Try to be consistent with them.

44.2.2 How to handle modified distfiles with the 'old' name

Sometimes authors of a software package make some modifications after the software was released, and they put up a new distfile without changing the package's version number. If a package is already in pkgsrc at that time, the checksum will no longer match. The contents of the new distfile should be compared against the old one before changing anything, to make sure the distfile was really updated on purpose, and that no trojan horse or so crept in. Then, the correct way to work around this is to set `DIST_SUBDIR` to a unique directory name, usually based on `PKGNAME_NOREV`. In case this happens more often, `PKGNAME` can be used (thus including the `nbX` suffix) or a date stamp can be appended, like `${PKGNAME_NOREV}-YYYYMMDD`. Do not forget regenerating the `distinfo` file after that, since it contains the `DIST_SUBDIR` path in the filenames. Furthermore, a mail to the package's authors seems appropriate telling them that changing distfiles after releases without changing the file names is not good practice.

44.3 Configuration gotchas

44.3.1 Shared libraries - libtool

pkgsrc supports many different machines, with different object formats like a.out and ELF, and varying abilities to do shared library and dynamic loading at all. To accompany this, varying commands and options have to be passed to the compiler, linker, etc. to get the Right Thing, which can be pretty annoying especially if you don't have all the machines at your hand to test things. The `devel/libtool` pkg can help here, as it just "knows" how to build both static and dynamic libraries from a set of source files, thus being platform-independent.

Here's how to use libtool in a pkg in seven simple steps:

1. Add `USE_LIBTOOL=yes` to the package Makefile.
2. For library objects, use "${LIBTOOL} --mode=compile ${CC}" in place of "${CC}". You could even add it to the definition of `CC`, if only libraries are being built in a given Makefile. This one command will build both PIC and non-PIC library objects, so you need not have separate shared and non-shared library rules.
3. For the linking of the library, remove any "ar", "ranlib", and "ld -Bshareable" commands, and instead use:

   ```
   ${LIBTOOL} --mode=link ${CC} -o ${.TARGET:.a=.la} ${OBJS:.o=.lo} -rpath ${PREFIX}/lib -v
   ```

   Note that the library is changed to have a `.la` extension, and the objects are changed to have a `.lo` extension. Change `OBJS` as necessary. This automatically creates all of the `.a`, `.so.major.minor`, and ELF symlinks (if necessary) in the build directory. Be sure to include "-version-info", especially when major and minor are zero, as libtool will otherwise strip off the shared library version.

   From the libtool manual:

   ```
        So, libtool library versions are described by three integers:

   CURRENT
          The most recent interface number that this library implements.

   REVISION
          The implementation number of the CURRENT interface.

   AGE
          The difference between the newest and oldest interfaces that this
          library implements.  In other words, the library implements all the
          interface numbers in the range from number 'CURRENT - AGE' to
          'CURRENT'.

        If two libraries have identical CURRENT and AGE numbers, then the
   dynamic linker chooses the library with the greater REVISION number.
   ```

   The "-release" option will produce different results for a.out and ELF (excluding symlinks) in only one case. An ELF library of the form "libfoo-release.so.*x*.*y*" will have a symlink of "libfoo.so.*x*.*y*" on an a.out platform. This is handled automatically.

   The "-rpath argument" is the install directory of the library being built.

In the PLIST, include only the .la file, the other files will be added automatically.

4. When linking shared object (.so) files, i.e. files that are loaded via dlopen(3), NOT shared libraries, use "-module -avoid-version" to prevent them getting version tacked on.

   The PLIST file gets the foo.so entry.

5. When linking programs that depend on these libraries *before* they are installed, preface the cc(1) or ld(1) line with "${LIBTOOL} --mode=link", and it will find the correct libraries (static or shared), but please be aware that libtool will not allow you to specify a relative path in -L (such as "-L../somelib"), because it expects you to change that argument to be the .la file. e.g.

   ```
   ${LIBTOOL} --mode=link ${CC} -o someprog -L../somelib -lsomelib
   ```

   should be changed to:

   ```
   ${LIBTOOL} --mode=link ${CC} -o someprog ../somelib/somelib.la
   ```

   and it will do the right thing with the libraries.

6. When installing libraries, preface the install(1) or cp(1) command with "${LIBTOOL} --mode=install", and change the library name to .la. e.g.

   ```
   ${LIBTOOL} --mode=install ${BSD_INSTALL_DATA} ${SOMELIB:.a=.la} ${PREFIX}/lib
   ```

   This will install the static .a, shared library, any needed symlinks, and run ldconfig(8).

7. In your PLIST, include only the .la file (this is a change from previous behaviour).

44.3.2 Using libtool on GNU packages that already support libtool

Add USE_LIBTOOL=yes to the package Makefile. This will override the package's own libtool in most cases. For older libtool using packages, libtool is made by ltconfig script during the do-configure step; you can check the libtool script location by doing **make configure; find work*/ -name libtool**.

LIBTOOL_OVERRIDE specifies which libtool scripts, relative to WRKSRC, to override. By default, it is set to "libtool */libtool */*/libtool". If this does not match the location of the package's libtool script(s), set it as appropriate.

If you do not need *.a static libraries built and installed, then use SHLIBTOOL_OVERRIDE instead.

If your package makes use of the platform-independent library for loading dynamic shared objects, that comes with libtool (libltdl), you should include devel/libltdl/buildlink3.mk.

Some packages use libtool incorrectly so that the package may not work or build in some circumstances. Some of the more common errors are:

- The inclusion of a shared object (-module) as a dependent library in an executable or library. This in itself isn't a problem if one of two things has been done:

  1. The shared object is named correctly, i.e. libfoo.la, not foo.la
  2. The -dlopen option is used when linking an executable.

- The use of libltdl without the correct calls to initialisation routines. The function lt_dlinit() should be called and the macro LTDL_SET_PRELOADED_SYMBOLS included in executables.

44.3.3 GNU Autoconf/Automake

If a package needs GNU autoconf or automake to be executed to regenerate the configure script and Makefile.in makefile templates, then they should be executed in a pre-configure target.

For packages that need only autoconf:

```
AUTOCONF_REQD=    2.50     # if default version is not good enough
USE_TOOLS+= autoconf   # use "autoconf213" for autoconf-2.13
...

pre-configure:
          cd ${WRKSRC}; autoconf

...
```

and for packages that need automake and autoconf:

```
AUTOMAKE_REQD=  1.7.1      # if default version is not good enough
USE_TOOLS+= automake   # use "automake14" for automake-1.4
...

pre-configure:
          cd ${WRKSRC};                                          \
          aclocal; autoheader;                                   \
          automake -a --foreign -i; autoconf

...
```

Packages which use GNU Automake will almost certainly require GNU Make.

There are times when the configure process makes additional changes to the generated files, which then causes the build process to try to re-execute the automake sequence. This is prevented by touching various files in the configure stage. If this causes problems with your package you can set `AUTOMAKE_OVERRIDE=NO` in the package Makefile.

44.4 Building considerations

44.4.1 CPP defines

To port an application to NetBSD, it's usually necessary for the compiler to be able to judge the system on which it's compiling, and we use definitions so that the C pre-processor can do this.

To test whether you are working on a 4.4 BSD-derived system, you should use the BSD definition, which is defined in `<sys/param.h>` on said systems.

```
#include <sys/param.h>
```

and then you can surround the BSD-specific parts of your package's C/C++ code using this conditional:

```
#if (defined(BSD) && BSD >= 199306)
    ...
```

```
#endif
```

Please use the "__NetBSD__" definition sparingly - it should only apply to features of NetBSD that are not present in other 4.4-lite-derived BSDs.

44.5 Package specific actions

44.5.1 User interaction

Occasionally, packages require interaction from the user, and this can be in a number of ways:

- help in fetching the distfiles
- help to configure the package before it is built
- help during the build process
- help during the installation of a package

The INTERACTIVE_STAGE definition is provided to notify the pkgsrc mechanism of an interactive stage which will be needed, and this should be set in the package's Makefile, e.g.:

```
INTERACTIVE_STAGE= build
```

Multiple interactive stages can be specified:

```
INTERACTIVE_STAGE= configure install
```

44.5.2 Handling licenses

A package may be covered by a license which the user has or has not agreed to accept. For these cases, pkgsrc contains a mechanism to note that a package is covered by a particular license, and the package cannot be built unless the user has accepted the license. (Installation of binary packages are not currently subject to this mechanism.) Packages with licenses that are either Open Source according to the Open Source Initiative or Free according to the Free Software Foundation will not be marked with a license tag. Packages with licenses that have not been determined to meet either definition will be marked with a license tag referring to the license. This will prevent building unless pkgsrc is informed that the license is acceptable, and enables displaying the license.

The license tag mechanism is intended to address copyright-related issues surrounding building, installing and using a package, and not to address redistribution issues (see RESTRICTED and NO_SRC_ON_FTP, etc.). However, the above definition of licenses for which tags are not needed implies that packages with redistribution restrictions should have tags.

Denoting that a package is covered by a particular license is done by placing the license in pkgsrc/licenses and setting the LICENSE variable to a string identifying the license, e.g. in graphics/xv:

```
LICENSE=        xv-license
```

When trying to build, the user will get a notice that the package is covered by a license which has not been accepted:

```
% make
===> xv-3.10anb9 has an unacceptable license: xv-license.
===>     To view the license, enter "/usr/bin/make show-license".
===>     To indicate acceptance, add this line to your /etc/mk.conf:
===>     ACCEPTABLE_LICENSES+=xv-license
*** Error code 1
```

The license can be viewed with **make show-license**, and if it is considered appropriate, the line printed above can be added to `/etc/mk.conf` to indicate acceptance of the particular license:

```
ACCEPTABLE_LICENSES+=xv-license
```

When adding a package with a new license, the license text should be added to `pkgsrc/licenses` for displaying. A list of known licenses can be seen in this directory as well as by looking at the list of (commented out) `ACCEPTABLE_LICENSES` variable settings in `pkgsrc/mk/defaults/mk.conf`.

The use of `LICENSE=shareware`, `LICENSE=no-commercial-use`, and similar language is deprecated because it does not crisply refer to a particular license text. Another problem with such usage is that it does not enable a user to denote acceptance of the license for a single package without accepting the same license text for another package. In particular, this can be inappropriate when e.g. one accepts a particular license to indicate to pkgsrc that a fee has been paid.

44.5.3 Installing score files

Certain packages, most of them in the games category, install a score file that allows all users on the system to record their highscores. In order for this to work, the binaries need to be installed setgid and the score files owned by the appropriate group and/or owner (traditionally the "games" user/group). The following variables, documented in more detail in `mk/defaults/mk.conf`, control this behaviour: `SETGIDGAME`, `GAMEDATAMODE`, `GAMEGRP`, `GAMEMODE`, `GAMEOWN`.

Note that per default, setgid installation of games is disabled; setting `SETGIDGAME=YES` will set all the other variables accordingly.

A package should therefor never hard code file ownership or access permissions but rely on `INSTALL_GAME` and `INSTALL_GAME_DATA` to set these correctly.

44.5.4 Packages containing perl scripts

If your package contains interpreted perl scripts, set `REPLACE_PERL` to ensure that the proper interpreter path is set. `REPLACE_PERL` should contain a list of scripts, relative to `WRKSRC`, that you want adjusted.

44.5.5 Packages with hardcoded paths to other interpreters

Your package may also contain scripts with hardcoded paths to other interpreters besides (or as well as) perl. To correct the full pathname to the script interpreter, you need to set the following definitions in your `Makefile` (we shall use **tclsh** in this example):

```
REPLACE_INTERPRETER+=     tcl
  _REPLACE.tcl.old=      .*/bin/tclsh
  _REPLACE.tcl.new=      ${PREFIX}/bin/tclsh
  _REPLACE_FILES.tcl=    ...list of tcl scripts which need to be fixed,
                         relative to ${WRKSRC}, just as in REPLACE_PERL
```

44.5.6 Packages installing perl modules

Makefiles of packages providing perl5 modules should include the Makefile fragment `../../lang/perl5/module.mk`. It provides a **do-configure** target for the standard perl configuration for such modules as well as various hooks to tune this configuration. See comments in this file for details.

Perl5 modules will install into different places depending on the version of perl used during the build process. To address this, pkgsrc will append lines to the `PLIST` corresponding to the files listed in the installed `.packlist` file generated by most perl5 modules. This is invoked by defining `PERL5_PACKLIST` to a space-separated list of paths to packlist files, e.g.:

```
PERL5_PACKLIST=    ${PERL5_SITEARCH}/auto/Pg/.packlist
```

The variables `PERL5_SITELIB`, `PERL5_SITEARCH`, and `PERL5_ARCHLIB` represent the three locations in which perl5 modules may be installed, and may be used by perl5 packages that don't have a packlist. These three variables are also substituted for in the `PLIST`.

44.5.7 Packages installing info files

Some packages install info files or use the "makeinfo" or "install-info" commands. Each of the info files:

- is considered to be installed in the directory `${PREFIX}/${INFO_DIR}`,
- is registered in the Info directory file `${PREFIX}/${INFO_DIR}/dir`,
- and must be listed as a filename in the `INFO_FILES` variable in the package Makefile.

`INFO_DIR` defaults to "info" and can be overridden in the package Makefile. `INSTALL` and `DEINSTALL` scripts will be generated to handle registration of the info files in the Info directory file. The "install-info" command used for the info files registration is either provided by the system, or by a special purpose package automatically added as dependency if needed.

A package which needs the "makeinfo" command at build time must define the variable `USE_MAKEINFO` in its Makefile. If a minimum version of the "makeinfo" command is needed it should be noted with the `TEXINFO_REQD` variable in the package `Makefile`. By default, a minimum version of 3.12 is required. If the system does not provide a **makeinfo** command or if it does not match the required minimum, a build dependency on the `devel/gtexinfo` package will be added automatically.

The build and installation process of the software provided by the package should not use the **install-info** command as the registration of info files is the task of the package `INSTALL` script, and it must use the appropriate **makeinfo** command.

To achieve this goal, the pkgsrc infrastructure creates overriding scripts for the **install-info** and **makeinfo** commands in a directory listed early in `PATH`.

The script overriding **install-info** has no effect except the logging of a message. The script overriding **makeinfo** logs a message and according to the value of USE_MAKEINFO and TEXINFO_REQD either run the appropriate **makeinfo** command or exit on error.

44.5.8 Packages installing GConf2 data files

If a package installs .schemas or .entries files, used by GConf2, you need to take some extra steps to make sure they get registered in the database:

1. Include ../../devel/GConf2/schemas.mk instead of its buildlink3.mk file. This takes care of rebuilding the GConf2 database at installation and deinstallation time, and tells the package where to install GConf2 data files using some standard configure arguments. It also disallows any access to the database directly from the package.
2. Ensure that the package installs its .schemas files under ${PREFIX}/share/gconf/schemas. If they get installed under ${PREFIX}/etc, you will need to manually patch the package.
3. Check the PLIST and remove any entries under the etc/gconf directory, as they will be handled automatically. See Section 36.14 for more information.
4. Define the GCONF2_SCHEMAS variable in your Makefile with a list of all .schemas files installed by the package, if any. Names must not contain any directories in them.
5. Define the GCONF2_ENTRIES variable in your Makefile with a list of all .entries files installed by the package, if any. Names must not contain any directories in them.

44.5.9 Packages installing scrollkeeper data files

If a package installs .omf files, used by scrollkeeper, you need to take some extra steps to make sure they get registered in the database:

1. Include ../../textproc/scrollkeeper/omf.mk instead of its buildlink3.mk file. This takes care of rebuilding the scrollkeeper database at installation and deinstallation time, and disallows any access to it directly from the package.
2. Check the PLIST and remove any entries under the libdata/scrollkeeper directory, as they will be handled automatically.
3. Remove the share/omf directory from the PLIST. It will be handled by scrollkeeper.

44.5.10 Packages installing X11 fonts

If a package installs font files, you will need to rebuild the fonts database in the directory where they get installed at installation and deinstallation time. This can be automatically done by using mk/fonts.mk, which you need to include in your Makefile.

When the file is included, you can list the directories where fonts are installed in the FONTS_*type*_DIRS variables, where *type* can be one of “TTF”, “TYPE1” or “X11”. Also make sure that the database file fonts.dir is not listed in the PLIST.

Note that you should not create new directories for fonts; instead use the standard ones to avoid that the user needs to manually configure his X server to find them.

44.5.11 Packages installing GTK2 modules

If a package installs GTK2 immodules or loaders, you need to take some extra steps to get them registered in the GTK2 database properly:

1. Include `../../x11/gtk2/modules.mk` instead of its `buildlink3.mk` file. This takes care of rebuilding the database at installation and deinstallation time.
2. Set `GTK2_IMMODULES=YES` if your package installs GTK2 immodules.
3. Set `GTK2_LOADERS=YES` if your package installs GTK2 loaders.
4. Patch the package to not touch any of the GTK2 databases directly. These are:
   - `libdata/gtk-2.0/gdk-pixbuf.loaders`
   - `libdata/gtk-2.0/gtk.immodules`
5. Check the PLIST and remove any entries under the `libdata/gtk-2.0` directory, as they will be handled automatically.

44.5.12 Packages installing SGML or XML data

If a package installs SGML or XML data files that need to be registered in system-wide catalogs (like DTDs, sub-catalogs, etc.), you need to take some extra steps:

1. Include `../../textproc/xmlcatmgr/catalogs.mk` in your `Makefile`, which takes care of registering those files in system-wide catalogs at installation and deinstallation time.
2. Set `SGML_CATALOGS` to the full path of any SGML catalogs installed by the package.
3. Set `XML_CATALOGS` to the full path of any XML catalogs installed by the package.
4. Set `SGML_ENTRIES` to individual entries to be added to the SGML catalog. These come in groups of three strings; see xmlcatmgr(1) for more information (specifically, arguments recognized by the 'add' action). Note that you will normally not use this variable.
5. Set `XML_ENTRIES` to individual entries to be added to the XML catalog. These come in groups of three strings; see xmlcatmgr(1) for more information (specifically, arguments recognized by the 'add' action). Note that you will normally not use this variable.

44.5.13 Packages installing extensions to the MIME database

If a package provides extensions to the MIME database by installing `.xml` files inside `${PREFIX}/share/mime/packages`, you need to take some extra steps to ensure that the database is kept consistent with respect to these new files:

1. Include `../../databases/shared-mime-info/mimedb.mk` (avoid using the `buildlink3.mk` file from this same directory, which is reserved for inclusion from other `buildlink3.mk` files). It takes care of rebuilding the MIME database at installation and deinstallation time, and disallows any access to it directly from the package.
2. Check the PLIST and remove any entries under the `share/mime` directory, *except* for files saved under `share/mime/packages`. The former are handled automatically by the update-mime-database program, but the latter are package-dependent and must be removed by the package that installed them in the first place.
3. Remove any `share/mime/*` directories from the PLIST. They will be handled by the shared-mime-info package.

44.5.14 Packages using intltool

If a package uses intltool during its build, include the `../../textproc/intltool/buildlink3.mk` file, which forces it to use the intltool package provided by pkgsrc, instead of the one bundled with the distribution file.

This tracks intltool's build-time dependencies and uses the latest available version; this way, the package benefits of any bug fixes that may have appeared since it was released.

44.5.15 Packages installing startup scripts

If a package contains a rc.d script, it won't be copied into the startup directory by default, but you can enable it, by adding the option `PKG_RCD_SCRIPTS=YES` in `/etc/mk.conf`. This option will copy the scripts into `/etc/rc.d` when a package is installed, and it will automatically remove the scripts when the package is deinstalled.

44.6 Feedback to the author

If you have found any bugs in the package you make available, if you had to do special steps to make it run under NetBSD or if you enhanced the software in various other ways, be sure to report these changes back to the original author of the program! With that kind of support, the next release of the program can incorporate these fixes, and people not using the NetBSD packages system can win from your efforts.

Support the idea of free software!

Chapter 45
Debugging

To check out all the gotchas when building a package, here are the steps that I do in order to get a package working. Please note this is basically the same as what was explained in the previous sections, only with some debugging aids.

- Be sure to set `PKG_DEVELOPER=1` in `/etc/mk.conf`
- Install `pkgtools/url2pkg`, create a directory for a new package, change into it, then run **url2pkg**:

```
% mkdir /usr/pkgsrc/category/examplepkg
% cd /usr/pkgsrc/category/examplepkg
% url2pkg http://www.example.com/path/to/distfile.tar.gz
```

- Edit the `Makefile` as requested.
- Fill in the `DESCR` file
- Run **make configure**
- Add any dependencies glimpsed from documentation and the configure step to the package's `Makefile`.
- Make the package compile, doing multiple rounds of

```
% make
% pkgvi ${WRKSRC}/some/file/that/does/not/compile
% mkpatches
% patchdiff
% mv ${WRKDIR}/.newpatches/* patches
% make mps
% make clean
```

  Doing as non-root user will ensure that no files are modified that shouldn't be, especially during the build phase. **mkpatches**, **patchdiff** and **pkgvi** are from the `pkgtools/pkgdiff` package.
- Look at the `Makefile`, fix if necessary; see Section 37.1.
- Generate a `PLIST`:

```
# make install
# make print-PLIST >PLIST
# make deinstall
# make install
# make deinstall
```

  You usually need to be `root` to do this. Look if there are any files left:

```
# make print-PLIST
```

  If this reveals any files that are missing in `PLIST`, add them.
- Now that the `PLIST` is OK, install the package again and make a binary package:

```
# make reinstall
# make package
```

- Delete the installed package:

```
# pkg_delete blub
```

- Repeat the above **make print-PLIST** command, which shouldn't find anything now:

```
# make print-PLIST
```

- Reinstall the binary package:

```
# pkgadd .../blub.tgz
```

- Play with it. Make sure everything works.
- Run **pkglint** from `pkgtools/pkglint`, and fix the problems it reports:

```
# pkglint
```

- Submit (or commit, if you have cvs access); see Chapter 46.

Chapter 46
Submitting and Committing

46.1 Submitting your packages

You have to separate between binary and “normal” (source) packages here:

- precompiled binary packages

  Our policy is that we accept binaries only from pkgsrc developers to guarantee that the packages don’t contain any trojan horses etc. This is not to annoy anyone but rather to protect our users! You’re still free to put up your home-made binary packages and tell the world where to get them. NetBSD developers doing bulk builds and wanting to upload them please see Section 35.3.8.

- packages

  First, check that your package is complete, compiles and runs well; see Chapter 45 and the rest of this document. Next, generate an uuencoded gzipped tar(1) archive, preferably with all files in a single directory. Finally, **send-pr** with category “pkg”, a synopsis which includes the package name and version number, a short description of your package (contents of the COMMENT variable or DESCR file are OK) and attach the archive to your PR.

  If you want to submit several packages, please send a separate PR for each one, it’s easier for us to track things that way.

  Alternatively, you can also import new packages into pkgsrc-wip (“pkgsrc work-in-progress”); see the homepage at http://pkgsrc-wip.sourceforge.net/ for details.

46.2 General notes when adding, updating, or removing packages

Please note all package additions, updates, moves, and removals in `pkgsrc/doc/CHANGES`. It’s very important to keep this file up to date and conforming to the existing format, because it will be used by scripts to automatically update pages on www.NetBSD.org (http://www.NetBSD.org/) and other sites. Additionally, check the `pkgsrc/doc/TODO` file and remove the entry for the package you updated or removed, in case it was mentioned there.

There is a make target that helps in creating proper `CHANGES` entries: **make changes-entry**. It uses the optional `CTYPE` and `NETBSD_LOGIN_NAME` variables. The general usage is to first make sure that your `CHANGES` file is up-to-date (to avoid having to resolve conflicts later-on) and then to **cd** to the package directory. For package updates, **make changes-entry** is enough. For new packages, or package moves or removals, set the `CTYPE` variable on the command line to "Added", "Moved", or "Removed". You can set `NETBSD_LOGIN_NAME` in `/etc/mk.conf` if your local login name is not the same as your NetBSD login name. Don’t forget to commit the changes to `pkgsrc/doc/CHANGES`!

46.3 Committing: Importing a package into CVS

This section is only of interest for pkgsrc developers with write access to the pkgsrc repository. Please remember that cvs imports files relative to the current working directory, and that the pathname that you give the **cvs import** command is so that it knows where to place the files in the repository. Newly created packages should be imported with a vendor tag of "TNF" and a release tag of "pkgsrc-base", e.g:

```
% cd .../pkgsrc/category/pkgname
% cvs import pkgsrc/category/pkgname TNF pkgsrc-base
```

Remember to move the directory from which you imported out of the way, or cvs will complain the next time you "cvs update" your source tree. Also don't forget to add the new package to the category's `Makefile`.

The commit message of the initial import should include part of the `DESCR` file, so people reading the mailing lists know what the package is/does.

For new packages, "cvs import" is preferred to "cvs add" because the former gets everything with a single command, and provides a consistent tag.

46.4 Updating a package to a newer version

Please always put a concise, appropriate and relevant summary of the changes between old and new versions into the commit log when updating a package. There are various reasons for this:

- A URL is volatile, and can change over time. It may go away completely or its information may be overwritten by newer information.
- Having the change information between old and new versions in our CVS repository is very useful for people who use either cvs or anoncvs.
- Having the change information between old and new versions in our CVS repository is very useful for people who read the pkgsrc-changes mailing list, so that they can make tactical decisions about when to upgrade the package.

Please also recognize that, just because a new version of a package has been released, it should not automatically be upgraded in the CVS repository. We prefer to be conservative in the packages that are included in pkgsrc - development or beta packages are not really the best thing for most places in which pkgsrc is used. Please use your judgement about what should go into pkgsrc, and bear in mind that stability is to be preferred above new and possibly untested features.

46.5 Moving a package in pkgsrc

1. Make a copy of the directory somewhere else.
2. Remove all CVS dirs.

   Alternatively to the first two steps you can also do:

   ```
   % cvs -d user@cvs.NetBSD.org:/cvsroot export -D today pkgsrc/category/package
   ```

and use that for further work.

3. Fix `CATEGORIES` and any `DEPENDS` paths that just did "../package" instead of "../../category/package".
4. **cvs import** the modified package in the new place.
5. Check if any package depends on it:

   ```
   % cd /usr/pkgsrc
   % grep /package */*/Makefile* */*/buildlink*
   ```

6. Fix paths in packages from step 5 to point to new location.
7. **cvs rm (-f)** the package at the old location.
8. Remove from `oldcategory/Makefile`.
9. Add to `newcategory/Makefile`.
10. Commit the changed and removed files:

    ```
    % cvs commit oldcategory/package oldcategory/Makefile newcategory/Makefile
    ```

    (and any packages from step 5, of course).

Appendix A.
Information

A.1 Where to get this document

This document is currently available in the following formats:

- HTML (http://www.NetBSD.org/guide/en/index.html)
- gzip'd PDF (http://www.NetBSD.org/guide/download/netbsd-en.pdf.gz)
- gzip'd PostScript (http://www.NetBSD.org/guide/download/netbsd-en.ps.gz)

In addition, this guide is also sold on occasion in printed form at tradeshows and exhibitons, with all profits being donated to the NetBSD Foundation. On demand printing may at some point be available as well. If you are interested in obtaining a printed and bound copy of this document, please contact <www@NetBSD.org>.

A.2 Guide history

This guide was born as a collection of sparse notes that Federico Lupi, the original author of the NetBSD Guide, wrote mostly for himself. When he realized that they could be useful to other NetBSD users he started collecting them and created the first version of the guide using the groff formatter. In order to "easily" get a wider variety of output formats (e.g. HTML and PostScript/PDF), he made the "mistake" of moving to SGML/DocBook, which is the current format of the sources. Maintainership was picked up by the NetBSD project and its developers later, and the format was changed to XML/DocBook later due to better tools and slightly more knowhow on customisations.

The following open source tools were used to write and format the guide:

- the vi editor which ships with NetBSD (nvi).
- the libxslt parser from GNOME for transforming XML/DocBook into HTML.
- the TeX system from the NetBSD packages collection. TeX is used as a backend to produce the PS and PDF formats.
- the tgif program for drawing the figures.
- the gimp and xv programs for converting between image formats and making small modifications to the figures.

Many thanks to all the people involved in the development of these great tools.

Appendix B.
A simple example package: bison

We checked to find a piece of software that wasn't in the packages collection, and picked GNU bison. Quite why someone would want to have **bison** when Berkeley **yacc** is already present in the tree is beyond us, but it's useful for the purposes of this exercise.

B.1 files

B.1.1 Makefile

```
# $NetBSD$
#

DISTNAME=       bison-1.25
CATEGORIES=     devel
MASTER_SITES=   ${MASTER_SITE_GNU}

MAINTAINER=     thorpej@NetBSD.org
HOMEPAGE=       http://www.gnu.org/software/bison/bison.html
COMMENT=        GNU yacc clone

GNU_CONFIGURE=  yes
INFO_FILES=     bison.info

.include "../../mk/bsd.pkg.mk"
```

B.1.2 DESCR

```
GNU version of yacc.  Can make re-entrant parsers, and numerous other
improvements.  Why you would want this when Berkeley yacc(1) is part
of the NetBSD source tree is beyond me.
```

B.1.3 PLIST

```
@comment $NetBSD$
bin/bison
man/man1/bison.1.gz
```

```
share/bison.simple
share/bison.hairy
```

B.1.4 Checking a package with pkglint

The NetBSD package system comes with `pkgtools/pkglint` which helps to check the contents of these files. After installation it is quite easy to use, just change to the directory of the package you wish to examine and execute **pkglint**:

```
$ pkglint
looks fine.
```

Depending on the supplied command line arguments (see pkglint(1)), more verbose checks will be performed. Use e.g. **pkglint -v** for a very verbose check.

B.2 Steps for building, installing, packaging

Create the directory where the package lives, plus any auxiliary directories:

```
# cd /usr/pkgsrc/lang
# mkdir bison
# cd bison
# mkdir patches
```

Create `Makefile`, `DESCR` and `PLIST` (see Chapter 37) then continue with fetching the distfile:

```
# make fetch
>> bison-1.25.tar.gz doesn't seem to exist on this system.
>> Attempting to fetch from ftp://prep.ai.mit.edu/pub/gnu//.
Requesting ftp://prep.ai.mit.edu/pub/gnu//bison-1.25.tar.gz (via ftp://orpheus.amdahl.com:80
ftp: Error retrieving file: 500 Internal error

>> Attempting to fetch from ftp://wuarchive.wustl.edu/systems/gnu//.
Requesting ftp://wuarchive.wustl.edu/systems/gnu//bison-1.25.tar.gz (via ftp://orpheus.amdah
ftp: Error retrieving file: 500 Internal error

>> Attempting to fetch from ftp://ftp.freebsd.org/pub/FreeBSD/distfiles//.
Requesting ftp://ftp.freebsd.org/pub/FreeBSD/distfiles//bison-1.25.tar.gz (via ftp://orpheus
Successfully retrieved file.
```

Generate the checksum of the distfile into `distinfo`:

```
# make makesum
```

Now compile:

```
# make
>> Checksum OK for bison-1.25.tar.gz.
===>  Extracting for bison-1.25
===>  Patching for bison-1.25
```

```
===>   Ignoring empty patch directory
===>  Configuring for bison-1.25
creating cache ./config.cache
checking for gcc... cc
checking whether we are using GNU C... yes
checking for a BSD compatible install... /usr/bin/install -c -o bin -g bin
checking how to run the C preprocessor... cc -E
checking for minix/config.h... no
checking for POSIXized ISC... no
checking whether cross-compiling... no
checking for ANSI C header files... yes
checking for string.h... yes
checking for stdlib.h... yes
checking for memory.h... yes
checking for working const... yes
checking for working alloca.h... no
checking for alloca... yes
checking for strerror... yes
updating cache ./config.cache
creating ./config.status
creating Makefile
===>  Building for bison-1.25
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DXPFILE=\"/usr/pkg/share/bison.simple\"  -DXPFILE1=\"/usr/pkg/share/bison.hairy\" -DS
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc -c -DSTDC_HEADERS=1 -DHAVE_STRING_H=1 -DHAVE_STDLIB_H=1 -DHAVE_MEMORY_H=1 -DHAVE_ALLOCA=1
cc  -g -o bison LR0.o allocate.o closure.o conflicts.o derives.o files.o         getargs.o g
./files.c:240: warning: mktemp() possibly used unsafely, consider using mkstemp()
rm -f bison.s1
sed -e "/^#line/ s|bison|/usr/pkg/share/bison|" < ./bison.simple > bison.s1
```

Everything seems OK, so install the files:

```
# make install
>> Checksum OK for bison-1.25.tar.gz.
===>  Installing for bison-1.25
```

```
sh ./mkinstalldirs /usr/pkg/bin /usr/pkg/share  /usr/pkg/info /usr/pkg/man/man1
rm -f /usr/pkg/bin/bison
cd /usr/pkg/share; rm -f bison.simple bison.hairy
rm -f /usr/pkg/man/man1/bison.1 /usr/pkg/info/bison.info*
install -c  -o bin -g bin -m 555 bison /usr/pkg/bin/bison
/usr/bin/install -c -o bin -g bin -m 644 bison.s1 /usr/pkg/share/bison.simple
/usr/bin/install -c -o bin -g bin -m 644 ./bison.hairy /usr/pkg/share/bison.hairy
cd .; for f in bison.info*;  do /usr/bin/install -c -o bin -g bin -m 644 $f /usr/pkg/info/$f
/usr/bin/install -c -o bin -g bin -m 644 ./bison.1 /usr/pkg/man/man1/bison.1
===>  Registering installation for bison-1.25
```

You can now use bison, and also - if you decide so - remove it with **pkg_delete bison**. Should you decide that you want a binary package, do this now:

```
# make package
>> Checksum OK for bison-1.25.tar.gz.
===>  Building package for bison-1.25
Creating package bison-1.25.tgz
Registering depends:.
Creating gzip'd tar ball in '/u/pkgsrc/lang/bison/bison-1.25.tgz'
```

Now that you don't need the source and object files any more, clean up:

```
# make clean
===>  Cleaning for bison-1.25
```

Appendix C.
Build logs

C.1 Building figlet

```
# make
===> Checking for vulnerabilities in figlet-2.2.1nb2
=> figlet221.tar.gz doesn't seem to exist on this system.
=> Attempting to fetch figlet221.tar.gz from ftp://ftp.figlet.org/pub/figlet/program/unix/.
=> [172219 bytes]
Connected to ftp.plig.net.
220 ftp.plig.org NcFTPd Server (licensed copy) ready.
331 Guest login ok, send your complete e-mail address as password.
230-You are user #5 of 500 simultaneous users allowed.
230-
230-  ___ _              _ _
230- |  _| |_ ___   ___| |_|___   ___ ___ ___
230- |  _|  _| . |_| . | | | . |_| . |  _| . |
230- |_| |_| |  _|_|  _|_|_|_  |_|___|_| |_  |
230-          |_|   |_|     |___|         |___|
230-
230-** Welcome to ftp.plig.org **
230-
230-Please note that all transfers from this FTP site are logged. If you
230-do not like this, please disconnect now.
230-
230-This arhive is available via
230-
230-HTTP:  http://ftp.plig.org/
230-FTP:   ftp://ftp.plig.org/     (max 500 connections)
230-RSYNC: rsync://ftp.plig.org/   (max  30 connections)
230-
230-Please email comments, bug reports and requests for packages to be
230-mirrored to ftp-admin@plig.org.
230-
230-
230 Logged in anonymously.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Type okay.
250 "/pub" is new cwd.
250-"/pub/figlet" is new cwd.
250-
250-Welcome to the figlet archive at ftp.figlet.org
250-
250- ftp://ftp.figlet.org/pub/figlet/
```

```
250-
250-The official FIGlet web page is:
250- http://www.figlet.org/
250-
250-If you have questions, please mailto:info@figlet.org. If you want to
250-contribute a font or something else, you can email us.
250
250 "/pub/figlet/program" is new cwd.
250 "/pub/figlet/program/unix" is new cwd.
local: figlet221.tar.gz remote: figlet221.tar.gz
502 Unimplemented command.
227 Entering Passive Mode (195,40,6,41,246,104)
150 Data connection accepted from 84.128.86.72:65131; transfer starting for figlet221.tar.gz
38% |**************                         | 65800      64.16 KB/s    00:01 ETA
226 Transfer completed.
172219 bytes received in 00:02 (75.99 KB/s)
221 Goodbye.
=> Checksum OK for figlet221.tar.gz.
===> Extracting for figlet-2.2.1nb2
===> Required installed package ccache-[0-9]*: ccache-2.3nb1 found
===> Patching for figlet-2.2.1nb2
===> Applying pkgsrc patches for figlet-2.2.1nb2
===> Overriding tools for figlet-2.2.1nb2
===> Creating toolchain wrappers for figlet-2.2.1nb2
===> Configuring for figlet-2.2.1nb2
===> Building for figlet-2.2.1nb2
gcc -O2 -DDEFAULTFONTDIR=\"/usr/pkg/share/figlet\"  -DDEFAULTFONTFILE=\"standard.flf\"  figl
chmod a+x figlet
gcc -O2 -o chkfont chkfont.c
=> Unwrapping files-to-be-installed.
#
# make install
===> Checking for vulnerabilities in figlet-2.2.1nb2
===> Installing for figlet-2.2.1nb2
install -d -o root -g wheel -m 755 /usr/pkg/bin
install -d -o root -g wheel -m 755 /usr/pkg/man/man6
mkdir -p /usr/pkg/share/figlet
cp figlet /usr/pkg/bin
cp chkfont /usr/pkg/bin
chmod 555 figlist showfigfonts
cp figlist /usr/pkg/bin
cp showfigfonts /usr/pkg/bin
cp fonts/*.flf /usr/pkg/share/figlet
cp fonts/*.flc /usr/pkg/share/figlet
cp figlet.6 /usr/pkg/man/man6
===> Registering installation for figlet-2.2.1nb2
#
```

C.2 Packaging figlet

```
# make package
===> Checking for vulnerabilities in figlet-2.2.1nb2
===> Packaging figlet-2.2.1nb2
===> Building binary package for figlet-2.2.1nb2
Creating package /home/cvs/pkgsrc/packages/i386/All/figlet-2.2.1nb2.tgz
Using SrcDir value of /usr/pkg
Registering depends:.
#
```

Appendix D.
Layout of the FTP server's package archive

Layout for precompiled binary packages on ftp.NetBSD.org:

```
/pub/NetBSD/packages/
                distfiles/

                # Unpacked pkgsrc trees
                pkgsrc-current -> /pub/NetBSD/NetBSD-current/pkgsrc
                pkgsrc-2003Q4 -> N/A
                pkgsrc-2004Q1/pkgsrc

                # pkgsrc archives
                pkgsrc-current.tar.gz -> ../NetBSD-current/tar_files/pkgsrc.tar.gz
                pkgsrc-2003Q4.tar.gz -> N/A
                pkgsrc-2004Q1.tar.gz -> N/A

                # Per pkgsrc-release/OS-release/arch package archives
                pkgsrc-2003Q4/
                        NetBSD-1.6.2/
                                i386/
                                        All/
                                        archivers/
                                                foo -> ../All/foo
                                        ...
                pkgsrc-2004Q1/
                        NetBSD-1.6.2/
                                i386/
                                        All/
                                        ...
                        NetBSD-2.0/
                                i386/
                                        All/
                                        ...
                        SunOS-5.9/
                                sparc/
                                        All/
                                        ...
                                x86/
                                        All/
                                        ...

                # Per os-release package archive convenience links
```

```
NetBSD-1.6.2 -> 1.6.2
1.6.2/
        i386 -> ../pkgsrc-2004Q1/NetBSD-1.6.2/i386
        m68k/
                All/
                archivers/
                        foo -> ../All/foo
                ...
        amiga -> m68k
        atari -> m68k
        ...

2.0 -> NetBSD-2.0       # backward compat, historic
NetBSD-2.0/
        i386 -> ../pkgsrc-2004Q1/NetBSD-2.0/i386
SunOS-5.9/
        sparc -> ../pkgsrc-2004Q1/SunOS-5.9/sparc
        x86 -> ../pkgsrc-2004Q1/SunOS-5.9/x86
```

To create:

1. Run bulk build, see Section 35.3
2. Upload /usr/pkgsrc/packages to

```
ftp://ftp.NetBSD.org/pub/NetBSD/packages/\
    pkgsrc-2004Q4/\                         # pkgsrc-branch
    `uname -s`-`uname -r`/                  # OS & version
    `uname -p`                              # architecture
```

3. If necessary, create a symlink **ln -s ‘uname -m‘ ‘uname -p‘** (amiga -> m68k, ...)

Appendix E.
Editing guidelines for the pkgsrc guide

This section contains information on editing the pkgsrc guide itself.

E.1 Targets

The pkgsrc guide's source code is stored in `pkgsrc/doc/guide/files`, and several files are created from it:

- `pkgsrc/doc/pkgsrc.txt`
- `pkgsrc/doc/pkgsrc.html`
- `http://www.NetBSD.org/Documentation/pkgsrc/`: the documentation on the NetBSD website will be built from pkgsrc and kept up to date on the web server itself. This means you *must* make sure that your changes haven't broken the build!
- `http://www.NetBSD.org/Documentation/pkgsrc/pkgsrc.pdf`: PDF version of the pkgsrc guide.
- `http://www.NetBSD.org/Documentation/pkgsrc/pkgsrc.ps`: PostScript version of the pkgsrc guide.

E.2 Procedure

The procedure to edit the pkgsrc guide is:

- Make sure you have the packages needed to re-generate the pkgsrc guide (and other XML-based NetBSD documentation) installed. These are "netbsd-doc" for creating the ASCII and HTML versions, and "netbsd-doc-print" for the PostScript and PDF versions. You will need both packages installed, to make sure documentation is consistent across all formats. The packages can be found in `pkgsrc/meta-pkgs/netbsd-doc` and `pkgsrc/meta-pkgs/netbsd-doc-print`.
- Edit the XML file(s) in `pkgsrc/doc/guide/files`.
- Run **make extract && make do-lint** in `pkgsrc/doc/guide` to check the XML syntax, and fix it if needed.
- Run **make** in `pkgsrc/doc/guide` to build the HTML and ASCII version.
- If all is well, run **make install-doc** to put the generated files into `pkgsrc/doc`.

- **cvs commit pkgsrc/doc/guide/files**
- **cvs commit -m re-generate pkgsrc/doc/pkgsrc.{html,txt}**
- Until the webserver on www.NetBSD.org is really updated automatically to pick up changes to the pkgsrc guide automatically, also run **make install-htdocs HTDOCSDIR=../../../htdocs** (or similar, adjust `HTDOCSDIR`!).
- **cvs commit htdocs/Documentation/pkgsrc**

Appendix F. *Contributing to the NetBSD guide*

There is a interest for both introductory and advanced documentation on NetBSD: this is probably a sign of the increased popularity of this operating system and of a growing user base. It is therefore important to keep adding new material to this guide and improving the existing text.

Whatever your level of expertise with NetBSD, you can contribute to the development of this guide. This appendix explains how, and what you should know before you start.

If you are a beginner and you found this guide helpful, please send your comments and suggestions to <www@NetBSD.org>. For example, if you tried something described here and it didn't work for you, or if you think that something is not clearly explained, or if you have an idea for a new chapter, etc: this type of feedback is very useful.

If you are an intermediate or advanced user, please consider contributing new material to the guide: you could write a new chapter or improve an existing one.

If you have some spare time, you could translate the guide into another language.

Whatever you choose to do, don't start working before having contacted us, in order to avoid duplicating efforts.

F.1 Translating the guide

If you want to translate the guide the first thing to do is, as already said, to contact <www@NetBSD.org> or to write to the <netbsd-docs@NetBSD.org> mailing list. There are several possible scenarios:

- someone else is already working on a translation into your language; you could probably help him.
- nobody is currently working on a translation into your language, but some chapters have already been translated and you can translate the remaining chapters.
- you start a new translation. Of course you don't need to translate all the guide: this is a big effort, but if you start translating one or two chapters it'll be a good starting point for someone else.

Even if a translation is already available, it is always necessary to keep it up to date with respect to the master version when new material is added or corrections are made: you could become the mantainer of a translation.

F.1.1 What you need to start a translation

In short, all you need is:

- the guide sources. They are part of “htdocs”, check it out from (anonymous) CVS like you would check out “src” or “pkgsrc” as described in Chapter 27.
- a text editor, such as vi or emacs.

> **Important:** Don’t start working with HTML or other formats: it will be very difficult to convert you work to XML/DocBook, the format used by the NetBSD guide.

F.1.2 Writing XML/DocBook

In order to translate the guide you don’t need to *learn* XML/DocBook; get the XML/DocBook sources of the NetBSD guide and work directly on them, in order to reuse the existing format (i.e. tags) in your work. For example, to translate the previous note, you would do the following:

1. load the english source of the current chapter, `ap-contrib.xml`, in your editor.
2. find the text of the previous note. You will see something like:

```
<important>
  <para>
    Don’t start working with HTML or other formats:
    it will be very difficult to convert you work
    to XML/DocBook, the format used by the NetBSD
    guide.
  </para>
</important>
```

3. add your translation between the tags, after the english version. The text now looks like this:

```
<imporant>
  <para>
    Don’t start working with HTML or other formats:
    it will be very difficult to convert you work
    to XML/DocBook, the format used by the NetBSD
    guide.
    your translation goes here
    your translation goes here
    your translation goes here
  </para>
</important>
```

4. delete the four lines of english text between the *tags* leaving your translation.

```
<important>
  <para>
    your translation goes here
    your translation goes here
    your translation goes here
  </para>
</important>
```

When you write the translation please use the same indentation and formatting style of the original text. See Section F.3 for an example.

One problem that you will probably face when writing the DocBook text is that of national characters (e.g. accented letters like “è”). You can use these characters in your source document but it’s preferable to replace them with XML *entities*. For example, “è” is written as “è”. Of course this makes your source text difficult to write and to read; the first problem, writing, can be solved using a good editor with macro capabilities. Vi and emacs, which are very popular choices, both have this feature and you can map the accented keys of your keyboard to generate the required entities automatically. For example, for vi you can put a line like the following in your `.exrc` file:

```
map! è &egrave;
```

Appendix G explains how to install the software tools to generate HTML and other formats from the DocBook sources. This is useful if you want to check your work (i.e. make sure you didn’t inadvertedly delete some tag) or to see what the output looks like, but it is not a requirement for a translation. If you don’t want to install the software tools, send your patches and sources to <www@NetBSD.org> and we’ll check them and create the various output formats.

F.2 Sending contributions

If you want to contribute some material to the guide you have several options, depending on the amount of text you want to write. If you just want to send a small fix, the easiest way to get it into the guide is to send it to <www@NetBSD.org> via e-mail. If you plan to write a substantial amount of text, such as a section or a chapter, you can choose among many formats:

- XML/DocBook; this is the preferred format. If you choose to use this format, please get the guide sources and use them as a template for the indentation and text layout, in order to keep the formatting consistent.
- text; if the formatting is kept simple, it is not difficult to convert text to XML format.
- other formats are also accepted if you really can’t use any of the previous ones.

F.3 XML/DocBook template

For the guide I use a formatting style similar to a program. The following is a template:

```
<chapter id="chap-xxxxx">
  <title>This is the title of the chapter</title>

  <para>
    This is the text of a paragraph.  This is the text of a paragraph.
    This is the text of a paragraph.  This is the text of a paragraph.
    This is the text of a paragraph.
  </para>

  <!-- ============================================================ -->
```

```
 <sect1>
   <title>This is the title of a sect1</title>

   <para>
     This is the text of a paragraph.  This is the text of a paragraph.
     This is the text of a paragraph.  This is the text of a paragraph.
     This is the text of a paragraph.
   </para>

   <!-- ..................................................... -->

   <sect2>
     <title>This is the title of a sect2</title>

     <para>
A sect2 is nested inside a sect1.
     </para>
   </sect2>

 </sect1>

 <!-- ============================================================ -->

 <sect1>
   <title>This is the title of another sect1</title>

   <para>
     An itemized list:
     <itemizedlist>
<listitem>
  <para>
    text
  </para>
</listitem>
<listitem>
  <para>
    text
  </para>
</listitem>
     </itemizedlist>
   </para>

 </sect1>
</chapter>
```

The defaults are:

- two spaces for each level of indentation
- lines not longer than 72 characters.
- use separator lines (comments) between sect1/sect2.

Appendix G.
Getting started with XML/DocBook

This appendix describes the installation of the tools needed to produce a formatted version of the NetBSD guide. Besides that it contains instructions that describe how to build the guide. This appendix assumes knowledge of pkgsrc, see Part VI in *The NetBSD & pkgsrc Guide* for details.

G.1 What is XML/DocBook

XML (eXtensible Markup Language) is a language which is used to define other languages based on markups, i.e. with XML you can define the grammar (i.e. the valid constructs) of markup languages. HTML, for example, can be defined using XML. If you are a programmer, think of XML like the BNF (Backus-Naur Form): a tool used to define grammars.

DocBook is a markup template defined using XML; DocBook lists the valid tags that can be used in a DocBook document and how they can be combined together. If you are a programmer, think of DocBook as the grammar of a language specified with the BNF. For example, it says that the tags:

```
<para> ... </para>
```

define a paragraph, and that a <para> can be inside a <sect1> but that a <sect1> cannot be inside a <para>.

Therefore, when you write a document, you write a document in DocBook and not in XML: in this respect DocBook is the counterpart of HTML (although the markup is richer and a few concepts are different).

The DocBook specification (i.e. the list of tags and rules) is called a DTD (Document Type Definition).

In short, a DTD defines how your source documents look like but it gives no indication about the format of your final (compiled) documents. A further step is required: the DocBook sources must be converted to some other representation like, for example, HTML or PDF. This step is performed by a tool like Jade, which applies the DSSSL transforms to the source document. DSSSL (Document Style Semantics and Specification Language) is a format used to define the *stylesheets* necessary to perform the conversion from DocBook to other formats. The build structure for the guide also supports the XSL (Extensible Stylesheet Language) stylesheet language. The **xsltproc** program is used for transforming XML with XSL stylesheets.

G.2 Installing the necessary tools

All the tools that are needed to generate the guide in various formats can be installed through the *netbsd-www*, *netbsd-doc*, and *netbsd-doc-print* meta-packages. Together the *netbsd-doc* and *netbsd-www* packages install everything that is needed to generate the HTML version of the guide. To be able to generate printable formats, such as Postscript and PDF, install the *netbsd-doc-print* meta-package.

Supposing that a current pkgsrc tree is installed at `/usr/pkgsrc`, you can install all these meta-packages with:

```
$ cd /usr/pkgsrc/meta-pkgs/netbsd-www
$ make install
$ cd /usr/pkgsrc/meta-pkgs/netbsd-doc
$ make install
$ cd /usr/pkgsrc/meta-pkgs/netbsd-doc-print
$ make install
```

G.3 Using the tools

This section provides an overview of how the guide can be compiled from XML to any of the following target formats: *html*, *html-split*, *ascii*, *ps*, and *pdf*. Creating all formats is the default. To produce any of the above output formats, run **make** with the format(s) as argument.

Let's look at a few examples.

Before looking at the output generated in any of the above-mentioned formats, integrity of the XML structure has to be ensured. This can be done by running **make lint**:

```
$ cd htdocs/guide/en
$ make lint
```

Fix any errors you may get. When working on the contents of the guide, you may want to produce the HTML version to have a look at it for proofreading:

```
$ cd htdocs/guide/en
$ make html-split
```

After this, please update the Postscript and PDF versions of the guide too. The command for this is:

```
$ cd htdocs/guide/en
$ make pdf
```

Before you go and commit the generated files, please make sure that you commit the XML files first, then re-generated all formats, i.e. the procedure would be something like:

```
$ cd htdocs/guide/en
$ cvs commit *.xml
$
$ make lint
$ make
$ make install-doc
$
```

```
$ cd ..
$ cvs commit en download
```

When running **make** with no argument, all formats will be re-generated. This is the default way to build the guide for the NetBSD.org website.

G.4 Language-specific notes

G.4.1 Enabling hyphenation for the Italian language

The NetBSD guide is currently available in three languages: English, French and Italian. Of these, only English and French are automatically hyphenated by TeX. To turn on hyphenation for the Italian language, some simple steps are required:

Edit `/usr/pkg/share/texmf/tex/generic/config/language.dat` and remove the comment (`%`) from the line of the Italian hyphenation. I.e.

```
%italian ithyph.tex
```

becomes

```
italian ithyph.tex
```

> **Note:** As more translations of the guide become available, you will probably need to enable other hyphenation patterns as well.

Now the latex and pdflatex formats must be recreated:

```
# cd /usr/pkg/share/texmf/web2c
# fmtutil --byfmt latex
# fmtutil --byfmt pdflatex
```

If you check, for example, `latex.log` you will find something like:

```
Babel <v3.6Z> and hyphenation patterns for american, french, german,
ngerman, italian, nohyphenation, loaded.
```

Please note that there are many ways to perform these operations, depending on your level of expertise with the TeX system (mine is very low). For example, you could use the "texconfig" interactive program, or you could recreate the formats by hand using the **tex** program.

If you know a better way of doing the operations described in this appendix, please let me know.

G.5 Links

The official DocBook home page (http://www.oasis-open.org/docbook/) is where you can find the definitive documentation on DocBook. You can also read online or download a copy of the book

DocBook: The Definitive Guide (http://www.oasis-open.org/docbook/documentation/reference/) by Norman Walsh and Leonard Muellner.

For DSSSL start looking at http://nwalsh.com.

XSL is described at http://www.w3.org/Style/XSL/.

Jade/OpenJade sources and info can be found on the OpenJade Home Page (http://openjade.sourceforge.net/).

If you want to produce Postscript and PDF documents from your DocBook source, look at the home page of JadeTex (http://sourceforge.net/projects/jadetex).

The home page of Markus Hoenicka (http://ourworld.compuserve.com/homepages/hoenicka_markus/) explains everything you need to know if you want to work with SGML/DocBook on the Windows NT platform.

Appendix H.
Acknowledgements

The NetBSD Guide was originally written by Federico Lupi who managed the sources, coordinated updates, and merged all contributions on his own. Since then, it has been updated and maintained by the NetBSD www team. The Guide has progressed thanks to the contributions of many people who have volunteered their time and effort, supplied material and sent in suggestions and corrections.

H.1 Original acknowledgements

Federico's original credits are:

- Paulo Aukar
- Grant Beattie, converted to XML DocBook.
- Manolo De Santis, Audio Chapter
- Eric Delcamp, Boot Floppies
- Hubert Feyrer, who contributed the Introduction to TCP/IP Networking in Chapter 21 including Next generation Internet protocol - IPv6 and the section on getting IPv6 Connectivity & Transition via 6to4 in Section 26.4. He also helped with the SGML to XML transition.
- Jason R. Fink
- Daniel de Kok, audio and linux chapters fixes.
- Reinoud Koornstra, CVS chapter and rebuilding `/dev` in the Misc chapter.
- Brian A. Seklecki <`lavalamp@burghcom.com`> who contributed the CCD Chapter.
- Guillain Seuillot
- Martti Kuparinen, RAIDframe documentation.
- David Magda

H.2 Current acknowledgements

This document is currently maintained by the NetBSD www team. Thanks to their efforts, the document is kept up to date and available online at all times. In addition, special thanks go to (in alphabetical order):

- Hubert Feyrer, for getting the guide up to speed for NetBSD 2.0, and for making numerous improvements to all chapters.
- Jason R. Fink, for maintaining this document and integrating changes.

- Andreas Hallman, for his information in Section 26.4.10.
- Daniel de Kok, for constant contributions of new chapters, maintenance of existing chapters and his translation work.
- Hiroki Sato, for allowing us to build PDF and PS versions of this document.
- Jan Schaumann, for maintenance work and www/htdocs managment.
- Lubomir Sedlacik, for some details on using CGD for swap in Section 14.5.
- Dag-Erling Smørgrav, for the article on Chapter 17. See Section H.3.2 for the license accompanying license.
- Florian Stöhr, for Section 14.4.

H.3 Licenses

H.3.1 Federico Lupi's original license of this guide

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by Federico Lupi for the NetBSD Project.
4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

H.3.2 Networks Associates Technology's license on the PAM article

Copyright (c) 2001-2003 Networks Associates Technology, Inc.
All rights reserved.

This software was developed for the FreeBSD Project by ThinkSec AS and
Network Associates Laboratories, the Security Research Division of
Network Associates, Inc. under DARPA/SPAWAR contract N66001-01-C-8035
("CBOSS"), as part of the DARPA CHATS research program.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote
   products derived from this software without specific prior written
   permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS “AS IS” AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

Appendix I.
Bibliography

Bibliography

[AeleenFrisch] Aeleen Frisch, 1991, O'Reilly & Associates, *Essential System Administration*.

[CraigHunt] Craig Hunt, 1993, O'Reilly & Associates, *TCP/IP Network Administration*.

[RFC1034] P. V. Mockapetris, 1987, *RFC 1034: Domain names - concepts and facilities*.

[RFC1035] P. V. Mockapetris, 1987, *RFC 1035: Domain names - implementation and specification*.

[RFC1055] J. L. Romkey, 1988, *RFC 1055: Nonstandard for transmission of IP datagrams over serial lines: SLIP*.

[RFC1331] W. Simpson, 1992, *RFC 1331: The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links*.

[RFC1332] G. McGregor, 1992, *RFC 1332: The PPP Internet Protocol Control Protocol (IPCP)*.

[RFC1933] R. Gilligan and E. Nordmark, 1996, *RFC 1933: Transition Mechanisms for IPv6 Hosts and Routers*.

[RFC2004] C. Perkins, 1996, *RFC 2003: IP Encapsulation within IP*.

[RFC2401] S. Kent and R. Atkinson, 1998, *RFC 2401: Security Architecture for the Internet Protocol*.

[RFC2411] R. Thayer, N. Doraswamy, and R. Glenn, 1998, *RFC 2411: IP Security Document Roadmap*.

[RFC2461] T. Narten, E. Nordmark, and W. Simpson, 1998, *RFC 2461: Neighbor Discovery for IP Version 6 (IPv6)*.

[RFC2529] B. Carpenter and C. Jung, 1999, *RFC 2529: Transmission of IPv6 over IPv4 Domains without Explicit Tunnels*.

[RFC3024] G. Montenegro, 2001, *RFC 3024: Reverse Tunneling for Mobile IP*.

[RFC3027] M. Holdrege and P. Srisuresh, 2001, *RFC 3027: Protocol Complications with the IP Network Address Translator*.

[RFC3056] B. Carpenter and K. Moore, 2001, *RFC 3056: Connection of IPv6 Domains via IPv4 Clouds*.