PREDGOVOR

Ovu knjigu sam napisao zato što verujem da ste u stanju da shvatite šta radi vaš računar. Osim toga, trebalo bi da znate i da naterate svoj softver da radi ono što želite da radi (unutar razumnih granica njegovih mogućnosti, razume se). Ključ za sticanje te moći leži u razumevanju osnova onoga šta i kako softver radi, a to je i suština ove knjige. Ne bi nikad trebalo da dođete u situaciju da se morate „boriti“ s računarom.

Linux je odlična platforma za učenje zato što ništa ne skriva od vas. Konkretno, najveći deo konfiguracije sistema može se naći u običnim tekstualnim datotekama koje se lako čitaju. Jedini zapetljan deo je pronalaženje koji su delovi za šta odgovorni i kako se sve sklapa u celinu.

Ko bi trebalo da pročita ovu knjigu?

Razlozi zbog kojih želite da naučite kako radi Linux mogu biti različiti. U profesionalnoj oblasti, potrebno je da korisnici i projektanti softvera znaju gotovo sve što ćete naći u ovoj knjizi. Arhitekte i programeri softvera za Linux treba da poznaju ovaj materijal i zato da bi najbolje iskoristili mogućnosti operativnog sistema. Istraživači i studenti, koji često koriste vlastite Linux sisteme, takođe će otkriti da ova knjiga pruža korisna objašnjenja o tome *zašto* se stvari odvijaju baš tako kako se odvijaju.

Zatim dolaze radoznalci – ljudi koji prosto vole da se igraju sa svojim računarima zbog zabave, zarade ili i jednog i drugog. Želite da znate zašto neke stvari rade, dok druge ne rade? Želite da znate šta će se dogoditi ako nešto premestite? Vi ste verovatno radoznali.

Predznanje

Mada programeri vole Linux, nije neophodno da budete programer da biste čitali ovu knjigu; treba vam samo osnovno znanje o upotrebi računara. To znači da bi trebalo da znate kako da se snađete u grafičkom korisničkom interfejsu (naročito u delu koji omogućava instaliranje i podešavanje vaše distribucije Linuxa) i da znate šta su to datoteke i direktorijumi („folderi“). Trebalo bi i da budete spremni da pretražujete dodatnu dokumentaciju na svom sistemu i na vebu. Kao što smo već napomenuli, najvažniji su spremnost i želja da se igrate sa svojim računarom.

Kako treba čitati ovu knjigu

Sticanje potrebnog znanja je izazov koji se pojavljuje pri savladavanju svake stručne teme. Kada treba objasniti kako rade softverski sistemi, stvari mogu postati *zaista* komplikovane. Previše detalja „guši“ čitaoca i otežava razumevanje važnih delova (ljudski mozak jednostavno ne može da obradi preveliki broj novih pojmova istovremeno), ali premalo detalja ostavlja čitaoca u pomrčini i nepripremljenog za materijal koji sledi.

Većinu poglavlja sam osmislio tako da obrađuju prvo najvažniji materijal: osnovne informacije koje će vam trebati da biste dalje napredovali. Ponekad sam stvari pojednostavio da se ne bismo odmakli od glavne teme. Tokom čitanja poglavlja, nailazićete na sve više detalja, naročito u poslednjim odeljcima. Da li je potrebno da sve te deliće savladate odmah? Uglavnom nije, kao što to često napominjem. Ako vam oči počnu da lutaju kada naiđete na previše novih detalja o nečemu što ste tek saznali, ne ustručavajte se da preskočite na sledeće poglavlje ili da napravite pauzu. Ti detalji će vas i dalje čekati.

Praktičan pristup

Međutim, ako se opredelite da nastavite čitanje ove knjige, trebalo bi da pred sobom imate Linux računar, po mogućstvu neki koji možete da „zlostavljate“ eksperimentisanjem. Možda bi vam više odgovaralo da se igrate s virtuelnom

instalacijom – ja sam najveći deo materijala iz ove knjige testirao pomoću softvera VirtualBox. Potreban vam je pristup sa ovlašćenjima administratora (korisnik root), ali bi trebalo da najveći deo vremena koristite neki standardni korisnički nalog. Uglavnom ćete raditi s komandne linije, u terminalskom prozoru ili udaljenoj sesiji. Ako dosad niste mnogo radili u takvom okruženju, nije problem; poglavlje 2 će vam omogućiti da uhvatite korak.

U ovoj knjizi komande će uglavnom biti slične sledećoj:

```
$ ls /
[neki rezultat]
```

Vi treba da upišete tekst istaknut podebljanim slovima; običan tekst koji sledi je ono čime se mašina odaziva. Znak $ je odzivni znak za standardni korisnički nalog. Ako kao odzivnik vidite znak #, trebalo bi da ste administrator. (Više informacija o tome naći ćete u poglavlju 2.)

Kako je ova knjiga organizovana

Poglavlja knjige grupisao sam u četiri glavna dela. Prvi je uvodni, koji vam pruža pogled na sistem iz ptičje perspektive i omogućava da steknete praktično iskustvo s određenim alatkama koje će vam trebati sve dok koristite Linux. Nakon toga, prelazite na detaljnije istraživanje sistema, od upravljanja uređajima do konfigurisanja mreže, u skladu sa opštim redosledom operacija pri pokretanju sistema. I najzad, obići ćete neke delove funkcionalnog sistema, naučiti nekoliko suštinskih veština i steći određeni uvid u alatke s kojima rade programeri.

Izuzev poglavlja 2, najveći deo početnih poglavlja bavi se Linuxovim jezgrom, ali tokom čitanja tog dela knjige stići ćete i do korisničkog prostora. (Ako ne razumete o čemu je reč, ne brinite; objasniću u poglavlju 1.)

Materijal u ovoj knjizi zamišljen je tako da bude što nezavisniji od konkretne distribucije Linuxa. Ako to imamo u vidu, pošto bi bilo nezgrapno da razmatramo sve moguće varijacije sistemskog softvera, pokušao sam da obradim dve najvažnije porodice distribucija Linuxa: Debian (uključujući i Ubuntu) i RHEL/Fedora/CentOS. Osim toga, knjiga je usredsređena na serverske instalacije i instalacije namenjene korisničkim računarima. Značajan deo funkcionalnosti je prenet u ugrađene sisteme, kao što su Android i OpenWRT, ali je na vama da sami otkrijete razlike na tim platformama.

Šta je novo u drugom izdanju?

Prvo izdanje ove knjige bavilo se prvenstveno aspektom Linux sistema koji se tiče korisnika. Bilo je usredsređeno na to kako rade pojedine komponente sistema i kako učiniti da one dobro sarađuju. U to vreme, bilo je teško instalirati i pravilno podesiti mnoge sastavne delove Linuxa.

Srećom, više nije tako zahvaljujući velikom trudu ljudi koji pišu softver i prave distribucije Linuxa. Imajući to u vidu, izostavio sam određeni deo starijeg i

možda manje važnog gradiva (kao što je detaljno objašnjavanje postupka štampanja) u korist podrobnijeg razmatranja uloge Linuxovog jezgra u svakoj distribuciji Linuxa. Sa jezgrom komunicirate verovatno više nego što ste toga svesni, a ja sam se posebno potrudio da istaknem gde se sve to događa.

Razume se, tokom godina, mnogo toga što je predmet ove knjige pretrpelo je izmene, a ja sam se namučio tražeći u materijalu iz prvog izdanja mesta gde je potrebno ažuriranje. Naročito je važan postupak pokretanja Linuxa i to kako on upravlja uređajima. Postarao sam se i da gradivo preuredim tako da odgovara interesima i potrebama današnjih čitalaca.

Jedna od stvari koja je ostala nepromenjena jeste veličina knjige. Želim da vam pružim znanje koje vam treba kako biste brzo uhvatili korak, a to podrazumeva usputno objašnjavanje određenih detalja koji se možda teže shvataju, ali ne želim da postanete dizač tegova kako biste mogli podići ovu knjigu. Kada u njoj naiđete na početak neke važne teme, ne bi trebalo da vam bude teško da i izvan nje pronađete i razumete detaljnije informacije.

Izostavio sam i neke istorijske informacije kojih je bilo u prvom izdanju, prvenstveno zato da se ne biste odmicali od glavne teme. Ako vas zanima Linux i njegova istorija, nabavite knjigu Petera H. Salusa *The Daemon, the Gnu, and the Penguin* (izdavač je Reed Media Services, 2008) – vrlo dobro objašnjava kako je softver koji koristimo evoluirao tokom vremena.

Napomena o terminologiji

Postoji mnoštvo oprečnih mišljenja o imenima nekih elemenata operativnih sistema. Čak je „Linux“ sporan – da li bi trebalo da to bude „Linux“, ili „GNU/Linux“ kako bi odražavalo činjenicu da operativni sistem sadrži delove iz GNU Projecta? U celoj knjizi sam se trudio da koristim najuobičajenija imena.

SADRŽAJ

PREDGOVOR **xv**

Ko bi trebalo da pročita ovu knjigu? ... xvi
Predznanje ... xvi
Kako treba čitati ovu knjigu ... xvi
Praktičan pristup ... xvi
Kako je ova knjiga organizovana ... xvii
Šta je novo u drugom izdanju? ... xvii
Napomena o terminologiji ... xviii

ZAHVALNICA **xix**

1 OPŠTA SLIKA **1**

1.1 Nivoi i slojevi apstrakcije Linux sistema ... 2
1.2 Hardver: glavna memorija ... 3
1.3 Jezgro ... 4
1.3.1 Upravljanje procesima ... 5
1.3.2 Upravljanje memorijom ... 6
1.3.3 Drajveri i upravljanje uređajima ... 6
1.3.4 Sistemski pozivi i podrška ... 7
1.4 Korisnički prostor ... 8
1.5 Korisnici ... 9
1.6 Šta sledi ... 10

2 OSNOVNE KOMANDE I HIJERARHIJA DIREKTORIJUMA **11**

2.1 Ljuska Bourne: /bin/sh ... 12
2.2 Upotreba ljuske ... 12
2.2.1 Prozor ljuske ... 12
2.2.2 Komanda cat ... 13
2.2.3 Standardni ulaz i standardni izlaz ... 13
2.3 Osnovne komande ... 14
2.3.1 ls ... 14
2.3.2 cp ... 15
2.3.3 mv ... 15
2.3.4 touch ... 15

2.3.5 rm 16
2.3.6 echo 16
2.4 Kretanje kroz direktorijume 16
2.4.1 cd 17
2.4.2 mkdir 17
2.4.3 rmdir 17
2.4.4 Globiranje u ljusci (džokerski znakovi) 17
2.5 Komande srednjeg nivoa 18
2.5.1 grep 18
2.5.2 less 19
2.5.3 pwd 20
2.5.4 diff 20
2.5.5 file 20
2.5.6 find i locate 20
2.5.7 head i tail 21
2.5.8 sort 21
2.6 Menjanje lozinke i ljuske 21
2.7 Datoteke s tačkom 21
2.8 Promenljive okruženja i promenljive ljuske 22
2.9 Komandna putanja 23
2.10 Specijalni znaci 23
2.11 Ažuriranje sadržaja komandne linije 24
2.12 Editori teksta 25
2.13 Ugrađena pomoć 26
2.14 Ulaz i izlaz podataka u ljusci 27
2.14.1 Standardna greška 28
2.14.2 Preusmeravanje standardnog ulaza 29
2.15 Poruke o greškama 29
2.15.1 Anatomija UNIX-ove poruke o grešci 29
2.15.2 Uobičajene greške 30
2.16 Prikazivanje liste procesa i manipulisanje njima 31
2.16.1 Opcije komande ps 32
2.16.2 Okončavanje procesa 32
2.16.3 Upravljanje poslovima 33
2.16.4 Pozadinski procesi 33
2.17 Režimi pristupa datotekama i ovlašćenja 34
2.17.1 Menjanje ovlašćenja 35
2.17.2 Simboličke veze 36
2.17.3 Izrada simboličkih veza 37
2.18 Arhiviranje i komprimovanje datoteka 38
2.18.1 gzip 38
2.18.2 tar 38
2.18.3 Komprimovane arhive (.tar.gz) 39
2.18.4 zcat 39
2.18.5 Druge alatke za komprimovanje 40
2.19 Uvod u hijerarhiju Linuxovih direktorijuma 40
2.19.1 Ostali poddirektorijumi korenskog direktorijuma 42
2.19.2 Direktorijum /usr 42
2.19.3 Lokacija jezgra 43
2.20 Izvršavanje komandi s ovlašćenjima administratora 43
2.20.1 sudo 43
2.20.2 /etc/sudoers 43
2.21 Šta sledi 44

3 UREĐAJI **45**

3.1 Datoteke uređaja 46
3.2 Putanje uređaja u sistemu datoteka sysfs 47
3.3 Program dd i uređaji 48
3.4 Šta treba znati o imenima uređaja 49
3.4.1 Čvrsti diskovi: /dev/sd* 50
3.4.2 CD i DVD uređaji: /dev/sr* 51
3.4.3 PATA čvrsti diskovi: /dev/hd* 51
3.4.4 Terminali: /dev/tty*, /dev/pts/* i /dev/tty 51
3.4.5 Serijski priključci: /dev/ttyS* 52
3.4.6 Paralelni priključci: /dev/lp0 i /dev/lp1 52
3.4.7 Audio uređaji: /dev/snd/*, /dev/dsp, /dev/audio i drugi 53
3.4.8 Izrada datoteka uređaja 53
3.5 udev 54
3.5.1 devtmpfs 54
3.5.2 Rad i konfigurisanje programa udevd 55
3.5.3 udevadm 57
3.5.4 Nadgledanje uređaja 57
3.6 Detaljan opis: SCSI i Linuxovo jezgro 58
3.6.1 USB uređaji za skladištenje podataka i SCSI 61
3.6.2 SCSI i ATA 62
3.6.3 Generički SCSI uređaji 62
3.6.4 Više načina pristupanja istom uređaju 63

4 DISKOVI I SISTEMI DATOTEKA **65**

4.1 Deljenje disk uređaja na particije 67
4.1.1 Pregledanje tabele particija 68
4.1.2 Menjanje tabela particija 69
4.1.3 Geometrija diska i particija 70
4.1.4 Diskovi bez pokretnih delova (SSD) 72
4.2 Sistemi datoteka 72
4.2.1 Tipovi sistema datoteka 73
4.2.2 Izrada sistema datoteka 74
4.2.3 Montiranje sistema datoteka 75
4.2.4 UUID sistem datoteka 76
4.2.5 Baferisanje diska, keširanje podataka i sistemi datoteka 77
4.2.6 Opcije za montiranje sistema datoteka 78
4.2.7 Ponovno montiranje sistema datoteka 79
4.2.8 Tabela sistema datoteka u /etc/fstab 79
4.2.9 Alternative za /etc/fstab 81
4.2.10 Kapacitet sistema datoteka 81
4.2.11 Proveravanje i popravljanje sistema datoteka 82
4.2.12 Sistemi datoteka za posebne namene 85
4.3 Prostor za razmenu 85
4.3.1 Upotreba cele particije na disku kao prostora za razmenu 86
4.3.2 Upotreba datoteke kao prostora za razmenu 86
4.3.3 Koliko vam prostora za razmenu zaista treba? 86
4.4 Šta sledi: diskovi i korisnički prostor 87
4.5 Tradicionalni sistem datoteka viđen iznutra 88
4.5.1 Prikazivanje podataka koje i-čvor sadrži 90
4.5.2 Rad sa sistemima datoteka u korisničkom prostoru 91
4.5.3 Evolucija sistema datoteka 92

5 KAKO SE POKREĆE LINUXOVO JEZGRO 93

5.1 Poruke koje se pojavljuju tokom podizanja sistema 94
5.2 Inicijalizovanje jezgra i opcije pri pokretanju 95
5.3 Parametri jezgra 96
5.4 Programi za pokretanje jezgra 97
5.4.1 Zadaci programa za pokretanje jezgra 97
5.4.2 Pregled programa za pokretanje jezgra 98
5.5 Uvod u program GRUB 99
5.5.1 Pregledanje uređaja i particije pomoću komandne linije programa GRUB 101
5.5.2 Konfigurisanje programa GRUB 103
5.5.3 Instaliranje programa GRUB 105
5.6 Problemi u vezi sa bezbednim UEFI podizanjem sistema 107
5.7 Lančano pokretanje drugih operativnih sistema 107
5.8 Detaljniji uvid u program za pokretanje jezgra 108
5.8.1 MBR način pokretanja 108
5.8.2 UEFI način pokretanja 108
5.8.3 Kako radi GRUB 109

6 KAKO SE POKREĆE KORISNIČKI PROSTOR 111

6.1 Uvod u program init 112
6.2 Radni nivoi verzije System V 113
6.3 Identifikovanje instalirane verzije programa init 114
6.4 systemd 114
6.4.1 Jedinice i tipovi jedinica 115
6.4.2 Zavisnosti systemd jedinica 116
6.4.3 Konfigurisanje programa systemd 118
6.4.4 Upotreba programa systemd 120
6.4.5 Dodavanje jedinica programu systemd 122
6.4.6 Kako systemd prati i inicijalizuje procese 124
6.4.7 Pokretanje pomoću programa systemd na zahtev i uz paralelizovanje resursa 125
6.4.8 Kompatibilnost programa systemd s verzijom System V 129
6.4.9 Pomoćni programi za systemd 129
6.5 Upstart 130
6.5.1 Postupak inicijalizovanja programa Upstart 130
6.5.2 Upstartovi poslovi 132
6.5.3 Konfigurisanje programa Upstart 134
6.5.4 Kako radi Upstart 138
6.5.5 Upstartovi dnevnici 139
6.5.6 Upstartova kompatibilnost sa System V radnim nivoima 139
6.6 Verzija System V init 140
6.6.1 System V init: sekvence komandi koje se izvršavaju pri podizanju 142
6.6.2 Farma veza s kojima radi System V init 143
6.6.3 Alatka run-parts 144
6.6.4 Upravljanje radom programa System V init 144
6.7 Spuštanje sistema 145
6.8 Inicijalni RAM sistem datoteka 146
6.9 Hitno podizanje sistema i jednokorisnički režim rada 148

7 KONFIGURISANJE SISTEMA: DNEVNICI, SISTEMSKO VREME, KOMANDNE DATOTEKE I KORISNICI 149

7.1 Struktura direktorijuma /etc ... 150
7.2 Sistemski dnevnici ... 150
7.2.1 Program za beleženje u sistemski dnevnik ... 150
7.2.2 Konfiguracione datoteke ... 151
7.3 Datoteke za upravljanje korisnicima ... 153
7.3.1 Datoteka /etc/passwd ... 153
7.3.2 Specijalni korisnici ... 154
7.3.3 Datoteka /etc/shadow ... 155
7.3.4 Rad s korisnicima i datotekama ... 155
7.3.5 Rad s grupama ... 155
7.4 Program getty i prijavljivanje u sistem ... 157
7.5 Podešavanje sistemskog vremena ... 157
7.5.1 Predstavljanje vremena jezgra i vremenske zone ... 158
7.5.2 Mrežno vreme ... 158
7.6 Višekratno izvršavanje programa u zadato vreme pomoću servisa cron ... 159
7.6.1 Instaliranje crontab datoteka ... 160
7.6.2 Sistemske crontab datoteke ... 161
7.6.3 Budućnost servisa cron ... 161
7.7 Jednokratno izvršavanje poslova pomoću servisa at ... 161
7.8 Korisnički identifikatori i zamena korisnika ... 162
Vlasništvo nad procesom, efektivni UID, stvarni UID i sačuvani UID ... 162
7.9 Identifikovanje i provera verodostojnosti korisnika ... 164
7.9.1 Upotreba biblioteka radi pribavljanja podataka o korisnicima ... 165
7.10 PAM ... 166
7.10.1 Konfigurisanje PAM-a ... 166
7.10.2 Napomene o PAM-u ... 170
7.10.3 PAM i lozinke ... 170
7.11 Šta sledi ... 171

8 DETALJNIJI UVID U PROCESE I UPOTREBU RESURSA 173

8.1 Praćenje procesa ... 174
8.2 Pronalaženje otvorenih datoteka pomoću programa lsof ... 174
8.2.1 Tumačenje rezultata komande lsof ... 174
8.2.2 Upotreba programa lsof ... 175
8.3 Praćenje izvršavanja programa i sistemskih poziva ... 176
8.3.1 strace ... 176
8.3.2 Komanda ltrace ... 177
8.4 Niti izvršavanja ... 178
8.4.1 Jednonitni i višenitni procesi ... 178
8.4.2 Prikazivaje niti ... 178
8.5 Uvod u nadgledanje resursa ... 179
8.6 Merenje vremena procesora ... 180
8.7 Podešavanje prioriteta izvršavanja procesa ... 180
8.8 Prosečno opterećenje ... 182
8.8.1 Upotreba komande uptime ... 182
8.8.2 Visoka opterećenja ... 182
8.9 Memorija ... 183
8.9.1 Kako radi memorija ... 183
8.9.2 Greška pristupanja stranici ... 184

8.10 Praćenje performansi procesora i memorije pomoću alatke vmstat ... 185
8.11 Praćenje ulazno/izlaznih operacija ... 187
8.11.1 Upotreba alatke iostat ... 187
8.11.2 Praćenje potrošnje U/I resursa po procesu: iotop ... 188
8.12 Praćenje pojedinačnih procesa pomoću alatke pidstat ... 189
8.13 Teme za dalje proučavanje ... 190

9 MREŽA I NJENO KONFIGURISANJE 191

9.1 Osnove mreža ... 192
9.1.1 Paketi ... 192
9.2 Mrežni slojevi ... 193
9.3 Internet sloj ... 194
9.3.1 Prikazivanje IP adresa računara ... 195
9.3.2 Podmreže ... 196
9.3.3 Uobičajene maske podmreža i CIDR notacija ... 196
9.4 Mrežne putanje i tabela rutiranja jezgra ... 197
9.4.1 Podrazumevani mrežni prolaz ... 198
9.5 Osnovne ICMP i DNS alatke ... 198
9.5.1 Alatka ping ... 199
9.5.2 Alatka traceroute ... 200
9.5.3 DNS i komanda host ... 200
9.6 Fizički sloj i Ethernet mreže ... 201
9.7 Mrežni interfejsi jezgra ... 201
9.8 Uvod u konfigurisanje mrežnih interfejsa ... 202
9.8.1 Ručno dodavanje i brisanje mrežnih putanja ... 203
9.9 Konfigurisanje mreže pri podizanju sistema ... 203
9.10 Problemi s ručnim konfigurisanjem mreže i konfigurisanjem pri podizanju sistema ... 204
9.11 Programi za konfigurisanje mreža ... 205
9.11.1 Kako radi NetworkManager ... 205
9.11.2 Interakcija s NetworkManagerom ... 206
9.11.3 Konfigurisanje NetworkManagera ... 206
9.12 Razrešavanje imena umreženih računara ... 208
9.12.1 Datoteka /etc/hosts ... 209
9.12.2 Datoteka resolv.conf ... 209
9.12.3 Keširanje DNS podataka i DNS bez konfigurisanja ... 209
9.12.4 Datoteka /etc/nsswitch.conf ... 210
9.13 Localhost ... 211
9.14 Transportni sloj: TCP, UDP i servisi ... 211
9.14.1 TCP priključci i veze ... 212
9.14.2 Uspostavljanje TCP veza ... 212
9.14.3 Brojevi priključaka i datoteka /etc/services ... 213
9.14.4 Karakteristike protokola TCP ... 214
9.14.5 UDP ... 214
9.15 Povratak u jednostavnu lokalnu mrežu ... 216
9.16 Protokol DHCP ... 216
9.16.1 Linuxov DHCP klijent ... 217
9.16.2 Linux DHCP serveri ... 217
9.17 Konfigurisanje Linuxa da radi kao usmerivač ... 217
9.17.1 Veza s internetom ... 218
9.18 Privatne mreže ... 219
9.19 Prevođenje mrežnih adresa (IP maskiranje) ... 219
9.20 Usmerivači i Linux ... 221

9.21 Zaštitni bedemi . . . 222
9.21.1 Osnove Linuxovog zaštitnog bedema . . . 222
9.21.2 Zadavanje pravila zaštitnog bedema . . . 223
9.21.3 Strategije za zaštitni bedem . . . 225
9.22 Ethernet, IP i ARP . . . 227
9.23 Bežični Ethernet . . . 228
9.23.1 iw . . . 229
9.23.2 Zaštita bežičnih mreža . . . 230
9.24 Sažetak . . . 230

10 MREŽNE APLIKACIJE I SERVISI 233

10.1 Osnove mrežnih servisa . . . 234
10.1.1 Detaljniji uvid . . . 234
10.2 Mrežni serveri . . . 236
10.3 Protokol SSH . . . 237
10.3.1 Server SSHD . . . 238
10.3.2 SSH klijent . . . 240
10.4 Dimoni inetd i xinetd . . . 241
10.4.1 TCP omotači: tcpd, /etc/hosts.allow i /etc/hosts.deny . . . 242
10.5 Dijagnostičke alatke . . . 242
10.5.1 lsof . . . 243
10.5.2 tcpdump . . . 244
10.5.3 netcat . . . 246
10.5.4 Skeniranje mrežnih priključaka . . . 246
10.6 Daljinsko pozivanje procedura . . . 247
10.7 Zaštita mreže . . . 248
10.7.1 Tipične ranjivosti . . . 249
10.7.2 Internet resursi iz oblasti zaštite mreža . . . 250
10.8 Šta dalje . . . 250
10.9 Mrežne utičnice: kako procesi komuniciraju s mrežom . . . 250
10.10 Utičnice Unix domena . . . 252
10.10.1 Prednosti za projektante aplikacija . . . 252
10.10.2 Prikazivanje utičnica Unix domena . . . 253

11 UVOD U KOMANDNE SKRIPTOVE 255

11.1 Osnove komandnih skriptova . . . 255
11.1.1 Ograničenja komandnih skriptova . . . 256
11.2 Navodnici i literali . . . 256
11.2.1 Literali . . . 257
11.2.2 Polunavodnici . . . 258
11.2.3 Navodnici . . . 258
11.2.4 Prosleđivanje komandi polunavodnika unutar literala . . . 258
11.3 Specijalne promenljive . . . 259
11.3.1 Pojedinačni argumenti: $1, $2, . . . 259
11.3.2 Ukupan broj argumenata: $# . . . 260
11.3.3 Svi argumenti: $@ . . . 260
11.3.4 Ime skripta: $0 . . . 261
11.3.5 ID procesa: $$ . . . 261
11.3.6 Izlazni kôd: $? . . . 261
11.4 Izlazni kodovi . . . 261

11.5 Uslovno izvršavanje komandi. . . . 262
11.5.1 Problem prazne liste parametara 263
11.5.2 Upotreba drugih komandi za ispitivanje uslova 263
11.5.3 elif. . . . 263
11.5.4 Logički konstrukti && i || 264
11.5.5 Ispitivanje svojstava 264
11.5.6 Ispitivanje znakovnih nizova pomoću naredbe case 267
11.6 Petlje 268
11.6.1 Petlje for 268
11.6.2 Petlje while 268
11.7 Zamena komandi 269
11.8 Upravljanje privremenim datotekama 270
11.9 Heredoc sintaksa. . . . 271
11.10 Važne alatke za rad sa komandnim skriptovima 271
11.10.1 basename 271
11.10.2 awk 272
11.10.3 sed 272
11.10.4 xargs 273
11.10.5 expr 274
11.10.6 exec 274
11.11 Podljuske. . . . 274
11.12 Umetanje sadržaja drugih datoteka u skriptove 275
11.13 Učitavanje podataka koje je korisnik uneo 275
11.14 Kada (ne) treba koristiti komandne skriptove 276

12 PRENOŠENJE DATOTEKA MREŽOM 277

12.1 Brzo kopiranje 278
12.2 rsync 278
12.2.1 Osnove programa rsync 278
12.2.2 Izrada identične kopije strukture direktorijuma. . . . 280
12.2.3 Upotreba završne kose crte. . . . 280
12.2.4 Isključivanje određenih datoteka i direktorijuma 281
12.2.5 Integritet prenetih podataka, načini zaštite i opširni režimi 282
12.2.6 Komprimovanje podataka. . . . 283
12.2.7 Ograničavanje propusnog opsega 283
12.2.8 Kopiranje datoteka sa udaljenog na lokalni računar 284
12.2.9 Druge teme u vezi s programom rsync. . . . 284
12.3 Uvod u deljenje datoteka 284
12.4 Deljenje datoteka pomoću kompleta Samba. . . . 285
12.4.1 Konfigurisanje servera. . . . 285
12.4.2 Kontrola pristupa serveru. . . . 286
12.4.3 Lozinke 287
12.4.4 Pokretanje servera 288
12.4.5 Dijagnostika i dnevničke datoteke 288
12.4.6 Konfigurisanje datoteka za deljenje. . . . 289
12.4.7 Matični direktorijumi 289
12.4.8 Deljenje štampača 290
12.4.9 Upotreba Samba klijenta 290
12.4.10 Pristupanje datotekama u svojstvu klijenta 291
12.5 NFS klijenti 292
12.6 Druge mogućnosti za rad s datotekama u mreži i njihova ograničenja 292

13 KORISNIČKA OKRUŽENJA **295**

13.1 Smernice za izradu pokretačkih datoteka . . . 296
13.2 Kada treba menjati postojeće pokretačke datoteke . . . 296
13.3 Elementi pokretačke datoteke ljuske . . . 296
13.3.1 Komandna putanja . . . 297
13.3.2 Putanja stranica priručnika . . . 298
13.3.3 Komandni odzivnik . . . 298
13.3.4 Alijasi . . . 299
13.3.5 Maska podrazumevanih ovlašćenja . . . 299
13.4 Redosled i primeri pokretačkih datoteka . . . 300
13.4.1 Ljuska bash . . . 300
13.4.2 Ljuska tcsh . . . 302
13.5 Podrazumevane vrednosti parametara korisnika . . . 303
13.5.1 Podrazumevana ljuska . . . 303
13.5.2 Editor . . . 304
13.5.3 Podrazumevana komanda za pregledanje tekstualnih datoteka . . . 304
13.6 Klopke u pokretačkim datotekama koje treba izbegavati . . . 304
13.7 Ostale teme u vezi s pokretačkim datotekama . . . 305

14 KRATAK OPIS LINUXOVE RADNE POVRŠINE **307**

14.1 Komponente radne površine . . . 308
14.1.1 Upravljači prozorima . . . 308
14.1.2 Kompleti alatki . . . 309
14.1.3 Okruženja za radnu površinu . . . 309
14.1.4 Aplikacije . . . 309
14.2 Detaljniji uvid u X Window System . . . 310
14.2.1 Upravljač prikazima . . . 310
14.2.2 Mrežna transparentnost . . . 311
14.3 Pregledanje X klijenata . . . 311
14.3.1 X događaji . . . 312
14.3.2 Podešavanje podrazumevanih vrednosti X parametara . . . 313
14.4 Budućnost X Window Systema . . . 315
14.5 D-Bus . . . 316
14.5.1 Sistemska instanca i instance sesija . . . 316
14.5.2 Praćenje poruka na D-Busu . . . 317
14.6 Štampanje . . . 317
14.6.1 CUPS . . . 318
14.6.2 Konverzija formata i filtri za štampanje . . . 318
14.7 Ostale teme u vezi s radnom površinom . . . 319

15 RAZVOJNE ALATKE **321**

15.1 Kompajler jezika C . . . 322
15.1.1 Kombinovanje više izvornih datoteka . . . 323
15.1.2 Datoteke zaglavlja i direktorijumi za umetanje . . . 323
15.1.3 Povezivanje s bibliotekama . . . 325
15.1.4 Deljene biblioteke . . . 327
15.2 Alatka make . . . 330
15.2.1 Jednostavna datoteka Makefile . . . 331
15.2.2 Ugrađena pravila . . . 332
15.2.3 Završno sklapanje programa . . . 332
15.2.4 Održavanje ažurnosti . . . 332

15.2.5 Argumenti i opcije na komandnoj liniji . . . 333
15.2.6 Standardni makroi i promenljive . . . 334
15.2.7 Standardna odredišta . . . 334
15.2.8 Organizovanje datoteke Makefile . . . 335
15.3 Programi za otkrivanje grešaka . . . 336
15.4 Lex i Yacc . . . 337
15.5 Jezici za pisanje skriptova . . . 338
15.5.1 Python . . . 339
15.5.2 Perl . . . 339
15.5.3 Drugi jezici za skriptove . . . 339
15.6 Java . . . 340
15.7 Šta sledi: prevođenje paketa . . . 341

16 UVOD U PREVOĐENJE SOFTVERA IZ C IZVORNOG KODA 343

16.1 Sistemi za sklapanje softvera . . . 344
16.2 Raspakivanje paketa C izvornog koda . . . 344
16.2.1 Odakle početi . . . 345
16.3 Alatka GNU Autoconf . . . 346
16.3.1 Primer upotrebe alatke Autoconf . . . 347
16.3.2 Instaliranje pomoću alatke za pakovanje datoteka . . . 348
16.3.3 Opcije skripta configure . . . 348
16.3.4 Promenljive okruženja . . . 349
16.3.5 Odredišta koja generiše alatka Autoconf . . . 350
16.3.6 Dnevničke datoteke alatke Autoconf . . . 350
16.3.7 Program pkg-config . . . 351
16.4 Praktični saveti u vezi sa instaliranjem paketa . . . 352
16.4.1 Gde instalirati . . . 353
16.5 Ugradnja ispravki softvera . . . 353
16.6 Ispravljanje grešaka pri prevođenju i instaliranju paketa . . . 354
16.6.1 Konkretne greške . . . 355
16.7 Šta dalje . . . 357

17 NADOGRADNJA OSNOVA 359

17.1 Veb serveri i aplikacije . . . 360
17.2 Baze podataka . . . 360
17.2.1 Vrste baza podataka . . . 361
17.3 Virtuelizovanje . . . 361
17.4 Distribuirano računarstvo i računarstvo na zahtev . . . 362
17.5 Ugrađeni sistem . . . 362
17.6 Završne napomene . . . 363

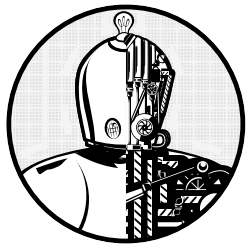
BIBLIOGRAFIJA 365

SPISAK TERMINA KORIŠĆENIH U KNJIZI 369

INDEKS 371

1

OPŠTA SLIKA



Na prvi pogled, jedan savremeni operativni sistem kao što je Linux izgleda vrlo komplikovano, s vrtoglavim brojem sastavnih delova koji rade istovremeno i međusobno komuniciraju. Na primer, veb server može „pričati“ sa serverom baza podataka, koji pak koristi deljenu biblioteku koju koriste i mnogi drugi programi. Ali kako sve to radi zajedno?

Kako radi operativni sistem najlakše ćete razumeti pomoću *apstrakcije* – što je slikovit način da se kaže da mnoge detalje možete zanemariti. Na primer, kada vozite auto, obično nema potrebe da razmišljate o detaljima kao što su zavrtnji koji drže motor unutar karoserije, ili ljudima koji grade i održavaju put po kojem vozite. Ako ste saputnik u vozilu, treba da znate samo šta vozilo radi (prevozi vas nekud) i ono osnovno o upotrebi tog vozila (kako se otvaraju vrata i vezuje sigurnosni pojas).

Ali ukoliko ste vozač auta, potrebno je da znate više. Treba da naučite kako se koriste njegove komande (kao što je volan i pedala gasa) i šta ćete uraditi ako nešto krene naopako.

Na primer, recimo da se tokom vožnje vozilo trese. Tada apstrakciju „vozilo se kreće po putu“ možete podeliti na tri dela: vozilo, put i način na koji vozite. To vam pomaže da ustanovite razlog problema: ako je put neravan, krivicu ne možete prebaciti na vozilo, niti na način na koji vozite. Umesto toga, možda ćete poželeti da saznate zašto je put u lošem stanju ili, ako je put nov, zašto su njegovi graditelji loše uradili svoj posao.

Kada grade operativni sistem i aplikacije za njega, projektanti softvera koriste apstrakciju kao alatku. Postoji više naziva za apstrahovani sastavni deo računarskog softvera, kao što su *podsistem*, *modul* i *paket* – ali u ovom poglavlju koristićemo reč *komponenta* zato što je jednostavna. Kada prave neku softversku komponentu, projektanti obično ne razmišljaju mnogo o internoj strukturi drugih komponenata, ali vode računa o tome koje druge komponente mogu da iskoriste i kako.

Ovo poglavlje pruža opšti uvid u komponente koje čine jedan Linux sistem. Mada se svaka sastoji od ogromnog broja tehničkih detalja koji opisuju njenu internu strukturu, te detalje ćemo zanemariti i usredsrediti se na to šta komponente rade i u kakvoj je to vezi s celim sistemom.

1.1 Nivoi i slojevi apstrakcije Linux sistema

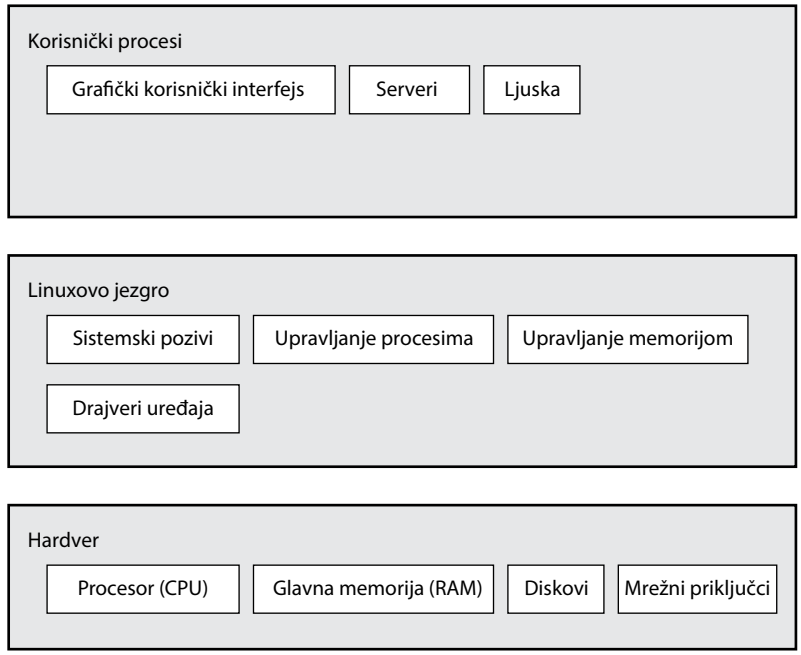
Upotreba apstrakcije radi deljenja računarskih sistema na komponente olakšava razumevanje stvari, ali je za to neophodna organizacija. Komponente razvrstavamo u slojeve ili nivoe. *Sloj* ili *nivo* se dobija klasifikacijom (tj. grupisanjem) date komponente na osnovu mesta gde se komponenta nalazi između korisnika i hardvera. Čitači veba (engl. *web browsers*), igre i slične aplikacije nalaze se u najvišem sloju; u najnižem sloju imamo memoriju u računarskom hardveru – nule i jedinice. Operativni sistem zauzima više slojeva između pomenuta dva.

Linux sistem ima tri glavna nivoa. Slika 1-1 prikazuje te nivoe i neke od komponenata unutar svakog nivoa. Osnovni nivo čini *hardver*. Hardver se sastoji od memorije i jedne ili više centralnih procesorskih jedinica (engl. *central processing unit*, CPU) koje obavljaju računske operacije i učitavaju podatke iz memorije, odnosno upisuju podatke u memoriju. Uređaji kao što su diskovi i mrežni interfejsi takođe su sastavni delovi hardvera.

Sledeći nivo je *jezgro* (engl. *kernel*) što je „srce“ operativnog sistema. Jezgro je softver koji se učitava u memoriju i govori procesoru šta treba da uradi. Jezgro upravlja upotrebom hardvera i prevashodno igra ulogu interfejsa između hardvera i svakog programa koji se izvršava.

Procesi – programi koji se izvršavaju i kojima upravlja jezgro – zajedno čine najviši nivo sistema, koji se zove *korisnički prostor* (engl. *user space*). (Precizniji izraz za proces je *korisnički proces*, bez obzira na to da li korisnik neposredno komunicira s procesom. Na primer, svi veb serveri rade kao korisnički procesi.)

Postoji ključna razlika između načina na koji rade jezgro i korisnički procesi: jezgro radi u *režimu jezgra* (engl. *kernel mode*), dok korisnički procesi rade u *korisničkom režimu* (engl. *user mode*). Programski kôd koji radi u režimu jezgra ima neograničen pristup procesoru i glavnoj memoriji. To je moćna, ali i opasna privilegija jer procesu koji radi u režimu jezgra omogućava da lako sruši ceo sistem. Oblast kojoj može da pristupa samo jezgro, zove se *prostor jezgra* (engl. *kernel space*).



Slika 1-1: Opšta organizacija Linux sistema

S druge strane, korisnički režim omogućava pristup samo u (najčešće prilično manji) podskup memorije i izvršavanje bezbednih CPU operacija. *Korisnički prostor* predstavlja delove glavne memorije u koje je odobren pristup korisničkim procesima. Ako se u procesu javi greška i on se sruši, posledice su ograničene, a jezgro može da „sredi" stanje posle njih. To znači sledeće: ako se vaš veb server sruši, to verovatno neće prekinuti obavljanje naučnih proračuna koje se već danima odvija u pozadini.

Teorijski gledano, korisnički proces koji se oteo kontroli ne može načiniti ozbiljnu štetu u preostalom delu sistema. U stvarnosti, to zavisi od onoga što podrazumevate pod „ozbiljnom štetom" i od konkretnih privilegija tog procesa, budući da je nekim procesima dozvoljeno da rade više stvari nego drugima. Na primer, da li neki korisnički proces može potpuno uništiti podatke na disku? Da, ako ima odgovarajuća ovlašćenja – a to možete smatrati prilično opasnim. Međutim, postoje bezbednosni mehanizmi da se to spreči, a većini procesa se jednostavno ne dozvoljava da čine štetu na taj način.

1.2 Hardver: glavna memorija

Od celokupnog hardvera koji čini računarski sistem, *glavna memorija* je možda njegov najvažniji deo. U svom najjednostavnijem obliku, glavna memorija je samo veliko mesto za skladištenje gomile nula i jedinica. Svaka nula ili jedinica zove se *bit*. Memorija je mesto gde „borave" jezgro i procesi koji se u datom trenutku izvršavaju – to su samo velike zbirke bîta. Ceo unos sa perifernih uređaja i sve što se njima prosleđuje protiče kroz glavnu memoriju,

takođe u obliku gomile bîta. CPU je samo operater koji radi s memorijom; učitava instrukcije i podatke iz memorije i upisuje rezultujuće podatke nazad u memoriju.

U vezi s memorijom, procesima, jezgrom i drugim delovima računarskog sistema često ćete čuti reč *stanje* (engl. *state*). Strogo govoreći, stanje je konkretna kombinacija bîta. Na primer, ako u memoriji imate četiri bita, 0110, 0001 i 1011 predstavljaju tri različita stanja.

Kada uzmete u obzir da jedan proces lako može biti sačinjen od više miliona bîta u memoriji, često je lakše da kada govorite o stanjima, koristite apstraktne izraze. Umesto da stanje opišete pomoću bîta, opisujete šta je urađeno ili šta se radi u datom trenutku. Na primer, možete reći „proces čeka unošenje podataka“ ili „proces izvršava korak 2 postupka svog pokretanja“.

NAPOMENA *Pošto je uobičajeno da se stanje opisuje pomoću apstraktnih termina umesto pomoću stvarnih bîta, izraz* slika *(engl.* image*) predstavlja određenu fizičku kombinaciju bîta.*

1.3 Jezgro

Zbog čega govorimo o glavnoj memoriji i stanjima? Gotovo sve što jezgro radi tiče se glavne memorije. Jedan od zadataka jezgra jeste da deli memoriju na više odeljaka i da sve vreme održava određene informacije o stanju tih odeljaka. Svaki proces dobija svoj deo memorije, a jezgro mora da se stara o tome da svaki proces ima pristup samo u svoj deo.

Jezgro je odgovorno za upravljanje poslovima u sledeće četiri opšte sistemske oblasti:

Procesi Jezgro je odgovorno za određivanje kojim je procesima dozvoljeno da koriste CPU.

Memorija Jezgro mora da nadzire upotrebu celokupne memorije – šta je u datom trenutku dodeljeno konkretnom procesu, šta bi moglo da se deli između više procesa i šta je slobodno.

Drajveri uređaja Jezgro igra ulogu interfejsa između hardvera (kao što je disk) i procesa. Direktan rad s hardverom najčešće je posao jezgra.

Sistemski pozivi i podrška U normalnim okolnostima procesi komuniciraju s jezgrom pomoću sistemskih poziva.

Sada ćemo ukratko istražiti svaku od pomenutih oblasti.

NAPOMENA *Ako vas zanimaju detalji o tome kako radi jezgro, dva dobra priručnika su* Operating System Concepts*, 9. izdanje, čiji su autori Abraham Silberschatz, Peter B. Galvin i Greg Gagne (izdavač je Wiley, 2012) i* Modern Operating Systems*, 4. izdanje, čiji su autori Andrew S. Tanenbaum i Herbert Bos (izdavač je Prentice Hall, 2014).*

1.3.1 Upravljanje procesima

Upravljanje procesima obuhvata pokretanje, pauziranje, nastavljanje i okončavanje procesa. Koncepti pokretanja i okončavanja procesa prilično su lako razumljivi, ali je teže opisati kako proces koristi CPU tokom svog uobičajenog rada.

U svakom savremenom operativnom sistemu, mnogi procesi rade „istovremeno“ Na primer, na svom stonom računaru možete imati otvorene u isto vreme čitač veba i radni list. Međutim, stvari se ne odvijaju onako kako izgleda: procesi iza tih aplikacija uglavnom se ne odvijaju *baš* u isto vreme.

Razmotrite sistem čiji CPU ima samo jedno jezgro. Više procesa *može* koristiti taj CPU, ali u svakom datom trenutku samo jedan proces zaista koristi taj CPU. U praksi, svaki proces koristi CPU tokom kratkog dela sekunde, a zatim pauzira; u tom trenutku drugi proces dobija CPU tokom drugog delića sekunde; zatim neki treći proces preuzima CPU itd. Događaj kada jedan proces prepušta CPU drugom procesu zove se *promena konteksta* (engl. *context switch*).

Svaki takav delić vremena – koji se zove *isečak vremena* (engl. *time slice*) – dovoljno je dugačak da proces može obaviti značajnu količinu proračuna (često se događa da proces do kraja obavi ceo svoj tekući posao u samo jednom isečku). Međutim, pošto su ti isečci tako kratki, ljudi ih ne mogu primetiti, pa im deluje da sistem izvršava više procesa u isto vreme (to se zove *višeprogramski rad* /engl. *multitasking/*).

Za promene konteksta odgovorno je jezgro. Da biste razumeli kako to funkcioniše, zamislite situaciju u kojoj neki proces radi u korisničkom režimu i njegov isečak vremena je upravo istekao. Evo šta se događa:

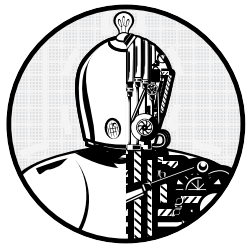
1. CPU (sam hardver) prekida tekući proces na osnovu internog tajmera, prelazi u režim jezgra i predaje kontrolu jezgru.
2. Jezgro beleži tekuće stanje CPU-a i memorije, što će kasnije biti ključno za nastavljanje procesa koji je upravo prekinut.
3. Jezgro obavlja poslove koji su možda bili pokrenuti tokom prethodnog isečka vremena (kao što je prikupljanje podataka koje je korisnik zadao ili podataka za ispisivanje, tj. ulazno/izlazne operacije).
4. Sada je jezgro spremno da omogući drugom procesu da se pokrene. Jezgro analizira listu procesa koji su spremni za pokretanje i iz nje bira jedan proces.
5. Jezgro priprema memoriju za taj novi proces, a onda priprema i CPU.
6. Jezgro obaveštava CPU koliko će trajati isečak vremena za novi proces.
7. Jezgro prebacuje CPU u korisnički režim i predaje CPU procesu.

Promena konteksta odgovara na važno pitanje *kada* jezgro radi. Ogovor je da radi tokom promena konteksta, u vremenskim isečcima *između* isečaka dodeljenih procesima.

U slučaju sistema s više CPU jedinica, stvari postaju nešto složenije zato što jezgro ne mora da prepusti svoj tekući CPU kako bi omogućilo datom procesu da se pokrene na drugoj CPU jedinici. Međutim, da bi maksimiralo upotrebu svih raspoloživih CPU jedinica, jezgro najčešće ipak baš to radi (a može primenjivati i određene trikove da bi za sebe prigrabilo malo više CPU vremena).

4

DISKOVI I SISTEMI DATOTEKA

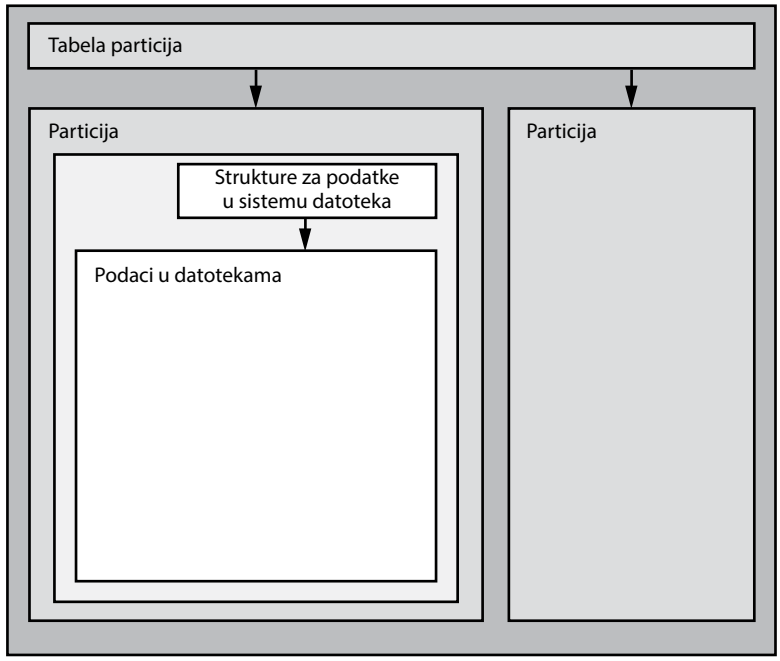


U poglavlju 3 razmatrali smo neke od disk uređaja najvišeg nivoa koje jezgro stavlja na raspolaganje u sistemu. U ovom poglavlju, detaljno opisujemo kako se radi s diskovima u Linux sistemu. Naučićete da delite diskove na particije, formirate i održavate sisteme datoteka na particijama i radite s prostorom za razmenu.

Podsećamo da disk uređaji imaju imena kao što je */dev/sda*, prvi disk u SCSI podsistemu. Ta vrsta blok uređaja predstavlja jedan ceo disk, ali unutar jednog diska postoji više raznih podela i slojeva.

Slika 4-1 ilustruje šemu tipičnog Linuxovog diska (imajte u vidu da slika ne prikazuje tačne razmere). Tokom napredovanja kroz ovo poglavlje saznaćete gde se svaki deo uklapa u celinu.

Particije (engl. *partitions*) jesu oblasti na koje je izdeljen ceo prostor diska. U Linuxu, one se obeležavaju rednim brojem iza imena blok uređaja, pa zato postoje imena uređaja kao što su */dev/sda1* i */dev/sdb3*. Jezgro predstavlja svaku particiju kao jedan blok uređaj, u istom obliku kao i ceo disk. Raspored particija je definisan u jednoj maloj oblasti diska koja se zove tabela particija (engl. *partition table*).



Slika 4-1: Tipična šema Linuxovog diska

NAPOMENA *Deljenje diska na više particija nekad je bilo uobičajeno na sistemima s velikim diskovima zato što su stariji PC računari mogli da se podižu samo sa određenih delova diska. Osim toga, administratori su koristili particije da bi rezervisali izvesnu količinu prostora u delovima namenjenim operativnom sistemu; na primer, nisu želeli da korisnici budu u stanju da zauzmu ceo sistem i tako onemoguće rad važnih sistemskih servisa. Ta praksa nije jedinstvena za Unix; i dalje ćete naći brojne nove Windows sisteme s više particija na jedinom disku. Osim toga, većina sistema ima posebnu particiju za razmenu.*

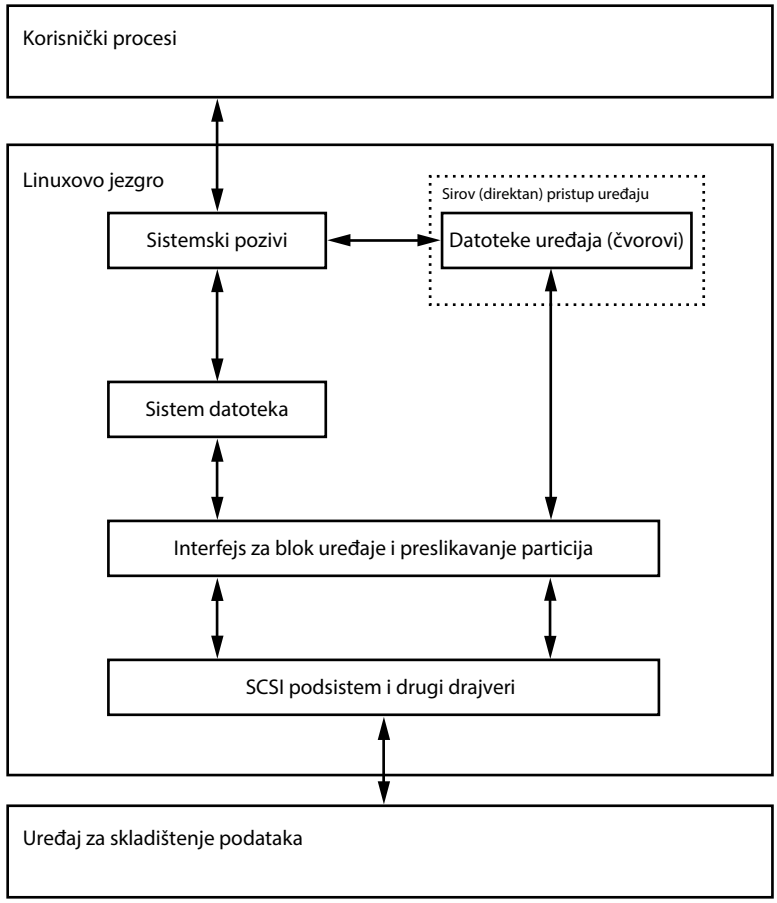
Mada jezgro pruža mogućnost da pristupate istovremeno i celom disku i jednoj od njegovih particija, to u normalnim okolnostima ne biste činili, osim kada kopirate ceo disk.

Sledeći sloj ispod particije jeste sistem datoteka (engl. *filesystem*), što je baza podataka datoteka i direktorijuma s kojom ste navikli da radite u korisničkom prostoru. Sisteme datoteka razmatramo u odeljku 4.2.

Kao što se vidi na slici 4-1, ako želite da pristupate podacima u nekoj datoteci, iz tabele particija morate pribaviti lokaciju odgovarajuće particije i zatim pretražiti bazu podataka sistema datoteka na toj particiji da biste dobili podatke iz tražene datoteke.

Kada pristupate podacima na disku, Linuxovo jezgro koristi sistem slojeva prikazan na slici 4-2. SCSI podsistem i sve ostalo opisano i odeljku 3.6 ovde je predstavljeno jednim pravougaonikom. (Imajte u vidu da s diskom možete raditi i kroz sistem datoteka i direktno pomoću disk uređaja. U ovom poglavlju radićete i jedno i drugo.)

Da biste stekli utisak kako se sve to zajedno uklapa, krenućemo od najnižeg nivoa, tj. od particija.



Slika 4-2: Šema načina na koji se disku pristupa iz jezgra

4.1 Deljenje disk uređaja na particije

Postoji više vrsta tabela particija. Tradicionalna tabela je format koji se nalazi u MBR (*Master Boot Record – glavni zapis za podizanje sistema*). Noviji standard koji počinje da stiče popularnost jeste *GPT (Globally Unique Identifier Partition Table – tabela particija s globalno jedinstvenim identifikatorima)*.

Evo kratkog pregleda brojnih alatki za deljenje diskova na particije u Linuxu:

`parted` Tekstualna alatka koja podržava i MBR i GPT.

`gparted` Grafička verzija alatke `parted`.

`fdisk` Tradicionalna tekstualna Linuxova alatka za izradu particija koja ne podržava GPT.

`gdisk` Verzija programa `fdisk` koja podržava GPT ali ne MBR.

Budući da podržava i MBR i GPT format, u ovoj knjizi ćemo koristiti program `parted`. Međutim, mnogi ljudi više vole interfejs programa `fdisk`, u čemu nema ničeg lošeg.

NAPOMENA *Mada program `parted` može da pravi nove sisteme datoteka i menja veličinu postojećih, nije preporučljivo da ga koristite za manipulisanje sistemima datoteka jer lako može doći do zabune. Postoji ključna razlika između deljenja diska na particije i manipulisanja sistemom datoteka. Tabela particija samo definiše određene granice na disku, dok je sistem datoteka znatno složeniji sistem za rad s podacima. Iz tog razloga, za deljenje diska na particije koristićemo program `parted` a sisteme datoteka pravićemo pomoću drugih alatki (videti odeljak 4.2.2). Čak se i u dokumentaciji programa `parted` savetuje da sisteme datoteka pravite pomoću zasebnih alatki.*

4.1.1 Pregledanje tabele particija

Tabelu particija u svom sistemu možete prikazati pomoću komande **`parted -l`**. Evo kako izgleda primer rezultata na računaru s dva disk uređaja i dve različite vrste tabela particija:

```
# parted -l
Model: ATA WDC WD3200AAJS-2 (scsi)
Disk /dev/sda: 320GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End    Size    Type      File system     Flags
 1      1049kB  316GB  316GB   primary   ext4            boot
 2      316GB   320GB  4235MB  extended
 5      316GB   320GB  4235MB  logical   linux-swap(v1)

Model: FLASH Drive UT_USB20 (scsi)
Disk /dev/sdf: 4041MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt

Number  Start   End     Size    File system  Name        Flags
 1      17.4kB  1000MB  1000MB               myfirst
 2      1000MB  4040MB  3040MB               mysecond
```

Prvi uređaj, */dev/sda*, koristi tradicionalnu MBR tabelu particija (koju `parted` zove „msdos“), a drugi disk sadrži tabelu tipa GPT. Obratite pažnju na to da se za svaku tabelu particija prikazuju različiti parametri, zato što su u pitanju drugačije tabele. Konkretno, u MBR tabeli nema kolone Name (ime) zato što u tom formatu ne postoji ime particije. (U GPT tabeli sam proizvoljno izabrao imena `myfirst` i `mysecond`.)

U ovom primeru, MBR tabela sadrži primarnu, proširenu i logičku particiju. *Primarna* (engl. *primary) particija* standardni je oblik podele diska na manje oblasti; takva particija je particija 1. Pošto je osnovni MBR ograničen na najviše četiri primarne particije, ako vam treba više od četiri, jednu od particija definišete kao proširenu (engl. *extended) particiju*. Proširenu particiju

dalje delite na *logičke* (engl. *logical) particije* koje operativni sistem može da koristi isto kao i svaku drugu particiju. U ovom primeru, particija 2 je proširena particija koja sadrži logičku particiju 5.

NAPOMENA *Podatak o tipu sistema datoteka (system ID) koji* `parted` *prikazuje, ne mora obavezno biti sadržaj polja system ID koje je definisano u većini MBR stavki. MBR-ov system ID je običan broj; na primer, 83 predstavlja standardnu Linuxovu particiju a 82 Linuxovu particiju za razmenu. To znači da* `parted` *pokušava da sam prepozna tip sistema datoteka. Ako apsolutno morate da znate sadržaj polja system ID u MBR tabeli, upotrebite* `fdisk -l`.

Početno učitavanje u jezgro

Kada na početku učitava MBR tabelu, Linuxovo jezgro šalje sledeću dijagnostičku poruku (podsećamo da je možete videti pomoću komande `dmesg`):

```
sda: sda1 sda2 < sda5 >
```

Deo `sda2 < sda5 >` znači da je */dev/sda2* proširena particija koja sadrži jednu logičku particiju, */dev/sda5*. Proširene particije ćete uglavnom zanemarivati jer ćete, u normalnim okolnostima, uvek pristupati samo logičkim particijama unutar proširene.

4.1.2 Menjanje tabela particija

Pregledanje tabela particija relativno je jednostavna i bezopasna operacija. Mada je menjanje postojećih tabela particija takođe relativno jednostavno, tu vrstu promena na disku prati i određeni rizik. Imajte na umu sledeće:

- Menjanje tabele particija značajno otežava restauriranje podataka na particijama koje izbrišete zato što se tako menja inicijalna referentna tačka u sistemu datoteka. Obavezno napravite rezervnu kopiju podataka ako disk čije particije menjate sadrži važne podatke.
- Postarajte se da nijedna particija na ciljnom disku nije u upotrebi. To je važan korak zato što većina Linux distribucija automatski montira svaki sistem datoteka koji nađe. (Više informacija o montiranju i demontiranju diskova naći ćete u odeljku 4.2.3.)

Kada budete spremni, izaberite program za deljenje na particije. Ako biste želeli da koristite `parted`, možete upotrebiti alatku za komandnu liniju `parted` ili neki grafički interfejs kao što je `gparted`; ako želite interfejs u stilu programa `fdisk`, upotrebite `gdisk` ako radite s GPT particijama. Za te alatke postoji ugrađena pomoć i lako se uče. (Isprobajte njihovu upotrebu ne nekom fleš uređaju ili nečem sličnom ako nemate rezervne diskove.)

Međutim, postoji bitna razlika između načina na koji `fdisk` i `parted` rade. Kada koristite `fdisk`, novu tabelu particija morate osmisliti pre nego što zaista izmenite disk; `fdisk` unosi izmene tek kad izađete iz tog programa. Nasuprot tome, `parted` pravi, menja i briše particije *čim izdate komandu*. Nemate priliku da pregledate tabelu particija pre nego što je izmenite.

Te razlike su važne i za razumevanje kako te dve alatke komuniciraju s jezgrom. I `fdisk` i `parted` menjaju particije u potpunosti u korisničkom prostoru; nema potrebe da se u jezgro ugrađuje podrška za ponovno pisanje tabele particija zato što se iz korisničkog prostora može čitati i menjati sadržaj celog blok uređaja.

Međutim, jezgro mora da u nekom trenutku učita tabelu particija kako bi moglo da ih predstavi u obliku blok uređaja. Alatka `fdisk` koristi relativno jednostavnu metodu: pošto izmeni tabelu particija, `fdisk` šalje sistemski poziv kojim obaveštava jezgro da treba da ponovo učita tabelu particija. Jezgro zatim generiše dijagnostičku poruku koju možete videti pomoću alatke `dmesg`. Na primer, ako na */dev/sdf* napravite dve particije, videćete sledeće:

```
sdf: sdf1 sdf2
```

S druge strane, alatke `parted` ne koriste sistemski poziv koji se odnosi na ceo disk. Umesto toga, signaliziraju jezgru kad god izmene neku particiju na disku. Pošto obradi izmenu koja se odnosi na jedni particiju, jezgro ne generiše dijagnostičku poruku kao u prethodnom slučaju.

Postoji nekoliko načina da saznate da je neka particija izmenjena:

- Upotrebite komandu `udevadm` da biste pratili događaje o izmenama koje jezgro šalje. Na primer, `udevadm monitor --kernel` pokazaće da su uklonjeni uređaji koji su predstavljali stare particije i da su dodati novi.
- Pregledajte */proc/partitions* ako vam trebaju kompletni podaci o particijama.
- Pregledajte */sys/block/device/* da biste videli sistemske interfejse izmenjenih particija odnosno */dev* da biste videli uređaje koji predstavljau izmenjene particije.

Ako morate biti apsolutno sigurni da ste zaista izmenili tabelu particija, pomoću komande `blockdev` možete poslati sistemski poziv u starom stilu koji `fdisk` koristi. Na primer, da biste naredili jezgru da ponovo učita tabelu particija na disku */dev/sdf*, zadajte sledeće:

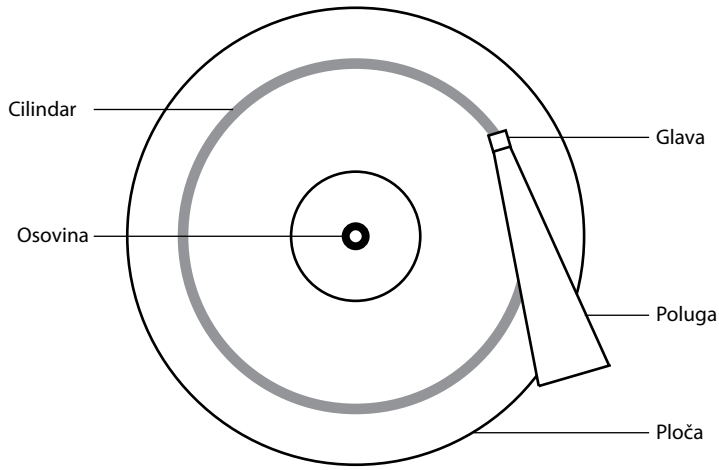
```
# blockdev --rereadpt /dev/sdf
```

Zasad je ovo sve što treba da znate o podeli diska na particije. Međutim, ako vas zanima nešto više o diskovima, nastavite čitanje. U suprotnom, pređite na odeljak 4.2 da biste naučili kako se formira nov sistem datoteka na disku.

4.1.3 Geometrija diska i particija

Svaki uređaj s pokretnim delovima uvodi dodatnu komplikaciju u softverski sistem zato što postoje fizički elementi koje je teško apstrahovati. Čvrsti disk nije izuzetak; čak i ako čvrsti disk zamislite kao blok uređaj s nasumičnim pristupanjem svakom bloku, postoje ozbiljne posledice po performanse ako ne vodite računa o tome kako raspoređujete podatke na disk. Razmotrite fizička svojstva jednostavnog diska s jednom pločom, prikazanog na slici 4-3.

Disk se sastoji od ploče koja se okreće oko svoje osovine, i od glave na pokretnoj poluzi koja može da se kreće duž radijusa diska. Kako se disk okreće ispod glave, glava učitava podatke. Kada se glava nalazi u datom položaju, može da čita podatke samo iz fiksnog kruga na disku. Taj krug se zove *cilindar* zato što veći diskovi imaju više od jedne ploče, koje su sve nanizane na istu osovinu. Svaka ploča može imati jednu ili dve glave, za početak i/ili kraj ploče, a sve glave su postavljene na istu polugu i kreću se zajedno. Pošto se poluga pokreće, na disku postoji veliki broj cilindara, od malih u blizini centra, do velikih blizu periferije diska. I najzad, cilindar se može podeliti na isečke koji se zovu *sektori*. Taj način predstavljanja geometrije diska zove se *CHS*, od *cylinder-head-sector* (cilindar-glava-sektor).



Slika 4-3: Prikaz elemenata čvrstog diska

NAPOMENA *Staza* (engl. *track) jeste ceo deo cilindra kojem može da pristupa jedna glava, pa je zato na slici 4-3, cilindar isto što i staza. Verovatno vam neće trebati da brinete i o stazama.*

Jezgro i razni programi za deljenje na particije mogu da vam kažu šta disk prijavljuje kao svoj broj cilindara (i *sektora*, što su isečci cilindara). Međutim, na savremenom čvrstom disku, te *prijavljene vrednosti ne odgovaraju ničem stvarnom*! Tradicionalan način adresiranja koji koristi CHS nije primenjiv na savremen hardver diskova, niti uzima u obzir činjenicu da možete smestiti više podataka na spoljašnje cilindre nego na unutrašnje. Hardver današnjih diskova podržava adresiranje po logičkim blokovima (*Logical Block Addressing, LBA)* gde se lokacija na disku adresira jednostavno po broju bloka, ali ostaci CHS-a i dalje žive. Na primer, MBR tabela particija sadrži i CHS podatke i njihove LBA ekvivalente, a neki programi za podizanje sistema i dalje su toliko ograničeni da veruju CHS vrednostima (ne brinite – većina programa za podizanje Linuxa koristi LBA vrednosti).

Uprkos tome, ideja cilindara je važna za podelu na particije zato što su cilindri savršene granice za particije. Tok podataka se sa cilindra učitava vrlo

brzo, zato što glave mogu da bez prekida primaju podatke tokom obrtanja ploče diska. Particija uređena kao skup susednih cilindara omogućava i brzo pristupanje podacima koji slede jedni drugima, zato što glava ne mora mnogo da se pomera između cilindara.

Neki programi za deljenje na particije „protestuju“ ako granice particija ne postavite tačno na granice cilindara. Zanemarite te proteste jer po tom pitanju možete malo toga uraditi pošto prijavljene CHS vrednosti na savremenim diskovima jednostavno nisu tačne. LBA podela diska obezbeđuje da vaše particije budu tačno tamo gde hoćete da budu.

4.1.4 Diskovi bez pokretnih delova (SSD)

Uređaji za skladištenje podataka bez pokretnih delova, kao što su elektronski diskovi (engl. *solid-state disks, SSDs)*, radikalno se razlikuju od elektromehaničkih diskova po karakteristikama pristupanja podacima. U njihovom slučaju, nasumično pristupanje nije problem zato što nema glave koja se kreće iznad ploče, ali određeni činioci ipak utiču na performanse.

Jedan od najznačajnijih činilaca koji utiče na performanse SSD uređaja jeste poravnavanje particija. Kada učitavate podatke sa SSD uređaja, to činite u blokovima – najčešće po 4096 bajtova odjednom – a čitanje mora da započne od nekog umnoška te veličine. Ako particija i podaci u njoj ne počinju na nekoj 4096-bajtnoj granici, možda će vam trebati dva učitavanja za jednu kratku i vrlo čestu operaciju, kao što je učitavanje sadržaja direktorijuma.

Budući da su mnoge alatke za deljenje na particije (na primer, `parted` i `gparted`) u stanju da novodefinisane particije postave na odgovarajuća rastojanja od početka diskova, verovatno nećete nikad morati da brinete o pogrešnom poravnavanju particija. Međutim, ako ste radoznali u vezi s time gde vaše particije počinju i želite samo da se uverite da zaista počinju na odgovarajućoj granici, do tog podatka možete lako doći ako pogledate u */sys/block*. Evo primera particije na */dev/sdf2*:

```
$ cat /sys/block/sdf/sdf2/start
1953126
```

Ova particija počinje na 1.953.126 bajtova od početka diska. Pošto taj broj nije deljiv s 4096, particija ne bi pružala optimalne performanse kada bi bila na SSD uređaju.

4.2 Sistemi datoteka

Poslednja karika lanca između jezgra i korisničkog prostora u slučaju diskova obično je sistem datoteka; to je ono s čime ste navikli da radite kada zadajete komande kao što su `ls` i `cd`. Kao što smo već pomenuli, sistem datoteka je oblik baze podataka, koja obezbeđuje strukturu pomoću koje se jednostavan blok uređaj pretvara u složenu hijerarhiju datoteka i poddirektorijuma koju korisnici mogu da razumeju.

Nekada su sistemi datoteka bili smešteni na diskove i druge fizičke medijume koji su se koristili isključivo za čuvanje podataka. Međutim, pošto su struktura u obliku stabla i ulazno/izlazni interfejs sistema datoteka prilično

raznovrsni, sistemi datoteka danas obavljaju razne vrste poslova, kao što su sistemski interfejsi koje vidite u */sys* i */proc*. Osim toga, sistemi datoteka se tradicionalno implementiraju u jezgru, ali inovacija 9P iz Plana 9 (*http://plan9.bell-labs.com/sys/doc/9.html*) inspirisala je razvoj sistema datoteka i u korisničkom prostoru. Mogućnost FUSE (*File System in User Space – sistem datoteka u korisničkom prostoru*) dozvoljava sisteme datoteka u korisničkom prostoru na Linuxu.

Sloj apstrakcije *VFS (Virtual File System – virtuelni sistem datoteka*) zaokružuje implementaciju sistema datoteka. Slično kao što SCSI podsistem standardizuje komunikaciju između pojedinih tipova uređaja i upravljačkih komandi iz jezgra, VFS obezbeđuje da sve konkretne implementacije sistema podržavaju standardizovan interfejs kako bi aplikacije iz korisničkog prostora mogle da na isti način pristupaju datotekama i direktorijumima. Podrška za VFS je omogućila Linuxu da podržava izuzetno veliki broj različitih sistema datoteka.

4.2.1 Tipovi sistema datoteka

Linuxova podrška za sistem datoteka obuhvata izvorne dizajne koji su optimizovani za Linux, spoljašnje tipove kao što je Windowsova FAT porodica, univerzalne sisteme datoteka kao što je ISO 9660 i mnoge druge. Sledeća lista sadrži najuobičajenije tipove sistema datoteka za skladištenje podataka. Imena tipova koje Linux prepoznaje navedena su između zagrada, pored imena sistema datoteka.

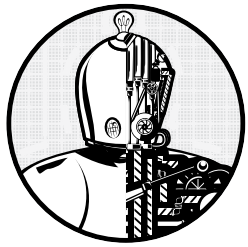
- *Fourth Extended filesystem* (ext4) tekuća je iteracija linije sistema datoteka koja je izvorno projektovana za Linux. Verzija *Second Extended filesystem* (ext2) dugo se podrazumevala u Linux sistemima koji su vodili poreklo od tradicionalnih Unixovih sistema datoteka kao što je UFS (Unix File System) i FFS (Fast File System). Verzija *Third Extended filesystem* (ext3) dodala je mogućnost koja se zvala žurnal (mali keš izvan normalne strukture za podatke sistema datoteka) i koja je poboljšavala integritet podataka i ubrzavala podizanje sistema. Verzija ext4 tog sistema datoteka inkrementno je poboljšanje koje je uvelo podršku za datoteke veće nego što su podržavale verzije ext2 ili ext3 i podršku za više poddirektorijuma.

  Postoji izvestan nivo kompatibilnosti unazad između članova ovog niza sistema datoteka. Na primer, svaku verziju ext2 i ext3 možete montirati kao da je ona druga, a ext2 i ext3 možete montirati i kao ext4, ali *ne možete* montirati ext4 kao ext2 ili ext3.

- *ISO 9660* (iso9660) jeste standard za CD-ROM. Većina CD-ROM uređaja koristi neku varijantu standarda ISO 9660.
- *FAT sistemi datoteka* (msdos, vfat, umsdos) razvijeni su za Microsoftove sisteme. Jednostavan tip msdos podržava vrlo primitivnu varijantu s jednom vrstom slova (samo velika) u MS-DOS sistemima. U većini savremenih sistema datoteka za Windows, trebalo bi da koristite sistem datoteka vfat kako biste imali pun pristup iz Linuxa. Sistem datoteka umsdos retko se koristi i svojstven je Linuxu. Podržava Unixove mogućnosti kao što su simboličke veze povrh sistema datoteka za MS-DOS.
- *HFS+* (hfsplus) Appleov je standard koji se koristi u većini Macintosh sistema.

9

MREŽA I NJENO KONFIGURISANJE



Umrežavanje je postupak međusobnog povezivanja više računara i razmene podataka između njih. To zvuči baš jednostavno, ali da biste razumeli kako to funkcioniše, morate sebi postaviti dva suštinska pitanja:

- Kako računar koji šalje svoje podatke zna *gde* da ih pošalje?
- Kada računar na odredištu primi podatke, kako zna *šta* je upravo primio?

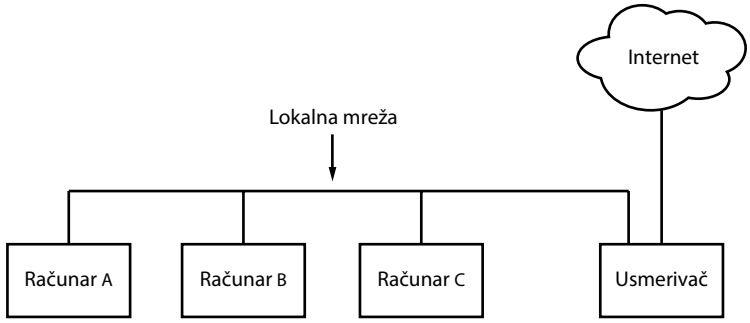
Računar odgovara na ta pitanja tako što koristi niz raznih komponenata, a svaka je odgovorna za određeni aspekt slanja, primanja i prepoznavanja podataka. Komponente su raspoređene u grupe koje čine *mrežne slojeve,* a oni se slažu jedan preko drugog da bi se formirao kompletan sistem. Linuxovo jezgro omogućava rad u mreži slično kao SCSI podsistem opisan u poglavlju 3.

Budući da je svaki sloj projektovan tako da ne zavisi od drugih slojeva, moguće je graditi mreže pomoću velikog broja raznih kombinacija komponenata. U toj fazi, konfiguracija mreže može postati vrlo složena, pa ovo poglavlje započinjemo razmatranjem slojeva u vrlo jednostavnim mrežama. Naučićete kako da pregledate parametre svoje mreže, a kada savladate osnove načina rada svakog sloja, bićete spremni da naučite kako da sami konfigurišete te slojeve. I

najzad, preći ćete na naprednije teme kao što su građenje vlastitih mreža i podešavanje zaštitnih bedema. (Ako ste počeli da širite oči u čuđenju, preskočite ovaj materijal; na njega se uvek možete vratiti kasnije.)

9.1 Osnove mreža

Pre nego što krenemo s teorijom mrežnih slojeva, pogledajte jednostavnu mrežu prikazanu na slici 9-1.



Slika 9-1: Tipična lokalna mreža sa usmerivačem koji omogućava pristup internetu

Ova vrsta mreže je univerzalna; većina kućnih mreža i mreža manjih organizacija konfigirisana je u tom obliku. Svaki računar povezan s mrežom zove se *umrežen računar*. Svi računari u mreži povezani su s *mrežnim usmerivačem* (engl. *router*), što je računar koji zna kako da usmerava podatke ka jednoj ili drugoj mreži. Sve te mašine (u ovom primeru, umreženi računari A, B i C) i usmerivač zajedno čine *lokalnu mrežu* (engl. *local area network,* LAN). Veze između računara unutar jednog LAN-a mogu biti žičane ili bežične.

Usmerivač je povezan i na internet – na slici je to oblak. Pošto je usmerivač istovrmeno povezan i u lokalnu mrežu i na internet, svi računari unutar lokalne mreže takođe imaju pristup internetu kroz usmerivač. Jedan od ciljeva ovog poglavlja je da shvatite kako usmerivač omogućava taj pristup.

Vaš početni predmet razmatranja biće Linux mašina kao što je Računar A u lokalnoj mreži na slici 9-1.

9.1.1 Paketi

Računar šalje podatke u mrežu u obliku malih blokova koji se zovu *paketi* i sastoje se od dva dela: *zaglavlja* (engl. *header*) i *korisnog dela* (engl. *payload*). Zaglavlje sadrži identifikacione podatke kao što su izvorni/odredišni računar i osnovni protokol. S druge strane, korisni deo čine stvarni podaci aplikacije koje računar želi da pošalje (na primer, HTML podaci ili podaci koji čine sliku).

Paketi omogućavaju izvornom računaru da komunicira s drugima „istovremeno“ zato što računari mogu da šalju, primaju i obrađuju pakete proizvoljnim redosledom, bez obzira na to da odakle su stigli ili kuda su upućeni. Deljenje poruka na manje jedinice takođe olakšava otkrivanje i ispravljanje grešaka tokom prenošenja poruka.

Najvećim delom nećete morati da brinete o prevođenju između sirovih paketa i podataka s kojim radi vaša aplikacija, zato što su u operativni sistem ugrađene komponente koje to rade za vas. Međutim, korisno je da znate ulogu paketa u mrežnim slojevima koje ćemo sada razmotriti.

9.2 Mrežni slojevi

Potpuno funkcionalna mreža sastoji se od skupa mrežnih slojeva koji se zove *mrežni slog* (engl. *network stack*). Svaka funkcionalna mreža ima takav slog. Tipičan internet slog, redom od najvišeg ka najnižem sloju, izgleda ovako:

Sloj aplikacija Sadrži „jezik" kojim međusobno komuniciraju aplikacije i serveri; najčešće neka vrsta protokola visokog nivoa. Uobičajeni protokoli sloja aplikacija su HTTP (Hypertext Transfer Protocol, koristi se za veb), SSL (Secure Socket Layer) i FTP (File Transfer Protocol). Protokoli sloja aplikacija često se mogu međusobno kombinovati. Na primer, SSL se često koristi u kombinaciji s HTTP-om.

Transportni sloj Definiše karakteristike za prenošenje podataka iz sloja aplikacija i u njega. Ovaj sloj obuhvata proveravanje integriteta podataka, izvorne i odredišne priključke i specifikacije za razdvajanje podataka iz aplikacija na pakete (ako to nije već obavio sloj aplikacija). TCP (Transmission Control Protocol) i UDP (User Datagram Protocol) najuobičajeniji su protokoli transportnog sloja. Transportni sloj se ponekad zove i *sloj protokola*.

Mrežni ili internet sloj Definiše kako treba prenositi pakete od izvornog do odredišnog računara. Pravila za prenošenje konkretnog paketa na internetu poznata su kao IP (Internet Protocol). Pošto ćemo u ovoj mreži razmatrati samo mreže internet tipa, govorićemo zapravo samo o internet sloju. Međutim, pošto su mrežni slojevi osmišljeni tako da ne zavise od konkretnog hardvera, na istom umreženom računaru možete konfigurisati više međusobno nezavisnih mrežnih slojeva (kao što su IP, IPv6, IPX i AppleTalk).

Fizički sloj Definiše kako se sirovi podaci prenose kroz dati fizički medijum, kao što je Ethernet ili modem. To se ponekad zove i *sloj veza* ili *sloj računar/mreža*.

Važno je da razumete strukturu mrežnog sloga zato što vaši podaci moraju proći kroz te slojeve barem dvaput pre nego što dođu do odredišne aplikacije. Na primer, ako šaljete podatke iz Računara A Računaru B na slici 9-1, vaši bajtovi napuštaju sloj aplikacija na Računaru A i putuju nadole kroz transportni i mrežni sloj na istom računaru; zatim prelaze niže u fizički medijum, putuju kroz njega, a onda na sličan način nagore kroz razne niže slojeve do sloja aplikacija na Računaru B. Ako šaljete nešto odredišnom računaru na internetu kroz usmerivač, to će proći kroz neke (ali uglavnom ne sve) slojeve na usmerivaču i na svemu drugom između njih.

Ponekad se slojevi međusobno preklapaju na čudne načine zato što može biti neefikasno da se svi redom obrađuju. Na primer, uređaji koji su tokom razvoja mreža „poslovali" samo s fizičkim slojem, danas „zaviruju" u podatke transportnog i internet sloja da bi te podatke brže filtrirali i usmeravali. (Nemojte brinuti o tome dok savladavate osnove.)

Prvo ćemo razmotriti kako naša Linux mašina uspostavlja vezu s mrežom da bismo odgovorili na pitanje *gde* s početka ovog poglavlja. To je najniži nivo mrežnog sloga – fizički i mrežni slojevi. Kasnije, razmotrićemo najviša dva sloja koja pružaju odgovor na pitanje *šta*.

NAPOMENA *Možda ste čuli za još jedan skup slojeva, poznat kao Referentni OSI (Open Systems Interconnection) model. To je sedmoslojni model mreže koji se često koristi za učenje i projektovanje mreža, ali u ovoj knjizi nećemo razmatrati OSI model zato što ćete vi raditi direktno s četiri ovde opisana sloja. Da biste saznali* znatno *više o slojevima (i uopšte o mrežama), pogledajte knjigu* Računarske mreže, *Andrew S. Tanenbaum i David J. Wetherall, 5. izdanje, 2010 (prevod Mikro knjiga, 2013).*

9.3 Internet sloj

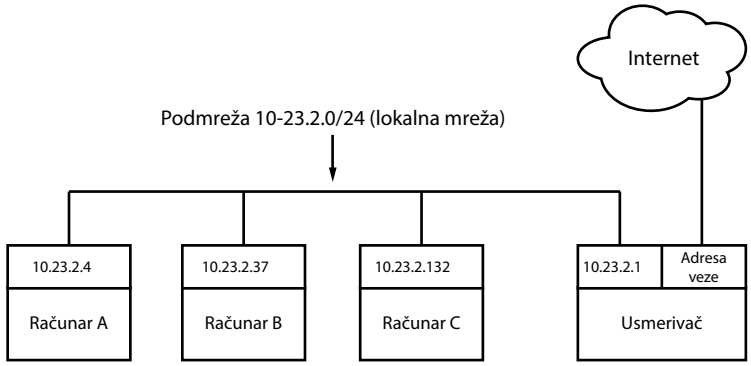
Umesto da krenemo od samog dna mrežnog sloga s fizičkim slojem, počećemo od mrežnog sloja zato što se on možda najlakše razume. Internet kakvog ga danas poznajemo zasniva se na IP-u (Internet Protocol), verzija 4 (IPv4), mada je i verzija 6 (IPv6) sve prihvaćenija. Jedan od najvažnijih aspekata internet sloja jeste to da je zamišljen kao softverska mreža koja ne postavlja nikakve zahteve koji bi se odnosili na dati hardver ili operativni sistem. Suština je u tome da možete slati i primati internet pakete na proizvoljnom hardveru i pod bilo kojim operativnim sistemom.

Internet topologija je decentralizovana; čine je manje mreže koje se zovu *podmreže* (engl. *subnets*). Suština je to da su sve podmreže na određeni način međusobno povezane. Na primer, na slici 9-1, lokalna mreža bi bila jedna od podmreža.

Jedan umreženi računar može biti povezan istovremeno s više podmreža. Kao što ste videli u odeljku 9.1, ta vrsta umreženog računara zove se usmerivač ako zna kako da prosleđuje podatke iz jedne podmreže u drugu (drugi izraz za usmerivač je *mrežni prolaz* /engl. *gateway*/). Slika 9-2 precizira sliku 9-1 tako što lokalnu mrežu definiše kao podmrežu i navodi internet adrese za svaki računar u mreži i usmerivač. Usmerivač na slici ima dve adrese: 10.23.2.1 u lokalnoj podmreži i vezu s internetom (ali je adresa veze sa internetom zasad nevažna pa je samo napisano „adresa veze“). Prvo ćemo razmotriti notaciju adresa a zatim i notaciju podmreža.

Svaki računar na internetu ima barem jednu numeričku *IP adresu* u formatu *a.b.c.d*, kao što je 10.23.2.37. Adresa u takvoj notaciji zove se *četvorodelna sekvenca s tačkama* (engl. *dotted-quad sequence*). Ako je računar povezan s više podmreža, ima barem jednu IP adresu po podmreži. Trebalo bi da IP adresa svakog umreženog računara bude jedinstvena na celom internetu, ali kao što ćete videti u nastavku poglavlja, privatne mreže i NAT mogu to malo da zakomplikuju.

NAPOMENA *Tehnički govoreći, IP adresa se sastoji od 4 bajta (tj. 32 bita),* abcd. *Bajtovi* a *i* d *su brojevi u opsegu od 1 do 254, a* b *i* c *su brojevi u opsegu od 0 do 255. Računar obrađuje IP adrese kao niz sirovih bajtova. Međutim, ljudima je znatno razumljivije da čitaju i pišu četvorodelnu adresu s tačkama, kao što je 10.23.2.37, umesto nečeg ružnog kao što je heksadecimalni oblik 0x0A170225.*



Slika 9-2: Mreža sa IP adresama

IP adrese su donekle nalik kućnim adresama. Da bi komunicirala s određenim računarom u mreži, vaša mašina mora da zna njegovu IP adresu.

Sada ćemo razmotriti adresu vaše mašine.

9.3.1 Prikazivanje IP adresa računara

Jedan računar može imati više IP adresa. Da biste videli adrese koje su aktivne na vašoj Linux mašini, zadajte komandu

```
$ ifconfig
```

Verovatno ćete dobiti mnogo izlaznih podataka, ali trebalo bi da budu slični sledećem:

```
eth0      Link encap:Ethernet  HWaddr 10:78:d2:eb:76:97
          inet addr:10.23.2.4  Bcast:10.23.2.255  Mask:255.255.255.0
          inet6 addr: fe80::1278:d2ff:feeb:7697/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85076006 errors:0 dropped:0 overruns:0 frame:0
          TX packets:68347795 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:86427623613 (86.4 GB)  TX bytes:23437688605 (23.4 GB)
          Interrupt:20 Memory:fe500000-fe520000
```

Rezultati komande `ifconfig` sadrže brojne podatke i iz internet sloja i iz fizičkog sloja. (Ponekad se čak ni ne prikazuje nikakva internet adresa!) U nastavku poglavlja razmotrićemo detaljnije te podatke, ali zasad, obratite pažnju na drugi red, koji pokazuje da je računar konfigurisan tako da ima jednu IPv4 adresu (`inet addr`) 10.23.2.4. U isto vreme, vrednost polja `Mask` je 255.255.255.0. To je *maska podmreže*, koja definiše podmrežu kojoj IP adresa pripada. Sada ćemo videti kako to radi.

NAPOMENA *Komandu `ifconfig`, kao i neke druge koje ćete videti u nastavku ovog poglavlja (kao što su `route` i `arp`), tehnički nadmašuje novija komanda `ip`. Komanda `ip` može uraditi više nego starije komande i preporučljivija je kada pišete*

skriptove. Međutim, mnogi ljudi i dalje koriste stare komande kada ručno rade s mrežom, a te komande se mogu koristiti i u drugim verzijama Unixa. Iz tih razloga, koristićemo starije komande.

9.3.2 Podmreže

Podmreža jeste grupa računara povezanih pomoću IP adresa određenog zajedničkog formata. Ti računari su najčešće u istoj fizičkoj mreži, kao što je ona na slici 9-2. Na primer, svi računari sa adresama od 10.23.2.1 do 10.23.2.254 mogu činiti jednu podmrežu, kao i računari čije su adrese u opsegu od 10.23.1.1 do 10.23.255.254.

Podmrežu definišete pomoću dva elementa: *prefiksa mreže* i maske podmreže (kao što je ona u rezultatima komande `ifconfig` u prethodnom odeljku). Recimo da želite da formirate podmrežu koja sadrži IP adrese od 10.23.2.1 do 10.23.2.254. Prefiks mreže je deo koji je *zajednički* svim adresama u podmreži; u ovom primeru, to je 10.23.2.0, a maska podmreže je 255.255.255.0. Pogledajmo zašto su to ispravne vrednosti.

Nije odmah jasno kako prefiks i maska deluju zajedno da bi vam dali sve moguće IP u podmreži. Razmatranje tih brojeva u binarnom obliku doprinosi lakšem razumevanju. Maska obeležava mesta svih bîta dela IP adrese koji je isti za celu podmrežu. Na primer, ovako izgledaju binarni oblici adresa 10.23.2.0 i 255.255.255.0:

```
10.23.2.0:        00001010 00010111 00000010 00000000
255.255.255.0:    11111111 11111111 11111111 00000000
```

Sada ćemo istaći podebljanim znacima mesta bîta u 10.23.2.0 gde se nalaze jedinice u 255.255.255.0:

```
10.23.2.0:        00001010 00010111 00000010 00000000
```

Pogledajte bîte koji *nisu* istaknuti. Da biste dobili ispravnu IP adresu u ovoj podmreži, možete dodeliti jedinice proizvoljnoj kombinaciji tih bîta, izuzev svima nule, ili svima jedinice.

Kada sklopimo sve u celinu, vidi se da računar čija je IP adresa 10.23.2.1 i maska podmreže 255.255.255.0 pripada istoj podmreži kao i svi drugi računari čije IP adrese počinju sa 10.23.2. Kompletnu podmrežu možete definisati kao 10.23.2.0/255.255.255.0.

9.3.3 Uobičajene maske podmreža i CIDR notacija

Ako imate sreće, bavićete se samo s lako razumljivim maskama podmreže kao što je 255.255.255.0 ili 255.255.0.0, ali možda vam sreća nije naklonjena, pa ćete naići na nešto poput 255.255.255.192, gde nije baš tako lako odrediti skup adresa koje pripadaju podmreži. Dalje, verovatno ćete naići i na drugačiji oblik predstavljanja podmreža, koji se zove CIDR (*Classless Inter-Domain Routing – besklasno međudomensko usmeravanje)* notacija, gde se podmreža kao što je 10.23.2.0/255.255.255.0 piše kao 10.23.2.0/24.

Da biste razumeli šta to znači, pogledajte binarni oblik maske (kao u primeru u prethodnom odeljku). Otkrićete da je gotovo svaka maska podmreže samo niz jedinica kojem sledi niz nula. Na primer, upravo ste videli da se binarni oblik adrese 255.255.255.0 sastoji od 24 jedinice kojima sledi 8 nula. CIDR notacija identifikuje masku podmreže pomoću broja *vodećih* jedinica u maski podmreže. Iz tog razloga, kombinacija kao što je 10.23.2.0/24 sadrži i prefiks podmreže i masku podmreže.

Tabela 9-1 prikazuje više primera maski podmreža i njihove CIDR oblike.

Tabela 9-1: Maske podmreža

Dugačak oblik	CIDR oblik
255.0.0.0	8
255.255.0.0	16
255.240.0.0	12
255.255.255.0	24
255.255.255.192	26

NAPOMENA *Ako ne poznajete dobro konverzije između decimalnog, binarnog i heksadecimalnog formata, možete upotrebiti neku kalkulatorsku alatku kao što je* `bc` *ili* `dc` *da biste konvertovali između raznih numeričkih osnova. Na primer, s programom* `bc`*, možete zadati komandu* `obase=2; 240` *da biste broj 240 prikazali u binarnom obliku (osnova 2).*

Prepoznavanje podmreža i računara koji im pripadaju prvi je gradivni blok za razumevanje rada interneta. Međutim, i dalje morate povezati podmreže.

9.4 Mrežne putanje i tabela rutiranja jezgra

Međusobno povezivanje internet podmreža uglavnom se svodi na postupak prepoznavanja računara koji su povezani u više od jedne podmreže. Ako se vratimo na sliku 9-2, razmotrite Računar A na IP adresi 10.23.2.4. Taj računar je povezan s lokalnom mrežom 10.23.2.0/24 i zato može direktno da dosegne ostale računare u toj mreži. Da bi dosegao računare u preostalom delu interneta, mora da komunicira kroz usmerivač na adresi 10.23.2.1.

Kako Linuxovo jezgro razlikuje ta dva odredišta? Tako što koristi odredišnu konfiguraciju koja se zove *tabela rutiranja* (engl. *routing table*) na osnovu koje prilagođava odredište podataka. Da biste videli sadržaj tabele rutiranja, zadajte komandu `route -n`. Evo šta biste mogli da vidite u slučaju nekog jednostavnog umreženog računara kao što je 10.23.2.4:

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.23.2.1       0.0.0.0         UG    0      0        0 eth0
10.23.2.0       0.0.0.0         255.255.255.0   U     1      0        0 eth0
```

Poslednja dva reda sadrže podatke za rutiranje (usmeravanje) podataka. Kolona `Destination` (odredište) sadrži prefiks mreže, a kolona `Genmask` sadrži masku koja odgovara toj mreži. Prikazani rezultati definišu dve mreže: 0.0.0.0/0 (koja predstavlja svaku adresu na internetu) i 10.23.2.0/24. Obe mreže u koloni `Flags` imaju znak `U`, što znači da je putanja upotrebljiva („up" – u aktivnom stanju).

Odredišta se razlikuju po kombinacijama vrednosti u njihovim kolonama `Gateway` i `Flags`. Mreža 0.0.0.0/0 ima znak `G` u koloni `Flags`, što znači da se komunkacija s tom mrežom mora odvijati kroz mrežni prolaz (engl. *gateway*) u koloni `Gateway` (10.23.2.1, u ovom slučaju). Međutim, mreža 10.23.2.0/24 nema znak `G` u koloni `Flags`, što znači da s tom mrežom postoji određena vrsta direktne veze. Vrednost 0.0.0.0 u koloni `Gateway` te mreže predstavlja bilo koju adresu. Zasad zanemarite vrednosti u ostalim kolonama.

Postoji jedna začkoljica: recimo da računar želi da pošalje nešto na adresu 10.23.2.132, koja ispunjava oba uslova u tabeli rutiranja, tj. odgovara i za mrežu 0.0.0.0/0 i za mrežu 10.23.2.0/24. Kako jezgro zna da treba da je odredište druga mreža? Tako što bira najduži odredišni prefiks koji odgovara adresi. Tu se CIDR notacija za mreže pokazuje kao posebno praktična: odgovara mreži 10.23.2.0/24, a dužina njenog prefiksa je 24 bita; mreža 0.0.0.0/0 takođe odgovara, ali je dužina njenog prefiksa nula bîta (tj. ona nema prefiks), pa se zato primenjuje pravilo 10.23.2.0/24.

NAPOMENA *Opcija `-n` nalaže komandi `route` da prikazuje IP adrese računara i mreža umesto njihovih imena. To je važna opcija koju treba zapamtiti zato što ćete je koristiti i uz druge komande za rad s mrežama, kao što je `netstat`.*

9.4.1 Podrazumevani mrežni prolaz

Stavka 0.0.0.0/0 u tabeli rutiranja ima specijalno značenje zato što predstavlja svaku adresu na internetu. To je *podrazumevana putanja* (engl. *default route*), a adresa zadata u koloni `Gateway` (u rezultatima komande `route -n`) podrazumevane putanje jeste *podrazumevani mrežni prolaz* (engl. *default gateway*). Kada nije ispunjen nijedan uslov za neku drugu putanju u tabeli rutiranja, uvek je ispunjen uslov za podrazumevanu putanju i podrazumevani mrežni prolaz je odredište na koje se šalju poruke kada ne preostane nijedna druga mogućnost. Umreženi računar možete konfigurisati i bez podrazumevanog mrežnog prolaza, ali onda on neće moći da dosegne druge računare koji se nalaze izvan eksplicitno zadatih odredišta u tabeli rutiranja.

NAPOMENA *U većini mreža čija je mrežna maska 255.255.255.0, usmerivač se obično nalazi na adresi 1 podmreže (na primer, 10.23.2.1 u mreži 10.23.2.0/24). Budući da je to obična konvencija, može biti izuzetaka.*

9.5 Osnovne ICMP i DNS alatke

Sada je vreme da razmotrimo neke osnovne praktične alatke za rad s umreženim računarima. Te alatke koriste dva posebno zanimljiva protokola: ICMP (Internet Control Message Protocol – Internet protokol za upravljačke

poruke), koji može biti od koristi kada rešavate probleme pri uspostavljanju veza i usmeravanju poruka, i DNS (Domain Name Service – servis za imena domena) sistem, koji preslikava imena u IP adrese kako ne biste morali da pamtite gomile brojeva.

9.5.1 Alatka ping

Alatka `ping` (videti na *http://ftp.arl.mil/~mike/ping.html*) jedna je od najosnovnijih alatki za otkrivanje grešaka u mreži. Ona šalje ICMP eho pakete pomoću kojih zahteva od odredišnog računara da primljeni paket vrati pošiljaocu. Ako odredišni računar primi taj paket i konfigurisan je tako da odgovori na njega, on vraća pošiljaocu ICMP odzivni eho paket.

Na primer, zadali ste **`ping 10.23.2.1`** i dobili sledeće rezultate:

```
$ ping 10.23.2.1
PING 10.23.2.1 (10.23.2.1) 56(84) bytes of data.
64 bytes from 10.23.2.1: icmp_req=1 ttl=64 time=1.76 ms
64 bytes from 10.23.2.1: icmp_req=2 ttl=64 time=2.35 ms
64 bytes from 10.23.2.1: icmp_req=4 ttl=64 time=1.69 ms
64 bytes from 10.23.2.1: icmp_req=5 ttl=64 time=1.61 ms
```

Prvi red pokazuje da šaljete 56-bajtne pakete (ukupne veličine 84 bajta, ako se uračunaju i zaglavlja) na odredišnu adresu 10.23.2.1 (standardno, po jedan paket u sekundi), a ostali redovi predstavljaju odgovore od 10.23.2.1. U ovim rezultatima, najvažniji delovi su redni broj paketa (`icmp_req`) i ukupno vreme prenošenja paketa (`time`). Veličina vraćenih paketa jednaka je veličini poslatog paketa plus 8. (Sadržaj paketa je nebitan za vas.)

Nedostajući redni brojevi paketa, kao što je onaj između 2 i 4, najčešće znače da postoji neki problem sa uspostavljanjem veze. Paketi mogu stizati izvan redosleda, a ako se to događa, postoji neki problem zato što `ping` šalje samo po jedan paket svake sekunde. Ukoliko odgovoru treba više od sekunde (1000 ms) da bi stigao, veza je izuzetno spora.

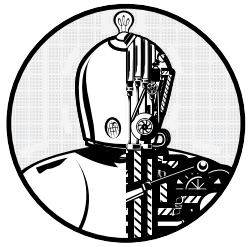
Ukupno vreme prenošenja paketa jeste vreme koje je proteklo od trenutka kada je paket sa zahtevom krenuo ka odredištu, do trenutka prijema odzivnog paketa. Ako je odredište nedosegljivo, poslednji usmerivač na putanju kroz koji je paket uspešno prošao vraća alatki `ping` ICMP paket „host unreachable“ (odredište nije dosegljivo).

U lokalnoj mreži sa žičanim vezama, trebalo bi da se ne gubi nijedan jedini paket i da ukupna vremena prenošenja budu vrlo niske vrednosti. (U prethodnom primeru prikazani su rezultati za bežičnu mrežu.) Osim toga, za očekivati je i da nema gubljenja paketa između vaše mreže i vašeg davaoca internet usluga i da su ukupna vremena prenošenja uvek približno jednake vrednosti.

NAPOMENA *Iz bezbednosnih razloga, na ICMP eho pakete ne odazivaju se svi računari na internetu, pa se zato može dogoditi da uspešno uspostavljate vezu s određenom veb lokacijom, ali ne dobijate odziv na komandu* `ping` *za nju.*

15

RAZVOJNE ALATKE



Linux i Unix su veoma popularni među programerima, ne samo zbog neverovatno bogatog izbora alatki i okruženja na raspolaganju, nego i zato što je sistem izuzetno dobro dokumentovan i transparentan. Na Linux mašini, ne morate biti programer da biste koristili razvojne alatke, ali kada radite na sistemu, trebalo bi da znate nešto o alatkama za programiranje zato što one igraju značajniju ulogu u upravljanju Unix sistemima nego u drugim operativnim sistemima. Potrebno je da barem umete da prepoznate razvojne alatke i steknete određeno znanje o njihovoj upotrebi.

Ovo poglavlje pakuje veliku količinu informacija na malom prostoru, ali nije neophodno da savladate baš sve navedeno. Pojedine delove gradiva lako možete preskočiti i vratiti se na njih kasnije. Odeljak o deljenim bibliotekama verovatno sadrži ono najvažnije što treba da znate. Ali da biste znali odakle potiču deljene biblioteke, prvo ćete se upoznati sa osnovama izrade programa.

15.1 Kompajler jezika C

Ako znate kako se koristi kompajler (prevodilac) za programski jezik C, imaćete prilično dobar uvid u poreklo programa koje vidite na svom Linux sistemu. Izvorni kôd većine alatki za Linux i mnogih aplikacija na Linux sistemima, napisan je na jeziku C ili C++. U ovom poglavlju koristićemo prevashodno primere na jeziku C, ali informacije do kojih ćete doći možete preneti i na C++.

C programi slede tradicionalni postupak razvoja: pišete programe, prevedete ih, a zatim izvršite, odnosno kada napišete C program i želite da ga izvršite, izvorni kôd koji ste napisali morate *prevesti* (engl. *compile*) u binarni oblik niskog nivoa koji rač unar razume. S druge strane, kada koristite jezik za skriptove – koji razmatramo u nastavku poglavlja – ne treba da prevodite ništa.

NAPOMENA *Većina distribucija standardno ne sadrži alatke koje su potrebne za prevođenje C koda zato što te alatke zauzimaju prilično mnogo mesta. Ako ne možete da nađete neke od alatki koje ovo poglavlje opisuje, možete instalirati paket build-essential za Debian/Ubuntu, ili sličan paket za Fedoru/CentOS pomoću komande* `yum groupinstall "Development Tools"`*. Ako to ne uspe, potražite na internetu odgovarajući paket pod odredniciom „C compiler“.*

Izvršiva komponenta kompajlera za jezik C u većini Unix sistema jeste GNU C kompajler, `gcc`, mada noviji prevodilac `clang` iz projekta LLVM stiče sve veću popularnost. Imena datoteka izvornog C koda završavaju se sa *.c*. Razmotrite samostalnu datoteku izvornog C koda čije je ime *hello.c*, a koju možete naći u knjizi *The C Programming Language*, 2. izdanje, čiji su autori Brian W. Kernighan i Dennis M. Ritchie (Prentice Hall, 1988):

```
#include <stdio.h>

main() {
    printf("Hello, World.\n");
}
```

Upišite taj izvorni kôd u datoteku koju ćete nazvati *hello.c* a zatim izvršite sledeću komandu:

```
$ cc hello.c
```

Rezultat je izvršiva datoteka *a.out*, koju možete izvršavati na isti način kao i svaku drugu izvršivu datoteku u sistemu. Međutim, verovatno bi izvršivoj datoteci trebalo nadenuti neko drugo ime (kao što je *hello*). Da biste to postigli, zadajte opciju kompajlera `-o`:

```
$ cc -o hello hello.c
```

Za veoma kratke programe, to je otprilike sve što treba uraditi po pitanju prevođenja koda. Može biti potrebno da dodate još neki direktorijum ili biblioteku za umetanje (videti odeljke 15.1.2 i 15.1.3), ali sada ćemo razmotriti nešto opsežnije programe pre nego što počnemo da se bavimo tim temama.

15.1.1 Kombinovanje više izvornih datoteka

Većina C programa je prevelika da bi mogli u celini stati u jednu datoteku izvornog koda. Datoteke mamutske veličine postaju suviše nepraktične za programera, a kompajleri ponekad čak imaju teškoća da analiziranju preopsežne datoteke. Iz tih razloga, projektanti grupišu više komponeneta izvornog koda i svaki njegov deo smeštaju u vlastitu datoteku.

Kada prevodite većinu *.c* datoteka, ne pravite odmah izvršivu datoteku. Umesto toga, upotrebite opciju `-c` kompajlera da biste napravili *objektne datoteke*. Da biste videli kako se to radi, recimo da imate dve datoteke, *main.c* i *aux.c*. Sledeće dve komande kompajlera obavljaju najveći deo sklapanja (engl. *building*) programa:

```
$ cc -c main.c
$ cc -c aux.c
```

Prethodne dve komande prevode dve izvorne datoteke u dve objektne datoteke *main.o* i *aux.o*.

Objektna datoteka je binarna datoteka koju procesor gotovo da razume, osim što i dalje postoji nekoliko nedorečenosti. Prvo, operativni sistem ne zna kako da izvrši objektnu datoteku, a drugo, verovatno ćete morati da kombinujete više objektnih datoteka i određene sistemske biblioteke kako biste sklopili kompletan (i upotrebljiv) program.

Da biste od jedne ili više objektnih datoteka sklopili jednu izvršivu datoteku, morate izvršiti *linker* (povezivač), što je u Unixu komanda `ld`. Programeri retko koriste `ld` s komandne linije, zato što sam C kompajler zna kako da pozove program za povezivanje (linker). Da biste od dve navedene objektne datoteke formirali izvršiv program koji se zove *myprog*, izvršite sledeću komandu kako biste ih povezali:

```
$ cc -o myprog main.o aux.o
```

Mada više izvornih datoteka možete prevesti i ručno, kao što prikazuje prethodni primer, može biti teško da ih sve pratite tokom postupka prevođenja kada se broj izvornih datoteka poveća. Sistem `make` opisan u odeljku 15.2 tradicionalni je Unixov standard za upravljanje postupkom prevođenja koda. Taj sistem je naročito važan za upravljanje datotekama koje su opisane u naredba dva odeljka.

15.1.2 Datoteke zaglavlja i direktorijumi za umetanje

C *datoteke zaglavlja* (engl. *header files*) dodatne su datoteke sa izvornim kodom koje obično sadrže deklaracije bibliotečkih tipova funkcija. Na primer, *stdio.h* je datoteka zaglavlja (videti jednostavan primer u odeljku 15.1).

Nažalost, veliki broj problema pri prevođenju koda potiče od datoteka zaglavlja. Većina grešaka nastaje kada kompajler ne uspe da nađe datoteke zaglavlja i biblioteke. Ponekad čak i programer zaboravi da umetne odgovarajuću datoteku zaglavlja, usled čega izvorni kôd ne može da se prevede.

Rešavanje problema s datotekama za umetanje

Određivanje ispravnih datoteka za umetanje nije uvek jednostavno. Ponekad je navedeno više datoteka za umetanje sa istim imenima ali iz različitih direktorijuma, a nije jasno koja je ispravna. Kada kompajler ne uspeva da pronađe potrebnu datoteku za umetanje, poruka o grešci izgleda ovako:

```
badinclude.c:1:22: fatal error: notfound.h: No such file or directory
```

Ova poruka vas obaveštava da kompajler ne može da nađe datoteku zaglavlja *notfound.h* koju datoteka *badinclude.c* referencira. Ta konkretna greška je direktan rezultat sledeće direktive u prvom redu datoteke *badinclude.c*:

```
#include <notfound.h>
```

Podrazumevani direktorijum za datoteke za umetanje na Unixu jeste */usr/include*; kompajler ih uvek tamo traži, osim ako mu izričito zadate da to ne čini. Međutim, kompajleru možete zadati i da potraži u drugim direktorijumima za umetanje (većina putanja koje sadrže datoteke zaglavlja imaju reč *include* negde u imenu).

NAPOMENA *U poglavlju 16 saznaćete više o tome kako se pronalaze nedostajuće datoteke za umetanje.*

Na primer, našli ste datoteku *notfound.h* u direktorijumu */usr/junk/include*. Kompajleru možete naložiti da traži u tom direktorijumu pomoću opcije `-I`:

```
$ cc -c -I/usr/junk/include badinclude.c
```

Sada kompajler više ne bi trebalo da se „sapliće“ o red u datoteci *badinclude.c* koji referencira datoteku zaglavlja.

Trebalo bi da obratite pažnju i na datoteke za umetanje navedene između navodnika (`" "`) umesto između šiljatih zagrada (`< >`), kao u sledećem primeru:

```
#include "myheader.h"
```

Navodnici znače da se datoteka za umetanje ne nalazi u sistemskom direktorijumu za umetanje nego da kompajler treba da je potraži u svom direktorijumu za umetanje. To često znači da se tražena datoteka za umetanje nalazi u istom direktorijumu kao datoteka izvornog koda. Ako naiđete na problem zbog navodnika, verovatno pokušavate da prevedete nekompletan izvorni kôd.

Šta je C pretprocesor (cpp)?

Posao traženja svih datoteka za umetanje zapravo ne obavlja C kompajler. Time se bavi *C pretprocesor*, što je program koji kompajler poziva da obradi vaš

izvorni kôd pre nego što kompajler analizira vaš program. Pretprocesor konvertuje izvorni kôd u oblik koji kompajler razume; to je alatka koja izvorni kôd čini lakšim za čitanje (i umeće u njega prečice).

Pretprocesorove komande u izvornom kodu zovu se *direktive* i počinju znakom `#`. Postoje tri osnovne vrste direktiva:

Datoteke za umetanje Direktiva `#include` nalaže pretprocesoru da umetne celu navedenu datoteku. Imajte u vidu da opcija `-I` kompajlera čini da pretprocesor traži datoteke za umetanje u zadatom direktorijumu, kao što ste videli u prethodnom odeljku.

Definicije makroa Red nalik na `#define BLABLA nešto` nalaže pretprocesoru da gde god u izvornom kodu naiđe na `BLABLA`, to treba da zameni sa `nešto`. Konvencije nalažu da se makroi pišu velikim slovima, ali nemojte se iznenaditi kada vidite da programeri ponekad koriste makroe čija imena podsećaju na funkcije ili promenljive. (S vremena na vreme, to je uzrok teških glavobolja. Mnogi programeri čak uživaju u zloupotrebi pretprocesora.)

NAPOMENA *Umesto da makroe definišete unutar izvornog koda, možete ih definisati i tako što ih prosledite kompajleru u obliku parametara: `-DBLABLA=nešto` ima isti efekat kao prehodna direktiva.*

Uslovno prevođenje Određene delove koda možete obeležiti sa `#ifdef`, `#if` i `#endif`. Direktiva `#ifdef` *`MAKRO`* ispituje da li je definisan pretprocesorov *`MAKRO`*, a direktiva `#if` *`uslov`* ispituje da li je *`uslov`* ispunjen. U obe direktive, ako uslov koji sledi „naredbi if“ nije ispunjen, pretprocesor ne prosleđuje kompajleru ceo tekst u opsegu od `#if` do sledećeg `#endif`. Ako nameravate da proučavate C kôd, počnite da se navikavate na to.

Sledi primer uslovne direktive. Kada pretprocesor vidi sledeći kôd, ispituje da li je definisan makro `DEBUG`, pa ako jeste, kompajleru prosleđuje red koji sadrži funkciju `fprintf()`. Ako nije, pretprocesor preskače taj red i nastavlja da obrađuje datoteku iza `#endif`:

```
#ifdef DEBUG
  fprintf(stderr, "Ovo je dijagnostička poruka.\n");
#endif
```

NAPOMENA *C pretprocesor ne zna ništa o sintaksi jezika C, promenljivama, funkcijama i drugim elementima jezika. On razume samo svoje makroe i direktive.*

Na Unixu, ime C pretprocesora je `cpp`, ali ga možete pokrenuti i pomoću komande `gcc -E`. Međutim, retko je potrebno da pretprocesor pokrećete kao samostalan program.

15.1.3 Povezivanje s bibliotekama

Kompajler za jezik C ne poznaje vaš sistem dovoljno da bi mogao sam da sklopi neki koristan program. Potrebne su vam *biblioteke* da biste sastavljali kompletne programe. C biblioteka je kolekcija zajedničkih unapred prevedenih

funkcija koju možete ugraditi u svoj program. Na primer, mnogi izvršivi programi koriste biblioteku math zato što ona sadrži trigonometrijske i druge matematičke funkcije.

Biblioteke dolaze do izražaja prvenstveno u vreme povezivanja koda, kada program za povezivanje (linker) formira od objektnih datoteka jednu izvršivu datoteku. Na primer, ako imate program koji koristi biblioteku gobject ali vi zaboravite da obavestite kompajler da poveže tu biblioteku, videćete greške pri povezivanju kao što je sledeća:

```
badobject.o(.text+0x28): undefined reference to 'g_object_new'
```

Najvažniji delovi tih poruka o greškama istaknuti su podebljanim slovima. Kada je program za povezivanje analizirao objektnu datoteku *badobject.o*, nije uspeo da nađe funkciju čije je ime istaknuto, a posledica toga je da nije mogao da sklopi izvršivu datoteku. U ovom konkretnom slučaju, mogli biste posumnjati da ste zaboravili biblioteku gobject zato što je ime nedostajuće funkcije `g_object_new()`.

NAPOMENA *Nedefinisane reference ne znače uvek da vam nedostaje određena biblioteka. Moguće je i da neka od objektnih datoteka programa nije navedena u komandi za povezivanje programa. Mada, lako je razlikovati funkcije iz biblioteka i funkcije iz objektnih datoteka.*

Da biste rešili taj problem, morate prvo pronaći biblioteku gobject i zatim zadati opciju kompajlera `-l` da biste svoj program povezali s tom bibliotekom. Isto kao i datoteke za umetanje, biblioteke su rasute po celom sistemu (*/usr/lib* je sistemska podrazumevana lokacija za biblioteke), mada se mnoge biblioteke nalaze u poddirektorijumu *lib*. Za prethodni primer, osnovna datoteka biblioteke gobject jeste *libgobject.a*, a ime biblioteke je `gobject`. Znajući to, svoj program biste povezali s tom bibliotekom na sledeći način:

```
$ cc -o badobject badobject.o -lgobject
```

Program za povezivanje (linker) morate obavestiti o bibliotekama na nestandardnim lokacijama; parametar za tu namenu je `-L`. Recimo da je programu `badobject` potrebna biblioteka *libcrud.a* koja se nalazi u */usr/junk/lib*. Da biste preveli kôd i napravili izvršivu datoteku, zadajte komandu sličnu sledećoj:

```
$ cc -o badobject badobject.o -lgobject -L/usr/junk/lib -lcrud
```

NAPOMENA *Ako želite da pronađete biblioteku u kojoj se nalazi data funkcija, upotrebite komandu* `nm`*. Očekujte veliku količinu podataka. Na primer, pokušajte sledeće:* ***`nm libgobject.a`****. (Možda će vam trebati komanda* `locate` *da biste pronašli biblioteku* libgobject.a*; mnoge distribucije danas smeštaju biblioteke u poddirektorijume* /usr/lib *koji su specifični za određene arhitekture.)*