
Pere László
GNU/Linux rendszerek
üzemeltetése I.

Bekapcsolástól a grafikus felületig

PÉCS, 2005



Pere László: GNU/Linux rendszerek üzemeltetése I.
Bekapcsolástól a grafikus felületig
© 2005 Pere László

Felelős kiadó a Kiskapu Kft. ügyvezető igazgatója
© 2005 Kiskapu Kft.

1081 Budapest, Népszínház u. 31., I. em., 7.
<http://kiado.kiskapu.hu/>
e-mail: kiado@kiskapu.hu

Lektorálta: Rézműves László
Borítóterv: Bognár Tamás
Műszaki szerkesztő: Csutak Hoffmann Levente

ISBN szám: 963 9301 92 2

Készült a debreceni Kinizsi Nyomdában
Felelős vezető: Bördős János

Tartalomjegyzék

1. Bevezetés	13
1.1. A jelölések és elnevezések	14
1.2. A könyvben található példák	15
2. Lemezrészek	17
2.1. A lemezrész	17
2.1.1. PC lemezrészek	18
2.2. Blokkeszköz-meghajtók	19
Blokkeszköz-meghajtók használata	22
2.3. Lemezrészek létrehozása	23
2.4. A GNU/Linux könyvtárszerkezete	30
2.4.1. A gyökérkönyvtár lemezrész	30
2.4.2. Egyéb lemezrészek	33
2.5. A lemezrészek terheltsége	35
3. Állományrendszerek	39
3.1. Az extended 2 állományrendszer	40
3.1.1. Az extended 2 állományrendszer felépítése	40
3.1.2. Közvetett és közvetlen hivatkozások	43
3.1.3. A foglalt lemezterület	47
3.1.4. A megnyitott állományok	48
3.1.5. A tulajdonos nyilvántartása	49
3.1.6. Az extended 2 állományrendszer kezelése	50
3.2. Az extended 3 állományrendszer	54
3.3. Az XFS állományrendszer	56
3.4. A csereterület	58
3.5. CD-ROM állományrendszer	61
3.5.1. Az iso9660 állományrendszer létrehozása	63
3.5.2. Az iso9660 állományrendszer kipróblálása	65
3.5.3. Az adatok felírása CD-ROM lemezre	66
3.5.4. Tömörített CD-ROM állományrendszer	67
3.6. A proc állományrendszer	69

3.7. A RAM állományrendszer	69
3.8. FAT típusú állományrendszerek	70
3.8.1. Az msdos állományrendszer	70
3.8.2. A vfat állományrendszer	71
3.9. Az állományrendszerek összefoglalása	72
4. Az állományrendszerek beillesztése	73
4.1. Beillesztés és lecsatolás	73
4.2. Az állományrendszer-táblázat	76
4.3. A hozzáférési lista használata	82
4.3.1. Az ACL-támogatás bekapcsolása	82
4.3.2. Az ACL bejegyzések használata	83
Új könyvtárbejegyzések ACL bejegyzései	90
5. A rendszermag betöltése	93
5.1. A BIOS	93
5.1.1. A BIOS merevlemezcímkészítési módszerei	94
5.1.2. Rendszerindítás merevlemezről	95
5.2. A LILO	95
5.2.1. A rendszertöltő használata	96
5.2.2. A rendszertöltő működése	97
A LILO állományai	101
5.2.3. A LILO beállítása	102
5.2.4. Indítólemez készítése	106
5.3. A GRUB	108
5.3.1. Lemezrészletek és állományok megadása	109
5.3.2. A rendszertöltő használata	110
5.3.3. A GRUB beállítása	112
5.3.4. Indítólemez készítése	116
5.4. A rendszermag paraméterei	117
6. Szolgáltatások indítása és leállítása	125
6.1. A futási szintek	125
6.2. A futási szintek kezelése	126
6.2.1. Az indítótáblázat	126
6.2.2. Az init használata	130
6.3. Alrendszerök indítása és leállítása	131
6.3.1. Az alrendszerök indítóprogramjai	134
7. Felhasználók nyilvántartása	139
7.1. Szöveges adatbázisok	140
7.1.1. Az adatbázis-állományok	140
7.1.2. Az árnyékjelszórendszer	142
7.2. A felhasználók kezelése	143

7.2.1.	Felhasználók létrehozása	144
7.2.2.	A jelszó megváltoztatása	145
7.2.3.	Felhasználók törlése	147
7.2.4.	Felhasználó tulajdonságainak módosítása	148
A héj megváltoztatása	148	
Személyes adatok megváltoztatása	149	
Egyéb adatok megváltoztatása	149	
7.2.5.	Felhasználók lekérdezése	150
7.3.	Csoportok kezelése	151
7.3.1.	Csoportok létrehozása	151
7.3.2.	Csoportok törlése	151
7.4.	Az árnyékjelszórendszer kezelése	152
8.	Be- és kijelentkezés	153
8.1.	Beléptetés	153
8.1.1.	A hagyományos bejelentkezés folyamata	153
Üzenetek bejelentkezéskor	153	
A beléptetés	154	
8.1.2.	A héj indítása	157
A rendszerszintű felhasználói profil	160	
8.2.	A biztonsági házirend	165
8.2.1.	A Linux-PAM felépítése	166
8.2.2.	A PAM beállítóállományok	167
8.2.3.	Rendszergazdai jogok biztosítása	169
8.2.4.	A modulok	171
Tiltás, engedélyezés és figyelmeztetés	171	
Hagyományos eszközök	172	
Az erőforrások korlátozása	175	
8.2.5.	A saját könyvtár automatikus létrehozása	178
8.3.	A lemezkorlát	178
8.3.1.	A lemezkorlát kezdeti beállítása	179
8.3.2.	Lemezkorlát rendelése felhasználóhoz	183
8.3.3.	Jelentések készítése	186
8.4.	A karakteres munkafelület beállításai	188
8.4.1.	Karakterkódolási szabványok	188
8.4.2.	A billentyűzet beállítása	190
A Unicode kódolás használata karakteres felületen	192	
Saját billentyűkiosztás létrehozása	193	
8.4.3.	Az egér használata karakteres felületen	197

9. Programcsomagok	199
9.1. Programok fordítása	199
9.1.1. A GNU C fordító	200
Egyeszerű C program fordítása	201
Összetett C programok	203
9.1.2. Projektek kezelése	205
A projektállomány alapszerkezete	206
A projektállomány makrói	209
Szabályok állománynév-végződés alapján	210
Hamis állományok használata	212
Összetett parancsok használata	213
9.1.3. A forrásprogram foltozása	214
9.1.4. Hordozható programok	217
Hordozható program készítése	217
Hordozható program fordítása	220
9.2. Programkönyvtárak	223
9.2.1. Programformátumok	223
9.2.2. Statikus programkönyvtárak	224
9.2.3. Megosztott programkönyvtárak	225
A dinamikus szerkesztő	225
A programkönyvtárak neve	227
9.2.4. A dinamikus könyvtárak nyilvántartása	228
9.2.5. Dinamikusan betöltött programkönyvtárak	228
9.3. A Red Hat csomagkezelő rendszere	229
9.3.1. A csomagok	229
9.3.2. A csomagkezelő szolgáltatásai	230
9.3.3. Csomagok lekérdezése	231
9.3.4. Csomagok telepítése	234
9.3.5. Csomagok eltávolítása	235
9.3.6. Csomagok ellenőrzése	236
9.3.7. Csomagok frissítése	238
9.3.8. Kicsomagolás	238
9.3.9. Csomagok készítése	238
A csomag elkészítése	239
A csomagleíró állomány alapelemei	240
Csomakészítés szimulált telepítéssel	248
Telepítéskor és eltávolításkor futtatóandó programok	252
A csomagok közti függőségek	253
9.4. A Debian csomagkezelő rendszere	255
9.4.1. A telepítőlemezek	257
9.4.2. A programcsomagok lekérdezése	259
9.4.3. Csomagok telepítése	262
9.4.4. Csomagok eltávolítása	262
9.4.5. A programcsomagok kezdeti beállítása	265

9.5. Programcsomagok automatikus letöltése	266
9.5.1. A programcsomagok forrása	266
A csomagforrások kezelése	270
9.5.2. Programcsomagok kezelése	271
9.5.3. Programcsomagok telepítése	273
9.5.4. Programcsomagok frissítése	274
9.5.5. Programcsomagok eltávolítása	276
9.5.6. A terjesztés frissítése	276
9.6. Programcsomagok lekérdezése	277
9.6.1. Adatok kérése az összes csomagról	278
9.6.2. Programcsomagok keresése	279
9.6.3. Csomagok adatainak lekérdezése	281
9.6.4. Telepítés grafikus felületen	282
10.A grafikus munkafelület	289
10.1. A grafikus felület indítása és leállítása	290
10.2. A betűk	290
10.2.1. A betűk nevei	290
10.2.2. A betűtípus-kiszolgáló	292
10.3. Az X kiszolgáló beállítása	293
10.3.1. Az állományokat leíró szakasz	297
10.3.2. A bemeneti eszközöket leíró szakasz	298
10.3.3. A videókártyát leíró szakasz	301
10.3.4. A monitort leíró szakasz	306
10.3.5. A monitor és a videókártya egymáshoz rendelése	309
10.3.6. Az elrendezés	311
10.3.7. Az X kiszolgáló különleges szolgáltatásai	315
10.4. Az X képernyőkezelő	322
10.4.1. A képernyőkezelő indítása	323
10.4.2. A Gnome képernyőkezelő beállítása	324
10.4.3. Az X képernyőkezelőjének beállítása	328
Az X erőforrásai	329
Az X képernyőkezelő erőforrásai	332
A bejelentkező ablak megjelenése	335
A munkafelület biztosítása	337

Táblázatok jegyzéke

2.1. Az mknod program	20
2.2. IDE csatolók blokkeszköz-meghajtói	20
2.3. SCSI merevlemezek blokkeszköz-meghajtói	21
2.4. SCSI CD-ROM meghajtók blokkeszköz-meghajtói	21
2.5. SCSI és IDE lemerzések blokkeszköz-meghajtói	22
2.6. Az fdisk program	23
2.7. A df program	36
3.1. A sync program	40
3.2. A stat program	41
3.3. A fins program	44
3.4. Az mke2fs program	51
3.5. A tune2fs program	53
3.6. Az e2fsck program	55
3.7. Az mkfs.xfs program	59
3.8. Az swapon/swapoff program	60
3.9. Az mkswap program	61
3.10. Az mkisofs program	64
3.11. A losetup program	65
3.12. A cdrecord program	67
3.13. Az mkzftree program	68
3.14. Állományrendszerek elnevezése	72
4.1. A mount program	75
4.2. Az umount program	75
4.3. A getfacl program	83
4.4. A setfacl program	87
5.1. A lilo program	98
5.2. Háttértárak és lemerzések GRUB nevei	110
5.3. Az mkinitrd program	118
5.4. A Vesa üzemmódok kódjai	120

6.1. Az runlevel program	126
6.2. Az init program	130
7.1. Az useradd program	144
7.2. Az passwd program	146
7.3. A userdel program	147
7.4. A chfn program	149
7.5. A usermod program	150
8.1. A login program	157
8.2. A BASH parancskérő jelének rövidítései	161
8.3. Az ulimit parancs	162
8.4. Az umask parancs	164
8.5. Az quotacheck program	181
8.6. Az quotaon program	183
8.7. Az setquota program	186
8.8. A quota program	187
8.9. A repquota program	188
8.10. A loadkeys program	190
8.11. A dumpkeys program	192
8.12. Módosítóbillentyűk és értékeik	196
9.1. Az gcc program	205
9.2. A make beépített makrói	210
9.3. Az RPM csomagok alapcsoportjai	242
9.4. Az RPM csomagok Applications csoportjai	243
9.5. A dselect billentyűkombinációi	285

Ábrák jegyzéke

2.1.	A lemezrész-táblázat	18
2.2.	PC lemezrészek	19
2.3.	PC lemezrészek nevei	22
2.4.	A /home/ könyvtár beillesztése	33
3.1.	Az adatblokkokat jelölő mutatók	42
3.2.	Könyvtár	43
3.3.	Közvetlen hivatkozás	45
3.4.	Közvetett hivatkozás	46
5.1.	A LILO rendszertöltő használata	96
5.2.	A GRUB rendszertöltő használata	111
5.3.	A rendszerindítás megkerülése	121
5.4.	Belépés rendszergazdaként, jelszó nélkül	122
5.5.	A gyökérkönyvtár nem érhető el	123
6.1.	Az automatikusan induló szolgáltatások beállítása	136
7.1.	A jelszókódolás	142
8.1.	Jelszókérés grafikus felületen	169
9.1.	A különbségi állomány a vim szövegszerkesztőben	216
9.2.	A debconf karakteres felülete	266
9.3.	A debconf Gnome felülete	267
9.4.	A dselect telepítőprogram főmenüje	283
9.5.	A dselect telepítőprogram bejelentkező képernyője	284
9.6.	A dselect telepítőprogram listaképernyője	284
9.7.	A synaptic grafikus telepítőprogram	286
9.8.	A synaptic telepítés közben	287
9.9.	A synaptic újabb változata	287
10.1.	Betűk kiválasztása az xfontsel program segítségével	292

10.2.A GLX kipróbálása	320
10.3.Grafikus bejelentkezés a gdm segítségével	323
10.4.A gdm általános beállításai	325
10.5.A gdm általános üdvözlőjének beállítása	326
10.6.A gdm grafikus üdvözlőjének beállítása	327
10.7.A gdm biztonsági beállításai	328
10.8.A gdm XDMCP beállításai	329
10.9.Az X erőforrás-szerkesztő	330
10.10A számológép erőforrások nélkül és erőforrásokkal	331
10.11Az X képernyőkezelője	336

1. fejezet

Bevezetés

Ma már közhelynek számít azt mondani, hogy a GNU/Linux egyre elterjedtebb, egyre többen használják, egyre nagyobb teret hódít. E könyv azonban éppen azoknak a szakembereknek és számítógép-tulajdonosoknak íródott, akik a GNU/Linux rendszerek üzemeltetését új ismeretként szeretnék elsajátítani, akik kevés tapasztalattal, de nagy lelkesedéssel vágnak neki a számukra új világ megismerésének. A kötetben megpróbáltuk közérthető és világos módon bemutatni azokat az alapismereteket, amelyek a rendszergazdák munkája és az otthoni számítógép-üzemeltetés során elengedhetetlenül fontosak.

A közérthető forma és az egyszerűség mellett igen fontos szempont volt az, hogy mindenkorral a működést, a felépítést, a rendszer megértését helyezzük előtérbe a konkrét műveletsor mechanikus leírásával szemben. A GNU/Linux és maga a UNIX olyan rugalmas, a rendelkezésre álló eszközök tárháza olyan hatalmas, hogy csak így volt esélyünk arra, hogy a gyakorlatban is használható tudást adjunk az olvasó kezébe. Egy-egy grafikus felületen használható beállítóprogramot bemutatva nagyon szűk körben használható és igen hamar elővülő ismeretanyagot gyűjtöttünk volna össze, amelyben az olvasó nem talál választ az igazán fontos kérdésekre.

A könyvben található konkrét példák és megoldások Debian és Red Hat terjesztések felhasználásával készültek. A Debian és a Red Hat gyűjtemény is rendelkezik többféle változattal és több leszármazott gyűjteménnyel, ezért a példák sokszor változtatás nélkül használhatók. A Fedora Core 3 terjesztés például a Red Hat utódjának tekinthető, az UHU-Linux pedig Debian vonásokat mutat, ezért ezek üzemeltetésében is segítséget nyújt a könyv.

Az olyan GNU/Linux terjesztések, amelyek mind a Debian, mind pedig a Red Hat gyűjteménytől függetlenül fejlődnek, többé-kevésbé eltérhetnek a könyvben bemutatott felépítéstől. Ez azonban természetesen nem jelenti azt, hogy ezeknek a terjesztéseknek az üzemeltetése gyökeresen új tudást igényel, a felépítés, a működés mindenkorral egységes, csak a nyelvjárás változik kissé. Ha tehát más GNU/Linux terjesztést kell üzemeltetnünk, a könyvben bemutatott ismeretek jó kiindulási alapot biztosíthatnak.

A könyvben tárgyalt ismeretanyag kiválasztása során GNU/Linux rendszereket főállásban üzemeltető, nagy tudású szakemberek véleményét és tapasztalatait is figyelembe vettettem, és megkíséreltem olyan kérdésekre megadni a választ, amelyeket a GNU/Linux-szal ismerkedő „kezdt” szakemberek tettek fel. Így a könyv nem azokat a programokat mutatja be, amelyeket könnyű használni, amelyekről érdekes beszélni, hanem azokat, amelyekkel minden napjai során a rendszergazdának dolgoznia kell. Nem szerepelnek a könyvben olyan megoldások és eszközök, amelyeket a „profik” nem használnak, akkor sem, ha azok szépek, érdekesek vagy könnyű a használatuk.

Ez úton is szeretnék köszönetet mondani mindeneknak a kollégáknak és olvasóknak, akik az eddigi könyveim hiányosságaira felhívták a figyelmet, tanácsaikkal, kérdéseikkel segítették a munkámat. Köszönet illeti a Pécsi Tudományegyetem érdeklődő hallgatóit, akik az ehelyütt tárgyalt ismeretek tanítása során kérdéseikkel hasznosabbá, érthetőbbé tették mondanivalóját. Szeretnék külön is köszönetet mondani a következő kollégáknak: CSIZMAZIA LÁSZLÓ, DR. HEGYI SÁNDOR, DR. KONIORCZYK MÁTYÁS, HARKA GYŐZŐ (CARLOS), IMREK GYULA, KISS GÁBOR, RÉBAY VIKTOR.

1.1. A jelölések és elnevezések

A szövegben található könyvtárbejegyzések neveit a szövegtől elütő betűtípussal emeltük ki (például /etc/fstab). A könyvtár típusú könyvtárbejegyzések neve / jellel végződik (például /home/). Reményeink szerint ez megkönnyíti a szöveg olvasását.

A programok és parancsok a szövegben szintén ki annak emelve (például 1s) épp úgy, ahogyan a programoknak adott paraméterek és kapcsolók (például -lahd). Hasonlóképpen kiemeltük a különféle beállítóállományokban elhelyezhető kulcsszavakat (például auto).

A könyvben a GNU/Linux és a Linux elnevezések nem véletlenszerűen követtik egymást. A GNU/Linux elnevezés a teljes telepített rendszert, a rendszer- és felhasználói programok összességét takarja. Ide tartoznak a programok, alkalmazások, a különféle beállítóállományok, egy szóval a teljes programrendszer. A Linux elnevezés a rendszermagot jelenti. A rendszermag egyetlen program, amely a számítógép hardverelemeinek közvetlen vezérlését végzi, és amely a programrendszer alapjául szolgál.

Néha nem lényeges ugyan a GNU/Linux és Linux elnevezések közti különbség, nem fontos, hogy a rendszermagról vagy a teljes rendszerről beszélünk, a legtöbb esetben azonban hasznos lehet tudni, mik azok a feladatok, amelyeket a rendszermag végez, és mi hárul az egyéb elemekre. Kínosan ügyeltünk tehát arra, mikor melyik elnevezést használjuk, a szöveg azonban a legtöbb esetben megérthető anélkül, hogy erre a különbségre egyáltalán felfigyelnénk.

Hasonlóképpen szigorúan ügyelünk a parancsok és a programok megkülönböztetésére. A héjnak parancsot adó felhasználónak nem feltétlenül kell tudnia, hogy

az általa begépeált parancsot a héj értelmezi és hajtja végre, vagy egy külső program elindulását jelenti, de a rendszert üzemeltető rendszergazdának illik tudnia, hogy belső parancsról vagy külső programról van szó.

Ha tehát egy kulcsszó értelmezését és végrehajtását a program végzi, amelynek begépeáltuk, parancsról, ha viszont a kulcsszó egy állomány neve, amelynek programként való indítását végzi az alkalmazás, programról beszélünk.

1.2. A könyvben található példák

A könyvben található néhány példaprogram és példaként felhasználható beállítóállomány. Ezek letölthetők a kiadó internetes oldalairól a <http://kiado.kiskapu.hu/98> cím felhasználásával.

A példákkal kapcsolatban fel kell hívnunk a figyelmet az óvatosságra! A programokat és beállítóállományokat ugyan többféle GNU/Linux gyűjteményen kipróbtáltuk, ez azonban nem jelenti azt, hogy azok garantáltan működnek minden számítógépen. A programok és beállítóállományok szabadon felhasználhatók és módosíthatók, valamint továbbadhatók, a használatukra azonban nem érvényes semmiféle garancia.

2. fejezet

Lemezrészek

A GNU/Linux-szal ismerkedők, a kezdeti lépéseket fontolgatók komoly problémája, hogy miképpen juthatnának működő rendszerhez, amelyen a programrendszer alapjait megismerhetik. A programokhoz ma már könnyen hozzájut hat bárki, a telepítés azonban sokaknak problémát okoz. Természetes, hogy akik a GNU/Linux-szal épp csak ismerkednek, nehézséget jelent a rendszer telepítése.

Növeli a bizonytalanságot, hogy a telepítés éppen az a munkafolyamat, ami az egyes terjesztésekben (*distribution*), GNU/Linux programgyűjteményekben a legváltozatosabb. minden GNU/Linux gyűjtemény más-más telepítőprogrammal rendelkezik. Ha ehhez még azt is hozzá tessük, hogy a telepítőprogramok állandóan változnak, fejlődnek, könnyen megérthető, miért riadnak vissza sokan a telepítés során.

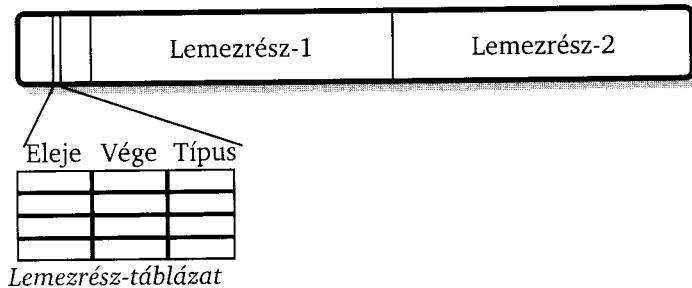
A GNU/Linux telepítése tulajdonképpen az operációs rendszer felmásolását jelenti a merevlemez háttérárra. Mielőtt azonban a felmásolás megtörténne, a merevlemez elő kell készíteni. Ennek az előkészítésnek az első lépése a merevlemez(ek) felosztása logikai egységekre, úgynevezett lemezrészekre (*partition*) [11].

A telepítés során az egyik legnehezebb feladat a lemezrészekre osztás, ami nagy előrelátást és némi tapasztalatot igényel. Igaz ugyan, hogy ma már léteznek olyan telepítőprogramok, amelyek a lemezrészekre osztást automatikusan elvégzik, a tapasztalt rendszergazda azonban mindenkor végzi ezt.

2.1. A lemezrész

A merevlemezen kialakított logikai egységek, a lemezrészek a következő alapvetően fontos tulajdonságokkal rendelkeznek:

- A lemezrészek nem lapolódhatnak át, nem lehet közös területük.



2.1. ábra. A lemezrész-táblázat

- A lemezrészek összefüggőek, kezdőpontjuktól a végpontjukig lefoglalják a lemez tárolóterületét.
- A merevlemezeknek csak a lemezrésszel lefedett területét tudjuk használni. Nem szükségszerű, hogy a lemezrészek lefedjék a merevlemez teljes területét, de azokat a részeket, amelyek egyetlen lemezrészhez sem tartoznak, nem tudjuk adattárolásra használni.

Felmerülhet a kérdés, hogy miért kell az egységes háttértárat logikai egységekre darabolni, „szétszabdalni”, miért nem lehet egyetlen lemezrészben elhelyezni a teljes GNU/Linux rendszert, egyszerűsítve ezzel a telepítést.

Valójában a GNU/Linux problémamentesen képes egyetlen lemezrészen is működni, nem szerencsés azonban így telepíteni. A lemezrészekre osztás biztonságosabbá, védettebbé teszi a rendszert, a rendszergazdának pedig megkönnyíti a munkáját. A későbbiekben látni fogunk módszereket, eszközöket, amelyeket csak helyesen felosztott merevlemezen használhatunk. Általanosságban elmondható, hogy ha egy számítógépet többen használnak, akkor mindenkihez gondosan kell felosztanunk a merevlemezeket, de akkor is érdemes több lemezrész használnunk, ha a számítógépet csak egyvalaki fogja használni.

A lemezrészek nyilvántartására a lemezrész-táblázatot (*partition table*) használjuk. A merevlemez elején található egy kis terület, ahol az egyes lemezrészek elejének és végének a pontos pontos helye van feljegyezve (2.1. ábra). A lemezrész-táblázatban általában egyéb tulajdonságokat is elhelyezünk, például a lemezrész típusát. A lemezrész típusa egy azonosítószám, amely utal arra, hogy milyen szabványok szerint szeretnénk kialakítani a lemezrészben belül található adatszerkezeteket.

2.1.1. PC lemezrészek

Sajnos az IBM rendszerű személyi számítógépek szabványainak kialakításakor a lemezrész-táblázat méretét úgy határozták meg, hogy abban csak négy lemezrész adatai férnek el. Az évek során ez a négy lemezrész kevésnek bizonyult, ezért



2.2. ábra. PC lemezrészek

a szabványokat módosítani kellett, de az eredeti szabványokkal való csereszabatosságot is meg kellett tartani. Ez az oka annak, hogy a személyi számítógépek lemezrészéinek kialakítása kissé bonyolultabb a szükségesnél.

A személyi számítógépek lemezrész-táblázatában elhelyezhető négy lemezrészt elsődleges lemezrésznek (*primary partition*) nevezzük. A lemezrészek kialakítását leíró szabvány módosítása lehetővé tette, hogy minden merevlemezen létrehozzunk egy elsődlegest lemezrész, 5-ös típuszámmal, amelyen belül újabb lemezrészeket hozhatunk létre. Az 5-ös típusú lemezrész neve kibővített lemezrész (*extended partition*) az abban létrehozott újabb lemezrészeket pedig logikai lemezrésznek (*logical partition*) nevezzük. Ne feledjük, hogy a kibővített lemezrész is elsődleges lemezrész, csak éppen a típusa miatt kibővített lemezrésznek nevezzük¹.

Egy logikai lemezrészről tartalmazó elrendezést mutat be a 2.2. ábra, ahol két elsődleges lemezrész található, melyek közül az egyik kibővített. Az ábrán látható kibővített lemezrész két logikai lemezrészről hordoz.

Mivel a merevlemezen legfeljebb 4 elsődleges lemezrész lehet, amelyből legfeljebb 1 lehet kibővített lemezrész és a kibővített lemezrészben 12 logikai lemezrész helyezhetünk el, 15 darab adattárolásra használható lemezrészre oszthatjuk fel a merevlemezeket.

Az IBM rendszerű személyi számítógép lemezrészéinek tárgyalása kapcsán meg kell említenünk, hogy más számítógépeken más korlátok lehetnek érvényben.

2.2. Blokkeszköz-meghajtók

A Linux az egyes háttértáratokat blokkeszköz-meghajtó állományokon (*block special file*) keresztül is elérhetővé teszi az alkalmazások számára. Ezen állományok mindegyike egy-egy kapu a hardver valamely eleme felé. Állományként viselkednek – azért, hogy állománykezelő parancsokkal tudjuk kezelni őket –, de némi-képpen különböznek a szokványos állományoktól. Ha ezekből az állományokból olvasunk, közvetlenül valamely hardverelemről olvasunk, ha ezekbe az állományokba írunk, közvetlenül valamely hardverelemre írunk.

¹Sok program és programleírás nem tekinti elsődleges lemezrésznek az 5-ös típusossal létrehozott kibővített lemezrészeket. Mi az egyszerűség érdekében elsődleges lemezrésznek nevezzük ezeket a lemezrészeket is, így minden lemezrész elsődleges, amelynek adatai a lemezrész-táblázatban találhatók.

2.1. tábla: mknod

Karaktereszköz és blokkeszköz-meghajtó állományok létrehozására szolgál.

mknod állománynév típus elsődleges másodlagos

A program létrehozza az eszközkezelő állományt a megadott néven, a megadott tulajdonságokkal. Azt, hogy az eszközkezelő állomány melyik hardverelemhez kapcsolódik, az elsődleges és a másodlagos eszközkezelő szám, valamint a típus határozza meg.

Típus	Jelentés
b	Blokkeszköz-meghajtó létrehozása.
c	Karaktereszköz meghajtó létrehozása.

Név	Hardverelem
/dev/hda	Elsődleges IDE csatolóra csatlakozó mester eszköz.
/dev/hdb	Elsődleges IDE csatolóra csatlakozó szolga eszköz.
/dev/hdc	Másodlagos IDE csatolóra csatlakozó mester eszköz.
/dev/hdd	Másodlagos IDE csatolóra csatlakozó szolga eszköz.

2.2. táblázat. IDE csatolók blokkeszköz-meghajtói

Az, hogy az eszközmeghajtó állományok melyik hardverelemhez nyújtanak kapcsolási pontot, az elsődleges- és másodlagos eszközsámtól (*major device number, minor device number*) függ.

Blokkeszköz-meghajtó állományt a rendszergazda bármikor létrehozhat az mknod program segítségével, vagy használhatja azokat, amelyeket a telepítőprogram a /dev/ könyvtárban elhelyezett. Az itt elhelyezett állományok nevei követtik a hagyományokat.

Az IDE (*integrated device electronics*, eszközre épített meghajtóelektronika) csatolón található merevlemezekhez tartozó blokkeszköz-meghajtó állományok nevei hagyományosan hd (*hard disk*, merevlemez) betűkkel kezdődnek. Az elnevezéseket a 2.2. táblázatban láthatjuk. A táblázatból látható, hogy a merevlemez csatlakoztatási módja egyértelműen meghatározza a blokkeszköz-meghajtó nevét, amelyet hagyományosan használnunk kell. A /dev/hdd például a másodlagos IDE csatoló szolga lemeze, függetlenül attól, hogy van-e mester lemez csatlakoztatva a másodlagos IDE csatolóra, vagy van-e eszköz az elsődleges IDE csatolón.

Az SCSI (*small computer system interface*, kisszámitógépes csatolófelület) csatolófelületen keresztül csatlakoztatott merevlemezek eszközmeghajtó állományai hagyományosan sd (*SCSI disk*, SCSI merevlemez) kezdetű nevet kapnak. Ezeknek az eszközöknek a nevét a 2.3. táblázatban találjuk.

A táblázatból látható, hogy az SCSI merevlemezek elnevezése nem független egymástól. Ha eltávolítjuk a rendszerből a legkisebb SCSI azonosítóval rendelkező merevlemezt, az összes többi SCSI merevlemez blokkeszköz-meghajtó állományának megváltozik a neve.

Név	Hardverelem
/dev/sda	A legkisebb SCSI azonosítóval rendelkező merevlemez.
/dev/sdb	A második legkisebb SCSI azonosítójú merevlemez.
/dev/sdc	A harmadik legkisebb SCSI azonosítójú merevlemez.

2.3. táblázat. SCSI merevlemezek blokkeszköz-meghajtói

Név	Hardverelem
/dev/scd0	A legkisebb SCSI azonosítóval rendelkező CD-ROM meghajtó.
/dev/scd1	A második legkisebb SCSI azonosítójú CD-ROM meghajtó.
/dev/scd2	A harmadik legkisebb SCSI azonosítójú CD-ROM meghajtó.

2.4. táblázat. SCSI CD-ROM meghajtók blokkeszköz-meghajtói

Ha a számítógépben IDE csatolófelülettel rendelkező CD-ROM meghajtó van, akkor azt ugyanúgy érjük el a /dev/hda.../dev/hdd állományokon keresztül, mint a merevlemezes meghajtókat. Más a helyzet viszont az SCSI csatolófelülettel szerelt CD-ROM eszközökkel. Ezek hagyományosan a scd (SCSI CD-ROM) kezdetű névvel ellátott eszközkezelő állományokon keresztül érhetőek el. Ezt a 2.4. táblázat mutatja be. Láthatjuk, hogy az SCSI CD-ROM eszközöket számokkal különböztetjük meg egymástól.

Mind az IDE csatolóra, mind pedig az SCSI csatolóra csatlakoztatott merevlemezen számokkal különböztetjük meg az egyes lemezrészeket. A PC szabványok nem tesznek különbséget az SCSI és az IDE csatolón keresztül kapcsolódó merevlemezek lemezrészinek kezelése között és hasonlóképpen egységes a lemezrészkekhez tartozó blokkeszköz-meghajtó állományok elnevezése is. Mindig az 1-es számot kapja az első elsődleges lemezrész, és az 5-ös számot az első logikai lemezrész. Különbség van azonban az elsődleges és a logikai lemezrészek számozása közt.

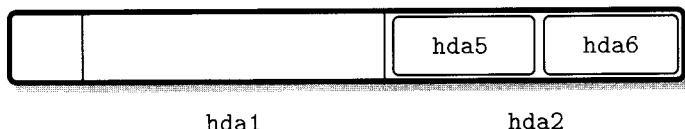
Az elsődleges lemezrészekhez rendelt számok azt mutatják meg, hogy a lemezrész-táblázatban a négy lehetséges hely közül melyikben tároltuk az adott lemezrész adatait. Lehetséges például, hogy létezik a merevlemezen a második lemezrész, de nem létezik az első. Ez azt jelenti, hogy a lemezrész-táblázat első bejegyzése üres, a második pedig ki van töltve.

A logikai lemezrészek számozása ezzel ellentétben minden folytonos számokkal történik (és minden 5-től indul). Ha létezik a 6-os lemezrész, akkor léteznie kell az 5-ös lemezrésznek is. Ez azt eredményezi, hogy ha törlünk egy logikai lemezrész, a felette található logikai lemezrészek számozása megváltozik, a felső lemezrészek száma eggyel csökken.

A lemezrészek blokkeszköz-meghajtójának elnevezésére a 2.5. táblázat és a 2.3. ábra ad példát.

Név	Hardverelem
/dev/hda2	Az elsődleges IDE csatoló mester lemezének lemezrész-táblázatában a második helyen található lemezrész.
/dev/sdb6	A második legkisebb SCSI azonosítóval rendelkező SCSI merevlemez második logikai lemezrésze.

2.5. táblázat. SCSI és IDE lemezrészek blokkeszköz-meghajtói



2.3. ábra. PC lemezrészek nevei

Blokkeszköz-meghajtók használata

Ha egy blokkeszköz-meghajtót állományként megnyitunk és abból olvasunk, a hozzá tartozó háttértárból, lemezrészről olvasunk. Az olvasás a legelső bájtnál kezdődik, és folyamatosan, a háttértár vagy a lemezrész végéig folytatható. Ha írásra nyitjuk meg a blokkeszköz-meghajtót, az írást az eszköz vagy lemezrész elején kezdjük, és egészen a végéig folytathatjuk.

Ha a megnyitott blokkeszköz-meghajtó létezik, de a hozzá tartozó fizikai eszköz vagy lemezrész nem, hibaüzenetet kapunk.

A blokkeszköz-meghajtók állományként való olvasása és írása természetesen biztonsági kérdéseket is felvet. Általában elmondható, hogy csak a rendszergazdának vagy a különleges jogokkal felruházott személyeknek van jog a közvetlenül, a blokkeszköz-meghajtó programokon keresztül kezelni a háttértárat. Kivétel általában ez alól a hajlékonylemez, amelyet a /dev/fd0, /dev/fd1,... blokkeszköz-meghajtó állományokon keresztül a legtöbb esetben a különleges jogokkal fel nem ruházott felhasználók is elérhetnek.

1. példa Tegyük fel, hogy két tökéletesen egyforma számítógép áll a rendelkezésünkre, amelyek közül az egyikre telepítettünk minden programot, és elvégeztük rajta minden beállítást. Annak érdekében, hogy a teljes telepített rendszert könnyen át tudjuk másolni, a telepített számítógép másodlagos IDE vezérlőjére csatlakoztattuk a másik számítógép merevlemezét (átadtuk a lemezt a másik számítógépbe). A következő parancs segítségével a merevlemez minden egyes bájtját átmásolhatjuk („klónozhatjuk”) a még nem telepített számítógép merevlemezére:

```
$ cat /dev/hda >/dev/dhc
$
```

A példánál feltételeztük, hogy minden merevlemez beállítása szerint mesterként csatlakozik a számítógéphez.

2.6. tábla: fdisk

Lemezrészek létrehozására és törlésére szolgáló interaktív program.

```
fdisk [kapcs] blokkeszköz
fdisk -l [kapcs] [blokkeszköz]
fdisk -s part_eszköz
```

Az első formában interaktív módon használható, a második formában kiírja a lemezrészek listáját, a harmadik formában a lemezrész méretét írja ki.

Kapcsoló *Jelentés*

- l Kiíratja a lemezrészek listáját, majd kilép. Ha nem adunk meg a kapcsolóval blokkeszközt, minden merevlemez adatait kiírja.
- u A kiíratáskor nem a henger (*cylinder*), hanem a szektor a mértékegység.
- s A kapcsoló után megadott blokkeszköznek egy lemezrészhez kell tartoznia, amelynek méretét írja ki.

2. példa Szeretnénk tudni, hogy az SCSI csatolóhoz van-e használható merevlemez csatlakoztatva. A legegyszerűbb módja, hogy megtudjuk az ha megpróbálunk adatokat olvasni a merevlemezről. A következő parancs tizenhatos számrendszerben írja ki a merevlemez első 32 bájtját:

```
$ head -c 32 /dev/sda | od -t x
0000000 8ec033fa 7c00bcd0 0750f48b fcfcfb1f50
00000020 b90600bf a5f20100 00061dea 07bebe00
0000040
$
```

Most próbálunk meg hasonló módon olvasni a második SCSI merevlemezről:

```
$ head -c 32 /dev/sdb | od -t x
head: /dev/sdb: Nincs ilyen eszköz vagy cím
0000000
$
```

Amint látjuk, nem volt lehetséges az olvasás. A hibaüzenetből megtudhatjuk, hogy nincs ilyen eszköz, vagyis az SCSI vezérlőhöz csak egy merevlemez csatlakozik.

2.3. Lemezrészek létrehozása

GNU/Linux rendszereken több program is használható lemezrészek létrehozására és törlésére. Az ilyen jellegű programok közül az fdisk minden GNU/Linux-ot futtató számítógépen elérhető, ezért ezt mutatjuk be.

Ha lemezrészeket akarunk törölni vagy létrehozni, az fdisk programot interaktív módon indítjuk. Ilyenkor egy merevlemezhez tartozó blokkeszköz-meghajtót kell megadnunk paraméterként. Ha a merevlemez elérhető, a program elindul, és egy parancskérő jellet jelzi, hogy kész a munkára:

```
$ fdisk /dev/sda
Command (m for help):
```

A munka során különféle parancsokkal módosíthatjuk a merevlemez lemezrészei-nek szerkezetét, és ha úgy gondoljuk, hogy elértek a célunkat, kiírhatjuk az eredményt a merevlemezre. Az fdisk csak a mentéskor változtatja a merevlemez tartalmát; ha valamit elrontottunk, bármikor kiléphetünk a változtatások elvetésével, megőrizve az eredeti elrendezést.

A lemezrészek elrendezésének mentése után nincs szükség a számítógép újraindítására, a módosítások azonnal életbe lépnek, az új lemezrészek használhatók.



*Soha ne módosítsunk olyan lemezrészet, amely a rendszerbe van illesztve!
Ha egy merevlemezen van beillesztett lemezrész, hajthatunk végre olyan módosításokat, amelyek nem érintik a beillesztett lemezrészet, de a beillesztett lemezrész módosítása problémákat okozhat!*

Ahogyan az fdisk parancskérő jele is utal rá, a használható parancsokról az m parancccsal kérhetünk listát.

```
Command (m for help): m
Command action
  a    toggle a bootable flag
  b    edit bsd disklabel
  c    toggle the dos compatibility flag
  d    delete a partition
  l    list known partition types
  m    print this menu
  n    add a new partition
  o    create a new empty DOS partition table
  p    print the partition table
  q    quit without saving changes
  s    create a new empty Sun disklabel
  t    change a partition's system id
  u    change display/entry units
  v    verify the partition table
  w    write table to disk and exit
  x    extra functionality (experts only)
```

```
Command (m for help):
```

A parancsok a következő műveleteket valósítják meg:

- a A lemezrészhez tartozó, betölthető/be nem tölthető kapcsoló ellentétre állítása.

Minden lemezrészhez tartozik egy kapcsoló, amely azt jelzi, hogy az adott lemezrész tartalmaz-e betölthető (*bootable*) operációs rendszert². A parancs ezt a kapcsolót állítja az ellentétre, így tudjuk be-, illetve kikapcsolni. Látni fogjuk, hogy a számítógép indításához nem feltétlenül van szükség erre a kapcsolóra, nem feltétlenül kell tehát használnunk ezt a parancsot.

- b A BSD lemezcímke kezelésére GNU/Linux rendszeren nem feltétlenül van szükség.
- c A DOS-szerű lemezrészek kezelésére szintén nem feltétlenül van szükség.
- d Lemezrész törlése.

Ha csak egyetlen lemezrész található a merevlemezen, az `fdisk` azonnal törli, ha több, egy kérdésre válaszul meg kell adnunk melyiket szeretnénk törölni.

1 Lemezrésztípusok kiíratása.

A parancs hatására a program egy emlékeztető listát ír a képernyőre, amelyben megtalálhatók a közismert lemezrésztípusok nevei és számai. Ezt a parancsot kell használnunk, ha nem tudjuk, hogy mi a kódja a lemezrésztípusnak, amit létre akarunk hozni.

m A parancsok és egysoros ismertetésük listája.

Ezt a parancsot használnunk kell, ha nem tudjuk, hogy az egyes műveleteket milyen parancssal érhetjük el.

n Új lemezrész létrehozása.

A program először megkérdezi, hogy milyen lemezrész kívánunk létrehozni (elsődleges, logikai stb.), majd megkérdezi, hogy hol kezdődjék és végződjék a létrehozott lemezrész. Amint a válaszokat megadtuk, a lemezrész létérehozza, de természetesen csak a mentéskor írja a merevlemezre.

- o Különleges, DOS-on használt lemezrész-táblázat létrehozása, nem feltétlenül szükséges használnunk.

p A lemezrészek kiíratása a képernyőre.

A program a parancs hatására táblázatos formában kiírja a képernyőre a lemezrészek szerkezetét, ahogyan a memoriában tárolja. Lehetséges, hogy a kiírt változat nem egyezik meg a merevlemezen tárolt változattal, a memoriában tárolt változat ugyanis csak a mentéskor kerül a merevlemezre.

²A betölthető rendszert tartalmazó lemezrész néha szokás aktív lemezrésznek is nevezni.

q Kilépés mentés nélkül.

Ha ezzel a parancsal lépünk ki, a merevlemezen megmarad az eredeti elrendezés. Ha tehát elrontottunk valamit, egyszerűen kiléphetünk mentés nélkül.

s Sun lemezcímke létrehozása nem feltétlenül szükséges személyi számítógépeken, ezért ezt a parancsot nem feltétlenül kell használnunk.

t Lemezrész típusának megváltoztatása.

Ha csak egy lemezrész van, csak az új típust kell megadnunk, ha több, meg kell adnunk, melyik lemezrész típusát akarjuk megváltoztatni, és azt is, hogy mi legyen az új típus. A típus megadásakor újra módunk nyílik a típusokat tartalmazó emlékeztető lista megtekintésére.

u A használt mértékegység megváltoztatása.

Valahányszor kiadjuk ezt a parancsot, a program megváltoztatja a használt mértékegységet, és kiírja a képernyőre.

v A lemezrészek elrendezésének ellenőrzése.

A parancs hatására a program ellenőri a memóriában tárolt elrendezést, és kiírja, ha például lemezrészekkel le nem fedett területek vannak a merevlemezen.

w Kilépés mentéssel.

Ha ezt a parancsot kiadjuk, a program a merevlemezre írja az új lemezrész-ellenőrzést, értesíti a módosításról a rendszert, majd kilép. A módosításokat ezek után csak akkor tudjuk visszavonni, ha valamilyen módon megjegyeztük az eredeti elrendezést.

x Szakértői üzemmód újabb parancsokkal.

3. példa Hozzunk létre egy merevlemezen egy 32 megabájt méretű elsődleges lemezrészét, és a maradék területet egyenlő arányban fedjük le két logikai lemezrésszel! A lemezrészek közül az elsődlegest és az egyik logikait a GNU/Linux számára készítsük, a másik logikait pedig az OpenBSD számára.

A program indításakor kérjünk egy listát a létező lemezrészkről, hogy szükség esetén törölni tudjuk azokat:

Command (m for help): **p** [Enter]

```
Disk /dev/sda: 131 MB, 131072000 bytes
5 heads, 50 sectors/track, 1024 cylinders
Units = cylinders of 250 * 512 = 128000 bytes
```

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------

```
/dev/sda1      1      1024    127975  83  Linux
```

Command (m for help):

Láthatjuk, hogy létezik egy elsődleges lemezrész, amely azonban nem felel meg a számunkra, mivel a rendelkezésre álló teljes területet lefoglalja. Töröljük tehát a lemezrészt:

Command (m for help): **d****[Enter]**
Selected partition 1

Command (m for help):

Most elkezdhetjük létrehozni a tervezett lemezrészeket.

Command (m for help): **n****[Enter]**
Command action
e extended
p primary partition (1-4)
p**[Enter]**
Partition number (1-4): **1****[Enter]**
First cylinder (1-1024, default 1): **[Enter]**
Using default value 1
Last cylinder or +size or +sizeM or
+sizeK (1-1024, default 1024): **+32M****[Enter]**

Command (m for help):

Amint látjuk, a program előbb megkérdezte, hogy elsődleges vagy kibővített lemezrészt kívánunk létrehozni, amire mi azt választottuk, hogy elsődlegest. A következő kérdés arra vonatkozott, hogy mi legyen a lemezrész száma, amire azt választottuk, hogy 1.

A következő kérdés arra vonatkozott, hogy hol kezdődjön a lemezrész. A program – a szabad terület ismeretében felajánlotta az 1. hengert (cylinder), amelyet mi el is fogadtunk egy **[Enter]** leütéssel.

A következő kérdés a lemezrész végére vonatkozott. Erre nem a konkrét henger megadásával választottunk, egy + jelrel jelezük, hogy a méretet adjuk meg, az M betűvel pedig azt, hogy a mértékegység megabájt lesz.

Következő feladatunk a két logikai lemezrész létrehozása lesz. Mivel logikai lemezrészt csak kibővített lemezrészen belül lehet létrehozni, készítenünk kell egy kibővített lemezrészt, amely lefedi a teljes fennmaradó területet.

Command (m for help): **n****[Enter]**
Command action
e extended
p primary partition (1-4)
e**[Enter]**
Partition number (1-4): **2****[Enter]**

```
First cylinder (252-1024, default 252):   

Using default value 252  

Last cylinder or +size or +sizeM or  

+sizeK (252-1024, default 1024):   

Using default value 1024
```

Command (m for help):

Amint látható, a kibővített lemezrész a második elsődleges lemezrész.

Amikor most az újabb lemezrészt elkezdjük létrehozni, a program lehetőségeként már a logikai lemezrész létrehozását is felajánlja, hiszen ha létezik kibővített lemezrész, logikai lemezrész is létrehozható.

```
Command (m for help):  n   

Command action  

  l  logical (5 or over)  

  p  primary partition (1-4)  

First cylinder (252-1024, default 252):  

Using default value 252  

Last cylinder or +size or +sizeM or  

+sizeK (252-1024, default 1024):  6  3  8 
```

Command (m for help):

Mivel az volt a tervünk, hogy a fennmaradó területet egyenlő arányban osztjuk el a logikai lemezrészek között, a lemezrész végét úgy adtuk meg, hogy még egy logikai lemezrészt létrehozhassunk ugyanilyen méretben. Ezt a következőképpen tesszük:

```
Command (m for help):  n   

Command action  

  l  logical (5 or over)  

  p  primary partition (1-4)  

First cylinder (639-1024, default 639):   

Using default value 639  

Last cylinder or +size or +sizeM or  

+sizeK (639-1024, default 1024):   

Using default value 1024
```

Command (m for help):

Ha most kiíratjuk a létrehozott lemezrészeket, láthatjuk, hogy körülbelül a tervezett elrendezést hoztuk létre:

```
Command (m for help):  p 
```

```
Disk /dev/sda: 131 MB, 131072000 bytes  

5 heads, 50 sectors/track, 1024 cylinders
```

Units = cylinders of 250 * 512 = 128000 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	251	31350	83	Linux
/dev/sda2		252	1024	96625	5	Extended
/dev/sda5		252	638	48350	83	Linux
/dev/sda6		639	1024	48225	83	Linux

Command (m for help):

Most már nincs más dolgunk, mint az utolsó lemezrész típusának beállítása. A beállítás során az 1 parancssal kérhetünk emlékeztető listát a lemezrészek típusáról. A helytakarékkosság érdekében tegyük fel, hogy emlékezünk a típusszámra.

Command (m for help): [Enter]

Partition number (1-6): [Enter]

Hex code (type L to list codes): [Enter]

Changed system type of partition 6 to a6 (OpenBSD)

Command (m for help):

Utolsó lépésként vessünk még egy pillantást a lemezrészek elrendezésére:

Command (m for help): [Enter]

Disk /dev/sda: 131 MB, 131072000 bytes

5 heads, 50 sectors/track, 1024 cylinders

Units = cylinders of 250 * 512 = 128000 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	251	31350	83	Linux
/dev/sda2		252	1024	96625	5	Extended
/dev/sda5		252	638	48350	83	Linux
/dev/sda6		639	1024	48225	a6	OpenBSD

Command (m for help):

Nincs más hátra mint a mentés és a kilépés, amelyet a w parancssal tehetünk meg.

Command (m for help): [Enter]

The partition table has been altered!

Calling ioctl() to re-read partition table.

Syncing disks.

\$

A kilépés után az új lemezrészek azonnal használhatók.

2.4. A GNU/Linux könyvtárszerkezete

Az előző fejezetből megtudhattuk, miképpen oszthatjuk merevlemezeinket lemezrészre, hogyan hozhatjuk létre a kívánt lemezrész-elrendezést. Ebben a fejezetben arról lesz szó, hogy milyen elrendezést hozunk létre, vagyis hogyan tervezzük meg a merevlemezek lemezrészéinek elrendezését. Mivel a lemezrészek elrendezésén utólag már nehéz változtatni, fontos, hogy előrelátóan tervezzük meg a szerkezetet, és pontosan ez az, amiben e fejezet a segítségünkre lesz.

A GNU/Linux könyvtárszerkezete követi a Unix évtizedek alatt kialakult könyvtárszerkezetét. Apróbb különbségek vannak a különféle Unix-változatok könyvtárszerkezete közt – hiszen a szerkezet ma is fejlődik –, de általában elmondható, hogy a Unix könyvtárszerkezet szabványos; mindennek megvan a helye a könyvtárak ágai közt.

A GNU/Linux az állományrendszer-szabványt (FHS, *filesystem hierarchy standard*) követi, ami leírja, hogy az egyes könyvtárakban minék kell lennie, a lemezrészeket hogyan kell kialakítani. A szabványosításnak köszönhetően a felhasználók, rendszergazdák és programozók mindenkor tudják, hogy az egyes állományokat hol keressék, hova tegyék. A szabvány lehetővé teszi számunkra azt is, hogy a lemezrészek elrendezését megtervezzük, és hatékony, könnyen felügyelhető, biztonságos rendszert hozzunk létre a telepítéskor.

A lemezrészek elrendezését ugyanis nekünk kell megtervezni a telepítéskor vagy előtte, hiszen az igények és a lehetőségek mindenkor változnak. Természetesen nem lehet egyforma lemezrész-elrendezést létrehozni egy öreg irodai számítógépen, ami kis méretű merevlemezzel kizárolag szövegszerkesztési feladatokat lát el és a vállalat legújabb, több millió forintért vásárolt központi kiszolgálóján. A lemezrészek elrendezésénél mindenkor figyelembe kell vennünk az igényeket, a feladatok jellegét és a lehetőségeket, és meg kell próbálnunk olyan elrendezést kialakítani, ami a jövőben felmerülő igényeknek is meg fog felelni.

2.4.1. A gyökérkönyvtár lemezrészé

A Unix rendszerek működéséhez mindenkor szükség van néhány állományra. Ha ezek az állományok elérhetetlenné válnak, a rendszer leáll, és még a karbantartás, helyreállítás sem végezhető el. A könyvtárszerkezet úgy van felépítve, hogy ezek a működéshez elengedhetetlenül szükséges állományok a gyökérkönyvár közelében helyezkedjenek el.



Ha a GNU/Linux gyökérkönyvtára vagy a gyökérkönyvtár közelében elhelyezett létfontosságú állományok valamelyike elérhetetlenné válik, a rendszer nem működik tovább, még a karbantartás, helyreállítás sem végezhető el. Ilyenkor külső adathordozóról – helyreállítólemezről – kell az állományokat biztosítanunk a rendszer helyreállításának idejére. Az Interneten több olyan GNU/Linux terjesztés található, amelyek kimondottan a helyreállításra képzültek.

A gyökérkönyvtárat tartalmazó lemezrész (*root partition*) kialakítására minden képpen szükség van a lemez részekre osztása során. Ha csak egyetlen lemezrészet hozunk létre és abba telepítjük a rendszert, akkor ez az egy lemezrész fogja tartalmazni a gyökérkönyvtárat. Nem szerencsés azonban minden egyetlen lemezrészbe telepíteni. Mivel a GNU/Linux számára a gyökérkönyvtár (és a közvetlen környéke) elengedhetetlenül szükséges, szerencsés, ha ez a lemezrész minél kisebb, hiszen kisebb lemezrészben kisebb valószínűséggel lesznek hibák.



Törekednünk kell arra, hogy a gyökérkönyvtárat tartalmazó lemezrészet minél kisebb mérettel hozzuk létre, hogy kisebb legyen a meghibásodás valószínűsége. A gyökérkönyvtárat tartalmazó lemezrész természetesen csak akkor lehet kicsi, ha mindeneket az állományokat, amelyek nem feltétlenül szükségesek a működéshez, más lemezrészeken helyezzük el.

Azt mondjuk, hogy a gyökérkönyvtárat tartalmazó lemezrészen csak azok az állományok helyezkedjenek el, amelyek már az induláskor is szükségesek. A rendszer a rendszerindítás bizonyos szakaszában beilleszti azokat a lemezrészeket, amelyeknek tartalmára a felhasználóknak ugyan szükségük van, de a rendszer működéséhez – a karbantartáshoz, szervizüzemhez – nem feltétlenül kellenek.

A GNU/Linux – néhány más Unix rendszerrel ellentétben – a gyökérkönyvtárban nem tart állományokat, minden állományt a gyökérkönyvtárban található alkönyvtárakban helyez el. A következő lista azokat a gyökérkönyvtárban található könyvtárakat sorolja fel, amelyeknek teljes tartalma a gyökérkönyvtárat tartalmazó lemezrészen kell hogy elhelyezkedjen. Ezeknek a könyvtáraknak – és a tartalmuknak – már az induláskor elérhetőnek kell lenniük.

A gyökérkönyvtárat tartalmazó lemezrészen a következő könyvtárak tartalmát szokás elhelyezni:

/bin/ Ez a könyvtár olyan programok futtatható állományait tartalmazza, amelyek már a rendszer indulásához is szükségesek, és amelyeket később a felhasználók is használnak (*binaries*, binárisok).

Ebben a könyvtárban található például a `mount` program, amely újabb lemezrészek beillesztésére használható. Erre a programra természetesen szükség van a rendszer indításához, mert enélkül a többi lemezrészket nem tudnánk beilleszteni.

/sbin/ E könyvtárban azoknak a programoknak a futtatható állományai találhatók, amelyekre már a rendszer indításakor szükség van, és amelyeket a felhasználók általában nem használnak (*system binaries*, rendszerbinárisok).

Ebben a könyvtárban található például az `fsck`, amely a háttértárat ellenőrzését végzi. Erre a programra már az induláskor szükség van, hiszen indulás közben a GNU/Linux ellenőrzi a merevlemezek tartalmát. A felhasználóknak általában nincs szükségük erre a programra, bár az használható a hajlékonylemez ellenőrzésére, ezért elérhetővé tehetjük a felhasználók számára is.

A legtöbb GNU/Linux gyűjtemény a `/sbin/` könyvtárat nem zárja el a felhasználók elől, de a könyvtár nem szerepel a keresési útban, ezért csak akkor használhatják, ha a teljes utat beírják parancsként (például `/sbin/fsck`) A rendszergazdának nem szükséges beírnia a teljes utat, az ő általa használt keresési út tartalmazza a `/sbin/` könyvtárat.

`/etc/` Ez a könyvtár azoknak a programoknak a beállítóállományait tartalmazza, amely programokra már a rendszer indítása során szükség van (*et cetera*, és így tovább).

`/root/` Ez a könyvtár a rendszergazda saját könyvtára (*root*, gyökér).

Célszerű e könyvtár tartalmát is a gyökérkönyvtárat tartalmazó lemezrészben tárolni, hogy egy esetleges rendszerösszeomlás után a helyreállítást végző rendszergazda elérhesse az anyagait, programjait.

Természetesen óvakodnia kell a rendszergazdának attól, hogy a rendszergazdai teendőkhöz nem feltétlenül szükséges nagy méretű állományokat ebben a könyvtárban tárolja.

`/lib/` E könyvtárban azoknak a programoknak a megosztott programkönyvtárai találhatók, amelyekre már az induláskor is szükség van (*libraries*, könyvtárak).

`/dev/` Eszközkezelő állományokat tartalmazó könyvtár (*devices*, eszközök).

Az itt található eszközkezelő állományokon keresztül teszi elérhetővé a Linux az egyes hardvereszközöket.

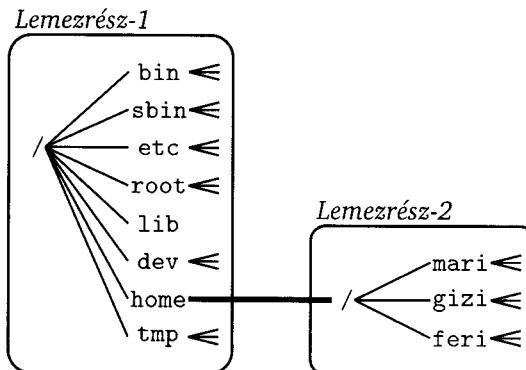
`/tmp/` Azok a programok hoznak létre ideiglenes állományokat ebben a könyvtárban, amelyekre már a rendszer indításakor is szükség van (*temporary*, ideiglenes).

A többi program számára a `/var/tmp/` könyvtár használata javasolt, sajnos azonban nagyon sok program létezik, amelyekre nincs ugyan szükség a rendszer indításakor, mégis ebben a könyvtárban hoznak létre ideiglenes állományokat. Ezzel természetesen feleslegesen terhelik a gyökérkönyvtárat tartalmazó lemezrészét, és így annak méretét feleslegesen nagyra kell állítanunk.

A megoldást az jelentheti, hogy egy külön lemezrész illesztünk be indulás után a `/var/tmp/` könyvtárba, így a rendszer indulása után ennek a könyvtárnak a tartalma már nem a gyökérkönyvtárat tartalmazó lemezrészben található.

`/boot/` Ez a könyvtár tartalmazza a rendszer betöltéséhez szükséges állományokat (*boot*, rendszertöltés).

Érdekes módon a rendszertöltő programok különleges működésének köszönhetően e könyvtár tartalmának általában nem kell a gyökérkönyvtárat tartalmazó lemezrészben lennie – sőt sokszor nem is szerencsés.



2.4. ábra. A /home/ könyvtár beillesztése

A gyökérkönyvtárat tartalmazó lemezrész mérete – ha minden más elemet külön lemezrészekre helyezünk – viszonylag kicsi lehet. Azt mondhatjuk, hogy legalább 128, legfeljebb néhány száz megabájtnak kell lennie ennek a lemezrésznek, ha a GNU/Linux egyéb elemei más lemezrészekre kerülnek.

2.4.2. Egyéb lemezrészek

A gyökérkönyvtárat tartalmazó lemezrészen található még jó néhány könyvtár, amelyekbe általában egyéb lemezrészeket illesztünk be. E könyvtárak tehát a gyökérkönyvtárat tartalmazó lemezrészre kerülnek, de tartalmuk nem. Ezt a szerkezetet mutatja be a 2.4. ábra.

A következőkben sorra vesszük azokat a könyvtárakat, amelyek a gyökérkönyvtárat tartalmazó lemezrészen helyezkednek el és amelyekbe egyéb lemezrészeket szokás beilleszteni.

/home/ Ebben a könyvtárban vannak a felhasználók saját könyvtárai (*home*, ott-hon). A felhasználók itt hozhatják létre állományaikat, könyvtárait, itt található minden saját adatuk.

Ennek a könyvtárnak a tartalmát mindenkorban külön lemezrészre érdemes helyezni, mivel az itt található állományok több szempontból is különlegesek:

- A felhasználóknak ezen a lemezrészen írási joguk van. Általában a könyvtárszerkezet más területén nincs a felhasználóknak írási joga. (Kivételt szokott ez alól képezni a /var/spool/mail/ könyvtár, ahol a felhasználók leveleslátása található és a /tmp/ könyvtár az ideiglenes állományok számára.)
- Az itt található adatok folyamatosan változnak.

- Az itt található adatok pótolhatatlanok. Ha egy program megsérül, az Internetről általában pótolni tudjuk, de a felhasználók adatai itt keletkeztek, ezért másol nem találhatók meg. Éppen ezért az itt található állományokat rendszeresen archiválnunk kell, hogy egy esetleges meghibásodás után helyreállíthatók legyenek.
- Az itt található állományok nem a GNU/Linux részei, ha tehát a rendszert cseréljük, ezek az adatok maradhatnak a régi állapotban.
Ez az oka annak, hogy még akkor is érdemes külön lemezrészét létrehozni, ha csak egyedül használjuk az adott számítógépet. Ha úgy döntenénk, hogy áttérünk egy másik GNU/Linux terjesztés használatára és előrelátóan külön lemezrészet hoztunk létre a /home/ könyvtár részére, az adataink könnyedén megőrizhetők a telepítés során.
- A felhasználók igényelhetik, hogy több számítógépen is elérjék az állományait, ezért ennek a lemezrésznek a tartalmát sokszor hálózati állományrendszerként több gép is használja.

Amikor ezt a lemezrészet készítjük, általában az összes „megmaradt” lemezterületet felhasználjuk. A rendszer számára kiosztva a szükséges merevlemezterületet, a fennmaradó hely a felhasználóké lesz, ezért a /home/ lemezrész nagyságát általában a lehetőségek és nem az igények szabják meg.

/usr/ Nagyméretű felhasználói programok, alkalmazások tárolására szolgáló könyvtár (*user resources*, a felhasználók erőforrásai).

Ezekre a programokra általában nincs szükség a rendszer működéséhez, de olyan szolgáltatásokat végeznek, olyan feladatokat látnak el, amelyeket a felhasználók használnak. Ide kerülnek a grafikus programok, az irodai programcsomagok, a programfejlesztő környezetek és így tovább.

A legtöbb rendszeren érdemes a /usr/ tartalmát külön lemezrészen elhelyezni, mivel igen nagy mennyiséggű állományról van szó, amelyeket érdemes külön kezelni.

A /usr/ könyvtár tartalma csak akkor változik, ha újabb alkalmazást telepítünk, telepített alkalmazást frissítünk vagy távolítunk el, ezért általában írásvédett formában használjuk a számára készített lemezrészet. Ha a /usr/ könyvtárat csak olvasható formában illesztjük be, kisebb a valószínűsége annak, hogy megsérüljenek az alkalmazások.

Az, hogy mekkora méretűre kell állítanunk a /usr/ lemezrészet, nagymértekben függ attól, hogy mennyi alkalmazást szeretnénk használni. Általában elmondható, hogy ma már sokszor 4-5 gigabájt méretű lemezrészet használunk az alkalmazások nagy száma és mérete miatt.

/contrib/ Ebben a könyvtárban nagyméretű felhasználói programcsomagok helyezkednek el, amelyekre nem feltétlenül van szüksége a felhasználóknak (*contribution, hozzájárulás*). Sok GNU/Linux rendszerről hiányzik e könyvtár.

A körültekintő rendszergazda a /contrib/ könyvtárba a kísérleti jelleggel telepített alkalmazásokat telepíti, esetleg azokat a felhasználói programokat, amelyek a felhasználók napi munkájához nem feltétlenül szükségesek. Így érhetjük el, hogy az alkalmazásokat könnyedén törölhessük, és az alkalmazás kísérleti változata ne zavarja meg a jól bevált változat működését.

/mnt/ A cserélhető adathordozót tartalmazó háttértárak beépítési pontjai találhatók itt (*to mount*, felszerel). Más rendszereken szokás a /floppy/, /cdrom/ könyvtárakat használni a hajlékonylemez, illetve a CD-ROM lemez tartalmának beillesztésére.

/proc/ Szimulált állományrendszer a folyamatok és a rendszermag adataival (*processes*, folyamatok).

Az itt található állományok nem foglalnak helyet a háttértáron, ezeket a Linux rendszermag szimulálja a felhasználói programokkal való kapcsolattartáshoz.

/var/ A minden nap használat során folyamatosan változó állományok tárolására szolgáló könyvtár (*variable*, változó). Itt általában adatokat találunk.

Hogy mekkora legyen a /var/ lemezrész, az nagymértékben függ attól, hogy mire szeretnénk használni a számítógépet. Mivel ide kerülnek az adatbázis-kiszolgálók, webkiszolgálók által kezelt adatok, a szükséges méret elérheti a több száz gigabájtot is.

Ha nem kívánunk nagyobb adatbázisokat kezelní a számítógéppel, elegendő lehet akár 128 megabájt terület is, amelyen elférhetnek a bejövő elektronikus levelek, a rendszernapló állományai és a nyomtatást kezelő alrendszer ideiglenes állományai. Mivel ezeket az állományokat a felhasználók által indított programok hozzák létre, a szükséges hely nagysága a felhasználók számával együtt növekedhet.

2.5. A lemezrészek terheltsége

A rendszergazdának nem csak a lemezrészek kialakításakor kell gondosan eljárnia, hanem később, a számítógép üzemeltetése során is figyelemmel kell kísérnie az egyes lemezrészek terhelését. Szükség esetén közbe kell lépni, hogy egyetlen lemezrész se teljen meg teljesen adatokkal, hiszen az lehetetlenné tenné újabb állományok létrehozását, újabb adatok tárolását.

A rendszergazda a df parancs segítségével figyelheti az egyes lemezrészek terhelését. A parancs kiírja a rendelkezésre álló üres területet és a lemezrész terhelt séget százalékos értékben.

4. példa A következő példa egy irodai feladatokat ellátó számítógép lemezrészzeit mutatja be. Figyeljük meg, hogy a számítógépen egy lemezrészet alakítottunk

2.7. tábla: df

Lemezrészek szabad és üres területeinek méretét vizsgáló program.

df [kapcs] [könyvtárbejegyzés]

A program felsorolja, hogy az egyes lemezrészek mérete mekkora, mennyi a foglalt és a szabad terület, milyen a lemezrészek kihasználtsága. Ha a program egy könyvtár vagy egy szabályos állomány nevét kapja, csak annak a lemezrésznek az adatait írja ki, amelyen az adott könyvtárbejegyzés elhelyezkedik.

Kapcsoló Jelentés

- h Kerekített, mértékegységgel ellátott, könnyen olvasható formában írja ki az adatokat.
- i Nem az adatterületek méretét, hanem a mutatópontok számát írja ki.

ki a gyökérkönyvtár számára, egy pedig a felhasználók saját könyvtárait hordozza. Nyilvánvaló, hogy a lemezrészek kialakításakor az egyszerűség volt a cél. A rendszer elhelyezéséhez szükséges terület kialakítása után a teljes fennmaradó részt a felhasználók adatainak számára foglaltuk le.

```
$ df -h
Fájlrendszer      Méret  Fogl. Szab.  % Csatl. pont
/dev/sda1          2.8G   2.0G  695M  75% /
/dev/sda3          31G    16G   14G  54% /home
```

```
$ fdisk -l /dev/sda
Disk /dev/sda: 36.4 GB, 36400267264 bytes
254 heads, 63 sectors/track, 4442 cylinders
Units = cylinders of 16002 * 512 = 8193024 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1	*	1	366	2928334+	83	Linux
/dev/sda2		367	428	496062	82	Linux swap
/dev/sda3		429	4442	32116014	83	Linux

A példában bemutatott, mintegy 2,8 gigabájtos lemezrész képes hordozni azokat a programokat, amelyekre egy irodai számítógépen szükségünk lehet.

5. példa A következő példa egy kiszolgáló lemezrészéinek elrendezését mutatja be. Itt már jóval gondosabban kell eljárnunk, hiszen a számítógépet több ezer felhasználó használja.

```
Fájlrendszer      Méret  Fogl. Szab.  % Csatl. pont
/dev/hda1          251M   74M  165M  31% /
```

/dev/hda2	31M	9.6M	20M	34%	/boot
/dev/hda5	251M	18K	238M	1%	/tmp
/dev/hda6	251M	28M	210M	12%	/var
/dev/hda7	4.0G	453M	3.4G	12%	/usr
/dev/hda8	14G	5.6G	6.9G	45%	/home
/dev/hde1	57G	667M	54G	2%	/home1
/dev/hde2	56G	3.3G	50G	7%	/oracle

A példa első sorában láthatjuk a gyökérkönyvtár számára fenntartott lemezrészét, amely meglehetősen kis méretű, mivel az alkalmazásokat szétoztották a további lemezrészek között.

A második sor a /boot/ könyvtár tartalma számára van fenntartva. Ebben a könyvtárban a rendszerindításhoz szükséges állományok helyezkednek el.

A következő lemezrész a /tmp/ könyvtár tartalmát hordozza. A rendszerindítás első szakaszában – amikor ez a lemezrész még nincs beillesztve – az ideiglenes állományok a gyökérkönyvtárat tartalmazó lemezrészre kerülnek, később azonban – a /tmp/ könyvtár számára készült lemezrész beillesztése – egy külön lemezrészre. Ez bizonyos védettséget jelent, hiszen a /tmp/ könyvtár esetleges túlterhelése esetén a gyökérkönyvtárat tartalmazó lemezrész nem telik meg.

A következő lemezrész a /var/ könyvtár számára van fenntartva. Az itt tárolt adatok közül a legfontosabbak a naplóállományok és a felhasználók levelesláfai. A példában szereplő néhányszor tíz megabájtnyi terhelésnél sokkal több is felléphet ezen a lemezrészen, ha a helyi szabályok a felhasználóknak nagyobb levelesládat is megengednek vagy részletesebb naplázási beállítások vannak érvényben.

A /usr/ könyvtár – a felhasználói programok – száma ára a példában néhány gigabájt van fenntartva, de ebből csak viszonylag keveset használunk fel. A tapasztalat azt mutatja, hogy 4-6 gigabájt még akkor is elegendő, ha igen sok alkalmazást telepítünk a számítógépre.

A felhasználók adatai a példában a /home/ és a /home1 könyvtárban vannak. Ha sok felhasználói adatot akarunk a kiszolgálón tárolni, használhatjuk ezt a megoldást az adatok elosztására. A felhasználókat csoportokba osztjuk, az egyes csoportok adatait pedig ekkor más-más merevlemezen tároljuk, így érve el a folyamatos bővítési lehetőséget. A felhasználók számának folyamatos növekedése esetén újabb és újabb felhasználói csoportokat hozunk létre, újabb és újabb merevlemezt vásárolunk.

A példa utolsó sora, az /oracle/ könyvtár az adatbázis-kezelő rendszernek ad helyet. Erre nem minden számítógépen van szükség, ott azonban, ahol adatbázis-kiszolgálót üzemeltetünk, szerencsés az adatokat külön lemezrészen tárolni.

A példában bemutatott szerkezetet a következő lemezrészekkel valósítottuk meg:

```
$ fdisk -l /dev/hda
Disk /dev/hda: 20.0 GB, 20020396032 bytes
255 heads, 63 sectors/track, 2434 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1		1	33	265041	83	Linux
/dev/hda2		34	37	32130	83	Linux
/dev/hda3		38	103	530145	82	Linux swap
/dev/hda4		104	2434	18723757+	5	Extended
/dev/hda5		104	136	265041	83	Linux
/dev/hda6		137	169	265041	83	Linux
/dev/hda7		170	692	4200966	83	Linux
/dev/hda8		693	2434	13992583+	83	Linux

```
$ fdisk -l /dev/hde
Disk /dev/hde: 122.9 GB, 122942324736 bytes
16 heads, 63 sectors/track, 238216 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hde1		1	120000	60479968+	83	Linux
/dev/hde2		120001	238216	59580864	83	Linux

Figyeljük meg, hogy a sok lemezrész miatt kibővített és logikai lemezrészek használatára is szükség volt!

3. fejezet

Állományrendszerek

A merevlemezen a lemezrészek kialakítása után még nem tárolhatunk adatokat, azokon előbb még létre kell hoznunk valamilyen állományrendszeret (*filesystem*), a lemezrész „formáznunk” kell.

Az állományrendszer egy szabványos adatszerkezet, amely meghatározza, hogy az adatokat és az adatok visszakereséséhez szükséges adminisztratív információkat milyen formában tároljuk a háttértáron. Sokféle szabvány létezik, sokféleképpen tárolhatjuk az adatokat a háttértáron, és ezek közül a Linux sokat támogat. E fejezetben a legfontosabb támogatott állományrendszereket vesszük sorra.

Az állományrendszerek tárgyalása előtt azonban néhány szót kell szólnunk a lemezgyorsítótárról. A lemezgyorsítótár a számítógép sebességét nagymértékben növeli azáltal, hogy a háttértárokra írt és az onnan olvasott adatok egy részét a memóriában megőrzi. A következő olvasási művelet így sokkal gyorsabb lesz – hiszen az adatok már a memóriában vannak –, az írást pedig el lehet halasztani, az írást kérő folyamat azonnal futhat tovább.

A gyorsítótár a háttértárkezelés legalsós szintjén helyezkedik el, a háttértárkezelést közvetlenül végző programrészeken. Ez lehetővé teszi, hogy a lemezgyorsítótárat a folyamatok, felhasználók közösen használják. Ha például egy felhasználó elindított egy programot, a következő felhasználónak már sokkal gyorsabban indul el, hiszen megtalálható a lemezgyorsítótárban.

A lemezgyorsítótár méretével kapcsolatban meg kell jegyeznünk, hogy a Linux minden rendelkezésre álló memóriát megpróbál lemezgyorsítótárként felhasználni. Ha egy folyamat memóriát kér a Linuxtól, a lemezgyorsítótár mérete csökken. Ha valamelyik folyamat memóriát szabadít fel, a lemezgyorsítótár növekedhet. Ha tehát azt látjuk, hogy a memória teljes egészé foglalt, az általában nem arra utal, hogy a számítógép túl van terhelve, hanem arra, hogy a lemezgyorsítótár a teljes rendelkezésre álló memóriaterületet használja.

A lemezgyorsítótár használata során a háttértárra történő írás nem azonnal törtenik meg. A Linux ugyan visszajelzi az írást kérő folyamat felé, hogy az írás megvolt, de az adatok ilyenkor még általában csak a lemezgyorsítótárba kerültek

3.1. tábla: sync

A lemezgyorsítótár mentése a háttértárra.

`sync`

A program az íráskor használt lemezgyorsítótárat fizikailag is a háttértárra írja, az íráskor használt lemezgyorsítótárat pedig üríti.

be. Ha például a számítógépet kikapcsoljuk, az adatok – amelyekről a folyamat úgy tudja, hogy biztonságban vannak a merevlemezen – elvesznek.

A felhasználók és a folyamatok bármikor kérhetik, hogy az íráskor használt lemezgyorsítótárat a Linux a háttértárra írja. Ha a gyorsítótár még nem mentett területeit a háttértárra szeretnénk írni, a `sync` programot használhatjuk, de az egyes programok is képesek ezt kérni az írás során.

3.1. Az extended 2 állományrendszer

A Linux számára több saját állományrendszert is kifejlesztettek, amelyek közül az egyik legelterjedtebb az *extended 2 filesystem* (2. továbbfejlesztett állományrendszer). Az extended 2 állományrendszer a Unix rendszerek körében elterjedt mutatópont felépítést használja, amely több mint harminc éves múltra tekint vissza.

3.1.1. Az extended 2 állományrendszer felépítése

Az extended 2 állományrendszer hagyományos, Unix szerkezetű, mutatópont alapú állományrendszer.

Az *inode* a mutatópont szóra utal, a könyvtárbejegyzésekhez tartozó adatszerkezet neve. minden könyvtárbejegyzéshez tartozik egy – és csak egy – mutatópont, amely a könyvtárbejegyzés legfontosabb adatait hordozza. A mutatópontban tárolt legfontosabb adatok a következők:

sorszám minden mutatópont rendelkezik egyedi azonosítószámmal, amely egyértelműen azonosítja az állományrendszeren belül.

típus megadja a könyvtárbejegyzés típusát (könyvtár, szabályos állomány stb.).

mutatók száma A mutatók száma azt határozza meg, hogy az adott könyvtárbejegyzéshez hány név tartozik.

tulajdonos minden könyvtárbejegyzéshez tartozik egy felhasználó, aki a könyvtárbejegyzés tulajdonosa. A mutatópont tárolja a könyvtárbejegyzéshez tartozó tulajdonos felhasználói azonosítószámát, az UID-t.

3.2. tábla: stat

A program segítségével a könyvtárbejegyzések legfontosabb tulajdonságait ismerhetjük meg.

stat [kapcs] könyvtárbejegyzés

Kapcsoló Jelentés

- | | |
|----|------------------------------------|
| -F | Közvetett hivatkozások követése. |
| -t | Az adatok gépi formájának kiírása. |

tulajdonoscsoport minden könyvtárbejegyzéshez tartozik egy felhasználói csoport, a tulajdonoscsoport. A mutatópont tárolja a könyvtárbejegyzéshez tartozó tulajdonoscsoport azonosítószámát, a GID-t.

hozzáférési jogok A hozzáférési jogok meghatározzák, hogy a tulajdonos, a tulajdonoscsoportba tartozó felhasználók és a többi felhasználó milyen jogokkal rendelkezik a könyvtárbejegyzés felett.

méret A könyvtárbejegyzés mérete.

időbeliyegek A mutatópont tárolja az állomány utolsó módosításának pontos idejét, azt, hogy mikor használtuk utoljára az állományt, és azt is, hogy az állomány tulajdonságait – magát a mutatópontot – mikor módosították utoljára.

adatblokkok A könyvtárbejegyzés tartalmát hordozó adatblokkokat kijelölő mutatók. A mutatók közül néhány nem az állomány tartalmát hordozó adatblokkokat jelöl ki, hanem olyan adatblokkokat, amelyek mutatókat tartalmaznak a tartalmat hordozó adatblokkokra, így igen nagy méretű állományokat is képes kezelní a rendszer.

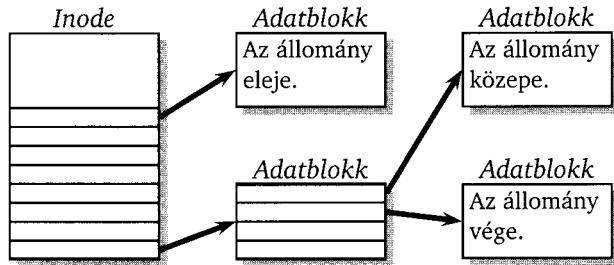
A mutatópont és az adatblokkok elrendezését a 3.1. ábra mutatja be. Az ábrán csak néhány adatblokkot jelöltünk a helytakarékosság érdekében.

A könyvtárbejegyzésekhez tartozó mutatópontokban tárolt legfontosabb adatokat az ls és a stat programok segítségével tudhatjuk meg.

6. példa Írassuk ki a / könyvtár mutatópont azonosítóját a ls -i kapcsolójával!

```
$ ls -lid /
2 drwxr-xr-x 21 root root 4096 nov 1 08:04 /
$
```

A gyökérkönyvtár azonosító 2, amely a program által kiírt táblázat első oszlopában található.



3.1. ábra. Az adatblokokat jelölő mutatók

7. példa Írassuk ki a / könyvtár legfontosabb adatait a stat program segítségével!

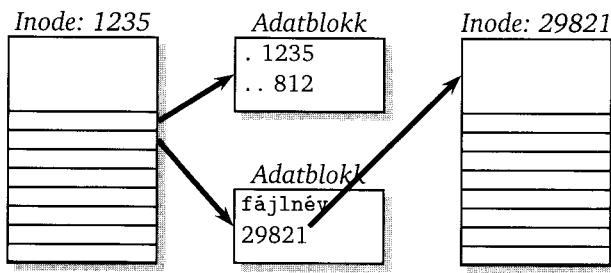
```
$ stat /
  File: '/'
  Size: 4096          Blocks: 8      IO Block: 4096 Directory
Device: 301h/769d     Inode: 2       Links: 21
Access: (0755/drwxr-xr-x)  Uid: (      0/root) Gid: (  0/root)
Access: 2003-11-01 08:22:19.000000000 +0100
Modify: 2003-11-01 08:04:48.000000000 +0100
Change: 2003-11-01 08:04:48.000000000 +0100
$
```

Láttuk, hogy a mutatópont nem tartalmazza a könyvtárbejegyzés nevét. Az adott mutatóponthoz tartozó könyvtárbejegyzés nevét az a könyvtár tartalmazza, amelyben a könyvtárbejegyzés található. A következő részből megtudhatjuk, milyen felépítés alapján tárolja az extended 2 állományrendszer a könyvtárakat.

Ha a könyvtárbejegyzés könyvtár, a mutatópont által kijelölt adatblokkok tartalma formalag kötött. A 3.2. ábra ezt az esetet mutatja be. Amint láthatjuk, a könyvtárat tároló adatterület egy listát tárol, amely az állományok neveit és a hozzájuk tartozó mutatópontok számát tartalmazza. A könyvtár tehát egy különleges könyvtárbejegyzés, amely könyvtárbejegyzések listáját tartalmazza.

A könyvtárakban már a létrehozásukkor megtalálható két könyvtárbejegyzés: a . a saját mutatópontra mutat, míg a .. a szülőkönyvtár mutatóponjára. Azt a könyvtárat tekinthetjük üresnek, amely csak ezt a két könyvtárbejegyzést tartalmazza.

A könyvtárbejegyzések minden könyvtárbejegyzéshez egyérelműen hozzárendelünk egy mutatópont számot. Fordítva azonban már nem egyértelmű a hozzárendelés: egy mutatópontra több könyvtárbejegyzés hivatkozhat, egy könyvtárbejegyzésnek pedig több neve is lehet. Ha egy könyvtárbejegyzésnek több neve van, akkor közvetlen hivatkozásról beszélünk.



3.2. ábra. Könyvtár

Ha figyelembe vesszük, hogy a könyvtár állományok listáját tartalmazza azokban a blokkokban, ahol a szabályos állományok az adatokat, világossá válik, hogy hogyan értelmezzük az alkönyvtárakra vonatkozó jogosultságokat, tulajdonosokat, időbelyeget és méretet. A könyvtár-állomány megkülönböztetés tehát a kezdő felhasználó számára hasznos lehet, a rendszergazdának azonban tudnia kell, hogy a szabványos kezelés érdekében kialakított egységes szerkezetről van szó. Ez igen fontos, mivel segíti a Unix állományrendserek tulajdonságainak mélyebb megértését.

3.1.2. Közvetett és közvetlen hivatkozások

A könyvtárak adatterületein tehát hivatkozást találunk, amely összeköti a könyvtárbejegyzés nevét a mutatóponttal, amely a könyvtárbejegyzés adatait tartalmazza és kijelöli a könyvtárbejegyzéshez tartozó adatterületeket. minden névhez tartozik tehát egy mutatópont.

Egy mutatóponthoz azonban nem csak egy név tartozhat. Az extended 2 állományrendszer lehetővé teszi, hogy több név is jelölje a mutatópontot. Ezt az elrendezést mutatja be a 3.3. ábra. Figyeljük meg, hogy az elrendezés szerint a könyvtárbejegyzésnek több neve van! Azt mondjuk ilyenkor, hogy az állományra közvetlen hivatkozást hoztunk létre.

A közvetlen hivatkozás nem egy különleges tulajdonságú könyvtárbejegyzés, egyszerűen csak azt az szerkezetet jelöljük ezzel a kifejezéssel, ahol ugyanarra a mutatópontra több állománynév is hivatkozik.

Figyeljük meg az ábrán, hogy a könyvtárbejegyzések listájában tároljuk ugyan a hivatkozott mutatópont számát, a mutatópontban azonban nem tároljuk azt, hogy melyik nevek hivatkoznak rá. A mutatópont tartalmaz egy számlálót, amely megmutatja, hogy az állományrendszerben hány neve van az adott könyvtárbejegyzésnek, de hogy ezek mely könyvtárakban vannak, azt nem lehet kideríteni a mutatópont vizsgálatával. A könyvtárbejegyzés esetleges további neveit csak az állományrendszer végigkeresésével lehet megtalálni. Erre a `find` program használható, ahogyan a következő példában láthatjuk.

8. példa Ha kiíratjuk az `ls` parancs segítségével a `/bin/gzip` állomány adatait, azt találjuk, hogy az állománynak több neve is létezik:

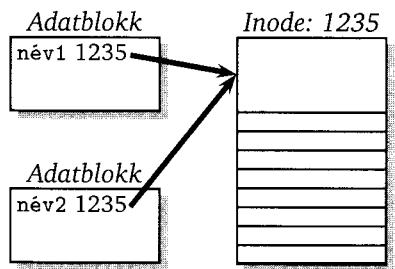
3.3. tábla: find

Könyvtárbejegyzések keresését végző program.

`find [kiindulási könyvtár] [keres] [művelet]`

A program a parancssorban megadott könyvtár teljes tartalmát sorra véve adott tulajdonságokkal rendelkező könyvtárbejegyzéseket keres. A megtalált könyvtárbejegyzésekben különféle műveleteket képes elvégezni. A program számára a számokat megadhatjuk előjel nélkül, a pontos egyezéshez, vagy $+n$ (nagyobb, mint n) és $-n$ (kisebb, mint n) alakban is.

<i>Keres</i>	<i>Jelentés</i>
<code>-maxdepth mély</code>	A keresést csak az adott mélységgig folytatja.
<code>-mount</code>	A keresés közben nem hagyja el az állományrendszert.
<code>-amin n</code>	Az állományt n perccel ezelőtt használták utoljára.
<code>-atime n</code>	Az állományt n nappal ezelőtt használták utoljára.
<code>-cnewer fájl</code>	Az állomány újabb, mint a megadott állomány.
<code>-ctime n</code>	Az állományt n nappal ezelőtt módosították utoljára.
<code>-gid g</code>	Az állomány tulajdonosának azonosítója g .
<code>-group név</code>	Keresés a tulajdonoscsoporthoz tartozó névvel.
<code>-uid u</code>	Az állomány tulajdonosának azonosítója u .
<code>-user név</code>	Keresés a tulajdonos nevével.
<code>-inum i</code>	A fájl mutatópont száma i .
<code>-lname fájl</code>	A fájl az adott állományra mutató közvetett hivatkozás.
<code>-mtime n</code>	A fájl tartalmát n nappal ezelőtt módosították utoljára.
<code>-mmin n</code>	A fájl tartalmát n perccel ezelőtt módosították utoljára.
<code>-name minta</code>	Az állomány neve illeszthető a megadott mintára.
<code>-nouser</code>	A tulajdonos nem létezik.
<code>-nogroup</code>	A tulajdonoscsoporthoz tartozó névvel.
<code>-and</code>	Keresési feltételek „logikai és” kapcsolattal.
<code>-or</code>	Keresési feltételek „logikai vagy” kapcsolattal.
 <i>Művelet</i>	 <i>Jelentés</i>
<code>-exec parancs</code>	A parancs végrehajtása. A parancs végét a ; jel, a megtalált fájl nevét a {} kifejezés jelzi.
<code>-ok parancs</code>	Ugyanaz, mint a -exec, de minden végrehajtás előtt megerősítést kér a felhasználótól.
<code>-print</code>	A talált állomány nevénak kiíratása.



3.3. ábra. Közvetlen hivatkozás

```
$ ls -li /bin/gzip
409140 -rwxr-xr-x 3 root root 53716 jan 31 2003 /bin/gzip
```

A második oszlop szerint ennek az állománynak három neve is található az állományrendszerben. A *find* parancs segítségével megkereshetjük, hol található az állomány többi neve:

```
$ find /bin/ -inum 409140 -exec ls -li \{\} \;
409140 -rwxr-xr-x 3 root root 53716 jan 31 2003 /bin/gunzip
409140 -rwxr-xr-x 3 root root 53716 jan 31 2003 /bin/gzip
409140 -rwxr-xr-x 3 root root 53716 jan 31 2003 /bin/zcat
$
```

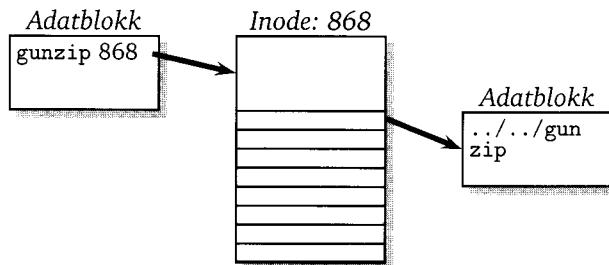
A *find -inum* kapcsolója azt jelzi a program számára, hogy az adott mutatópont számhoz tartozó neveket keressük.

A példában a *find -exec* kapcsolójával végrehajtattuk az *ls* parancsot, az *ls -i* kapcsolójával pedig kiírtattuk a könyvtárbejegyzéshez tartozó mutatópont számot is. Jól látható, hogy ugyanarról az állományról van szó minden név esetében.

A közvetlen hivatkozások bemutatott szerkezetéből könnyen következtethetünk azok alaptulajdonságaira.

A közvetlen hivatkozás több nevet rendel egy könyvtárbejegyzéshez. Ezek a nevek egyenértékűek. Teljesen mindegy, hogy melyik nevet hoztuk létre előbb és melyiket később, a nevek nem különböznek egymástól. A mutatópont által jelzett adatterület akkor szabadul fel, amikor az utolsó nevét is megszüntettük, azaz törlöttük a könyvtárszerkezetből. Amíg egyetlen név is létezik, az adatok elérhetők.

A nevek ugyanarra a mutatópontra mutatnak, ezért minden név ugyanazt az állományt jelzi. A könyvtárbejegyzéshez tartozó jogok, a tulajdonos, az utolsó módosítás dátuma mind olyan jellemzők, amelyek a mutatópontban tárolódnak, ezért ezek a tulajdonságok minden név esetében megegyeznek. Ha a könyvtárbejegyzést módosítjuk az egyik név felhasználásával, a többi néven elérhető könyvtárbejegyzés is megváltozik.



3.4. ábra. Közvetett hivatkozás

Az adatok csak egy helyen vannak elhelyezve, ezért a könyvtárbejegyzés csak annyi helyet foglal, amekkorát akkor foglalna, ha csak egy neve volna. (A további nevek tárolására használt lemezterület azoknak a könyvtáraknak a méretét növeli elhanyagolható mértékben, amelyekhez tartoznak.)

A közvetlen hivatkozások a mutatópont számát használják azonosításra. Mivel a mutatópont száma csak az állományrendszeren belül alkalmas azonosításra, közvetlen hivatkozást csak állományrendszeren belül hozhatunk létre.

Annak érdekében, hogy a könyvtárszerkezetben ne legyenek hurkok – a programok ne „tévedjenek el” –, a Linux nem teszi lehetővé könyvtárakra mutató közvetlen hivatkozások létrehozását. Tehát minden könyvtárnak csak egy neve lehet. (Ez alól kivételezett a rendszer által létrehozott . és .. könyvtárbejegyzések jelentenek, amelyek másodlagos nevei a könyvtárnak és a szülőkönyvtárnak.)

A közvetlen hivatkozások használatára bevezetett két korlátozás szerint tehát csak állományrendszeren belül lehet őket használni és könyvtárak esetében még ott sem. A közvetett hivatkozásokat éppen azért vezették be, hogy ezeket a korlátozásokat kivédhessük a segítségükkel.

A közvetett hivatkozások különleges könyvtárbejegyzések, amelyek más könyvtárbejegyzések nevét és elérési útját tartalmazzák az adatterületükön. Amikor a Linux egy közvetett hivatkozást nyit meg, nem az adatterületet nyitja meg, hanem az adatterület által leírt más könyvtárbejegyzést. Ezt az elrendezést mutatja be a 3.4. ábra.

A közvetett hivatkozásra nem érvényesek a közvetlen hivatkozásra bevezetett korlátozások. Közvetett hivatkozás mutatható állományrendszeren kívülre is, hiszen egy könyvtárbejegyzés nevét tartalmazza, amely bármely beillesztett állományrendszeren lehet.

A közvetett hivatkozás ráadásul mutatható könyvtár típusú könyvtárbejegyzésre is. Emiatt az állományrendszerben hurkok alakulhatnak ki, de ezek ellen a Linux egyszerűen tud védekezni. Ha egy könyvtárbejegyzés megnyitásakor „túlságosan sok” közvetett hivatkozáson kell áthaladnia, egyszerűen úgy dönt, hogy hurok van az állományrendszerben, és hibajelzéssel megtagadja a további keresést.

Figyeljük meg a 3.4. ábrán, hogy a közvetett hivatkozás tartalmazza ugyan a hivatkozott könyvtárbejegyzés nevét és elérési útját, a hivatkozott könyvtárbejegy-

zés azonban nem tartalmaz információt arról, hogy rá közvetett hivatkozás mutat. Ha meg akarjuk tehát tudni, hogy egy könyvtárbejegyzésre mely közvetett hivatkozások mutatnak, a teljes könyvtárszerkezetet végig kell keresnünk. Erre a feladatra a `find` program használható, ahogyan a következő példa bemutatja.

9. példa Azt gyanítjuk, hogy a `/usr/` könyvtárszerkezetben létezik közvetett hivatkozás a `gzip` parancsra. Keressük meg a hivatkozást és írassuk ki!

```
$ ls -l $(find /usr -lname "*gzip")  
lrwxrwxrwx 1 root root 14 júl 10 15:27 /usr/bin/gzip ->  
.../bin/gzip
```

Amint látható, közvetett hivatkozások felkutatására a `find` parancs `-lname` kapcsolója használható.

A közvetett hivatkozás szerkezetének ismeretében könnyen megérhetjük annak alapvető tulajdonságait.

Közvetett hivatkozást az állományrendszeren kívülre hivatkozva is létrehozhatunk.

A követett hivatkozás és a hivatkozott könyvtárbejegyzés nem egyenrangú. A közvetett hivatkozást törölve az eredeti könyvtárbejegyzés elérhető marad ugyan, de ez fordítva nem igaz. Ha egy közvetett hivatkozás által jelzett könyvtárbejegyzést törlünk, az adatok elvesznek, a közvetett hivatkozás pedig egy nem létező helyre mutat. Azt mondjuk ilyenkor, hogy a hivatkozás megszakadt, érvénytelen helyre mutat. Az ilyen hivatkozásokat nem lehet megnyitni és használni, hiszen az adatok elvesztek.

A közvetett hivatkozásokhoz rendelt hozzáférési jogok lényegtelenek, hiszen a használati jogot a hivatkozott könyvtárbejegyzéshez rendelt jogok döntik el.

3.1.3. A foglalt lemezterület

Az állományrendszerben az állományok nem akkora helyet foglalnak, amekkora a méretük. Az utolsó foglalt adatblokkban fellép valamekkora veszteség, ha az állomány mérete nem egész számú többszöröse a blokkméretnek, mivel a könyvtárbejegyzéseknek minden adatblokk elején kell kezdődniük.

Az extended 2 állományrendszer adatblokkjainak mérete az állományrendszer létrehozásakor beállítható 1024, 2048 vagy 4096 bájtosra. Ha az állományrendszerben sok kisméretű állományt fogunk tárolni, szerencsés kis blokkméretet használnunk, hogy minél kevesebb legyen a veszteség. Másrészről viszont nagy állományokat nem szerencsés kis blokkméretű állományrendszerben tárolni, mivel a sok apró blokk betöltése lassítja az állományrendszer használatát.

10. példa A következő példa bemutatja, hogy a `tar` programmal egy állományba másolt könyvtárszerkezet kevesebb helyet foglal a lemezen, mint az eredeti könyvtárszerkezet, mivel a veszteség jóval kevesebb.

```
$ tar cf examples.tar examples/
$ du -hs examples*
568K    examples
156K    examples.tar
$
```

Látható, hogy a veszteség miatt a tároláshoz szükséges hely több mint háromszorosára növekedett. Ez természetesen csak sok kicsi állomány esetében fordulhat elő.

3.1.4. A megnyitott állományok

Amikor egy program használni kívánja valamelyik állományt, megnyitja azt. A program a megnyitáskor jelzi, hogy milyen módon fogja használni azt az állományt, olvasni vagy írni szeretné, esetleg olvasni és írni is.

A Unix rendszerek az alkalmazások által megnyitott állományokat a mutatópont számuk alapján tartják nyilván és kezelik. A könyvtárbejegyzés mutatópont számhoz való hozzárendelése az állomány megnyitása során történik, utána már a mutatópont alapján történik az állomány kezelése, függetlenül attól, hogy az állománynak megváltozott-e a neve, esetleg átkerült egy másik könyvtárba a rá való hivatkozás.



Ha egy alkalmazás megnyitott egy állományt írásra vagy olvasásra, és ezt az állományt egy másik program állományrendszeren belül áthelyezi vagy átnevezi, az állomány írása vagy olvasása annak ellenére sem hiúsul meg, hogy az állomány az eredeti néven már nem érhető el.

Az állomány hozzáférési jogait a rendszer az állomány megnyitásakor ellenőrzi. Ha a megnyitás sikeres volt, a Linux a hozzáférést engedélyezi írásra vagy olvasásra addig, amíg az állományt a program megnyitva tartja. Nyilvánvalóan takarékos megoldás ez, hiszen az állomány olvasása vagy írása során nem kell ellenőrzéseket végezni, de az egyszerűsítés azt eredményezi, hogy a megnyitás után módosított hozzáférési jogok hatástalanok maradnak.



Ha megváltoztatjuk egy állomány hozzáférési jogait, azok a folyamatok, amelyek már megnyitották az állományt, a régi jogokkal kezelhetik azt. A folyamatok, amelyek a könyvtárbejegyzést megnyitották, addig használhatják azokat, amíg le nem zájják őket, vagy be nem fejezik futásukat. A rendszergazda természetesen bármelyik folyamatot megszakíthatja.

A rendszergazdának tudnia kell tehát, hogy ha valamilyen állományjogot elvesz, azok a folyamatok, amelyek már megnyitották az állományt, zavartalanul folytathatják annak használatát.

3.1.5. A tulajdonos nyilvántartása

Mivel a mutatópont nem a tulajdonos és tulajdonoscsoporthoz tartozik, hanem azonosítószámát tartalmazza, az állománykezelő programoknak használniuk kell a felhasználói adatokat tartalmazó felhasználói adatbázist. Az `ls` parancs például a felhasználói adatbázisból kérdezi le a könyvtárbejegyzések tulajdonosainak és tulajdonoscsoporthoz tartozó nevét, így képes azokat a könyvtárbejegyzések listájába írni.

Ha egy másik GNU/Linux rendszerre másolunk át állományokat, előfordulhat, hogy azok nem az eredeti tulajdonos tulajdonában maradnak, hanem annak a felhasználónak a tulajdonába kerülnek, akinek az azonosítója megegyezik az eredeti tulajdonosnak az eredeti rendszeren található azonosítószámával. Ha több gépet használó Unix/Linux rendszert üzemeltetünk, mindenkorban javasolt a felhasználói adatbázisok egységesítése, hogy az állományok jellemzői között tárolt felhasználói és csoportazonosító minden számítógépen ugyanazt a felhasználót és csoportot jelentse. E felhasználói azonosítószámok egységesítésére több programrendszer is használható (NIS, NIS+ stb.) [14].

Előfordulhat az is, hogy a mutatópont olyan tulajdonosazonosítót tartalmaz, ami nem szerepel a felhasználói adatbázisban. Ilyen probléma általában a felhasználók törlése után fordul elő.

Ha a felhasználót vagy tulajdonoscsoportot töröltük a rendszerből, akkor a mutatópont olyan felhasználónak a tulajdonában van, aki nem is létezik. Ha később egy új felhasználó létrehozásakor az azonosítót újra kiosztja a rendszer, az újonnan létrehozott felhasználó birtokába kerülnek ezek az állományok, ő kezelheti őket, az ő lemezkvótáját terhelik. Ennek elkerülése érdekében a rendszergazdának fontos lehet, hogy a háttértárakat átnézve felkutassa a „gazdátlan” állományokat.

A tulajdonos, illetve tulajdonoscsoport nélkül maradt állományok felkutatására a `find` parancs használható a `-nouser`, illetve `-nogroup` kapcsolóval.

11. példa A következő példa bemutatja, hogyan kutathatjuk fel a gazdátlanul maradt könyvtárbejegyzéseket az állományrendszerben.

```
$ find /home/ -nouser -print
/home/fred
/home/fred/.bashrc
$ ls -ld /home/fred
drwx----- 4 501 502 4096 júl 14 17:39 /home/fred
$
```

Amint láthatjuk, a `/home/fred/` könyvtár tulajdonosa már nincs nyilvántartva a felhasználói adatbázisban. Ezt az `ls` parancs használatakor is megfigyelhetjük, hiszen a listában a felhasználó nevének helyén csak az azonosítószám található.

3.1.6. Az extended 2 állományrendszer kezelése

Ha létrehoztunk egy lemezrész, amelynek típusa 0x82, abban extended 2[4] állományrendszert hozhatunk létre. (Természetesen más típusú lemezrészben is létrehozhatunk extended 2 állományrendszeret, vagy akár a hajlékonylemezünket is használhatjuk ilyen állományrendszerrel, a hagyomány azonban az, hogy a 0x82 kódú lemezrészeken használunk extended 2 állományrendszert.)

Az általunk létrehozott lemezrészben adatokat tárolni csak akkor tudunk, ha rajta állományrendszert hozunk létre. Habár a ma használt hajlékonylemezeket általában már a gyárban ellátják állományrendszerrel, azokra is igaz, hogy csak állományrendszerrel ellátva alkalmasak állományok tárolására.



Az állományrendszer létrehozásakor az adathordozón található adatok elvesznek. Ha tehát állományrendszert hozunk létre valamely lemezrészben vagy hajlékonylemezen, ügyelnünk kell, hogy az esetleg ott tárolt adatokat előtte mentsük, ellenkező esetben elvesznek.

Extended 2 állományrendszert létrehozni az `mke2fs` program segítségével lehet. (A program a legtöbb GNU/Linux rendszeren elérhető az `mkfs.ext2` néven is.) Az `mke2fs` program számára annak a blokkeszköz-meghajtó állománynak a nevét kell megadnunk, amely kijelöli az adathordozót – lemezrész vagy hajlékonylemez –, amelyen az állományrendszert létre akarjuk hozni. Az állományrendszer létrehozása a blokkeszköz-meghajtó írásával történik, az állományrendszer létrehozásához tehát írási jogra van szükségünk az adott blokkeszköz-meghajtó állományra.

12. példa Legyen a `/dev/sda` egy hordozható adattár, amelyen létrehoztunk egy elsődleges lemezrész. Hozzunk létre a lemezrészben extended 2 állományrendszer úgy, hogy a rendszergazda számára a hibajavításra ne foglaljunk le adatblokkokat!

```
$ mke2fs -m 0 /dev/sda1
mke2fs 1.32 (09-Nov-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
32000 inodes, 127975 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
16 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
          8193, 24577, 40961, 57345, 73729
```

3.4. tábla: mke2fs, mkfs.ext2

Extended 2 és extended 3 állományrendszer létrehozására használható program.

mke2fs [kapcs] [blokköz-meghajtó]

A program létrehoz egy extended 2 állományrendszert a paraméterként átadott meghajtó által meghatározott lemezrészben. A program kapcsolóival a létrehozott állományrendszer finomhangolható.

Kapcsoló	Jelentés
-b m	Az állományrendszerben használt adatblokkméret megadása.
-c	Hibás blokkok keresése a lemezrészben. Ha kétszer adjuk meg a kapcsolót, a háttértár ellenőrzése írást is magában foglal.
-i n	A létrehozandó mutatópontok számának meghatározása. A program minden n bájt számára egy mutatópontot hoz létre.
-m f	A rendszergazda számára előre lefoglalt blokkok számának megadása százalékos formában. Alapértelmezés szerint az adatblokkok 5%-a a rendszergazda számára van lefoglalva.
-q	Kevés üzenetet használó üzemmód.

```
Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
```

```
This filesystem will be automatically checked every 21 mounts
or 180 days, whichever comes first. Use tune2fs -c or -i to
override.
$
```

Figyeljük meg, hogy a program 1024 bájtosra választotta a blokkmérétet, így 127 975 adatblokkot hozott létre 32 000 mutatóponttal. Ez átlagosan 3 kilobájtos állománymréretet tesz lehetővé. Ha ennél lényegesen nagyobb vagy kisebb a tervezett átlagos állománymréret, a program megfelelő kapcsolóival újra létre kell hoznunk az állományrendszert az optimális működés érdekében.

Az mke2fs által létrehozott állományrendszer néhány tulajdonsága a későbbieken megváltoztatható, de a blokkmérét és a mutatópontok száma nem. Gondosan kell tehát megválasztanunk ezeket az értékeket, mert csak az állományrendszer létrehozásakor állíthatjuk be őket. A program által választott értékek általában megfelelnek az átlagos igényeknek, csak akkor érdemes eltérnünk az alapértéktől, ha komoly okunk van rá.

Az extended 2 állományrendszer megváltoztatható tulajdonságait a tune2fs program segítségével állíthatjuk be.

Az állományrendszeren a használat során hibák jelenhetnek meg. Okozhatja ezeket hibás háttértár, áramszünet, a számítógép váratlan (megfelelő leállítás nélküli) kikapcsolása vagy programhiba is.

13. példa Azt szeretnénk, ha a /dev/sda1 lemezrészén létrehozott állományrendszer logikai hibája esetén a Linux csak olvasható módon illesztené be újra azt. Ez az állományokat elérhetővé teszi hiba után is, de megakadályozza, hogy a sértült állományrendszer módosításával halmozódjanak a hibák. A feladatra a tune2fs programot használhatjuk.

```
$ tune2fs -e remount-ro /dev/sda1
tune2fs 1.32 (09-Nov-2002)
Setting error behavior to 2
$ tune2fs -l /dev/sda1 | grep Error
Errors behavior:           Remount read-only
$
```

Amint láthatjuk, a program elvégezte a kívánt módosítást.

Az extended 2 állományrendszerben jelentkező logikai hibák kijavítása az e2fsck program segítségével történhet. A program megvizsgálja, hogy az állományrendszeren jelentkeztek-e hibák a használat során, és ha igen, azokat összefűtött módszerek segítségével megkíséri kijavítani.

14. példa Végezzük el a /dev/sda1 lemezrészén található extended 2 állományrendszer ellenőrzését, hibajavítását! Erre a program a következő formában használható:

```
$ e2fsck /dev/sda1
e2fsck 1.32 (09-Nov-2002)
/dev/sda1: clean, 22/32000 files, 25814/127975 blocks
$
```

Amint láthatjuk, a program úgy találta, hogy az állományrendszeren nem találhatók hibák, ezért a részletes ellenőrzést nem kell lefolytatni. Ha mindenkiéppen el akarjuk végezni a részletes ellenőrzést, a program -f kapcsolójával megtehetjük:

```
$ e2fsck -f /dev/sda1
e2fsck 1.32 (09-Nov-2002)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/sda1: 22/32000 files (0.0% non-contiguous), 25814/127975
blocks
$
```

3.5. tábla: tune2fs

Extended 2 és extended 3 állományrendszerek tulajdonságainak beállítása.

tune2fs [kapcs] [blokkész köz-meghajtó]

A program segítségével az extended 2 és extended 3 állományrendszerek néhány tulajdonsága használat közben megváltoztatható. Más tulajdonságok megváltoztatásához az állományrendszert újra létre kell hozni.

Kapcsoló	Jelentés
-c n	A kapcsoló után megadható, hogy hány beillesztés után végezzen a rendszer teljes állományrendszer-ellenőrzést. Tiltás 0 értékkel lehetséges.
-e vis.	A kapcsoló után megadható, hogy mit tegyen a Linux, ha működés közben állományrendszer-hibát talál. A kapcsoló után megadható continue (folytatás), remount-ro (újraillesztés csak olvashatóan) vagy panic (azonnali rendszerleállás).
-i idő	A kapcsoló után megadható, hogy mennyi időnként végezzen a rendszer teljes állományrendszer-ellenőrzést beillesztéskor. A szám után d, m vagy w betűvel jelezhetjük a napot, hónapot vagy hetet. Tiltás 0 értékkel történhet.
-j	Napló hozzádása az állományrendszerhez. Ezzel a kapcsolóval az állományrendszerből extended 3 állományrendszert készíthetünk használat közben.
-l	Az állományrendszer tulajdonságainak kiíratása.
-m n	A rendszerüzem számára fenntartott blokkok százalékos megadása.

Láthatjuk, hogy a részletes ellenőrzés sem talált hibát.

Tudnunk kell, hogy a legtöbb GNU/Linux gyűjteményben a e2fsck futtatása automatikusan megtörténik a rendszerindításkor. A legtöbb esetben ez semmi fennakadást nem okoz, hiszen a program azonnal kilép, amint látja, hogy az állományrendszer tökéletes állapotban van. Bizonyos esetekben azonban részletes ellenőrzést végez a program.

Ha az állományrendszer túlságosan sokszor lett beillesztve ellenőrzés nélkül vagy ha régóta nem történt ellenőrzés, az e2fsck részletesen ellenőrzi az állományrendszert épp úgy, mint akkor, ha azon hibák vannak. A részletes ellenőrzés időnként tehát automatikusan megtörténik, amikor a számítógépet újraindítjuk. Ez némi fennakadást jelent a rendszerindítás során, de általában nem okoz problémát a rendszergazda számára.

Egész más a helyzet azonban akkor, ha az e2fsck olyan komoly hibát talál az állományrendszerben a rendszerindításkor, amelyet nem képes emberi közbeavatkozás nélkül kijavítani. Ilyen esetben az indítást végző héjprogram rendszergazdai jelszót kér és karbantartási üzemmódban indítja el a számítógépet. A rendszergazda feladata és felelőssége, hogy a jelszó begépelése után futassa az e2fsck programot és a hibákat helyreállítsa.

A rendszergazda helyreállító munkája a legtöbb esetben kimerül abban, hogy elindítva az e2fsck programot és annak minden kérdésére igennel válaszolva engedélyt ad a veszélyesebb módosításokra. Ügyelnünk kell azonban, mert a hibás állományrendszeren eszközölt módosítások esetleg annyira összekeverhetik az állományrendszert, hogy azt később nem tudjuk többé helyreállítani.



A tapasztalat azt mutatja, hogy az e2fsck minden helyre tud állítani, nyugodtan megadhatjuk neki a felhatalmazást a legsúlyosabb lépések megtételére is. Valójában azonban az e2fsck csak egy program, amely megpróbál helyreállítani néhány (vagy éppen sok) összekeveredett adatszerkezetet és természetesen senki sem garantálhatja, hogy ez sikerül is neki. Érdemes tehát végiggondolnunk, hogy mennyire fontosak az állományrendszerben található adatok mielőtt megadnánk a felhatalmazást.

3.2. Az extended 3 állományrendszer

A merevlemezek rohamosan növekvő adattároló kapacitása miatt szükségessé vált az extended 2 állományrendszer továbbfejlesztése. A fejlesztést leginkább az indokolta, hogy a nagyméretű merevlemezeken található állományrendszerek logikai ellenőrzése és helyreállítása igen időigényes műveletté vált. A nagyméretű állományrendszerek időigényes helyreállításának kihívására válaszul elkészült az extended 3 állományrendszer[25], amely az extended 2 állományrendszer naplóval kiegészített változata.

3.6. tábla: e2fsck

Extended 2 és extended 3 állományrendszerek ellenőrzése és hibajavítása.

`tune2fs [kapcs] [blokkeszköz-meghajtó]`

A program segítségével az extended 2 és extended 3 állományrendszerek üzem közben kialakult logikai hibáit deríthetjük fel és javíthatjuk.

Kapcsoló *Jelentés*

- b n A kapcsolóval az állományrendszer elején található szuperblokk sérülése esetén a szuperblokk másolat helyét adhatjuk meg. A másolat blokkmérettől függően a 8193 (1kb), 16384 (2kb) és 32768 (4kb) számokkal érhető el.
- c Hibás adatblokkok keresése a háttértár teljes végigolvasásával.
- f A kapcsoló hatására a program mindenképpen elvégzi a részletes ellenőrzést.

Az állományrendszer helyreállítására a legtöbb esetben azért van szükség, mert az operációs rendszernek nem volt lehetősége valamilyen írási művelet befejezésére. Amikor a Linux elkezd egy írási műveletet (új állományt hoz létre, meglévő állományt módosít) az állományrendszer logikai rendje egy időre felborul. A rend akkor áll helyre, amikor az írási művelet befejeződik. Ha a számítógépet kikapcsoljuk az írási művelet befejezése előtt, a logikai rend nem áll helyre, az állományrendszerben a félig végrehajtott írási művelet miatt logikai hiba keletkezik. Az időigényes helyreállítási műveletre a logikai rend helyreállításának érdekében van szükség.

A probléma megoldásának érdekében ma már a legtöbb Unix rendszerben naplózó állományrendszert használnak. A napló működésének lényege az, hogy az írási műveletek először egy naplóban, az állományrendszer különálló, kisméretű részében történnek meg, így a helyreállítás igen gyorsan és hatékonyan elvégezhető. Az extended 3 állományrendszer ilyen naplót tartalmaz, egyébként teljesen megegyezik a jól bevált extended 2 állományrendszerrel.



A napló használata nem garantálja, hogy az írási művelet közben kikapcsolt számítógépen az épenn írt adatok nem vesznek el. A napló csak azt garantálja, hogy az állományrendszer logikai rendje gyorsan helyreállítható, de ez sokszor a félig végrehajtott írási művelet teljes visszavonását eredményezi.

Az extended 3 állományrendszer használatához a rendszeren telepíteni kell a megfelelő programokat. A rendszermagnak tartalmaznia kell a megfelelő programrészeket az állományrendszer használatához, és a segédprogramok (`mke2fs`, `e2fsck`, `tune2fs`) újabb – extended 3 állományrendszert támogató – változatait

kell használnunk. Ma már a legtöbb GNU/Linux telepítőkészlet tartalmazza ezeket az elemeket.

Az extended 2 állományrendszert az adatok mentése és visszatöltése nélkül, könnyedén átalakíthatjuk extended 3 állományrendszerré a tune2fs program segítségével. Ezt mutatja be a következő példa.

15. példa A /dev/sda1 lemezrészén extended 2 állományrendszer található. Alakítsuk át az állományrendszert extended 3 állományrendszerre!

```
$ tune2fs -j /dev/sda1
tune2fs 1.32 (09-Nov-2002)
Creating journal inode: done
This filesystem will be automatically checked every 27 mounts
or 180 days, whichever comes first. Use tune2fs -c or -i to
override.
$
```

Amint látjuk, a program létrehozta az extended 3 állományrendszerre jellemző naplót, ezzel az átalakítás megtörtént.

Az e2fsck program újabb változatai automatikusan kezelik a naplót, így tökéletesen alkalmasak az extended 3 állományrendszer helyreállítására. A hibás állományrendszer helyreállítása legtöbb esetben az állományrendszeren található napló érvényesítését jelenti, így néhány másodperc alatt elvégzhető.

Az extended 3 állományrendszer a könyvtárszerkezetbe illeszthető extended 3 és extended 2 állományrendszerként is. Ha az utóbbi módszert választjuk, az állományrendszer korlátozás nélkül használható, de a naplázás nem történik meg. Ezt mutatja a következő példa.

16. példa Az előző példában naplóval ellátott állományrendszert illesszük be extended 2 állományrendszerként a könyvtárszerkezetbe!

```
$ mount -t ext2 /dev/sda1 /home/
$
```

Amint láthatjuk, a csereszabatosság eredményeként az extended 3 állományrendszerű adathordozók használhatók olyan számítógépen is, ahol csak az extended 2 állományrendszer a támogatott.

3.3. Az XFS állományrendszer

A GNU/Linux világban az utóbbi években komoly változások történtek. Mondhatnánk azt is, hogy a GNU/Linux felnőttkorába lépett, amikor a „nagy testvérek”, a Linux előtt született UNIX-változatok keztek komolyan venni a legfiatalabb társukat. A GNU/Linux hatalmas szövetségesekre lelt, kölcsönös előnyökkel kecsegtető megállapodások születtek.

Az XFS állományrendszer egy ilyen szövetség eredményeképpen került a Linux rendszermag forrásába, a Silicon Graphics Inc. jóvoltából.

Az XFS állományrendszer ízig-vérig nagyszámitógépes állományrendszer, amely a személyi számítógép világán kívül született, de ma már a személyi számítógépeken is használható [12]. Már a felületes vizsgálat is megmutatja, hogy az XFS állományrendszert kitűnően használhatjuk nagyobb GNU/Linux rendszerek, ki-szolgálók állományrendszereként.

Az XFS állományrendszer beépített támogatást tartalmaz a könyvtárbejegyzések jogosultsági listájának (ACL) és a lemezkorlát kezelésére. Ezeknek az eszközöknek a kezelésére a későbbiekben visszatérünk.

Az XFS állományrendszer naplózó állományrendszer. Ez nagy állományrendszelek esetében ma már elengedhetetlenül fontos.

Az XFS állományrendszer erőssége hatékony párhuzamos működés. Ez azt jelenti, hogy nagy rendszerek esetén minél több folyamat használja egyszerre az állományrendszert, annál jobb eredményeket produkál az XFS más állományrendserekkel összehasonlítva.

Az XFS állományrendszer belső felépítése három részre tagolható.

- Az adatterület szolgál a szokásos adattárolásra, hordozza a könyvtárbejegyzések rövidített nézetét, amelyeket nyilvántartott adatokat és a könyvtárbejegyzések adatterületeit (például az állományok tartalmát).
- A naplóterület hordozza az állományrendszer naplóbejegyzéseit, amelyek lehetővé teszik, hogy az állományrendszer helyreállításához ne kelljen a teljes állományrendszert átvizsgálni.
- A valós idejű adatterület hordozza azokat a valós idejű (*real time*) könyvtárbejegyzéseket, amelyek gyors elérése és kezelése létfontosságú.

Az alkalmazások a könyvtárbejegyzések létrehozásakor megadhatják, hogy az adott könyvtárbejegyzés a valós idejű területre kerüljön, hogy elérése elsőbbséget élvezzen.

Az XFS állományrendszer minden területe külön fizikai eszközre helyezhető. Az állományrendszer létrehozásakor megadhatjuk például azt, hogy a valós idejű állományok egy gyorsabb merevlemezre kerüljenek. Ezt az izgalmas lehetősséget az XFS előtt GNU/Linux rendszereken nem használhattuk.

XFS állományrendszert az `mkfs.xfs` programmal hozhatunk létre. Ezt mutatja be a következő példa.

17. példa Szeretnénk egy lemezrészben az ott található állományrendszert törlni és XFS állományrendszert létrehozni. Használjuk az `mkfs.xfs` programot erre a célra.

```
$ fdisk -l /dev/sda
```

```
Disk /dev/sda: 131 MB, 131072000 bytes
```

```
5 heads, 50 sectors/track, 1024 cylinders
Units = cylinders of 250 * 512 = 128000 bytes

      Device Boot Start     End   Blocks Id System
/dev/sda1            1   1024  127975  83 Linux
$ mkfs.xfs -f /dev/sda1
meta-data=/dev/sda1  isize=256    agcount=7, agsize=4096 blks
                  =       sectsz=512
data     =           bsize=4096   blocks=28672, imaxpct=25
          =           sunit=0    swidth=0 blks, unwritten=1
naming   =version 2   bsize=4096
log      =internal log bsize=4096   blocks=1200, version=1
          =           sectsz=512  sunit=0 blks
realtime =none        extsz=65536  blocks=0, rtextents=0
$
```

Figyeljük meg, hogy a lemezrész típusának beállításakor semmilyen különleges lépést nem kellett tennünk, az mkfs.xfs -f kapcsolóját azonban használnunk kellett, hogy a meglévő állományrendszert felülírjuk.

Megfigyelhető az is, hogy az mkfs.xfs a futása közben kiírta, milyen értékekkel hozta létre az állományrendszert.

Az XFS állományrendszert beilleszteni a szokásos módon lehet. Megadhatjuk az xfs állományrendszert a mount -t kapcsolója után, de bízhatunk az állományrendszer automatikus felismerésében is. Az állományrendszer beillesztését mutatja be a következő példa.

18. példa Illessük be az előző példában formázott állományrendszert a szokásos módon!

```
$ mount /dev/sda1 /mnt
$ mount | grep sda1
/dev/sda1 on /mnt type xfs (rw)
$
```

Amint látjuk, a beillesztés nem igényelt különleges eljárást, a szokásos módszer használható volt.

3.4. A csereterület

A csereterület (*swap space*) nem valódi állományrendszer, hiszen állományok tárolására nem alkalmas, az állományrendszerekhez való hasonlósága miatt azonban az állományrendszerek közt tárgyaljuk. A csereterületen használt adatszerkezetre a swap kulcsszóval hivatkozhatunk a különféle beállítóállományokban, ezért ezt az adatszerkezetet szokás swap állományrendszernek is nevezni.

3.7. tábla: mkfs.xfs

A program XFS állományrendszer létrehozására használható.

mkfs.xfs [kapcsolók] blokkész köz-meghajtó

A program segítségével XFS állományrendszt hozhatunk létre lemezrészen vagy állományban. A program kapcsolóival az XFS állományrendszer tulajdon-ságai finomhalgolhatók.

Kapcsoló Jelentés

-f Meglévő állományrendszer felülírása.

A csereterület a számítógépbe épített memória (operatív memória) tehermentesítésére, kiterjesztésére használható. A Linux rendszermag képes arra, hogy a memoriának az éppen nem használt területeit a csereterületre mentse, és így memóriát szabadítson fel. A rendszermag természetesen automatikusan visszatölki a mentett memóriatartalmat, amint arra szükség van. Mivel a mentést és visszatöltést a Linux automatikusan elvégzi, semmiféle feladat nem hárul a rendszeren futó programokra vagy a rendszert használó felhasználókra.

Mivel a csereterület a gépre épített memóriát bővíti – úgy viselkedik, mintha operatív memória volna –, szokás látszólagos memóriának (*virtual memory*) is nevezni. Fontos azonban, hogy soha ne feledjük szem elől a látszólagos memória és a valódi memória közti különbséget: a látszólagos memória nagyságrendekkel lassabban működik, mint a számítógép operatív memóriája.

Amíg ritkán kell az adatokat a csereterület és a memória közt mozgatni, a sebeségkülönbség nem lassítja jelentősen a számítógép működését. Ha azonban a csereterületre másolás és az onnan való visszatöltés sűrűn jelentkezik, a számítógép működése jelentősen lassul. Minél kisebb az operatív memória a csereterülethez képest, az adatok másolására annál többször van szükség, hiszen minél kisebb az operatív memória, annál nagyobb az esély arra, hogy a keresett adatok éppen a csereterületen vannak.

Nem könnyű általános érvényességű szabályt adni a létrehozandó csereterület nagyságára anélkül, hogy pontosan tudnánk, milyen jellegű feladatokat lát el az adott számítógép, de annyit kijelenthetünk, hogy nem szerencsés, ha a csereterület nagysága meghaladja az operatív memória nagyságának háromszorosát. A legtöbb rendszeren az operatív memória nagyságának kétszeresénél nem nagyobb csereterületet használunk.

Ha meg akarjuk tudni, hogy a Linux mennyi csereterületet használ, a /proc/swaps látszólagos állományból kiolvashatjuk, ahogyan a következő példa is bemutatja.

19. példa Szeretnénk megtudni, hogy a Linux mennyi csereterületet használ. A

3.8. tábla: swapon, swapoff

Csereterület használatának be- és kikapcsolása.

swapon [kapcs] [blokkész köz-meghajtó]
swapoff [kapcs] [blokkész köz-meghajtó]

A programot **swapon** névvel indítva bekapcsolhatjuk, **swapoff** névvel indítva pedig kikapcsolhatjuk a csereterületek használatát.

Kapcsoló Jelentés

- a Az összes csereterület bekapcsolása és kikapcsolása, amely az állományrendszer-táblázatban fel van sorolva, illetve használatban van.

cat parancs segítségével a /proc/swaps állományból kiolvashatjuk a pillanatnyi értéket:

```
$ cat /proc/swaps
Filename      Type          Size    Used   Priority
/dev/hda3     partition    265064  38184  -1
$
```

Amint látjuk, egyetlen csereterület van használatban, amelyet a /dev/hda3 lemezrész helyeztünk el. Ennek mérete mintegy 250 megabájt, ebből foglalt – memóriatartalmat hordoz – mintegy 34 megabájt.

Amint az előző példában is láttuk, a csereterületet általában teljes lemezrészek lefoglalásával valósítjuk meg. A merevlemez lemezrészekre osztásakor külön lemezrészről hozunk létre a csereterület számára. Linux csereterületek létrehozására a 0x82 típusú lemezrészeket használjuk.

Ha ki akarjuk kapcsolni a csereterület használatát, a **swapoff** programot kell futtatnunk. Ha viszont használatba akarjuk venni a csereterületet, a **swapon** parancssal tehetjük meg. A csereterület előkészítését a **mkswap** parancssal végezhetjük el. Ezt mutatja be a következő példa.

20. példa Tegyük fel, hogy néhány percre szükségünk van egy lemezterületre, amelyen állományokat helyezhetünk el. Erre a célról felhasználhatjuk a csereterületet, amelyet időlegesen felszabadítunk állományok tárolására.

Először kikapcsoljuk a csereterület használatát, majd létrehozunk rajta egy extended 2 állományrendszeret:

```
$ swapoff /dev/hda3
$ mke2fs -q /dev/hda3
mke2fs 1.32 (09-Nov-2002)
$
```

3.9. tábla: mkswap

A program segítségével a csereterületet előkészíthetjük használatra.

mkswap [kapcs] [blokkész köz-meghajtó]

Kapcsoló Jelentés

-c Hibás blokkok keresése a merevlemezen.

Most beilleszhetjük az újonan létrehozott állományrendszert a könyvtárszerkezetbe, és állományok tárolására használhatjuk. Ha már nincs szükség a lemezterületre, újra felkészítjük csereterületként való használatra, és igénybe is vesszük mint csereterület:

```
$ mkswap /dev/hda3
Setting up swap space version 1, size = 271429 kB
$ swapon /dev/hda3
```

Arra azonban ügyelnünk kell, ha ezt a módszert választjuk, hogy ne indítsuk újra a számítógépet! A rendszerindításkor ugyanis a Linux csereterületként próbálja használni a lemezrészét, mivel a rendszerbeállításokon nem változtattunk.

3.5. CD-ROM állományrendszer

A CD-ROM lemezeken használható állományrendszert az ISO-9660 számú szabvány írja le, ezért GNU/Linuxon az ilyen állományrendszereket iso9660 állományrendszernek is nevezik.

Az iso9660 állományrendszer meglehetősen egyszerű. Egyszerű azért, mert az elkészítésekor az volt a cél, hogy a legegyszerűbb operációs rendszerek is kezelni tudják, és egyszerű azért, mert nem teszi lehetővé állományok és könyvtárszerkezetek létrehozását. Az iso9660 állományrendszert a létrehozása után nem lehet módosítani, mivel eredetileg csak olvasható adathordozókon való használatra terveztek.

Az iso9660 állományrendszer eredeti formájában nem alkalmas Unix rendszereken való használatra. Nem teszi lehetővé például 255 karakter hosszú állománynevek használatát és tetszőleges mélységű könyvtárszerkezetek létrehozását. Ha tehát egy eredeti iso9660 állományrendszert a GNU/Linux könyvtárszerkezetbe illesztünk, azon nem tárolhatunk olyan néven állományokat, amilyen néven a könyvtárszerkezet egyéb részein tároljuk az állományainkat. Az iso9660 eredeti formájában tehát egyszerűsége miatt megbontja a könyvtárszerkezet egy-ségeségét, ezért nem használjuk.

Szerencsére az ISO-9660 szabvány készítői gondoltak a további fejlesztésekre, fenntartottak területeket a későbbi bővítések számára. A Rock Ridge szabványbővítés ezeken a helyeken tárolja a kiegészítő információkat, amelyek a CD-ROM lemezeket a Unix állományrendszerek teljes jogú tagjává tették.

GNU/Linux rendszeren tehát a legtöbbször a Rock Ridge bővítéssel használjuk az iso9660 állományrendszeret. A bővítés lehetővé teszi, hogy hosszú állományneveket, mély könyvtárszerkezeteket, közvetett hivatkozásokat és egyéb, a GNU/Linux könyvtárszerkezetén belül használható eszközöket tároljunk a CD-ROM lemezeken.

A Rock Ridge bővítéssel készített lemezek teljesítik az ISO 9660 szabványt. Más operációs rendszerekben, ahol nem áll rendelkezésre Rock Ridge bővítés, csak az eredeti ISO-9660 szabvány, használhatjuk a bővítéssel készített lemezt. Lesznek természetesen olyan információk, amelyek ezeken a rendszereken nem érhetők el – például a hosszú állománynevek helyett rövid állománynevek jelennek meg –, de az adatok elérhetők maradnak.

Az elmúlt évek során azok az operációs rendszerek, amelyek miatt az ISO-9660 állományrendszer oly egyszerűre és korlátozott képességűre sikerült, komoly fejlődésen mentek keresztül. Számukra készítették az ISO-9660 egy másik bővítését, amelyet Joliet névre kereszteltek. Ez a bővítés azonban olyan elemeket is tartalmaz, amelyek ellentmondásban vannak az ISO-9660 szabvánnyal, ezért a Joliet nem tekinthető szabványbővítésnek a szó szoros értelmében.



A Rock Ridge egy meglehetősen összetett eljárás keretében áthelyezi a könyvtákat más könyvtárakba, ha azok túlságosan mélyre kerülnek a könyvtárszerkezetben, és az operációs rendszerre bízza, hogy azokat ott jelenítse meg, ahol eredetileg voltak. Ily módon a Rock Ridge bővítéssel készült lemezek teljesítik a szabványt, mégis tartalmazhatnak mély könyvtárszerkezeteket.

A Joliet szabvány egyszerűen „megengedi” a mély könyvtárszerkezetek létrehozását, ezért mély könyvtárszerkezetek esetén a Joliet szabvány alapján készített lemezek nem teljesítik az ISO-9660 előírásait, hiszen túlságosan mély könyvtárszerkezeteket használnak.

Hasonló módon kezeli a Joliet például a könyvtárbejegyzések neveire vontkozó korlátozásokat is.

A Linux képes kezelni az eredeti ISO-9660 szabvány alapján készített lemezeket, valamint automatikusan érzékeli a Rock Ridge és Joliet szabványokat, és a megfelelő módon használja őket. GNU/Linux rendszeren futtatható programok segítségével létrehozhatunk – és a lemezre írhatunk – iso9660, valamint Rock Ridge bővítéssel vagy Joliet szabványmódosítással készült állományrendszereket.

3.5.1. Az iso9660 állományrendszer létrehozása

Az iso9660 állományrendszer létrehozására GNU/Linux rendszereken az `mkisofs` program szolgál. A programot igen egyszerűen használhatjuk – amint a következő példában látni fogjuk – de rengeteg kapcsolója segítségével szinte mindenre képes a CD-ROM állományrendszerek létrehozásában [24].

21. példa Szeretnénk létrehozni egy iso9660 állományrendszt Rock Ridge bővítéssel. A CD-ROM lemezen a forrasok/ könyvtár tartalmát szeretnénk elhelyezni (a forrasok/ könyvtárat nem, csak a tartalmát). Futtassuk az `mkisofs` programot a következő kapcsolókkal:

```
$ mkisofs -R -o /tmp/forras.iso forrasok/
```

A program a parancs hatására létrehozza a /tmp/forras.iso állományt, benne a teljes állományrendszerrel. Ez az állomány a CD-ROM lemez teljes térképét tartalmazza, minden bájt megtalálható benne, aminek a lemezre kell kerülnie.

A példában látható formában megadott `-R` kapcsoló hatására a `mkisofs` Rock Ridge bővítéssel hozza létre az állományrendszert. Hasonlóképpen használható a `-J` kapcsoló a Joliet szabványmódosítás bekapsolására.

Az `mkisofs` program segítségével könnyen készíthetünk olyan lemeztérképet, amelyről a CD-ROM lemezre való írás után az operációs rendszer indítható (*bootable CD-ROM*).

3.10. tábla: mkisofs

ISO-9660 szabvány szerint készített állományrendszer létrehozása.

`mkisofs [kapcs] -o kim.állomány könyvtár [könyvtári ...]`

A program segítsével CD-ROM-on használható ISO-9660 szerint felépített állományrendszt készíthetünk állományba. A programmal elkészített állományt később a cdrecord programmal a CD-ROM lemezre írhatjuk.

Kapcsoló	Jelentés
-o fájl	A kimenőállomány nevének megadása.
-R	A Rock Ridge szabványbővítés használata.
-J	Joliet szabványú állományrendszer létrehozása.
-dvd-video	DVD-Video csereszabatos UDF állományrendszer létrehozása.
-b fájl	Indítólemez létrehozása. A kapcsoló után megadott állománnév egy hajlékonylemezeiről készített térképállomány neve.
-z	Tömörített állományrendszer létrehozása. Szükségessé teszi a Rock Ridge szabványbővítés és az <code>mkzftree</code> program használatát.

Az `mkisofs` El Torito rendszerű rendszertöltő lemezeket készít. Ennek lényege, hogy a CD-ROM betöltő része hajlékonylemezt utánoz, azaz a betöltés során a CD-ROM lemez betöltést végző része hajlékonylemezként működik.

Rendszertöltő CD-ROM lemeztérkép készítésekor először létre kell hoznunk egy 1200, 1440, vagy 2880 kbájtos lemeztérképet, amely a rendszer betöltését végzi, majd el kell helyeznünk ezt a könyvtában, amely a CD-ROM lemezre kerülő anyagokat tartalmazza. Az `mkisofs` használatakor a -b kapcsoló után ennek a térképállománynak a nevét kell megadnunk, de ügyelni kell arra, hogy a név megadásakor a lemezre kerülő anyagokat tartalmazó könyvtáron belüli elérési utat használjuk. Ezt mutatja be a következő példa.

22. példa Elhelyeztük a CD-ROM lemezre írándó állományokat a `cd-re/` nevű könyvtárban. Másoljuk fel a rendszerindító lemez térképállományát a hajlékonylemezes meghajtóról, és készítsünk rendszertöltésre alkalmas CD-ROM lemeztérképet!

```
$ mkdir cd-re/images
$ cat /dev/fd0 >cd-re/images/boot.img
$ mkisofs -R -b images/boot.img -o cdrom.iso cd-re/
```

A példában a hajlékonylemez térképe közvetlenül a hajlékonylemezeiről származott, de természetesen másképp is előállítható ilyen lemeztérkép. A legtöbb GNU/Linux gyűjtemény például tartalmazza a megfelelő hajlékonylemez

3.11. tábla: losetup

Szabályos állományt köt össze blokkeszköz-meghajtó állománnal és választ le róla. A blokkeszköz-meghajtó állományokhoz kapcsolt szabályos állományok a könyvtárszerkezetbe illeszthetők.

```
losetup blokk_eszköz szab_állomány
losetup -d blokk_eszköz
```

Kapcsoló *Jelentés*

-d A blokkeszköz-meghajtó és a szabályos állomány szétválasztása.

terképeket-rendszertöltésre alkalmas CD-ROM lemez készítéséhez, amelyekkel a telepítést megkezdhetjük.

3.5.2. Az iso9660 állományrendszer kipróbálása

Az elkészített iso9660 állományrendszer általában CD-ROM lemezre írjuk, ami viszonylag lassú folyamat és a legtöbb esetben visszafordíthatatlan (a lemez nem törölhető). Szerencsés tehát, ha vettünk egy pillantást a lemez tartalmára, mielőtt felírjuk az adatokat a CD-ROM lemezre.

Ha állományban elhelyezett állományrendszer térképet szeretnénk az állományrendszerbe illeszteni, a beillesztés előtt az állományrendszer blokkeszköz-meghajtó állományhoz kell kapcsolnunk, hiszen a beillesztés során ilyen blokkeszköz-meghajtó állományra van szükségünk. Állományok blokkeszköz-meghajtó állományon keresztül való elérésére szolgálnak a /dev/loop0, /dev/loop1 stb. könyvtárbejegyzések, amelyeket állományhoz kötni a losetup program segítségével lehetséges.

Az mkisofs programmal elkészített állományrendszer CD-ROM lemezre való felírás előtti kipróbálásához előbb hozzá kell csatolni az állományt egy blokkeszköz-meghajtó állományhoz. Ez után a blokkeszköz-meghajtó állományon keresztül az állományrendszer beilleszthető a könyvtárszerkezetbe, és az állomány tartalma átvizsgálható. Ha már nincs szükségünk az állományokra, az állományrendszer lecsatoljuk, majd megszüntetjük a kapcsolatot a blokkeszköz-meghajtó és az állomány között. Ezeket a lépéseket mutatja be a következő példa.

23. példa Letöltöttünk egy teljes CD-ROM térképet, és a lemezre való felírás előtt szeretnénk átvizsgálni annak tartalmát.

Első lépésként csatlakoztassuk az állományt a /dev/loop0 blokkeszköz-meghajtó állományhoz, és illessük be az állományrendszerbe az állományt:

```
$ losetup /dev/loop0 1.iso
$ mount -t iso9660 /dev/loop0 /mnt/cdrom/
```

Most az állományban található állományrendszer be van illesztve a /mnt/cdrom/ könyvtárba – oda, ahol ákkor is be lesz illesztve, ha már felírtuk a lemezre –, tehát a tartalma megvizsgálható, a lemez kipróbálható.

Ha már nincs szükségünk az állományrendszer tartalmára, csatoljuk le azt, és szüntessük meg a /dev/loop0 felé kiépített kapcsolatot.

```
$ umount /mnt/cdrom
$ losetup -d /dev/loop0
$
```

A két parancs kiadásával az eredeti állapotot visszaállítottuk, az állomány nem kapcsolódik blokkeszköz-meghajtó állományhoz.

Az előző példában bemutatott módszer természetesen nem csak az iso9660 állományrendszerrel használható. Bármelyik állományt beilleszthetjük így a könyvtárszerkezetbe, amely állományrendszert tartalmaz.

3.5.3. Az adatok felírása CD-ROM lemezre

Az adatok CD-ROM lemezre írását könnyedén elvégezhetjük a cdrecord program segítségével. A programnak – helyes beállítás után – csak a térképalomány nevét kell megadnunk, az az állományrendszert a lemezre írja.

A cdrecord csak SCSI programozófelülettel ellátott CD-írókat képes kezelni. Szerencsére a Linux képes nem SCSI felületen csatlakozó eszközök számára ilyen programfelületet biztosítani. A legtöbb GNU/Linux gyűjtemény ma már automatikusan SCSI-utánzást biztosít a CD-író eszközök számára, így azok SCSI eszköz-ként jelennek meg, a cdrecord képes kezelní őket.

A cdrecord a -scanbus kapcsoló hatására kiírja a szabványos kimenetre, hogy melyik azonosítószámon milyen eszközt talált. Válasszuk ki az eszközök közül azt, amelyiken a lemezt írni szeretnénk, és adjuk meg a három számból álló azonosítót a cdrecord dev= paramétere után.

Ha sokszor használjuk a programot, szerencsés az eszköz azonosítócímét és az írási sebességet környezeti változóban elhelyezni. A következő példa ezt mutatja be.

24. példa A cdrecord a -scanbus kapcsoló hatására kiírt listában jelezte, hogy csak egyetlen CD-író eszköz található a számítógépen, mégpedig a 0,0,0 címen. A gyártó szerint ezzel az eszközzel nyolcszoros sebességgel lehet lemezeket írni.

Helyezzük el a következő sorokat a /etc/profile állományban:

```
export CDR_SPEED=8
export CDR_DEVICE=0,0,0
```

A cdrecord a következő bejelentkezés után már egyszerűen, a térképalomány nevének megadásával indítható.

3.12. tábla: cdrecord

CD-ROM lemezek írását végző program.

`cdrecord [kapcs] térkép_állomány`

Kapcsoló	Jelentés
<code>-v</code>	A lemezre írás során a folyamatot a képernyőn követhetjük.
<code>-eject</code>	Az írás után a lemezt a program kiadja.
<code>dev=i,j,k</code>	A CD-író eszköz pontos címét várja a <code>dev=</code> paraméter után.
<code>-scanbus</code>	A program az összes fellelhető eszköz nevét és számát kiírja a szabványos kimenetre.
<code>speed=n</code>	Az írás sebességének megadása.
<code>-overburn</code>	A kapcsoló hatására a program megkísérel több adatot felírni a lemezre, mint amennyit a gyártó szerint lehet.

Természetesen automatikusan is beállíthatjuk a CD-író címét a `cdrecord` kiemenetén kapott listából. Helyezzük el például a következő sort a `/etc/profile` állományban:

```
export CDR_DEVICE=$(cdrecord -scanbus 2>/dev/null | \
grep CDRW | awk '{print $1}')
```

A bemutatott megoldás a CDRW kifejezést keresi a `cdrecord` kimenetén kapott listában, és a talált sorból az első oszlopot használja fel címként. Ezt a címet helyezi el az értékadás a `CDR_DEVICE` környezeti változóban, ahol a `cdrecord` program keresi a CD-író címét.

3.5.4. Tömörített CD-ROM állományrendszer

A Linux támogatja a Rock Ridge tömörített állományok kezelésére készített bővítesét. Ez a bővítés azonban csak a Linux használatával érhető el, az így készített lemezek más operációs rendszerrel nem használhatók.

A tömörített CD-ROM állományrendszer működése igen egyszerű. A lemezre írandó állományokat az `iso9660` állományrendszer elkészítése előtt tömörítenünk kell, a kicsomagolást pedig automatikusan elvégzi a Linux, amikor az állományokat később használjuk. A kicsomagolás automatikus, a program, amelyik a CD lemezen található adatokat olvassa, már a kicsomagolt állományokat kapja a Linuxtól, így a tömörített lemez használata nem okozhat nehézséget.

A tömörített állományrendszer létrehozása egyszerű. Az állományokat a térképpállomány létrehozása előtt tömörítenünk kell az `mkzftree` program segítségével és a térképpállomány létrehozásakor a `-z` kapcsolót kell megadnunk az `mksofs` programnak. Ezt mutatja be a következő példa:

3.13. tábla: mkzftree

Teljes könyvtárszerkezetek tömörítése tömörített CD-ROM állományrendszer létrehozásához.

mkzftree [kapcs] eredeti másolat

A program az eredeti könyvtár tartalmát lemásolja az utolsó paramétereként megadott könyvtárba, miközben tömöríti. A tömörített könyvtárszerkezetből tömörített CD-ROM állományrendszer készíthető.

Kapcsoló **Jelentés**

-z n A tömörítés mértékének megadása 1-től 9-ig. A nagyobb szám hatékonyabb – és lassabb – tömörítést tesz lehetővé.

25. példa Az anyag/ nevű könyvtár tartalmát szeretnénk felírni egy CD-ROM lemezre, de annak teljes mérete jóval meghaladja a 800 megabájtot. Készítsük el a könyvtár tömörített változatát az **mkzftree** program segítségével!

```
$ mkzftree anyag tomor
$ du -hs *
882M    anyag
638M    tomor
$
```

Amint láthatjuk az újonan létrehozott tomor/ állomány bőven elfér a CD-ROM lemezen. Hozzunk létre egy térképállományt a tömörített könyvtárból!

```
$ mkisofs -z -R -o 1.iso tomor/
Warning: using transparent compression. This is a nonstandard
Rock Ridge extension. The resulting filesystem can only be
transparently read on Linux. On other operating systems you
need to call mkzftree by hand to decompress the files.
```

Figyeljük meg, hogy használnunk kellett a -z és a -R kapcsolókat!

Az így elkészített térképállományt a CD-ROM lemezre írhatjuk és a tömörített CD-ROM állományrendszer támogató Linux rendszereken a megszokott módon használhatjuk.



Mielőtt használni kezdenénk a tömörített CD-ROM állományrendszt, egy dologgal mindenki által tisztában kell lennünk a tömörítési eljárásokat illetően.

Amit egy tömörítőprogrammal tömörítettünk, azt általában nem tudjuk továbbtömöríteni – legalábbis jelentős mértékben – más tömörítőprogram használatával. A hang és videóanyagok tárolására használt népszerű állományformátumok tömörítettek, így nincs esélyünk arra, hogy jelentős helymegtakarítást érjünk el ezek tárolásakor a tömörített CD-ROM állományrendszer segítségével.

3.6. A proc állományrendszer

A proc állományrendszer nem valódi állományrendszer, a Linux rendszermag által szimulált könyvtárbejegyzéseket és adatokat tartalmazza. Mindazon állományok és könyvtárak tehát, amelyek a proc állományrendszer beillesztésével válnak elérhetővé, nem a merevlemezen vagy a számítógépbe beépített egyéb háttértáron tárolódnak, hanem csak a memóriában léteznek, és csak akkor, amikor éppen használjuk őket.

Ha egy program a proc állományrendszeren belül található állományt olvassa, a Linux rendszermag által a kéréskor előállított adatokat olvassa. Ha a program ilyen szimulált állományba ír, az adatokat a rendsermagnak küldi el, amely azokat értelmezi és felhasználja. A proc állományrendszer tehát ablak a rendszer magja felé, kapcsolattartási felület, amelyen keresztül a rendszergazda a Linux mélyére hatol.

Az elmondottaknak megfelelően a proc állományrendszerhez nem köthető adattároló, így ilyen állományrendszer a háttértáron nem hozható létre. A proc állományrendszert egyszerűen beilleszthetjük a könyvtárszerkezetbe és használhatjuk a kapcsolattartásra. Ezt mutatja be a következő példa:

26. példa Hozzunk létre egy könyvtárat, illesszük be a proc állományrendszert, és olvassuk ki belőle, hogy milyen típusú processzor van a számítógépbe építve!

```
$ mkdir proc  
$ mount -t proc none proc/  
$ grep "model name" proc/cpuinfo  
model name : Mobile Intel(R) Pentium(R) 4 - M CPU 1.70GHz
```

A parancs végrehajtása után a proc állományrendszer a könyvtárszerkezetről lecsatolható.

A legtöbb GNU/Linux gyűjtemény automatikusan beilleszti a proc állományrendszert a /proc/ könyvtárba a rendszerindítás során. Ehhez a könyvtárhoz természetesen csak a rendszergazdának van jog a hozzáférni.

3.7. A RAM állományrendszer

A RAM állományrendszer – ahogyan neve is mutatja – a számítógép operatív memoriáját használja állományok tárolására. A RAM állományrendszer ennek megfelelően írható és olvasható, de tartalma elvész, ha kikapcsoljuk a számítógépet vagy lecsatoljuk az állományrendszeret. A RAM állományrendszert a beállítóállományokban és a programok paramétereiben ramfs névvel jelöljük.

A RAM állományrendszer mérete mindenkorának megfelelően nagy, amelyre területre a tárolt könyvtárbejegyzések számára szükség van. A RAM állományrendszer alapértelmezésben azonban legfeljebb az operatív memória felét használja fel.

A proc állományrendszerhez hasonlóan a RAM állományrendszer sem kell létrehoznunk háttértáron, csak egyszerűen be kell illesztenünk a könyvtárszerkezetbe. A RAM állományrendszer beillesztését és használatát mutatja be a következő példa.

27. példa Illesszünk be a könyvtárszerkezetbe egy RAM állományrendszer!

```
$ mkdir ideiglenes
$ mount -t ramfs -o,maxsize=10240 none ideiglenes/
$
```

Amint látjuk a RAM állományrendszer használatakor beállíthatjuk az állományrendszer legnagyobb méretét. A használt mértékegység a kilobájt.

Használat után a RAM állományrendszer célszerű lecsatolni, hogy ne foglalja többé a memóriát. Ezt a szokásos módon végezhetjük el:

```
$ umount ideiglenes/
$
```

3.8. FAT típusú állományrendszerek

A Linux támogatja a FAT (*file allocation table*, állományfoglaltsági táblázat) rendszerre épülő állományrendszerek használatát. Ezeknek az állományrendszereknek a segítségével más operációs rendszerekkel létesíthetünk kapcsolatot.

3.8.1. Az msdos állományrendszer

Egyesrűsége és elterjedtsége miatt gyakran használjuk hajlékonylemezen az msdos állományrendszer, amelyet eredetileg az MS-DOS programrendszer számára fejlesztettek ki. Mivel használatát ma is sok operációs rendszer támogatja, hasznos lehet, ha háttértáron akarunk adatokat cserélni.

Az msdos állományrendszer könnyedén létrehozható az `mkdosfs` parancs segítségével és beilleszthető a könyvtárszerkezetbe. Ezt mutatja be a következő példa.

28. példa Hozzunk létre msdos állományrendszert a /dev/sda1 lemezrészén, és illesszük be a könyvtárszerkezetbe!

```
$ mkdosfs /dev/sda1
mkdosfs 2.8 (28 Feb 2001)
$ mount -t msdos /dev/sda1 /mnt
$ df -h /mnt/
Fájlrendszer      Méret Fogl. Szab.  % Csatl. pont
/dev/sda1          125M    0  125M   0% /mnt
$
```

Másoljunk egy állományt a beillesztett állományrendszerre, majd vizsgáljuk meg, milyen tulajdonságokkal hozta azt létre a Linux!

```
$ cp /etc/syslog.conf /mnt/
$ ls -lh /etc/syslog.conf /mnt/syslog.con
-rw-r--r-- 1 root root 937 ápr 14 2003 /etc/syslog.conf
-rwxr-xr-x 1 root root 937 okt 16 19:52 /mnt/syslog.con
```

Amint látjuk, az állomány létrejött, de a neve és néhány egyéb tulajdonsága megváltozott. Ennek oka az, hogy az msdos állományrendszer olyan egyszerű, hogy nem képes kezelni a könyvtárbejegyzések hosszú nevét, a hozzájuk kapcsolódó jogokat és tulajdonosokat, valamint még néhány más tulajdonságukat.

Az msdos állományrendszer csak nagy körültekintéssel és óvatossággal használható, egyszerűsége sok probléma okozója lehet. Az msdos állományrendszer által kezelt blokkok mérete az állományrendszer méretétől függ, már kisebb állományrendszer esetén is olyan nagy lehet, hogy a rendelkezésre álló tárkapacitás 60-80%-át elvesztegetjük. Főleg kisebb állományok tárolására használt állományrendszer esetén hasznos, ha a -F 32 kapcsolóval 32 bites állományrendszer-formátumot kapcsolunk be. Így az állományrendszer által kezelt blokkok mérete kisebb lesz és csökken a veszendőbe ment háttérítőterület.



Manapság a legtöbb memóriakártyát használó hordozható eszköz – fényképezőgép, mp3-lejátszó stb. – msdos állományrendszerrel előkészítve vásárolhatók, amiből következik, hogy sok kis állomány esetén a memóriakártyák kapacitásának jelentős része, és így természetesen a memóriakártya árának tetemes része is veszendőbe megy.

Érdemes GNU/Linux rendszer segítségével 32 bites msdos állományrendszer létrehozni a vásárlás után és kipróbálni, hogy a hordozható eszköz képes-e azt használni. Így akár több tízezer forintot takaríthatunk meg egyetlen parancs begépelésével.

Komoly problémákat okozhat az is, hogy az msdos állományrendszer gyökérkönyvtárában nem helyezhetünk el tetszőlegesen sok könyvtárbejegyzést. Ha például sok állományt akarunk felmásolni az msdos állományrendszerre, kénytelenek vagyunk egy alkönyvtárat létrehozni annak gyökérkönyvtárában és az állományokat oda másolni. A könyvtárakban – az állományrendszer gyökérkönyvtárával ellentében – tetszőlegesen sok könyvtárbejegyzés lehet.

3.8.2. A vfat állományrendszer

Az msdos állományrendszer hiányosságainak kiküszöbölésére született a vfat szabványbővítés. Ennek a szabványbővítésnek a használatával akár 255 karakter hosszú állományneveket is használhatunk. A vfat bővítés sem tesz azonban

különbséget alapbeállítás esetén a kisbetűk és a nagybetűk közt és semmiképpen nem alkalmas a könyvtárbejegyzések tulajdonosainak és a hozzájuk kapcsolódó jogoknak a hordozására.

A vfat állományrendszer használata igen egyszerű. Ha az msdos állományrendszert vfat állományrendszerként illesztjük a rendszerbe, használhatjuk a hosszú állományneveket. Ha az adathordozót ezután msdos állományrendszerként illesztjük be, a könyvtárbejegyzéseket csak torzított, rövidített néven érhetjük el.

3.9. Az állományrendszerek összefoglalása

Az állományrendszerekről szóló fejezet végén összefoglalásképpen megadjuk az állományrendszerek Linuxon használt rövidített neveit. Ezek a nevek a programok paramétereként adhatók meg, valamint a különféle beállítóállományokban szerepelhetnek.

Olyan állományrendszerek neveit is megadjuk, amelyek részletes tárgyalása nem található meg a könyvben, de a Linux által támogatott állományrendszerek listája még ebben a formájában sem teljes.

Név	Rövid leírás
ext2	Az extended 2 állományrendszer.
ext3	Az extended 3 állományrendszer.
iso9660	Az ISO-9660 szabvány alapján készített CD-ROM állományrendszer.
jfs	Az IBM által kifejlesztett naplázó állományrendszer.
msdos	Az MS-DOS számára kifejlesztett egyszerű állományrendszer.
ncpfs	A Novell Netware kiszolgálók elérésére használt hálózati állományrendszer.
nfs	A Sun Microsystems által kifejlesztett hálózati állományrendszer.
proc	A rendszermaggal való kapcsolattartásra használható lát-szólagos állományrendszer.
ramfs	Memoriában elhelyezett RAM állományrendszer.
reiserfs	Állományok gyors kikeresésére készített naplázó állományrendszer.
smbfs	Az IBM által kifejlesztett SMB hálózati állományrendszer.
umsdos	Az msdos állományrendszer továbbfejlesztett változata.
vfat	Az msdos állományrendszer továbbfejlesztett változata.
xfs	A Silicon Graphics által kifejlesztett naplázó állományrendszer.
swap	Látszólagos memória hordozására alkalmas adatszerkezet.

3.14. táblázat. Állományrendszerek elnevezése

4. fejezet

Az állományrendszer beillesztése

Amint a 2.4. ábrán láttuk, az állományrendszer beillesztése az állományrendszeren található gyökérkönyvtár csatolása egy, a könyvtárszerkezetben található könyvtárhoz, a beillesztési ponthoz.

A beillesztett állományrendszer tartalma ilyenkor elfedi a beillesztési ponton belül található könyvtárbejegyzések – ha vannak ilyenek. Ezek a lefedett könyvtárbejegyzések a háttértáron helyet foglalnak, de használni őket nem lehet. Amíg a beillesztett állományrendszer – amely letakarja őket – nem csatoljuk le, a lefedett könyvtárbejegyzések használhatatlanok.

4.1. Beillesztés és lecsatolás

Állományrendszereket a könyvtárszerkezetbe illeszteni a `mount` programmal lehet. A programot a legtöbbször három paraméterrel indítjuk. Az első paraméter adja meg a beillesztendő állományrendszer típusát, a második annak a blokközött-meghajtó állománynak a nevét, ahol az állományrendszer található, a harmadik pedig a beillesztési pontot határozza meg.

29. példa Illesszük be a `/dev/hda6` eszközön található extended 3 állományrendszer a `/tmp/` könyvtárba:

```
$ mount -t ext3 /dev/hda6 /tmp  
$
```

A parancsot ebben a formájában azonban csak a rendszergazda használhatja, hiszen egyedül ő az, aki bármelyik állományrendszer beillesztheti tetszőleges

könyvtárba. A felhasználók a programot egyszerűbb formában indíthatják, kizárálag a blokkeszköz vagy a beillesztési pontot adhatják meg. A mount ilyen esetben a további adatokat az állományrendszer-táblázatból olvassa ki, azaz a felhasználó csak azok szerint a beállítások szerint végezhet el a beillesztést, amelyeket a rendszergazda ebben a táblázatban elhelyezett.

A mount program számára a beillesztés során a -o kapcsoló segítségével olyan módosítók adhatók meg, amelyek meghatározzák, milyen módon kerüljön be az állományrendszer a könyvtárszerkezetbe. Ha az állományrendszer már be van illesztve, akkor szintén a -o segítségével adhatjuk meg a remount kulcsszót, amellyel a beillesztés módját változtathatjuk meg anélkül, hogy az állományrendszer lecsatolnánk és újra beillesztenénk. Ezt a fontos lépést mutatja be a következő példa.

30. példa Létre szeretnénk hozni egy állományt a /usr/ könyvtárat tartalmazó állományrendszeren belül, de az meghiúsul:

```
$ touch /usr/bin/krt
touch: '/usr/bin/krt' létrehozása: Csak olvasható fájlrendszer
$
```

Amint látjuk, az állományrendszer csak olvasható módon van beillesztve a könyvtárszerkezetbe. Illessük újra az állományrendszer írható formában, hozzuk létre az állományt, majd térjünk vissza az írásvédett üzemmódra:

```
$ mount -o,remount,rw /usr
$ touch /usr/bin/krt
$ mount -o,remount,ro /usr
$
```

Fontos tudnunk, hogy a bemutatott újraillesztés nem akadályozta a rendszeren dolgozó felhasználók munkáját, hiszen az állományrendszer mindenkor előre maradt.

A -o kapcsoló után használható módosítók listáját a következő fejezetben, az állományrendszer-táblázat bemutatásának kapcsán adjuk meg.

Ha a mount programot kapcsolók és paraméterek nélkül indítjuk, kiírja a szabványos kimenetet a beillesztett állományrendserek listáját. A program ezt az /etc/mtab állományból olvassa ki, amit minden beillesztés során frissít.

Az /etc/fstab állomány azonban nem minden tartalmazza a valós adatokat. Ha például a /etc/ könyvtár nem volt írható a beillesztéskor, a mount nem tudta bejegyezni a beillesztést és az nem szerepel a listában.

Ha a /proc/ könyvtárba proc állományrendszer van illesztve, a rendszermag által nyilvántartott, hiteles beillesztések listája kiolvasható a /proc/mounts lászlólagos állományból. Ez a lista minden pontos, bár kevesebb információt tartalmaz.

4.1. tábla: mount

Állományrendszerek beillesztését végző program.

```
mount [kapcs] blokkeszköz beill_pont
mount -a [kapcs]
mount
```

A program első formájában beilleszti az adott blokkeszközön található állományrendszert az adott beillesztési pontra, a második formájában pedig az összes automatikusan beillesztendő állományrendszert beilleszti. A harmadik formában a program kiírja a beillesztett állományrendszerek listáját.

Kapcsoló	Jelentés
-a	Az összes automatikusan beillesztendő állományrendszer beillesztése (állományrendszer-táblázat).
-t tip	A beillesztendő állományrendszer típusa. A kapcsoló után az állományrendszer rövid neve található (ext2, ext3 stb.)
-n	Beillesztés a /etc/mtab frissítése nélkül.
-o mód	A beillesztés módjának megadása. Az állományrendszer viselkedését megadó – vesszőkkel elválasztott – módokra az állományrendszer-táblázat tárgyalásakor visszatérünk.

4.2. tábla: umount

Állományrendszerek lecsatolását végző program.

```
umount [kapcs] blokkeszköz
umount [kapcs] beill_pont
mount -a
```

A program segítségével a beillesztett állományrendszereket csatolhatjuk le a beillesztési pont vagy a blokkeszköz-meghajtó állomány ismeretében.

Kapcsoló	Jelentés
-a	Az állományrendszer-táblázatban leírt összes állományrendszer lecsatolása.
-n	Lecsatolás a /etc/mtab frissítése nélkül.
-r	Ha nem sikerül a lecsatolás, újraillesztés csak olvasható módon.

A mount programmal beillesztett állományrendszereket az umount programmal csatolhatjuk le. A parancs használata egyszerű, paraméterként csak a beillesztési pontot kell megadnunk, ahonnan az állományrendszert el akarjuk távolítani.

Azokat az állományrendszereket azonban, amelyeken a folyamatok dolgoznak – azaz könyvtárakat vagy egyéb könyvtárbejegyzéseket tartanak nyitva –, nem tudjuk lecsatolni. (Valójában a GNU/Linux újabb változatai képesek használatban lévő állományrendszerek lecsatolására az umount -l kapcsolójával, de ennek a módszernek a tárgyalásától eltekintünk.) Ha mindenkorban le akarjuk csatolni az állományrendszert, meg kell tudnunk, mely folyamatok használják azt.

Az fuser program segítségével megállapíthatjuk, hogy egy állományrendszert mely folyamatok használják. A program használatát mutatja be a következő egyszerű példa.

31. példa *Le szeretnénk csatolni az állományrendszert, amely a /home/ könyvtárat hordozza. Állapítsuk meg, mely folyamatok használják!*

```
$ fuser -m /home
/home:          1489  1489c  1558  1560c
$
```

A kiírt folyamatazonosítók alapján a folyamatok adatai könnyen visszakereshetők.

Az fuser programnál sokkal kifinomultabb és összetettebb lsof program is alkalmaz a folyamatok által használt állományrendszerek visszakeresésére. Ezt láthatjuk a következő példában.

32. példa *Le szeretnénk csatolni a /cdrom/ beillesztési pontba beillesztett állományrendszert. Írassuk ki, mely folyamatok használják!*

```
$ lsof /cdrom
COMMAND  PID USER   FD   TYPE DEVICE SIZE NODE NAME
xterm    3772 root    cwd   DIR     8,1 16384      1 /cdrom/
bash     3774 root    cwd   DIR     8,1 16384      1 /cdrom/
```

Ügyeljünk arra, hogy az lsof paramétereként átadott könyvtár végére ne írunk / jelet!

4.2. Az állományrendszer-táblázat

Az állományrendszer-táblázat a számítógépen található állományrendszerekről tartalmaz fontos információkat. Ennek a táblázatnak a legfontosabb szerepe az, hogy meghatározza, mely állományrendszereket kell beilleszteni a rendszer indításakor.

A rendszer indításáról később bővebben olvashatunk, de már most néhány szót kell ejtenünk arról, hogyan történik az indítás során az állományrendszer beillesztése. A beillesztés a következő szakaszokból áll:

0. A legelső szakaszban a betöltött rendszermag beilleszti a gyökérkönyvtárat tartalmazó állományrendszeret. A beillesztés csak olvasható módban történik.

Erre a szakasra azért van szükség, mert a rendszerindításhoz, az állományrendszer ellenőrzéséhez és a további állományrendszerek beillesztéséhez szükséges programok és beállítóállományok ezen az állományrendszeren találhatók.

A beillesztés csak olvasható formában történik, hogy az állományrendszeret ellenőrizni lehessen. Írható formában beillesztett állományrendszeret ellenőrizni nem lehet, hiszen az ilyen állományrendszer az ellenőrzés során megváltozhat.

1. A következő szakaszban a GNU/Linux elvégzi a gyökérkönyvtárat tartalmazó állományrendszer ellenőrzését és az esetleg szükséges helyreállítását.

Ha az állományrendszer ellenőrzése nem talált hibát, vagy sikeresen helyreállította az állományrendszeret, a gyökérkönyvtárat tartalmazó állományrendszer újraillesztése is megtörténik írható és olvasható formában.

2. Az állományrendszer-táblázatban található bejegyzések alapján az egyéb állományrendszerek ellenőrzése is megtörténik. Ezek az állományrendszerek ebben a szakaszban még nincsenek beillesztve, de adataik az állományrendszer-táblázatból kiolvashatók.

Ha minden állományrendszer ellenőrzése (helyreállítása) megtörtént, az állományrendszerek beillesztése megtörténik az állományrendszer-táblázatban található adatok alapján.

GNU/Linux rendszereken az állományrendszer-táblázat a /etc/fstab állományban található. Az állomány hat, szóközökkel vagy tabulátor karakterekkel elválasztott oszlopot tartalmaz, minden sorban egy állományrendszer adataival. A következő oszlopok találhatók meg az /etc/fstab állományban:

1. Az első oszlop tartalmazza az állományrendszeret tároló háttértár meghatározását. Itt általában egy blokkeszköz-meghajtó állomány nevét találjuk, de az újabb rendszerek megengedik az állományrendszer jelölésére az állományrendszer címkéjének (*label*) megadását is.

Ha az állományrendszer nem köthető háttértárhoz (például proc állományrendszer), az első oszlopba a *none* kulcsszót írjuk.

2. A második oszlop az állományrendszer beillesztési pontja, az a könyvtár, amelyből kiindulva az állományrendszer tartalma látszani fog a beillesztés után.

A látszólagos memória meghatározásakor ide a swap kulcsszó kerül.

3. A harmadik oszlop tartalmazza az állományrendszer típusát. A használható elnevezések rövidítése a 72. oldalon található 3.14. táblázatban olvashatunk.
4. A negyedik oszlop az állományrendszerre vonatkozó kapcsolókat tartalmazza. Ezek a kapcsolók az állományrendszer viselkedését módosítják a beillesztés után.

A kapcsolókból vesszővel elválasztva többet is felsorolhatunk, de szóközt vagy tabulátor karaktert nem lehetünk közéjük, hiszen az az oszlopok elválasztására van fenntartva.

A használható kapcsolókra a későbbiekben visszatérünk.

5. Az ötödik oszlopot a dump nevű archiválóprogram használja annak előláncsere, hogy mely állományrendszereket kell archiválni.

Ha az oszlop értéke 0, akkor a program az állományrendszeret nem menti, ha attól különböző, akkor igen. Az itt található szám azt állítja be, hogy a dump hányadik indításakor kell az állományrendszeret menteni, azaz ha például hetente készítünk archívumot, hány hetente.

6. A hatodik mezőt az fsck parancs használja annak előláncsere, hogy mely állományrendszereket mikor kell ellenőrizni a rendszer indítása során.

Ha az itt található szám 0, akkor nem történik ellenőrzés, egyébként a 77. oldalon található felsorolás alapján az 1 vagy a 2 szám írható ebbe az oszloomba.

A hatodik mező legyen mindenkor 0 azoknál az állományrendszereknél, amelyeket az fsck nem tud induláskor ellenőrizni (hajlékonylemez, CD-ROM stb.).

Amint láttuk a mount -o kapcsolója után és az állományrendszer-táblázat negyedik oszlopában vesszővel elválasztva olyan kapcsolók adhatók meg, amelyek befolyásolják az állományrendszer viselkedését a beillesztés után. Igen sok kapcsoló létezik, ami ezeken a helyeken szerepelhet, ráadásul bizonyos állományrendszerek különleges – csak általuk támogatott – kapcsolókat is képesek használni. Nem tudjuk tehát megadni az összes lehetséges kapcsolót és jelentésüket, de törekszünk arra, hogy az összes állományrendszer által elfogadott, általános kapcsolók listáját és azok magyarázatait összefoglaljuk. A legtöbb feladat megoldására ezek a kapcsolók elegendőek, a többi kapcsolóról pedig a mount program dokumentációjában olvashatunk bővebben.

Az állományrendszerek beillesztés utáni viselkedését a következő kapcsolókkal befolyásolhatjuk:

ac1 A hozzáférési lista (ACL, *access control list*) engedélyezése az adott lemezrészben. Az ACL rendszer használatáról a 4.3. szakaszban az 82. oldalon olvashatunk bővebben.

async A lemezgyorsítótár használata írási műveletek esetén.

Ha ez a kapcsoló be van kapcsolva, az adott háttértárra az adatok írása nem történik meg azonnal, így a háttértár használata gyorsabb lesz. Ezt a kapcsolót a legtöbb háttértár esetében használni szoktuk.

sync Az **async** kapcsoló ellentéte.

Bekapcsolása esetén az adatok az írási művelet során azonnal az adathordozóra kerülnek. Így az adathordozó használata lassabb lesz, de az adatok nem veszhetnek el a lemezgyorsítótár miatt.

atime Az állományhasználat bejegyzése.

Ha a kapcsoló be van kapcsolva, az állományrendszer könyvtárbejegyzéseiből történő minden olvasáskor frissül az utolsó olvasást jelző érték. (Ez tehát *nem* az utolsó módosítás – az az utolsó írás dátuma és időpontja!)

A legtöbb állományrendszeren használjuk ezt a kapcsolót, így visszakereshető, hogy az egyes állományokat mikor olvasta a rendszer utoljára. Az állományrendszer használatát felgyorsíthatja ennek a kapcsolónak a kikapcsolása.

noatime Az **atime** ellentéte.

auto A kapcsoló a rendszerindításkor való automatikus beillesztést írja elő. Ha ez a kapcsoló be van kapcsolva, a rendszerindítás során a GNU/Linux megpróbálja beilleszteni az állományrendszert.

Ha az ilyen kapcsolóval ellátott állományrendszer beillesztése a rendszerindítás során nem sikerül, a rendszer nem indul el, hanem szervizüzemre kapcsol. A kivehető adathordozót használó állományrendszereket (hajlékonylemez, CD-ROM stb.) ezért soha ne lássuk el ezzel a kapcsolóval.

Valójában a kapcsoló hatása az, hogy a `mount` beilleszti az adott állományrendszert, ha a -a kapcsolót kapja. Mivel a rendszerindítást vezérlő BASH programok a `mount` program -a kapcsolójával illesztik be az állományrendszereket a megfelelő szakaszban, az automatikusan beillesztendő állományok a könyvtárszerkezetbe kerülnek.

noauto Az **auto** ellentéte.

defaults Ugyanaz a hatása, mintha a következő kapcsolókat használtuk volna: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser` és `async`.

dev Engedélyezi a karaktereszköz- és blokkesköz-meghajtó állományok használatát az adott állományrendszeren belül.

A legtöbb merevlemezen engedélyezzük az eszközkezelő állományok használatát – azaz bekapsoljuk ezt a kapcsolót –, de soha nem szabad engedélyeznünk olyan háttértáron, amelyeken keresztül a felhasználók a saját

anyagaikat illeszthetik be a könyvtárszerkezetbe. Ha megengednénk a felhasználóknak a hozott eszközkezelő állományok használatát, a megfelelően előkészített adathordozóval korlátlan hatalmat szerezhetnek a rendszer felett!

nodev A dev kapcsoló ellenére.

exec Programok (bináris programok) futtatásának engedélyezése az adott állományrendszerből. Ezt a kapcsolót a legtöbb merevlemezen használjuk.

noexec Nem engedélyezzük a bináris programok futtatását az állományrendszerből. Ezt általában biztonsági okokból tesszük, ha például el akarjuk kerülni, hogy a felhasználók programokat juttathassanak a számítógépbe.

Fontos tudnunk, hogy ez az kapcsoló nem tiltja a futás közben értelmezett nyelveken készült programok – például a héjprogramok – futtatását.

nosuid Nem engedélyezzük a SUID és az SGID bitek értelmezését, ami azt fogja eredményezni, hogy az innen indított programok mindenkor az indító felhasználó jogával rendelkeznek.

Az olyan eszközök esetében, amelyeket a felhasználók az általuk hozott állományrendszer beillesztésére használhatnak, a SUID és SGID bitek értelmezését ki kell kapcsolnunk, ellenkező esetben a felhasználók könnyedén teljhatalmat szerezhetnek a rendszer felett.

suid Engedélyezzük a SUID és SGID bitek használatát, amelyek bekapcsolt állapota esetén az indított program a programfájl tulajdonosának, illetve tulajdonoscsoportjának nevében, az ő jogával felhatalmazva fut.

A legtöbb merevlemez esetében ezt a kapcsolót bekapcsoljuk, de sohasem kapcsoljuk be kivehető adathordozót használó eszközöknél.

nouser Nem engedélyezzük a beillesztést és a lecsatolást a felhasználóknak, csak a rendszergazdának.

A legtöbb állományrendszer esetében ezt a kapcsolót bekapcsoljuk. Kivételek csak azok az eszközök jelentenek, amelyek a felhasználók által hozott állományrendszer beillesztésére szolgálnak.

rw Beillesztés olvasható és írható módban. A legtöbb állományrendszer esetén ez a kapcsolót használjuk, azaz lehetővé tesszük az állományrendszeren található adatok módosítását.

Ha az állományrendszer CD-ROM lemezen vagy írásvédett hajlékonylemezen található, a Linux a kapcsoló ellenére az állományrendszert csak olvasható módon illeszti be.

ro Beillesztés csak olvasható módban.

user A kapcsoló használata esetén a felhasználók is beilleszthetik és lecsatolhatják az állományrendszert. Ez az kapcsoló bekapcsolja a noexec, nosuid és nodev kapcsolókat is.

A kapcsoló használata esetén a beillesztett állományrendszert csak az a felhasználó csatolhatja le, aki a beillesztést végezte. A rendszergazda természetesen lecsatolhatja az ilyen kapcsolóval beillesztett állományrendszert is.

users Bárminely felhasználó elvégezheti a beillesztést és a lecsatolást. Ez a kapcsoló minden hatásában megegyezik a user kapcsoló hatásaival, azzal a különbséggel, hogy nem csak az a felhasználó végezheti el a lecsatolást, aki a beillesztést végezte, hanem bárki.

owner Az állományrendszer beillesztését és lecsatolását a rendszergazdán kívül az állományrendszert tartalmazó blokkeszköz-meghajtó állomány tulajdonosa is elvégezheti. Ez az kapcsoló bekapcsolja a noexec, nosuid és nodev kapcsolókat is.

Általában olyan programmal együtt használjuk ezt a kapcsolót, amely a felhasználó belépését figyelve a blokkeszköz-meghajtó tulajdonosát megváltoztatja. Az ilyen, belépést figyelő program meg tudja állapítani, hogy a bejelentkezés számítógép-hálózaton keresztül történt-e, és csak akkor változtatja meg a blokkeszköz-meghajtó tulajdonosát, ha helyi számítógéphasználatról van szó. Ezzel a kapcsolóval tehát elérhetjük, hogy a kivehető adathordozókat mindenig az a felhasználó használhassa, aki a számítógép előtt ül.

33. példa Egy GNU/Linux-ot futtató számítógépen az itt látható bejegyzések találhatók a /etc/fstab állományrendszer-táblázatban. Értelmezzük az egyes sorokat!

```
$ cat /etc/fstab
LABEL=/1      /
/dev/hda2    /usr        ext3  defaults          1  1
LABEL=/home   /home       ext3  defaults          1  2
none         /proc       proc   defaults          0  0
/dev/hda3    swap        swap   defaults          0  0
/dev/fd0     /mnt/floppy auto   noauto,owner    0  0
/dev/cdrom   /mnt/cdrom  iso9660 noauto,owner,ro 0  0
/dev/sda1    /mnt/pendrive auto   noauto,user,rw  0  0
```

Az első sor a gyökérkönyvtárat tartalmazó állományrendszer leírását adja. Figyeljük meg, hogy az állományrendszert az első lépében kell ellenőrizni, és minden biztonsági mentés során archiválni kell. Az első sorban azt is megfigyelhetjük, hogy a telepítőprogram az állományrendszer azonosítását az állományrendszeren elhelyezett címke alapján írta elő. Erre a LABEL= kulcsszót használhatjuk.

A második sor a /usr/ könyvtár tartalmát hordozó állományrendszer beillesztését írja elő. Ennek ellenőrzése már a második lépében történik, mentése azonban minden archiválás folyamán megtörténik. Figyeljük meg, hogy az állományrendszer beillesztése csak olvasható módon történik.

A harmadik sor a /home/ állományrendszer leírása.

A negyedik sorban láthatjuk a proc állományrendszer beillesztésére vonatkozó részét. Figyeljük meg, hogy mivel utánzott állományrendszerrel van szó, az első oszlopba a none kulcsszó került.

A ötödik sorban a látszólagos memóriát tartalmazó lemezrész, a csereterület leírása látható.

A következő három sor a kivehető adathordozók leírását tartalmazza. Ezeket az eszközöket a negyedik oszlop értelmében nem kell beilleszteni és az ötödik oszlop értelmében nem kell ellenőrizni a rendszer indításakor.

4.3. A hozzáférési lista használata

A GNU/Linux terjesztések újabb változatai támogatják a hozzáférési lista (ACL, *access control list*) használatát a könyvtárbejegyzések jogainak beállítására. E szakaszban az ACL rendszer használatáról olvashatunk.

Az állományok jogosultsága UNIX rendszereken meglehetősen egyszerű, ezért sokak szerint nem alkalmas arra, hogy kifinomult biztonsági rendszert készítsünk a felhasználásával. Az állományjogosultságok kifinomultabb beállítására készült az IEEE 1003.1e draft 17 ("POSIX.1e") ajánlástervezet, amelyből úgy tűnik, soha sem lesz már szabvány, és amelyet ennek ellenére egyre több UNIX rendszer támogat. A tervezet leírja, hogyan lehet az ACL rendszert ügyesen bevezetni a UNIX állományrendszerekbe.

A könyv írásának pillanatában elkészült az ACL-támogatás az extended 2 és az extended 3 állományrendszerhez, valamint elérhetővé vált az XFS állományrendszer, amely szintén támogatja az ajánlástervezetben leírt ACL szerkezetet. A GNU/Linux gyűjteményekben megjelent az ACL-támogatást tartalmazó rendszermag és egyszerűen elérhetővé váltak az ACL rendszer támogatására fejlesztett segédprogramok.

4.3.1. Az ACL-támogatás bekapcsolása

Az ACL kiegészítés bekapcsolása meglehetősen egyszerű. Az extended 2 és az extended 3 állományrendszerben egyszerűen meg kell adnunk az acl állományrendszer-kapsolót az állományrendszer beillesztésekor. Ezt mutatja be a következő példa.

34. példa Ha azt szeretnénk, hogy az extended 2 vagy az extended 3 állományrendszer a számítógép indítása után támogassa az ACL szerkezetek hasz-

4.3. tábla: getfacl

A program segítségével megtekinthetjük a könyvtárbejegyzések ACL bejegyzéseit.

```
getfacl [kapcsolók] könyvtárbejegyzések
```

A program segítségével kiírathatjuk a könyvtárbejegyzésekhez tartozó ACL bejegyzéseket, amelyek finomítják a könyvtárbejegyzések jogosultságainak rendszerét.

<i>Kapcsoló</i>	<i>Jelentés</i>
-----------------	-----------------

- | | |
|----|---|
| -d | Az alapértelmezett ACL bejegyzések kiíratása. |
| -R | Alkönyvtárakkal és azok tartalmával együtt. |

nálatát, helyezzük el az `acl` állományrendszer-kapcsolót az állományrendszer-táblázat (`/etc/fstab`) 4. oszlopában:

1	<code>/dev/hda3</code>	<code>/home</code>	<code>ext3</code>	<code>defaults,acl</code>	<code>1 2</code>
---	------------------------	--------------------	-------------------	---------------------------	------------------

A legközelebbi rendszerindítás után az ACL rendszer használhatóvá válik.

Ha nem akarjuk újraindítani a számítógépet egy ilyen csekélyseg miatt, az adott állományrendszert egyszerűen illesszük be újra menet közben:

```
$ mount -o,remount /home
$
```

A parancs kiadása után az ACL rendszer használhatóvá válik.

Még egyszerűbb a dolgunk az XFS állományrendszer esetében. Az XFS állományrendszer külön kapcsoló vagy beállítás nélkül támogatja az ACL rendszer használatát, nem kell tehát semmit tennünk ahhoz, hogy ACL bejegyzéseket használhassunk XFS állományrendszeren.

4.3.2. Az ACL bejegyzések használata

Ha bekapcsoltuk az ACL rendszer használatát – vagy XFS állományrendszert használunk –, a `getfacl` program segítségével kiírathatjuk a könyvtárbejegyzésekhez tartozó ACL bejegyzéseket. A következő példa ezt mutatja be.

35. példa Írassuk ki egy állományhoz tartozó jogokat az `ls` program és a `getfacl` program segítségével! Használunk olyan állományt, amelynek ACL bejegyzéseit nem módosítottuk!

```
$ ls -lh test.html
-rw-rw-r-- 1 pipas pipas 71K jún 1 20:56 test.html
$ getfacl test.html
# file: test.html
# owner: pipas
# group: pipas
user::rw-
group::rw-
other::r--
$
```

Figyeljük meg, hogy a getfacl program kimenet tulajdonképpen megegyezik az ls program kimenetével.

Az előző példából is látható, hogy az ACL rendszer által kezelt jogok tulajdonképpen ugyanazok a jogok, amelyekkel a hagyományos UNIX jogosultságkezelés során már megismerkedtünk. Az r, w és x betűk jelentése gyakorlatilag semmit sem változott. Az ACL által bevezetett újdonságok abban jelentkeznek, hogy ki-nek adhatunk jogokat.

Az ACL rendszer által létrehozott bővítés tulajdonképpen abban áll, hogy a segítségével gyakorlatilag tetszőleges számú listabejegyzést készíthetünk a könyvtárbejegyzésekhez. minden listabejegyzés jogokat tartalmaz, valamint annak a meghatározását, hogy a jogok kire vonatkoznak. Az ACL rendszer a jogokat a következő kulcsszavakhoz rendeli:

user A jogok felhasználóra vonatkoznak.

A felhasználó az állomány tulajdonosa, ha a bejegyzés nem tartalmaz felhasználói nevet, egyébként pedig a listabejegyzésben olvasható felhasználó.

group A jogok felhasználói csoportra vonatkoznak.

A jogok az állomány tulajdoncsoportjára vonatkoznak, ha a bejegyzés nem tartalmaz csoportnevet, egyébként pedig a bejegyzésben található csoportra.

other A bejegyzésben található jogok azokra a személyekre (folyamatokra) vonatkoznak, amelyekre egyetlen más ACL bejegyzés sem vonatkozik.

mask Az ilyen jellegű bejegyzések a megadható legmagasabb jogokat mutatják, amely jog tehát nem szerepel ebben a maszkban, azt nem kaphatja meg senki, hiába szerepel a jog másik bejegyzésben.

Kivétel képez ez alól az állomány tulajdonosa és az other kulcsszóval megadott csoport, akiknek a jogait a maszk nem korlátozza.

Vizsgáljuk meg a jogokat az előző példa alapján egy olyan állományon, amelynek jogait nem módosítottuk ACL bejegyzések hozzáadásával! A jogok az előző példában a következőképpen alakultak:

```
$ ls -lh test.html
-rw-rw-r-- 1 pipas pipas 71K jún  1 20:56 test.html
$ getfacl test.html
# file: test.html
# owner: pipas
# group: pipas
user::rw-
group::rw-
other::r--
$
```

Figyeljük meg, hogy az állomány ACL listája semmi újat nem mond a hagyományos jogosultságokhoz képest, és ez természetes is, hiszen nem adtunk ACL-bejegyzéseket az állományhoz.

Az ACL-bejegyzések hozzáadására a `setfacl` parancs használható. A program számára a jogosultságokat a következő formában adhatjuk meg:

`user:felh:jogok` Felhasználóra vonatkozó jogok megadása. Ha a felhasználó nevét elhagyjuk, a bejegyzés a könyvtárbejegyzés tulajdonosára vonatkozik.

A `user` kulcsszó helyettesíthető az `u` kulcsszóval.

`group:csop:jogok` Felhasználói csoportra vonatkozó jogok megadása. Ha a felhasználói csoport nevét elhagyjuk, a jogok a könyvtárbejegyzés tulajdonos-csoportjára vonatkoznak.

A `group` kulcsszó helyettesíthető az `m` betűvel.

`other:jogok` A jogok megadása azon felhasználók számára, akikre nem vonatkozik egyetlen más ACL-bejegyzés sem.

A `other` kulcsszó rövidíthető az `o` betűvel.

`mask:jogok` A maszk megadása. A maszkban található jogoknál több jog senkit sem illethet, kivéve a könyvtárbejegyzés tulajdonosát és azokat, akikre az `other` kulcsszóval megadott jogok vonatkoznak.

A `mask` kulcsszó rövidíthető az `m` kulcsszóval.

A következő példa azt mutatja meg, hogyan adhatunk ACL-bejegyzést a könyvtárbejegyzéshez.

36. példa Adjunk olvasási és írási jogot a könyvtárbejegyzéshez egy bizonyos felhasználó számára, majd vizsgáljuk meg, hogy a művelet sikeres volt-e! ACL-bejegyzés hozzáadására a `setfacl -m` kapcsolója használható.

```
$ setfacl -m u:joe:rw  test.html
$ getfacl test.html
# file: test.html
# owner: pipas
```

```
# group: pipas
user::rw-
user:joe:rwt
group::rw-
mask::rw-
other::r--
$
```

Amint látjuk, a könyvtárbejegyzéshez tartozó ACL a felhasználóra vonatkozó bejegyzéssel gyarapodott. Megfigyelhető az is, hogy a program egy bejegyzést hozott létre a mask kulcsszóval. A setfacl a maszkot automatikusan kezeli.

Ha most az ls program segítségével kiíratjuk a könyvtárbejegyzéshez tartozó jogokat, a kiírt sorban egy + jel jelzi, hogy a könyvtárbejegyzés jogait ACL bejegyzések módosítják:

```
$ ls -lh test.html
-rw-rw-r--+ 1 pipas pipas 71K jún 16 12:25 test.html
$
```

Ez az egy karakter jelzi, hogy a könyvtárbejegyzés jogait nem érhetjük meg kizárolag a hagyományos UNIX jogosultságrendszer segítségével.

A következő példa azt mutatja meg, hogyan távolíthatunk el ACL bejegyzéseket a könyvtárbejegyzéshez tartozó listából.

37. példa Szeretnénk eltávolítani az ACL bejegyzést, amely egy bizonyos felhasználó számára biztosít hozzáférési jogot a könyvtárbejegyzéshez. Ilyen esetben használhatjuk a setfacl -x kapcsolóját.

Vizsgáljuk meg az ACL bejegyzéseket, változtassuk meg, majd vizsgáljuk meg újra, hogy lássuk a különbséget!

```
$ getfacl test.html
# file: test.html
# owner: pipas
# group: pipas
user::rw-
user:joe:rwt
group::rw-
mask::rw-
other::r--
$ setfacl -x user:joe test.html
$ getfacl test.html
# file: test.html
# owner: pipas
# group: pipas
user::rw-
group::rw-
mask::rw-
other::r--
$
```

4.4. tábla: setfacl

Könyvtárbejegyzések ACL bejegyzéseinek módosítására szolgáló program.

```
setfacl [kapcsolók] [{-m|-x} acl] állományok
```

A program segítségével az ACL bejegyzésekhez újat adhatunk, az ACL bejegyzéseket módosíthatjuk vagy törlhetjük.

Kapcsoló	Jelentés
-m acl	Bejegyzés vagy bejegyzések módosítása.
-x acl	Bejegyzés vagy bejegyzések törlése.
-b	Az összes bejegyzés törlése.
-k	Alapértelmezett ACL bejegyzések törlése.
-d	Alapértelmezett ACL bejegyzések kezelése.
-R	Teljes könyvtárágak kezelése.

A -m és -x kapcsolók után egy vagy több - , jellel elválasztott – ACL bejegyzést adhatunk meg a következő formában:

ACL	Jelentés
[d[efault]:] [u[ser]:] felh [:jogok]	Felhasználóra vonatkozó jogok megadása.
[d[efault]:] g[roup]:csop [:jogok]	Felhasználói csoportra vonatkozó jogok megadása.
[d[efault]:] m[ask] [:] [:jogok]	A jogokat korlátozó maszk megadása.
[d[efault]:] o[ther] [:] [:jogok]	ACL bejegyzéssel nem rendelkező felhasználók jogainak megadása.

Amint látjuk, a program kizárálag a célbavett bejegyzést szüntette meg. Mivel minden felhasználóra csak egy ACL bejegyzés készíthető, a jogokat nem is kellett megadnunk.

A következő példa bemutatja, miképpen távolíthatjuk el az összes ACL bejegyzést egy lépében.

38. példa A könyvtárbejegyzéshez tartozó összes ACL bejegyzés egy lépében törlhető a setfacl -b kapcsolójával. Töröljük a bejegyzéseket, amelyek egy bizonyos könyvtárbejegyzéshez tartoznak!

```
$ ls -lh test.html
-rw-rw-r--+ 1 pipas pipas 71K jún 16 12:25 test.html
$ setfacl -b test.html
$ ls -lh test.html
```

```
-rw-rw-r-- 1 pipas pipas 71K jún 16 12:25 test.html
$ getfacl test.html
# file: test.html
# owner: pipas
# group: pipas
user::rw-
group::rw-
other::r--
$
```

Amint látjuk, a program törölte az összes ACL bejegyzést, ami a könyvtárbejegyzéshez tartozott.

Az ACL bejegyzések segítségével a felhasználók jogokat adhatnak egymásnak a saját könyvtáruk különböző könyvtárbejegyzéseihez. Valójában ezt az ACL előtti időkben is megtehették, abban az időben azonban sokszor a rendszergazda segítségére szorultak.

A hagyományos UNIX jogosultsági rendszer segítségével a jogosultságokat általában felhasználói csoportok létrehozásával kezeljük. Ha néhány felhasználó számára jogot szeretnénk adni egy könyvtárban elhelyezett állományokhoz, akkor általában létrehozunk egy felhasználói csoportot, a csoportban elhelyezzük a felhasználókat, és magának a csoportnak adjuk a megfelelő jogokat a könyvtárra. A jogosultságok ilyen módon való kiosztása a rendszergazda közreműködését igényli. Az ACL rendszer segítségével a rendszergazda remélheti, hogy a felhasználók nem kérik tőle felhasználói csoportok létrehozását minden projekt indításakor, hanem egyszerűen megadják egymásnak a közös erőforrások használatához szükséges jogokat a megfelelő ACL bejegyzések létrehozásával.



Lehetséges, hogy a legtöbb esetben a jogok kiosztásának felhasználókra való bízása nem több, mint pusztta vágyáalom a rendszergazda részéről.

Igaz ugyan, hogy az ACL bejegyzések kezelése meglehetősen egyszerű, a legtöbb felhasználó számára azonban felesleges és megtanulhatatlan számítógépes zagyvaság marad.

A következő példa bemutatja, hogyan adhat a felhasználó a saját könyvtárának egy részére olvasási és írási jogot egy közös munka elkezdésekor.

39. példa A joe nevű felhasználó egy közös munka megkezdésekor szeretne jogokat adni más felhasználók számára a közösen használt adatokhoz.

Hozzon létre a felhasználó egy könyvtárat a munkakönyvtáron belül, és helyezze el abban a munkához szükséges állományokat és alkönyvtárakat:

```
$ cd
$ mkdir kozos
$ mv *.pdf kozos/
$
```

Adjon most a felhasználó *x* jogot a saját könyvtárhoz a *fred* nevű felhasználónak, hogy az hozzáférhessen a közös könyvtárhoz. A jog megadásakor kínosan kell ügyelnünk arra, hogy csak az *x* jogot adjuk meg a felhasználónak.

```
-m user:fred:--x $HOME
[joe@localhost joe]$ getfacl $HOME
getfacl: Removing leading '/' from absolute path names
# file: home/joe
# owner: joe
# group: joe
user::rwx
user:fred:--x
group::---
mask::--x
other::---
```

Figyeljük meg, hogy a felhasználó saját könyvtárhoz mások (other) nem rendelkeznek semmilyen joggal.

Adjon most a felhasználó jogot a közös munkára kiszemelt könyvtárra és annak teljes tartalmára is:

```
$ -R -m user:fred:rwx,o:--- kozos/
$ getfacl kozos/ kozos/smilplugin.pdf
# file: kozos
# owner: joe
# group: joe
user::rwx
user:fred:rwx
group::rwx
mask::rwx
other::---

# file: kozos/smilplugin.pdf
# owner: joe
# group: joe
user::rw-
user:fred:rwx
group::r--
mask::rwx
other::---
$
```

Láthatjuk a könyvtár és a példaként kiszemelt állomány jogain, hogy a *-R* kapcsoló segítségével a *fred* nevű felhasználó írási és olvasási jogot is kapott a közös könyvtár teljes tartalmára. Ettől a pillanattól kezdve a *fred* nevű felhasználó részt vehet a közös munkában.

Új könyvtárbejegyzések ACL bejegyzései

Amikor egy új könyvtárbejegyzést hozunk létre, esetleg kissé terhesnek érezhetjük a megfelelő ACL bejegyzések létrehozását, ami nagyobb munkacsoportok, összetett jogosultsági rendszerek esetében időrabló lehet. A munka megkönnyítésének érdekében a könyvtárakhoz alapértelmezett ACL bejegyzéset hozhatunk létre a default kulcsszóval.

Az alapértelmezett ACL bejegyzéseket a könyvtárakban létrehozott könyvtárbejegyzések a létrehozásuk pillanatában megkapják, mondhatnánk öröklik. A könyvtárban létrehozott könyvtárbejegyzések ACL bejegyzéseit nem kell egyenként beállítanunk, ha a könyvtár alapértelmezett ACL bejegyzései megfelelően vannak beállítva.

Az alapértelmezett ACL bejegyzések használatát mutatja be a következő példa:

40. példa Hozzunk létre egy könyvtárat és adjunk ACL bejegyzést a könyvtárhoz! A könyvtárban a jövőben létrehozandó könyvtárbejegyzések ACL bejegyzéseinak beállítására készítsünk alapértelmezett ACL bejegyzéseket!

```
$ mkdir test  
$ setfacl -m u:joe:r-- test/  
$ setfacl -m d:u:joe:r-- test/  
$
```

Figyeljük meg, hogy a default kulcsszó rövidíthető az o betűvel!

Hozzunk létre most egy új állományt és egy új könyvtárat az alapértelmezett ACL bejegyzésekkel ellátott könyvtárban, és vizsgáljuk meg az új elemek ACL bejegyzéseit!

```
$ touch test/new_file  
$ getfacl test/ test/new_file  
# file: test  
# owner: fred  
# group: fred  
user::rwx  
user:joe:r--  
group::rwx  
mask::rwx  
other::r-x  
default:user::rwx  
default:user:joe:r--  
default:group::rwx  
default:mask::rwx  
default:other::r-x  
  
# file: test/new_file  
# owner: fred  
# group: fred  
user::rw-
```

```
user:joe:r--  
group::rwx          #effective:rw-  
mask::rw-  
other::r--  
$
```

Amint látjuk, a szabályos állomány örökölte a szülőkönyvtára alapértelmezett ACL bejegyzéseit, amint a létrehozott alkönyvtár is örökölte azokat. Az alkönyvtár azonban alapértelmezett ACL bejegyzésként is örökölte az alapértelmezett ACL bejegyzéseket. Ez azt jelenti, hogy az alapértelmezett ACL bejegyzések tetszőleges könyvtármélyséig kifejtik hatásukat.

Mivel a cp program másoláskor tulajdonképpen új állományokat és könyvtákat hoz létre, az alapértelmezett ACL bejegyzéssel ellátott könyvtárba másolt teljes könyvtárszerkezetek öröklik az alapértelmezett ACL bejegyzéseket.

5. fejezet

A rendszermag betöltése

A GNU/Linux rendszer indításának első lépése a rendszermag betöltése. Miután a BIOS elvégezte a számítógép legfontosabb hardverelemeinek ellenőrzését és beállítását, a beállításainak megfelelő háttértárat vizsgálatával megkeresi és betölti a rendszertöltő programot (*boot loader*). A rendszertöltő megkeresi, betölti és elindítja a Linux rendszermagot, ami a számítógépet a továbbiakban vezérli. Ebben a fejezetben a rendszertöltő programok működését és használatát vizsgáljuk meg.

A rendszertöltő programok ismerete fontos a GNU/Linux rendszert üzemeltető rendszergazdák számára, ha ugyanis valamelyen hiba miatt nem működik helyesen a rendszertöltő program, a GNU/Linux nem indul el, és a számítógép használhatatlanná válik.

GNU/Linux rendszerek számára többféle rendszertöltő is készült. Az IBM-megfelelő személyi számítógépek körében a két legelterjedtebb rendszertöltő program a LILO (*Linux loader*, Linux rendszertöltő)[1, 2, 22] és a GRUB (*grand unified bootloader*, nagy egysített rendszertöltő)[16]. Mindkét rendszertöltő program alkalmas Linuxon kívül más operációs rendszerek betöltésére is, így egy számítógépen – felváltva – több operációs rendszert is használhatunk.

E fejezetben a LILO és a GRUB rendszertöltő programokkal ismerkedhetünk meg.

5.1. A BIOS

A BIOS feladata a rendszerbetöltés szempontjából egyszerű: meg kell találnia a rendszertöltő programot valamelyik háttértáron, be kell töltenie a memóriába, és el kell indítania. Az indítás után az összes feladatot a rendszertöltő program végzi a rendszer indításáig.

A BIOS a betöltés előtt megvizsgálja a számítógép beállításait (*BIOS setup*), hogy megállapítsa, melyik háttértárról kell betöltenie a rendszertöltő programot. A beállítások alapján esetleg több háttértárat vizsgálva sorra megkísérli betölteni

a rendszertöltőt róluk, és az első sikeres kísérlet után elindítja azt, amit betöltött. A BIOS nem képes ellenőrizni a betöltött programot. Ha az indítás után nem működik – nem tölti be a rendszermagot –, a számítógép használhatatlan lesz, „nem indul”.

A hajlékonylemezek adatterületének elején elhelyezkedő 512 bájt méretű adatterületet betöltősávnak (*boot sector*) nevezünk. A BIOS ezt a területet tölti be, és az itt található programot indítja el, ha hajlékonylemezről indítjuk a számítógépet. A hajlékonylemezek kis méretűek, felépítésük igen keveset változott az elmúlt években, ezért a BIOS általában gond nélkül be tudja tölteni a rendszertöltő programot a hajlékonylemezről. A hajlékonylemezek fizikai meghibásodása az egyetlen ok, ami megakadályozhatja a rendszertöltést a hajlékonylemezről.

A rendszertöltésre alkalmas CD-ROM lemezeken általában hajlékonylemez térképet helyezünk el rendszertöltés céljára. A rendszer indításakor a CD-ROM lemezen található térkép alapján a BIOS hajlékonylemezét utánoz, azaz a rendszertöltés úgy történik meg, mintha hajlékonylemezről indítottuk volna azt. A BIOS CD-ROM-ról történő rendszerindításkor a szimulált hajlékonylemez első 512 bájtját olvassa be és indítja el, hasonlóan a hajlékonylemezknél használt módszerhez. Habár a rendszerindítás ebben az esetben bonyolultabb, mint a hajlékonylemezet használó esetben, általában problémamentes.

A merevlemezkről való rendszerindítás azonban már kissé bonyolultabb és összehasonlíthatatlanul több probléma forrása. Valószínűleg azért okoz annyi bosszúságot a merevlemezről való rendszerindítás, mert a merevlemezek az elmúlt években igen sokat fejlődtek, változtak.

5.1.1. A BIOS merevlemezcímzési módszerei

Amikor a számítógépet bekapcsoljuk, az egyes hardverelemeket kezelő szoftverösszetevők még nincsenek a memóriába töltve, azokat a rendszertöltő által betöltött Linux rendszermag tartalmazza. Amíg a rendszermagot be nem töltöttük és el nem indítottuk, a merevlemezek kezelésének nehéz feladata a BIOS-ra hárul.

A merevlemezek blokkos háttértárrak, ami egyrészről azt jelenti, hogy az adatmozgatást blokkokba szervezett módon végzik, másrészről pedig azt, hogy a merevlemez-vezérlőn keresztül a programok előírhatják, hogy a merevlemezen található adatblokkok közül melyiket akarják írni vagy olvasni. A merevlemez tehát címezhető háttértárra, azaz az adatblokkok elérése címük alapján történik olvasáskor és íráskor is.

A BIOS három számmal jelöli ki a merevlemezen található adatblokkokat, három számmal leírt címzési rendszert használ.

Az első szám a henger száma (*cylinder*), ami azt adja meg, hogy az adattárolásra használt korongan található koncentrikus sávok melyikén található a keresett adatblokk.

A második szám a fej száma (*head*), ami megadja, hogy a lemezegységek melyik író/olvasó fejével lehet elérni az adott adatblokkot. Mivel minden korongnak két oldala van, a fejek száma megegyezik a korongok számának kétszeresével.

A harmadik szám az elérni kívánt ív száma (*sector*), és azt adja meg, hogy a ke-resett adatblokk az egymás után sorakozó blokkok közül melyik, azaz a korong milyen elfordulása mellet olvasható az adatblokk.

Valójában a ma használatos merevlemezek képesek a három számra épülő cím-zési rendszer meghamisítására. A beállítástól függően például működhet úgy egy merevlemez, mintha kétszer annyi fejet és fele annyi hengert használna, mint a valóságban. A BIOS beállításaiban szereplő hengerek, fejek és ívek száma tehát ma már nem feltétlenül tükrözi a merevlemez fizikai felépítését.

5.1.2. Rendszerindítás merevlemezről

A merevlemez első 512 bájtnyi területe a betöltősáv. Mivel a merevlemezeken több betöltősáv is helyet kapott, e területnek a neve elsődleges betöltősáv, rövi-dítve MBR (*master boot record*). Az elsődleges betöltősávban foglal helyet a 446–510. bájt közti 64 bájtos területen a lemezrész-táblázat is. A rendszertöltő programok íróinak tehát helyet kell szorítaniuk a lemezrész-táblázat számára is, ha a rendszertöltő programot az MBR területen kívánják elhelyezni.

A merevlemezeken azonban nem csak az elsődleges betöltősávban helyezhető el a rendszertöltő program. minden lemezrész elején 512 bájtnyi betöltősáv-terület van fenntartva a rendszertöltő programok számára. Azt mondjuk, hogy nem csak a merevlemeznek van betöltősávja (az MBR), hanem minden lemezrésznek is.

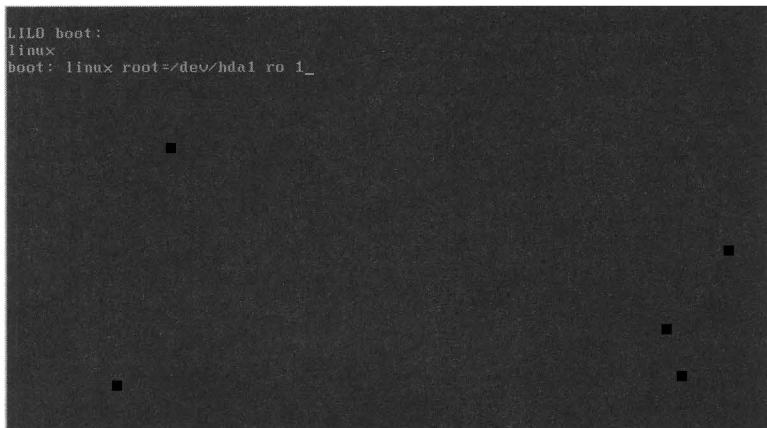
A legtöbb számítógépen a BIOS segítségével beállítható az is, hogy a merevlemez mely betöltősávját töltse be és indítsa el a BIOS a rendszerindítás első szaka-szában. Léteznek olyan számítógépek is, amelyek esetében a BIOS a lemezrész-táblázatból kiolvassa, melyik lemezrész aktív, azaz annak a lemezrésznek a betöl-tősávját tölti be, amelyik lemezrészét a lemezrészek kialakítása során betölthető-ként megjelöltük.

Amint láthatjuk, az Intel alapú személyi számítógépek esetében a rendszertöltő számára rendelkezésre álló terület 512 bájtnál kisebb. Ez komoly feladat elé állítja a rendszertöltő programok készítőit, és ez – amint a következő részekben látni fogjuk – rányomja a bélyegét a rendszertöltő programok viselkedésére.

5.2. A LILO

A LILO programcsomag segítségével rendszertöltő programot telepíthetünk haj-lékonylemezre, CD-ROM lemezen elhelyezett hajlékonylemez-térképre, az első merevlemez elsődleges betöltősávjára, illetve bármely elsődleges lemezrész be-töltősávjára.

A LILO csomag által készített és telepített rendszertöltő képes betölteni a Linux rendszermagot vagy más operációs rendszert a felhasználóval kialakított kapcso-lat alapján.



5.1. ábra. A LILO rendszertöltő használata

5.2.1. A rendszertöltő használata

A LILO által telepített rendszertöltőt igen egyszerű használni. Ki kell választanunk, hogy melyik operációs rendszert akarjuk elindítani, és be kell írnunk a megfelelő kulcsszót. Ha paramétert akarunk átadni az elindított rendszermagnak, azt is meg kell adnunk.

Amikor elindul a rendszertöltő, a beállításoktól függően egy parancskérő jelet ír a képernyőre (boot:). Ha a parancskérő jel nem jelenik meg, a felhasználó kérheti a **[Shift]** billentyű lenyomásával.

A parancskérő jel megjelenése után a **[Tab]** billentyű hatására a rendszertöltő kiírja a képernyőre, hogy milyen lehetőségeket ismer. A lehetőségek egy-egy operációs rendszer betöltését jelentik, de előfordulhat, hogy ugyanazt az operációs rendszert többféle üzemmódban is elindíthatjuk.

A **[Tab]** billentyű hatására megjelenő listából ki kell választanunk azt a rendszert, amelyiket el szeretnénk indítani és be kell írnunk a billentyűzet segítségével. Ha az indítás során a betöltendő rendszermagnak paramétereit szeretnénk átadni, azokat az alternatíva neve után begépelhetjük. Amint lenyomjuk az **[Enter]** billentyűt, a rendszertöltő megkíséri elindítani a megadott rendszert.

A LILO rendszertöltő használatát a 5.1. ábra mutatja be, ahol láthatjuk, hogyan adhatunk át paramétereit a rendszermagnak az indítás során.

Tudnunk kell, hogy a legtöbb esetben valamelyik rendszer alapértelmezettként van beállítva. Ha ezt akarjuk betölteni, elegendő lenyomnunk az **[Enter]** billentyűt, és a rendszertöltő betölti és elindítja a rendszermagot.

Fontos tudnunk azt is, hogy a rendszertöltő beállításai között általában szerepel egy időkorlát. Ha a rendszerindításkor a felhasználó nem nyom le egyetlen billentyűt sem, az időkorlát lejárta után a rendszertöltő automatikusan elindítja

az alapértelmezett rendszert. Az időkorlát lehetővé teszi, hogy a rendszertöltés automatikusan – emberi beavatkozás nélkül – megtörténjen.

A LILO rendszertöltő újabb változatai menüs felhasználói felületet biztosítanak a rendszerindításkor. A menüs felhasználói felület használata szintén nagyon egyszerű, a menü akár előképzettség nélkül is használható.

5.2.2. A rendszertöltő működése

A LILO által telepített rendszertöltő program működését alapjában véve a következő két tény határozza meg:

- A rendszertöltő programnak kevesebb, mint 512 bájton kell elférnie. Ez lehetetlenné teszi, hogy a rendszertöltő program összetett műveleteket megvalósító, bonyolult program legyen, sok adatot tartalmazzon és természeten azt is, hogy a rendszertöltő program tartalmazza a Linux rendszermagot, amelyet elindítani hivatott.
- Amikor a rendszertöltő program fut, a Linux rendszermag még nincs betöltve, a rendszertöltő program tehát nem élvezheti a Linux rendszermag támogatását. Mivel az állományrendszeret az alkalmazások a rendszermag segítségével érik el, és a rendszermag elindítása éppen a rendszertöltő program feladata, a rendszertöltő nem használhat állományokat.

A rendszertöltő programnak tehát állományok használatára van szüksége, de állományrendszer nem használhat. A megoldást az jelenti, hogy a LILO feljegyzi a betöltési folyamat során szükséges állományok adatblokkjainak fizikai elhelyezkedését, amikor a Linux rendszermag fut, azaz az állományrendszer használható, a betöltés során pedig az így elkészített térkép alapján keresi meg az állományok adatblokkjait.

Valójában a LILO kétszintű térképet használ a helyhiány miatt. Az állományok adatblokkjainak elhelyezkedését egy térképállományban helyezi el, a betöltő-sávba pedig a térképállomány adatblokkjainak elhelyezkedését jegyzi fel. A módszer hasonlít az extended 2 állományrendszerben használt közvetett blokkcímzéshez: az adatblokokban feljegyzett címek adatblokokra mutatnak.

A kétszintű térképrendszer segít az állományok elérésében, de a betöltést végző program még így sem fér el a betöltősvában. A betöltősvában egyszerűen túl kicsi a Linux rendszermagot betöltő és elindító program számára. A LILO megoldásként kétszintű rendszertöltőt használ.

A kétszintű rendszertöltő első szintje a program, ami a betöltősvában található. Ez a program a térképek alapján nem a rendszermagot, hanem egy másik programot, a rendszertöltő második fokozatát tölti be és indítja el. A rendszertöltő második fokozatának a feladata a Linux rendszermag betöltése és elindítása.

A betöltősváv kitöltése, az elsődleges rendszertöltő felmásolása és a térképek elkészítése a LILO részét képező lilo program feladata. Mivel ez a program készíti

5.1. tábla: lilo

A LILO rendszertöltő programjának felmásolása a beállítóállományban meghatározott helyre, a térképek elkészítése.

`lilo [kapcs]`

A program telepíti az elsődleges rendszertöltő programot a beállítóállomány által meghatározott helyre. A másolás során a program elkészíti vagy frissíti a térképet, amely alapján a rendszertöltő program képes megtalálni az állományrendszerben a szükséges állományokat a rendszermag támogatása nélkül is.

Kapcsoló *Jelentés*

- C fájl Beállítóállomány nevének megadása (alapértelmezés szerint `/etc/lilo.conf`).
- r kvt A program futása alatt az adott könyvtár legyen a gyökérkönyvtár.
- R parancs A rendszertöltő a legközelebbi indításkor (csak egyszer) az adott parancs alapján fogja betölteni a rendszermagot.

el az állományok elérésében oly fontos szerepet betöltő térképeket, szokás térkép-telepítőnek (*map installer*) is nevezni.

Fontos, hogy pontosan átlássuk, hogy a LILO hogyan végzi a Linux rendszermag betöltését és indítását, hiszen csak így van esélyünk arra, hogy megtaláljuk és kijavítsuk a hibát, ha a betöltés nem sikerül. A betöltés menete a következő:

1. A BIOS betölti a betöltősávot a memóriába és elindítja. Az elindított program az elsődleges rendszertöltő.
2. Az elsődleges rendszertöltő előkészíti a másodlagos rendszertöltő betöltését, majd kiírja a képernyőre az L karaktert.
3. Az elsődleges rendszertöltő betölti, majd kiírja az I betűt és elindítja a másodlagos rendszertöltőt.
4. A másodlagos rendszertöltő indulás után kiír egy L betűt.
5. A másodlagos rendszertöltő betölti a rendszermag adatblokkjainak eléréséhez szükséges térképet, majd kiír egy 0 betűt.
6. A másodlagos betöltő a felhasználó parancsára vagy automatikusan betölti a Linux rendszermagot és elindítja azt az esetleges paraméterek átadásával.

Amint látjuk, a LILO minden fontosabb lépést egy karakter kiírásával jelez. Hiba esetén a kiírt karakterekből következtethetünk arra, hogy hol akadt el a folyamat, mi a probléma. A következő lista a lehetséges hibajelenségeket sorolja fel.

„semmi” Ha a betöltés során semmi sem jelenik meg a képernyőn, a LILO egyetlen része sem töltődött be. Ez általában azt jelzi, hogy a betöltősáv hibás.

Megoldást jelenthet, ha hajlékonylemezről vagy CD-ROM lemezről indítjuk a gépet, és a `lilo` program futtatásával telepítjük az elsődleges rendszertöltőt a megfelelő helyre.

L hibakód Az elsődleges rendszertöltő elindult, de nem tudta betölteni a másodlagos rendszertöltőt. Az L betű után gyakran – de nem mindenkor – egy hibakódot ír az elsődleges rendszertöltő, és az is előfordulhat, hogy a hibakódot a végtelenséggel ismételgeti. A hibakódokat a későbbiekben részletesen ismertetjük.

Ezt a hibát többek között az is okozhatja, hogy a merevlemez BIOS beállításai megváltoztak a `lilo` térképtelepítő legutóbbi futtatása óta. Mivel az elsődleges rendszertöltő a futása alatt még nem támaszkodhat a Linux rendszermag szolgáltatásaira, kénytelen a BIOS eszközkezelő programjait használni a merevlemez kezelésére. Ha a BIOS merevlemezre vonatkozó beállítása, a merevlemez címzési módja megváltozott, előfordulhat, hogy a betöltősből található térkép nem létező adatblokkokra mutat. Ilyen esetben segíthet a BIOS beállítások meg változtatása.

L1 A másodlagos rendszertöltőt sikerült betölteni, de azt elindítani nem lehetett. Az elsődleges rendszertöltő olyan adatblokkokat olvasott a térkép alapján, amelyek nem a másodlagos rendszertöltőt tartalmazzák.

A hibát okozhatja az előző pontban bemutatott BIOS-beállítások, de a lemeztartalom megváltozása is. Ha a másodlagos rendszertöltő mozgatjuk, a térképet újra kell telepíteni. Ha ezt nem tesszük meg, a térkép a régi adatblokkokat jelöli, és ez a hiba léphet fel.

LIL? A másodlagos rendszertöltő rossz helyre töltődött be. Az ilyen hibát a BIOS-beállítások megváltozása, vagy a másodlagos rendszertöltő elhelyezésére használt állomány mozgatása okozhatja.

LIL A másodlagos rendszertöltő elindult, de nem tudja betölteni a térképállomány tartalmát.

Ez a hiba a merevlemezre vonatkozó BIOS beállítások megváltozása miatt következhet be.

LIL- A másodlagos rendszertöltő által betöltött térképállomány érvénytelen adatokat tartalmaz.

Ezt a hibát a BIOS merevlemezre vonatkozó beállításainak megváltoztatása, vagy a térképállomány mozgatása okozhatja.

Amint láttuk, bizonyos esetekben előfordulhat, hogy a LILO hibakódokat ír a képernyőre. A hibakódok kétszámjegyű számok tizenhatos számrendszerben. Fon-

tos, hogy tudjuk, hogy ezek a számok hibakódok, ha tehát a rendszertöltő vég-telen számsorokat ír a képernyőre ahelyett, hogy betöltené a Linuxot, a számok alapján megtalálhatjuk és kiküszöbölnihetjük a hibát. (Valójában a rendszertöltő a BIOS által visszaadott hibakódot írja a képernyőre, ezért a hibakódok jelentését a BIOS dokumentációjából is megismerhetjük.)

0x00 Belső programhiba.

Okozhatja ezt a hibaüzenetet az is, ha a BIOS nem támogatja az 1024. henger feletti lemezterületek elérését, és a betöltéshoz szükséges adatterületek a felső területen vannak.

Megoldást jelenthet a `linear` kulcsszó elhelyezése a LILO beállítóállományában és a `lilo` térképtelepítő újrafuttatása.

Ha ez a módszer nem segít, a merevlemez lemezrészéinek átszerkesztése szükséges. Alakítsuk ki a lemezrészeket úgy, hogy a `/boot/` könyvtárat tartalmazó állományrendszer teljes egészében az 1024. henger alatt helyezkedjen el.

0x01 Érvénytelen utasítás. A BIOS nem támogatja a háttértárat.

0x02 Címjel nem található. Általában lemezhibát jelöl. A LILO dokumentációja szerint érdemes néhányszor újrapróbálkozni, amihez mi csak annyit tennénk hozzá, hogy „egy új lemezzel”.

0x03 Írásvédett lemez. Ez a hiba csak írási művelet esetén fordulhat elő, amelyet a rendszertöltésnél ritkán, a rendszermagnak átadott paraméterek rögzítése során használunk.

0x04 Szektor nem található.

A hibát okozhatja a BIOS-beállítások megváltozása, esetleg segíthet a `compact` kulcsszó eltávolítása a beállítóállományból, illetve a `linear` kulcsszó elhelyezése a beállítóállományban.

0x06 Ideiglenes hiba. A LILO dokumentációja szerint érdemes újrapróbálkozunk.

0x07 A merevlemez kezdeti beállítása nem sikerült. A merevlemezre vonatkozó BIOS beállítások megváltoztatása vagy a kikapcsolás nélküli újraindítás segíthet.

0x08 DMA túlcordulás. A dokumentáció szerint érdemes újrapróbálkoznunk.

0x09 A DMA megkísérli átlépni a 64 kilobájtos határt. Próbáljuk meg eltávolítani a `compact` kulcsszót a beállítóállományból és újrafuttatni a `lilo` programot, hogy a rendszermag betöltése részletekben történjen.

0x0C Érvénytelen adathordozó. Lemezhibára utalhat.

0x10 CRC hiba. Lemezhibára utalhat, a hibás adatblokkok keresése és kiszűrése, az adathordozó cseréje segíthet.

0x11 Hibajavítás történt.

A rendszertöltő akkor is megszakítja a betöltési folyamatot, ha a hibajavítás az olvasás során sikeresen megtörtént. Érdemes újraindítani a betöltést, bár a lemez valószínűleg előbb-utóbb végervényesen tönkremegy.

0x20 A vezérlő meghibásodására utal.

0x40 Hengerkeresési hiba. A lemezmeghajtó író-olvasó feje nem találja a kere-
ssett hengert az adathordozón, ami a legtöbb esetben az adathordozó fizikai
meghibásodásának legbiztosabb jele.

0x80 A háttértár időtúllépése. A háttértár nem üzemkész, ami hajlékonylemez
esetében jelentheti azt, hogy a hajlékonylemezt eltávolítottuk. A legtöbb
esetben azt jelzi, hogy a hajlékonylemes egység vagy a merevlemez meg-
hibásodott, az adathordozót forgató motor nem indul.

0xBB BIOS hiba. A LILO dokumentációja szerint segíthet a `compact` kulcsszó eltá-
volítása, illetve a `linear` kulcsszó elhelyezése a LILO beállítóállományban.

A LILO állományai

A LILO által a betöltés során használt állományokat soroljuk fel ebben a szakasz-
ban. Az állományok neve és elérési útja a LILO beállítóállományban megváltoz-
tatható, ez a lista tehát csak az alapértékeket tartalmazza.

Az állományok közül jónéhányra a LILO rendszertöltő programjának van szük-
sége. Ezeket az állományokat a rendszertöltő a `lilo` térképtelepítő program által
készített térképek alapján tudja csak betölteni, ezért ha ezeket az állományokat
módosítjuk, áthelyezzük vagy egyszerűen csak töröljük és visszaírjuk a lemezre, a
térképtelepítőt minden futtatnunk kell.



*Mindig futtassuk a `lilo` térképtelepítő programot, ha megváltozik a be-
töltéshez szükséges állományok fizikai helye a lemezen, vagy a merev-
lemez címzési rendszere (henger, fej, ív átszámítási módja), különben
a betöltés meghiusul. Ha nem futtatjuk a térképtelepítőt, a rendszertöltő az ál-
lományokat nem találja meg a merevlemezen.*

Még ma is használnak olyan BIOS-okat, amelyek nem képesek elérni a merev-
lemeznek az 1024. henger feletti területeit. Mivel a rendszertöltő nem élvezheti
a Linux rendszermag támogatását, a BIOS ezen hibája miatt nem minden gépen
képes betölteni a rendszermagot, ha a betöltéshez használt állományok (vagy egy
részük) az elérhetetlen – 1024. henger feletti – területen van.



Néhány BIOS nem képes az 1024. hengernél magasabban lévő címek kezelésére. Ha ilyet használunk, azoknak az állományoknak, amelyek a betöltés során szükségesek, kivétel nélkül az 1024-es határ alatt kell lenniük, különben a betöltés nem fog sikerülni.

Megoldást jelenthet, hogy a betöltéskor szükséges állományoknak nem kell a gyökérkönyvtárat tartalmazó lemezrészben lenniük, hiszen a betöltés során azokat a térkép alapján találja meg a rendszertöltő. Ha tehát a számítógépben található BIOS nem képes elérni a merevlemez felső területeit, a betöltéskor szükséges állományokat egy külön lemezrészben helyezzük el, ami teljes egészében az 1024 hengeres határ alatt található. Ez garantálja, hogy minden állomány teljes egészében elérhető legyen a BIOS számára is.

A LILO alapértelmezés szerint a következő állományokat használja a térkép telepítésekor és a rendszertöltés során.

`/etc/lilo.conf` A térkép telepítésekor használt beállítóállomány. Ennek az állománynak a rendszertöltés során nem kell elérhetőnek lennie.

`/boot/boot.b` A másodlagos betöltőt tartalmazó állomány. Ennek az állománynak elérhetőnek kell lennie a rendszertöltéskor.

`/boot/boot.NNNN` Amikor a `lilo` térképtelepítő felülírja a betöltősávot, annak eredeti tartalmát ebbe az állományba menti, hogy szükség esetén helyreállíthassuk.

`/boot/chain.b` A csatolt betöltőprogramot tartalmazó állomány. Ezt az állományt használja a LILO, ha valamelyik lemezrész elején található betöltősávot kell elindítania. Erre az állományra akkor van szükség, ha a LILO idegen – nem Linux – operációs rendszert indít.

`/boot/map` A térképállomány, amely alapján az elsődleges rendszertöltő megtállja a másodlagos rendszertöltőt és az megtalálja a Linux rendszermagot. Ennek az állománynak a rendszertöltés során elérhetőnek kell lennie.

5.2.3. A LILO beállítása

A LILO térképtelepítőjének beállítóállománya a `/etc/lilo.conf` állomány. A `lilo` térképtelepítő ez alapján az állomány alapján végzi el a rendszertöltő telepítését.

A beállító állomány általában egy általános érvényű bevezető szakaszból, majd egy vagy több alternatív részből áll, amelyek a rendszer indításakor választható indítási módokat írják le. A felhasználó az indításkor ezek közül az indítási módok közül választhat, így döntve arról, hogy melyik operációs rendszert – esetleg ugyanazon operációs rendszer melyik változatát – indítsa el.

A bevezető szakasz minden indítási módra érvényes, habár sok itt található parancsot megismételhetünk és ezzel felülbírálhatunk később az alternatívák felsorolásakor. A bevezető szakaszban a következő parancsokat szokás elhelyezni:

boot= Megadja, hogy a `lilo` hova telepítse az elsődleges rendszertöltő programot.

A kulcsszó után annak a fizikai eszköznek vagy lemezrésznek a blokkesköz-meghajtó állományát kell írnunk, amelynek a betöltősávjában a rendszer-töltő programot el akarjuk helyezni.

compact Használata esetén a rendszertöltő megkíséri a háttértár olvasását egy lépéssben elvégezni. Ez meggyorsítja a betöltést, de – mivel nem minden háttértárat lehet így kezelní – meghiúsíthatja a rendszer indítását.

default= Megadja, hogy melyik legyen az alapértelmezett indítási mód. A kulcsszó után az alternatíva nevét kell megadnunk.

delay= Megadja, hogy a rendszertöltő az indulása után hány tizedmásodpercet várakozzon a felhasználó beavatkozására.

Ha ez alatt az idő alatt a felhasználó lenyomja a `Shift` billentyűt, a rendszer-töltő parancskérő jelet ír a képernyőre és a felhasználó parancsára vár. Ha ez a várakozási idő 0 vagy nincs megadva a beállítóállományban, a rendszer-töltő a beállítóállomány prompt és `delay=` kulcsszavai által meghatározott módon viselkedik.

install= A kulcsszó után megadhatjuk a rendszertöltő által elindítandó másodlagos rendszertöltőt tartalmazó állomány nevét.

A másodlagos rendszertöltő általában a `/boot/boot.b` állományban található.

linear A parancs hatására a térképkészítő a rendszertöltéskor szükséges állományok adatblokkjainak elhelyezkedését nem az adatblokkok sorszámaival jegyzi fel.

Nem a szokásos henger-fej-ív hármast használja tehát a térképtelepítő a térképek elkészítésekor, hanem azt jegyzi fel, hogy az adott adatblokk a merevlemez hányadik adatblokkja. A henger-fej-ív hármás meghatározását ilyen esetben a rendszertöltő végzi el közvetlenül a rendszer betöltésekor.

A rendszertöltés ennek megfelelően akkor is zavartalanul elvégezhető, ha a merevlemez címzési rendszerét átállítjuk, hiszen a rendszertöltő lekérdezi a címzési rendszert, és az alapján számítja át a sorszámot konkrét henger-fej-ív címhármassá.

Vegyük észre, hogy ezt a címzési rendszert használva a LILO térképtelepítője nem tudja meghatározni, hogy a betöltés során az adatblokkok melyik hengeren lesznek megtalálhatók, vagyis nem tudja ellenőrizni, hogy azok

a BIOS-ok, amelyek nem képesek betölteni az 1024. henger feletti területeket, be tudják-e tölteni majd az adatblokkokat.

lba32 A 32 bites LBA logikai blokkcímzés (*logical block addressing*) használatának engedélyezése, ha azt a BIOS támogatja.

lock A kulcsszó arra utasítja a rendszertöltő programot, hogy a felhasználó által a rendszermagnak adott paramétereket jegyezze meg, és a következő induláskor ismét adjá át a rendszermagnak.

map= Megadja a térképállomány nevét a könyvvártászerkezetben. A térképállomány a rendszertöltő által használt összes állomány (másodlagos rendszertöltő, rendszermag stb.) elhelyezkedését tartalmazza, alapértelmezés szerint /boot/map néven érhető el.

message= Egy szöveges állomány nevét adhatjuk meg a kulcsszó után. A szöveges állomány tartalma megjelenik a képernyőn, mielőtt a felhasználó kiválasztja az általa használni kívánt alternatívát.

prompt Előírja a rendszertöltő program számára, hogy mindenkorban kezdeményezzen kapcsolatot a felhasználóval, akkor is, ha az a rendszertöltő program indulásakor nem tartja nyomva a **Shift** billentyűt. Ha ez a kulcsszó szerepel a beállítóállományban, a rendszertöltő mindenkorban parancskérő jelet ír a képernyőre.

delay= A kulcsszó után tizedmásodpercben adhatjuk meg, hogy a rendszertöltő mennyi ideig várakozzon a felhasználóra a parancskérő jel kiírása után. Ha 0 az értéke, a rendszertöltés csak az **Enter** hatására indul el.

Az általános szakasz után a beállítóállományban az egyes rendszerindítási lehetőségek felsorolása és beállítása következik. A lehetőségeket minden esetben a következő két kulcsszó egyike vezeti be:

image= A Linuxot indító lehetőségek mindegyike ezzel a kulcsszóval kezdődik. Az egyenlőségjel után a rendszermagot tartalmazó állomány nevét és elérési útját kell megadnunk.

other= A többi operációs rendszert indító lehetőségek kezdődnek ezzel a sorral.

A kulcsszó után annak a lemezrésznek a blokkszköz-meghajtó állománya szerepel, amelynek a betöltősváját a lehetőség elindítja.

Az általános szakaszban – a kezdősort után – általában a következő kulcsszavakat használjuk.

label= Az lehetőség neve, amelyet a rendszertöltő a tabulátor hatására a képernyőre ír, és amellyel kiválaszthatjuk és elindíthatjuk a betöltést.

append= A kulcsszó után kettős idézőjelek között paramétereket adhatunk a Linux rendszermagnak, amelyet a már meglévő paraméterekhez akarunk fűzni.

A rendszermagnak átadható paraméterekre a későbbiekben még visszatérünk.

literal= A parancs segítségével ugyanúgy adhatunk meg paramétereket a rendszermag számára, mint az **append=** kulcszó esetében, de ez a kulcsszó törli az esetlegesen már megadott paramétereket.

password= Jelszó, amellyel védhetjük a rendszer indítását. Az adott rendszert csak az a felhasználó indíthatja el, aki ezt a jelszót ismeri.

Fontos tudnunk, hogy a rendszer indításakor a GNU/Linux védelmi rendszere még nem fut, ezért indításkor ez a jelszó védi a rendszert az illetéktelen behatolástól.

Ez a kulcsszó a beállítóállomány általános részében is használható, ha egy jelszóval akarjuk védeni az összes lehetőséget.

restricted A kulcsszó hatására a rendszertöltő csak akkor kéri a jelszót, ha az indítás során a felhasználó paramétereket próbál adni a betöltendő rendszermagnak.

Ez igen hasznos kulcsszó, hiszen a használatával elérhető, hogy a rendszert bárki elindíthassa, de a védelmi rendszer megkerülésére használható paraméterezést csak a jelszót ismerők érjék el.

read-only Ez a kulcsszó arra ad parancsot, hogy a rendszermag indítása után a gyökérkönyvtárat csak olvasható módon illessze be. A gyökérkönyvtárat általában a rendszerindítást végző BASH programok illesztik be újra írható módban az ellenőrzés után.

read-write A **read-only** kulcszóval ellentétben a gyökérkönyvtár írható formában történő beillesztésére ad utasítást.

root= Annak a blokkész köz-meghajtó állománynak a neve, amely a gyökérkönyvtárat hordozó állományrendszer jelöli ki.

vga= A kulcsszó segítségével meghatározhatjuk, hogy milyen üzemmódba állítsa a rendszermag a VGA csatolót, azaz milyen felbontású legyen a képernyő a rendszermag indítása után. A kulcsszó után adható értékekkel az 5.4. részben olvashatunk bővebben.

41. példa A következő példa egy olyan lilo.conf állományt mutat be, ami kétféle indítási módot biztosít:

```

1 boot=/dev/hda
2 map=/boot/map
3 install=/boot/boot.b
4 prompt
5 timeout=50
6 linear
7 default=linux
8
9 image=/boot/vmlinuz-2.2.14-5.0smp
10      label=linux
11      read-only
12      root=/dev/hda3
13
14 image=/boot/vmlinuz-2.2.14-5.0
15      label=linux-up
16      read-only
17      root=/dev/hda3

```

Láthatjuk, hogy a `linux` és a `linux-up` címekkel ellátott indítási lehetőségeket tartalmazza az állomány. Mindkettő az `image=` kulcsszóval készült, tehát minden kettő Linux indítására szolgál. Ha megvizsgáljuk a lehetőségeket, akkor láthatjuk, hogy azok csak a rendszermagban különböznek. Ezt a Linuxot tehát kétféle rendszermaggal lehet elindítani.

5.2.4. Indítólemez készítése

A LILO segítségével viszonylag egyszerűen készíthetünk indítólemezt, amelynek segítségével hiba esetén el tudjuk indítani számítógépünket. Az indítólemez egy rendszertöltő programot tartalmaz, valamint a Linux rendszermag egy példányát. Ha a számítógépet egy ilyen módon felszerelt hajlékonylemezeiről indítjuk, a rendszertöltő a hejlékonylemezről betölti a rendszermagot, az pedig a szokásos módon elindítja a teljes GNU/Linux rendszert a merevlemezről.

A rendszermag betöltése után akár el is távolíthatjuk a hajlékonylemezt, a rendszermag ugyanis a memoriában marad egészen a számítógép kikapcsolásáig.

42. példa A következő program a LILO segítségével hoz létre indítólemezt, ami a rendszer hajlékonylemezeiről való elindítását teszi lehetővé.

```

1 #!/bin/bash
2 #####
3 # Indítólemez létrehozása a Linux rendszermag      #
4 # hajlékonylemezre másolásával (lilo).          #
5 #####

```

```
6 #
7 #
8 # A hajlékonylemez blokkész köz-meghajtó állománya.
9 #
10 DEVICE=/dev/fd0
11 #
12 # A hajlékonylemez beillesztési pontja az indítólemez
13 # létrehozása alatt.
14 #
15 MDIR=/mnt/floppy
16 #
17 # A rendszermag, ami az indítólemezre kerül.
18 #
19 KERNEL=/boot/vmlinuz
20 #
21 #
22 # Létrehozzuk az állományrendszeret, és beillesztjük
23 # azt a könyvtárszerkezetbe. A beillesztés során
24 # engedélyezzük a meghajtóállományok használatát!
25 # (Csak rendszergazdák számára működik...)
26 #
27 mke2fs $DEVICE
28 mount -o,dev -t ext2 $DEVICE $MDIR
29 #
30 #
31 # Létrehozunk néhány könyvtárat a hajlékonylemezen.
32 #
33 mkdir $MDIR/boot $MDIR/dev $MDIR/etc
34 #
35 #
36 # Felmásoljuk a rendszermagot, és létrehozunk két
37 # blokkész köz-meghajtó állományt. mindenkorban létre
38 # kell hoznunk a lilo.conf-ban hivatkozott
39 # eszközkezelő állományokat!
40 #
41 cp $KERNEL $MDIR/boot
42 mknod $MDIR/dev/fd0 b 2 0
43 mknod $MDIR/dev/hda1 b 3 1
44 #
45 #
46 # Létrehozzuk a lilo beállítóállományát a
47 # hajlékonylemezen.
48 #
49 cat >$MDIR/etc/lilo.conf <<EOF
```

```

50 boot=/dev/fd0
51 timeout=100
52 prompt
53 image=/boot/vmlinuz
54         label=linux
55         root=/dev/hda1
56 EOF
57
58 #
59 # Felmásoljuk a másodlagos rendszertöltőt.
60 #
61 cp /boot/boot.b $MDIR/boot/
62
63 #
64 # Futtatjuk a térképtelepítőt. Számára a
65 # hajlékonylemez beillesztési pontja lesz a
66 # gyökérkönyvtár!
67 #
68 lilo -r $MDIR
69
70 #
71 # Kész van a lemez, lecsatoljuk a könyvtár-
72 # szerkezetről.
73 #
74 umount $MDIR

```

A bemutatott program által készített indítólemez a /dev/hda1 lemezrészén keressi a gyökérkönyvtárat tartalmazó állományrendszeret. Ha a gyökérkönyvtár nem ezen a lemezrészben található, a rendszertöltés után a Linux rendszermag leáll, mivel nem találja a szükséges állományokat.

Ilyen esetben a root= paraméterrel próbálkozhatunk, amellyel meg kell adnunk a rendszermagnak, hogy melyik lemezrészben található a gyökérkönyvtár.

5.3. A GRUB

A GRUB (*grand unified boot loader*, nagy egyesített rendszertöltő) egy rendszer-töltő, amely többek között képes a GNU/Linux operációs rendszer indítására is. A program igen fejlett, rugalmas és könnyen kezelhető, ezért egyre jobban terjed, egyre több terjesztés része.

A GRUB legfontosabb előnye a legtöbb rendszertöltővel szemben, hogy képes kezelni a legelterjedtebben használt állományrendszerket. Ez azt eredményezi,

hogy a GRUB már a Linux rendszermag indítása előtt is képes állományokat kezelní, így a használata és a működése sokkal egyszerűbb. Ha például a rendszermagot letöröljük, és újra létrehozzuk ugyanazon állománynév alatt, nem kell semmilyen további lépést tennünk annak érdekében, hogy a rendszertöltő megtalálja, hiszen az állományrendszeren belül az állomány neve nem változik.

Az állományrendszer kezeléséből fakad a GRUB azon kellemes tulajdonsága is, hogy a beállítóállomány megváltoztatása után nem kell semmiféle programot futtatnunk, hogy azt érvényesítsük. Sőt, a GRUB segítségével a rendszermag betöltése előtt a beállításokat szabadon megváltoztathatjuk. Ez azt jelenti, hogy a GRUB képes olyan rendszermagot is betölteni, amelynek létezéséről csak közvetlenül a betöltés előtt értesül! Ez igen hasznos lehet a rendszer helyreállítása során.

A GRUB képes kezelni az extended 2, extended 3 állományrendszereket, a ReiserFS állományrendszert, valamint a 16 és 32 bites FAT állományrendszereket.

Fontos tudnunk azonban, hogy a GRUB kezeli ugyan az állományrendszereket, de arról nincs információja, hogy az egyes állományrendszerek hogyan épülnek a könyvtárszerkezetbe, azaz nem tudja, hogy az egyes állományrendszereknek mi a beépítési pontja. Számára nem egy összefüggő fa létezik ezért, hanem annyi önálló fa, ahány állományrendszer van.

5.3.1. Lemezrészek és állományok megadása

Már láttuk, hogy a GRUB számára az állományokat az állományrendszerek helyével, vagyis a háttértárak és lemezrészek meghatározásával együtt kell megadunk. Ebben a szakaszban arról olvashatunk, milyen nyelvtani szabályok szerint adhatjuk meg az egyes állományok pontos helyét.

Az állomány nevének leírásakor először a lemezrészet kell megadnunk zárójelek között, majd az állomány pontos elérési helyét a lemezrészen belül. Ezt mutatja a következő példa:

```
splashimage=(hd0,0)/grub/image.xpm.gz
```

Itt a (hd0,0) a lemezrész neve, a /grub/ a lemezrészen belül található könyvtár neve, míg az image.xpm.gz az állománynév. Látható, hogy a lemezrészeket nem a szokásos módon – a /dev/ könyvtárban található blokkcsköz-meghajtó állományok nevével – adjuk meg (5.2. táblázat).

A zárójelen belül előbb a háttértár típusát kell megadnunk két betűvel, ami merevlemez esetén hd, hajlékonylemez esetén pedig fd. A GRUB nem tesz különbséget az IDE és SCSI csatolófelületen keresztül kapcsolódó lemezek között, egyszerűen sorra veszi az összes merevlemezt, és megszámozza őket.

A háttértár típusa után következik a sorszáma. A GRUB az eszközök számozását 0-tól kezdi, az első merevlemez sorszáma tehát 0, a másodiké 1 és így tovább.

A háttértár sorszáma után – ha nem az egész háttértárra hivatkozunk, hanem valamelyik lemezrészre – egy vesszővel elválasztva következhet a lemezrész sorszáma. A GRUB a lemezrészek sorszámozását 0-tól kezdi, a logikai lemezrészek sorszámozását pedig 4-től.

Név	Hardverelem
(fd0)	Az első hajlékony lemezes egység.
(hd0)	Az első merevlemezes egység.
(hd0, 0)	Az első elsődleges lemezrész az első merevlemezen.
(hd0, 4)	Az első logikai lemezrész az első merevlemezen.
(hd1, 0)	Az első elsődleges lemezrész a második merevlemezen.

5.2. táblázat. Háttértárak és lemezrészek GRUB nevei

5.3.2. A rendszertöltő használata

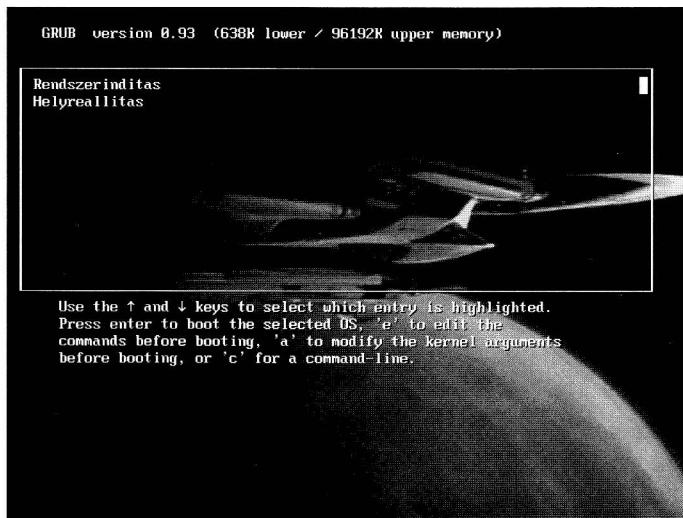
Amikor a GRUB által telepített rendszertöltő elindul, a felhasználónak módjában áll beavatkozni a rendszer betöltésének folyamatába. Általában a 5.2. ábrán látható formájú menü jelenik meg, amelyből a kívánt indítási mód, a kívánt operációs rendszer kiválasztható. A rendszertöltő használata közben a következő billentyűket használhatjuk:

- [Esc] Ha a menü a beállítások miatt nem jelenne meg, ezzel a billentyűvel kérhetjük a megjelenítését.
- [P] Ha a rendszertöltő jelszóval védett, a felhasználó ezzel a billentyűvel kezdeményezheti a jelszó megadását. Amikor a felhasználó beírta a helyes jelszót, a rendszertöltő különleges szolgáltatásai elérhetővé válnak.
- [C] A GRUB parancssor indítása. A GRUB parancssor jónéhány hasznos parancssal kényelmes kezelői felületet biztosít a rendszer indításához és a rendszertöltési folyamat zavarainak elhárításához.
A parancssorból kilépni, a menüt újra megjeleníteni az [Esc] billentyűvel lehet.
- [E] A menüben kiválasztott menüpont szerkesztését kérhetjük ezzel a billentyűvel.
- [B] Ha a menü szerkesztésével végezhetünk, ezzel a billentyűvel indíthatjuk az átszerkesztett menüpont betöltését.

A rendszertöltő programban a parancssor használata majdnem teljesen megegyezik az elindított rendszeren használható grub program viselkedésével. A GRUB parancssor meglehetősen sok hasznos parancsot ismer, amelyek közül jónéhány kimondottan a rendszer helyreállításának megkönyítésére szolgál.

A GRUB parancssor itt következő parancsai hasznosak lehetnek, ha a rendszertöltő parancssorát indítottuk el:

cat állomány Az állomány tartalmát a képernyőre írja.



5.2. ábra. A GRUB rendszertöltő használata

cmp *állomány1* *állomány2* A két állomány tartalmát összehasonlítja és kiírja, ha a két állomány különbözik egymástól.

configfile *állomány* Beállítóállományként tölti be az adott állományt.

find *állomány* Megkeresi és kiírja, hogy az adott állomány melyik háttértáron, melyik lemezrészben található. Az állománynévnek abszolút állományaíró-nak kell lennie, a parancs tehát nem arra ad választ, hogy az állomány melyik könyvtárban található, hanem arra, hogy melyik lemezegységen.

halt Leállítja a számítógépet.

help Felsorolja a parancsokat, vagy ha egy parancsot írunk be a help után, kiírja annak rövid használati utasítását.

install A parancs telepíti a rendszertöltő programot és létrehozza a szükséges térképeket.

A parancs használata meglehetősen bonyolult. Paraméterként meg kell adnunk az elsődleges rendszertöltő állomány nevét, a meghajtó nevét, ahová telepíteni szeretnénk a rendszertöltő programokat, a másodlagos rendszertöltő állomány nevét, illetve megadhatjuk a beállítóállomány nevét.

Az install bonyolultsága miatt használjuk inkább a setup parancsot, amely ezeket az elemeket automatikusan megkeresi, és azután indítja az install parancs végrehajtását.

quit Kilépés. Ezt a parancsot csak akkor használhatjuk, ha a programot a grub parancssal indítottuk. Ha a parancssorba a rendszerindítás során léptünk, a halt parancsot használhatjuk.

reboot A parancs újraindítja a számítógépet.

root meghajtó Beállítja, hogy melyik legyen az alapértelmezett meghajtó. Hasonlóképpen lehet használni, mint a héj cd parancsát.

Ennek a parancsnak köszönhetően kiadhatjuk például a cat /boot/grub.conf parancsot éppen úgy, mintha a héjprogramot futtatnánk.

setup meghajtó Automatikusan telepíti a rendszertöltőt a paraméterként megadott meghajtóra.

A parancs hatására a GRUB megkeresi a root parancssal beállított helyen a szükséges állományokat (elsődleges rendszertöltő, másodlagos rendszertöltő, beállítóállomány stb.)

Az utasítás valójában az állományok megkeresése után az install parancsot hajtja végre.

testvbe üzemmód Megpróbálja a grafikus csatolót az adott üzemmódba kapcsolni (VESA). Ha sikerül, egészen addig nem kapcsol vissza, amíg le nem nyomunk egy billentyűt. A tapasztalat azt mutatja, hogy a parancs csak akkor kapcsolja át a videókártya üzemmódját, ha az már grafikus üzemmódban van. Ezt elérhetjük a splashimage parancssal. Sajnos a GRUB jelenlegi változatánál azt is tapasztaltuk, hogy a videókártya üzemmódjának visszakapcsolása nem minden esetben sikeres, a számítógépet a próba után újra kell indítni.

Az üzemmód tízes és tizenhatos számrendszerben is megadható. A üzemmódokhoz tartozó kódok a 5.4. táblázatban találhatók a 120. oldalon.

vbeprobe Megvizsgálja és kiírja, hogy milyen grafikus módokat támogat a grafikus csatoló (VESA).

5.3.3. A GRUB beállítása

A grub beállítása a grub.conf állományban történhet. Az állományt a rendszertöltő tárolására kijelölt lemezegységen vagy lemezrészben helyezzük el a /grub/ vagy a /boot/grub/ könyvtárban. A GRUB által telepített rendszertöltő a beállítóállományt a neve alapján megkeresi és betölti, ami azt eredményezi, hogy a beállítóállomány módosítása után nem kell újratelepítenünk a rendszertöltőt. Ha módosítjuk a GRUB rendszertöltő beállításait, a módosítások a legközelebbi rendszerindításkor automatikusan érvényesülnek.

A GRUB beállítóállomány egy menürendszer ír le, amely a rendszertöltő számára meghatározza, hogy mit kell tennie. A beállítóállományban használhatjuk

a 5.3.2. részben leírt parancsokat, de további parancsok is a rendelkezésünkre állnak a menü felépítéséhez:

A beállítóállomány egy bevezető szakaszból és a menüpontok felsorolásából áll. A bevezető szakaszban a következő parancsokat szokás használni.

default szám A menük előtti bevezető részben helyezhetjük el ezt a parancsot.

Beállítja, hogy melyik menüpont legyen az alapértelmezett. A menük számozása 0-val kezdődik.

A szám helyén megadhatjuk a saved kulcsszót, amely mindenkor a savedefault által mentett értéket tartalmazza.

savedefault Az éppen aktuális menüpont számát menti a saved változóba. A menüpontokhoz tartozó végrehajtható utasítások között helyezzük el.

timeout mp A parancs segítségével megadhatjuk, hogy ha a felhasználó nem nyom le egyetlen billentyűt sem, hány másodperc után induljon el az alapértelmezett menüpont végrehajtása.

Ha 0 értéket adunk meg a parancs után, a rendszertöltő azonnal indítja a menüpontot, ellenben ha egyáltalán nem adjuk meg a timeout parancsot, a rendszertöltő tetszőlegesen hosszú ideig várakozik.

hiddenmenu Ha ezt a parancsot helyezzük el a menüpontok előtti bevezető részben, a menü nem jelenik meg. A felhasználó az billentyűvel kérheti a menü megjelenítését, amíg a beállított idő le nem jár.

password jelszó Ha valamelyik menüpontnál adjuk meg ezt a parancsot, a menüpont kiválasztása után a rendszertöltő jelszót kér és csak akkor hajtja végre a menüpont által előírt parancsokat, ha a felhasználó által begépelezett jelszó megegyezik a password parancs után begépelezett parancssal.

Ha a password parancs a menüpontok előtt, a bevezetőben szerepel, a GRUB csak a jelszó megadása után engedélyezi a beállítások megtekintését, szerkesztését és a rendszertöltő parancssorának használatát. A jelszó begépelését a felhasználó kezdeményezheti a billentyű lenyomásával.

Vegyük észre, hogy a menüpontban addig nincs értelme jelszót megadni, amíg az általános résznél meg nem adtuk, hiszen addig a felhasználó a password parancsot tartalmazó sort akár törlheti is az billentyűvel elérhető szerkesztőmódban.

A password parancs --md5 kapcsolója jelzi, hogy a jelszót MD5 kódolással adtuk meg. A jelszó kódolt formáját többek között az md5crypt GRUB parancssal is előállíthatjuk.

splashimage= állománynév A menü hátterében megjelenő képet tartalmazó állomány neve.

Az állománynak 640×480 méretű xpm formátumú képnek kell lennie 14 színnel. Az állományt tömöríthetjük a gzip programmal.

A bevezető szakasz után a menüpontok meghatározásai következnek egymás után. minden menüpontot ugyanaz a parancs vezet be.

title Szöveg... A menüpontot bevezető első sor, ami megadja a szöveget, amely a rendszertöltő menüjében megjelenik. A **title** parancs utáni teljes sor a menübe kerül. A **title** parancs utáni sorok a menüpont kiválasztásakor végrehajtandó utasítássorozatot alkotják.

A **title** a második, harmadik stb. menük pontok esetében lezárja az előző menüpontot alkotó utasításokat.

A menüpontokon belül a következő parancsokat használhatjuk a Linux rendszermag betöltésére:

kernel állomány kapcsolók A parancs hatására a rendszertöltő megkíséri betölteni az adott állományt. Az állomány neve után található szavakat a rendszertöltő a rendszermagnak paraméterként adja át.

A **kernel** parancs végrehajtása után a rendszermag a memóriában lesz, de nem indul el azonnal. A **boot** parancs indítja el a rendszermagot.

initrd fájlnév A Linux rendszermag lehetőséget ad arra, hogy egy külön állományból még a gyökérkönyvtár beillesztése előtt magmodulokat töltünk be.

Az **initrd** fájlnév parancs előkészíti a magmodulokat, és a megfelelő paraméterekkel jelzi a rendszermag számára, hogy be kell töltenie azokat. A parancs után a magmodulokat tartalmazó állomány nevét kell megadnunk. A rendszermag kezdeti moduljairól a 5.4. szakaszban olvashatunk bővebben.

Ezt a parancsot mindenkor a **kernel** parancs után használjuk.

boot A parancs hatására a rendszertöltő elindítja a már betöltött rendszermagot. Ezt a parancsot nem szükséges a menüben elhelyeznünk, mert a rendszertöltő a menü utolsó sora után automatikusan végrehajtja.

Idegen (nem Linux) operációs rendszerek betöltésekor szükségünk lehet a következő parancsokra:

rootnoverify meghajtó A parancs ugyanúgy használható, mint a **root** parancs, de az állományrendszeret nem kezeli. Akkor használjuk, ha olyan operációs rendszert akarunk betölteni, amelynek állományrendszerét a GRUB nem ismeri.

makeactive Az előzőleg a **rootnoverify** vagy a **root** parancsok segítségével beállított lemezrészét beállítja aktív lemezrésznek (*bootable*). Olyan operációs rendszerek esetében használhatjuk, amelyek csak az aktív lemezrészről tudnak betöltődni.

chainloader állomány A parancs idegen operációs rendszer rendszertöltő programjának betöltését végzi el.

Ha a parancs paramétereként a +1 kifejezést adjuk meg, a rendszertöltő egy 512 bájtos adatblokkot olvas a háttértár – vagy lemezrész – elejéről, azaz beolvassa a betöltősávot.

boot A boot parancs a chainloader által betöltött idegen rendszertöltőt is képes elindítani.

43. példa A következő beállítóállomány két operációs rendszert képes betölteni. A beállítóállományban elhelyezett megjegyzések megkönyyítik annak megértését.

```
1 # grub.conf: A GRUB rendszertöltő beállítóállománya
2 #
3 # A beállítások a következő elrendezést tükrözik:
4 #   o Idegen operációs rendszer a /dev/hda1 elsődleges le-
5 #     mezrészen. Az operációs rendszer betöltését a chain-
6 #     loader végzi, azaz az idegen rendszer saját betöltőszája
7 #     a /dev/hda1 lemezrész elején található.
8 #
9 #   o A Linux rendszer gyökérkönyvtára a /dev/hda3 lemezrészen
10 #     található.
11 #
12 #   o A Linux rendszeren belül a /boot/ külön lemezrészen ta-
13 #     lálható, amely a /dev/hda2.
14 #
15 default=0
16 timeout=50
17 splashimage=(hd0,1)/grub/splash.xpm.gz
18 title GNU/Linux (2.4.20-8)
19     root (hd0,1)
20     kernel /vmlinuz-2.4.20-8 root=/dev/hda3
21     initrd /initrd-2.4.20-8.img
22 title Masvalami
23     rootnoverify (hd0,0)
24     chainloader +1
```

A bemutatott beállítóállomány a gyakorlatban is használható, az egyes operációs rendszerek és könyvtárak helyét azonban szükséges lehet átírni!

5.3.4. Indítólemez készítése

A GRUB használatával is készíthetünk olyan rendszerindító lemezt, ami az esetleg szükséges helyreállítást lehetővé teszi.

44. példa A következő példaprogram hajlékonylemezre másolja a GRUB rendszertöltőt és a Linux rendszermagot, hogy a lemez segítségével elindíthassuk a számítógépet, ha a rendszermag vagy a rendszertöltő megsérült. Az elkezített rendszerindító lemez a gyökérkönyvtárat tartalmazó állományrendszert a /dev/hda1 lemezszeren keresi, de ez a rendszerindításkor könnyedén megváltoztatható.

```

1  #! /bin/bash
2
3 #
4 # A hajlékonylemez blokkeszköz-meghajtó állománya.
5 #
6 DEVICE=/dev/fd0
7 #
8 # A hajlékonylemez beillesztési pontja az indítólemez
9 # létrehozása alatt.
10 #
11 MDIR=/mnt/floppy
12 #
13 # A rendszermag, ami az indítólemezre kerül.
14 #
15 KERNEL=/boot/vmlinuz
16 #
17 # A GRUB stage1 és stage2 elsődleges és másodlagos rend-
18 # szertöltő programjainak fellelési helye.
19 #
20 IMAGES=/boot/grub
21 #
22 #
23 # Létrehozzuk az állományrendszert, és beillesztjük a könyvtár-
24 # szerkezetbe. A GRUB-nak nincs szüksége blokkeszköz-meghajtók-
25 # ra.
26 #
27 /sbin/mke2fs $DEVICE
28 mount $MDIR
29 #
30 #
31 # Létrehozzuk a boot/grub könyvtárakat a hajlékony-lemezen.
32 #
33 mkdir -p $MDIR/boot/grub

```

```
34 #
35 #
36 # Felmásoljuk az elsődleges és másodlagos rendszer-töltő prog-
37 # ramokat a hajlékonylemezre.
38 #
39 cp $IMAGES/stage* $MDIR/boot/grub/
40 cp $KERNEL /mnt/floppy/boot/vmlinuz
41
42 #
43 # Létrehozzuk a hajlékonylemezen a GRUB beállítóállományát.
44 #
45 cat <<EOF >$MDIR/boot/grub/grub.conf
46 #default=0
47 #timeout=10
48 title Inditas a~/dev/hda1 gyokerrel
49   root (fd0)
50   kernel /boot/vmlinuz ro root=/dev/hda1
51 EOF
52
53 #
54 # Lecsatoljuk a hajlékonylemezt.
55 #
56 umount /mnt/floppy
57
58 #
59 # Telepítjük a hajlékonylemezre a GRUB rendszertöltőt a root
60 # és setup parancsokkal. A parancsokat a grub hajtja végre
61 #
62 /sbin/grub --batch <<EOF
63 root (fd0)
64 setup (fd0)
65 quit
66 EOF
```

5.4. A rendszermag paraméterei

Amint azt az előző részekben már említettük, a Linux rendszermag paramétereit kaphat az indulásakor, éppen úgy, mint a felhasználói és rendszerprogramok a rendszer indítása után[10].

A paramétereket a rendszertöltő program adja át a Linux rendsermagnak, vagy azért, mert a beállításai tartalmazzák a paramétereket, vagy azért, mert a felhasználó, aki a rendszer indítását kérte, paramétereket adott át a rendszertöltőnek.

5.3. tábla: mkinitrd

A gyökérkönyvtár beillesztése előtt használatos térképállomány elkészítése.

`mke2fs [kapcs] térképállomány rendszermag_változatszáma`

A program segítségével létrehozhatunk egy térképállományt, amely a rendszermag számára a gyökérkönyvtár beillesztése előtt betöltendő magmodulokat tartalmazza. A program összegyűjti a különböző helyekről a rendszer által használt modulokat, és elhelyezi azokat a térképállományban.

Ha a parancssorban további modulokat adunk meg, azoknak csak a nevét kell beírnunk (például ext2), a program megkeresi az állományokat, amelyek az adott rendszermaghoz tartoznak.

Kapcsoló	Jelentés
<code>-f</code>	A meglévő térképállomány felülírása.
<code>--with=modul</code>	A kapcsoló után megadott modul is belekerül a térképállományba.
<code>--preload=modul</code>	Az adott modult a térképállományban úgy helyezi el, hogy a betöltésük az SCSI modulok előtt történjen meg.

A rendszermagnak átadott paraméterek a rendszermag viselkedését, működését befolyásolják. Általában azért van szükség a rendszermag paramétereire, hogy annak viselkedését már a futásának első pillanatában befolyásolhassuk. Néhány esetben előfordulhat, hogy a Linux rendszermag el sem képes indulni, ha nem kapja meg a megfelelő paramétereket. Ez általában a processzor vagy a számítógép egyéb elemeinek tervezési hibáira utal.

Miután a rendszer elindult, a rendszergazda a /proc/cmdline szimulált állományból kiolvashatja, hogy a rendszermag milyen paraméterekkel indult el. (Természetesen csak akkor, ha a proc állományrendszeret beillesztettük a /proc/könyvtárba.)

`init=` A kulcsszó után annak a programnak az állománynevét adhatjuk meg, amelyet a rendszermag indulása után elindít. Ez alapértelmezés szerint a /sbin/init, amely a különféle szolgáltatásokat elindítja (lásd 126. o.).

Ha valamilyen hiba miatt nem tudjuk elindítani a rendszert a szokásos módon, az összes szolgáltatást megkerülhetjük ezzel a paraméterrel. Ezt mutatja be a 5.4. példa.

Igen fontos tudnunk, hogy ezzel a paraméterrel a GNU/Linux jelszóellenőrző rendszere is megkerülhető! Ha a rendszertöltő elfogad és továbbít a magnak paramétereket anélkül, hogy jelszóellenőrzést végezne, bárki, aki a számítógéphez fizikai értelemben hozzáfér, teljhatalmat szerezhet a számítógép felett!

initrd= A gyökérkönyvtár beillesztése előtt beillesztendő kezdeti RAM állományrendszer (*initial RAM disk*) használatára ad utasítást ez a kapcsoló. Az ilyen kezdeti RAM lemezeket általában arra használjuk, hogy a gyökérkönyvtár beillesztéséhez szükséges magmodulokat elérhetővé tegyük a Linux rendszermag számára.

A kezdeti RAM lemeztérkép ennek megfelelően egy kis méretű, tömöritt állományrendszert tartalmaz, amelynek beillesztése gyökérkönyvtár-ként történik. A beillesztett kezdeti RAM lemeztérkép a benne található állományok segítségével betölti a gyökérkönyvtár beillesztéséhez szükséges magmodulokat, majd a gyökérkönyvtár újraillesztésével illeszti be a végeges gyökérkönyvtárat.

Az **initrd=** paraméter tulajdonképpen a rendszertöltő program számára jelzi, hogy a lemeztérképet be kell illeszteni, és a benne található programot le kell futtatni.

Kezdeti RAM lemeztérképet hozhatunk létre az **mkinitrd** program segítségével. Az **mkinitrd** a különféle beállítóállományokat olvasva kikeresi azokat a magmodulokat, amelyeket a kezdeti RAM állományrendszerben el kell helyezni, és elkészíti a lemeztérképet.

A kezdeti RAM lemeztérkép készítését a 5.4. példa, használatát pedig a 5.4., illetve 5.4. példák mutatják be.

vga= A VGA (*video graphics array*) videócsatoló üzemmódját adhatjuk meg, amelyet a karakteres üzemmódban használni akarunk. A kulcsszó után megadhatjuk az **ask** szót, amelynek hatására a felhasználónak kell megadnia rendszerindításkor az üzemmódot.

Az üzemmódokat a felhasználónak kódjukkal, tizenhatos számrendszerben kell megadnia, néhány rendszertöltő azonban csak tízes számrendszerben fogadja el ezeket a beállítóállományban.

Ha a számítógépbe épített videóáramkör teljesíti a VESA 2.0 (*video electronics standards association*) szabványt, lehetőségünk van a karakteres üzemmód grafikus üzemmódra váltására. (Nem csak VESA szabványú videókártyánál használható ez a módszer, további lehetőségekről olvashatunk a Linux rendszermag dokumentációjában.) Ilyen esetben a GNU/Linux karakteres képernyője a VGA csatoló grafikus üzemmódját használja. A VESA üzemmódok kódjait a 5.4. táblázat tartalmazza.

- ro** E paraméter hatására a Linux rendszermag a gyökérkönyvtárat csak olvasható módon illeszti be. Ez az alapértelmezett viselkedés.
- rw** E paraméter hatására a Linux rendszermag a gyökérkönyvtárat írható és olvasható módon illeszti be.
- root=** A paraméter segítségével megadhatjuk a gyökérkönyvtárat tartalmazó állományrendszer helyét. Ezt mutatja be a 5.4. példa.

	640×480	800×600	1024×768	1280×1024
256	0x301	0x303	0x305	0x307
32k	0x310	0x313	0x316	0x319
64k	0x311	0x314	0x317	0x31A
16M	0x312	0x315	0x318	0x31B

	640×480	800×600	1024×768	1280×1024
256	769	771	773	775
32k	784	787	790	793
64k	785	788	791	794
16M	786	789	792	795

5.4. táblázat. A Vesa üzemmódok kódjai

rootfstype= A paraméter segítségével a gyökérkönyvtárat tartalmazó állományrendszer típusát adhatjuk meg.

rootflags= A paraméter segítségével az állományrendszer beillesztésekor használt, az állományrendszer viselkedését módosító kapcsololókat adhatjuk meg (lásd a 78. oldalon).

S, 1, 2,... A rendszermag számára ezekkel a karakterekkel a futási szintet adhatjuk meg, amelyet a rendszer az indulás után felvesz. A futási szintekről bővebben a 125. oldalon, a 6.1. részben olvashatunk.

Valójában ezt a paramétert a rendszermag csak továbbítja az általa elindított init programnak, amely a futási szinteket kezeli.

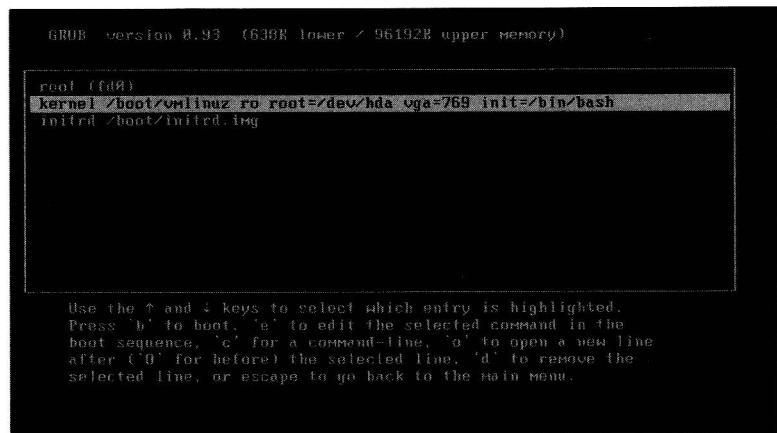
mem=méret A számítógéphez beépített memória méretének megadása. Akkor használjuk, ha a Linux rendszermag valami miatt nem tudja a memória méretét megállapítani.

A paraméterben megadott szám után a K, M és G mértékegységeket használhatjuk.

45. példa Tegyük fel, hogy valamelyen állomány sérülése miatt a GNU/Linux nem indítható a szokásos módon! Kíséreljük meg a helyreállításhoz elindítani a rendszert úgy, hogy a szokásos szolgáltatások helyett csak a héj induljon el!

A rendszerültő program segítségével adjuk át a rendszermagnak az `init=/bin/bash` paramétert (lásd 5.3. ábra).

Ha a rendszermagot ezzel a paraméterrel indítjuk, a gyökérkönyvtár beillesztése után a /bin/bash állományt indítja el, mégpedig rendszerüzemi jogokkal (lásd 5.4. ábra).



5.3. ábra. A rendszerindítás megkerülése

A képernyőképen látszik, hogy a rendszermag a BASH programot indította el, de úgy, hogy számára saját névként az init szót adta át. Ez az oka annak, hogy a parancskérő jelben az init szót olvashatjuk a BASH változatszámával.

46. példa Hozzunk létre kezdeti RAM lemeztérképet, és vizsgáljuk meg a tartalmát! A lemeztérkép tartalmazza az ide-cd és a cdrom magmodulokat azokon a modulokon kívül, amelyek a rendszer indításához feltétlenül szükség van!

A feladatot egy egyszerű BASH programmal végezzük el:

```

1 #! /bin/bash
2 #
3 #
4 # Hozzuk létre a kezdeti RAM lemeztérképet, és csomagoljuk ki!
5 #
6 mkinitrd --with=cdrom\
7     --with=ide-cd \
8         initrd.img.gz $(uname -r)
9
10 gunzip initrd.img.gz
11 #
12 #
13 # Illesszük be a lemeztérképet az újonan létrehozott ramdisk
14 # könyvtárba, hogy megvizsgálhassuk a tartalmát!
15 #
16 losetup /dev/loop0 initrd.img
17 mkdir ramdisk
18 mount /dev/loop0 ramdisk

```



```

md: ... autorun DONE.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
Linux IP multiclient 0.06 plus PIM-SM
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 116k freed
UFS: Mounted root (ext2 filesystem).
Red Hat nash version 3.4.42 starting
Loading cdrom.o module
Loading ide-cd.o module
hda: attached ide-cdrom driver.
hda: ATAPI 1X CD-ROM drive, 32KB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.12
Mounting /proc filesystem
Creating block devices
Creating root device
UFS: Mounted root (iso9660 filesystem) readonly.
Trying to move old root to /initrd ... failed
Unmounting old root
Trying to free ramdisk memory ... okay
Freeing unused kernel memory: 112k freed
init-2.05b#
```

5.4. ábra. Belépés rendszergazdaként, jelszó nélkül

A ramdisk könyvtárba beillesztett állományrendszerben megtalálhatjuk a linuxrc állományt, amely a modulok beillesztését és a gyökérkönyvtár újraillesztését végzi.

A linuxrc az újraillesztés során a /etc/fstab állományrendszer-táblázatban megadott gyökérkönyvtárat igyekszik beilleszteni. Ha ettől el akarunk tértí - mert például egy másik számítógépen szeretnénk használni az elkészített állományt -, az mkinitrd --fstab= kapcsolójával meg kell adnunk a használni kívánt állományrendszer-táblázat helyét.

47. példa Módosítsuk a 5.2.4. feladatban adott példaprogramot úgy, hogy az kezdeti RAM térképet használjon a magmodulok betöltésére! A módosítások a következő sorok hozzáadását jelentik:

1	45a47,49
2	> # Létrehozzuk a kezdeti állományrendszer térképét.
3	> mkinitrd \$MDIR/boot/initrd.img \$(uname -r)
4	>
5	56a61
6	> initrd=/boot/initrd.img

A különbséget tartalmazó állományt a következőképpen érvényesíthetjük az eredeti programon:

```
$ patch <inditolemez-lilo.patch inditolemez-lilo.sh
patching file inditolemez-lilo.sh
$
```



```

md: autodetecting RAID arrays.
md: autorun ...
md: ... autorun DONE.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 16384)
Linux IP multicast router 0.06 plus PIM-SM
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 116k freed
UFS: Mounted root (ext2 filesystem).
Red Hat nash version 3.4.42 starting
Loading cdrom.o module
Loading ide-cd.o module
hda: attached ide-cdrom driver.
hda: ATAPI 1X CD-ROM drive, 32KB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.1Z
Mounting /proc filesystem
Creating block devices
Creating root device
UFS: Cannot open root device "hda1" or 03:01
Please append a correct "root=" boot option
Kernel panic: UFS: Unable to mount root fs on 03:01

```

5.5. ábra. A gyökérkönyvtár nem érhető el

48. példa Módosítsuk a 5.3.4. példában bemutatott programot úgy, hogy az a betöltőlemez elkészítésekor kezdeti RAM állományrendszer is használjon! A módosítás a következő sorokat adja a programhoz:

```

1 42a43,45
2 > # Létrehozzuk a kezdeti állományrendszer térképét.
3 > mkinitrd $MDIR/boot/initrd.img $(uname -r)
4 >
5 52a56
6 >     initrd /boot/initrd.img

```

A különbséget tartalmazó állományt a következőképpen érvényesíthetjük az eredeti programon:

```

$ patch <inditolemez-grub.patch inditolemez-grub.sh
patching file inditolemez-grub.sh
$ 

```

49. példa Rendszerhiba miatt szeretnénk a számítógépet hajlékony lemez felhasználásával indítani. Az indítólemezet azonban egy másik számítógépen készítettük, így az nem a megfelelő helyen keresi a gyökérkönyvtárat tartalmazó állományrendszeret, ezért a Linux rendszermag betöltés után leáll (lásd 5.5. ábra).

A probléma megoldását az jelentheti, ha a Linux rendszermagnak a `root=` kifejezés után megadjuk a gyökérkönyvtárat tartalmazó állományrendszer eszközkezelő állományának nevét.

Mivel a gyökérkönyvtár beillesztése csak olvasható módon történik, a rendszerben akkor sem teszünk kárt, ha hibásan adjuk meg a gyökérkönyvtár helyét. Nyugodtan próbálkozhatunk tehát különféle értékekkel (`hda1`, `hda2 stb.`), amíg a gyökérkönyvtárat meg nem találjuk.

6. fejezet

Szolgáltatások indítása és leállítása

6.1. A futási szintek

A Unix rendszerek különféle üzemmódokat, futási szinteket (*runlevel*) különböztenek meg. A futási szintek az operációs rendszer állapotai, amelyek közül a rendszergazda választja ki, hogy éppen melyik felel meg a feladat elvégzésére.

Azt, hogy az adott GNU/Linux rendszeren melyik futási szint milyen viselkedést jelent, a beállítóállományok határozzák meg. minden GNU/Linux gyűjtemény a rá jellemző beállítóállományokkal a rá jellemző futási szinteket határozza meg.

A Red Hat alapú terjesztések például általában a következő futási szinteket különböztetik meg:

- 0 Leállítás. Ez a futási szint a számítógép leállítását, kikapcsolását végzi. Amelyik számítógép felveszi ezt a futási szintet, az rövid időn belül leáll.
- 1 Egyfelhasználós üzemmód, amelyet szokás S betűvel is jelölni. Ez az üzemmód szervizcélokra, helyreállításra használható.
Ha egy számítógép felveszi ezt a futási szintet, csak a rendszergazda számára engedi a rendszer használatát.
- 3 Többfelhasználós üzemmód. Ez a futási szint a számítógép normális használat közben használt futási szintje.
- 5 Többfelhasználós üzemmód grafikus felülettel. Ezt a futási szintet használva a felhasználók számára grafikus bejelentkezési felületet biztosíthatunk.
- 6 Újraindítás. Az a számítógép, amely felveszi ezt a futási szintet, rövidesen újraindul.

6.1. tábla: runlevel

Kiírja a képernyőre, hogy a rendszer milyen futási szinten működik és milyen futási szinten működött előtte.

```
runlevel
```

A program a /var/run/utmp állományból olvasott adatok alapján megállapítja, hogy a GNU/Linux milyen futási szinten üzemel és milyen futási szintről jutott erre a szintre, majd kiírja az előző futási szint értékét (N, ha nem volt előző szint) és a futási szint pillanatnyi értékét a szabványos kimenetre.

A használható futási szintekről általában a /etc/inittab szöveges beállítóállományban olvashatunk.

Azt, hogy a számítógép éppen milyen futási szinten van, a runlevel parancs segítségével állapíthatjuk meg.

6.2. A futási szintek kezelése

A futási szinteket az init program kezeli, amelyet maga a Linux rendszermag indít el a gyökérkönyvtár beillesztése után a /bin/ vagy a /sbin/ könyvtárból (Hacsak az init= paraméterrel a rendszertöltéskor más programot nem állítunk be).

A rendszermag csak egy programot indít el, az init programnak kell tehát kezdeményeznie a különféle szolgáltatások indítását. Az init által indított folyamatok természetesen indíthatnak újabb folyamatokat, de végső soron minden folyamat családfája visszavezethető az elsőként indított init programhoz. Az init tehát minden folyamat összülője és így igen fontos a rendszer működése szempontjából.

Mivel az init az első indított folyamat, a folyamatazonosítója (PID) mindig pontosan 1. A Linux rendszermag különlegesen kezeli az 1-es folyamatazonosítójú folyamatot. Ha például az 1-es folyamat olyan jelzést (*signal*) kap, ami a megszakítását eredményezné, a Linux rendszermag egyszerűen nem továbbítja a jelzést, hogy megvédeje a folyamatot.

6.2.1. Az indítótáblázat

Amikor az init elindul, a rendszer gyökérkönyvtára már elérhető, így az init eléri az indítótáblázatot (a /etc/inittab állományt), amely a beállításait tartalmazza.

A /etc/inittab különlegesen fontos állomány, mivel az init által a különféle futási szinteken indítandó szolgáltatások leírását tartalmazza. Ha a /etc/inittab

nem létezik vagy sérült, a számítógép csak egyfelhasználós üzemmódban indítható.

Az indítótáblázat egyszerű szöveges állomány, amely kettőspontokkal elválasztott oszlopokban tartalmazza az elindítandó szolgáltatások listáját. Az állományban a # jelrel helyezhetünk el megjegyzéseket: az init figyelmen kívül hagyja azokat a sorokat, amelyek ezzel a karakterrel kezdődnek.

Az indítótáblázat négy, kettősponttal elválasztott oszlopban tartalmazza az indítandó szolgáltatások listáját, minden sorban egy szolgáltatást leírva. Az egyes oszlopokba a következő adatok kerülnek:

1. Az első oszlopba a szolgáltatás egyedi azonosítója kerül, amely legalább egy, legfeljebb négy karakterből áll.

Ügyeljünk arra, hogy az egyes szolgáltatásokat egyedi azonosítóval lássuk el, hogy az init hibaüzenetei alapján az adott szolgáltatást egyértelműen vissza tudjuk keresni. Ha az indítótáblázatban több szolgáltatás tartalmazza ugyanazt az azonosítót, az init működik ugyan, de hibaüzenetet küld.

2. Futási szintek, amelyeken az adott szolgáltatást futtatni kell.

A második mezőben azt adjuk meg, hogy a bejegyzés milyen futási szinten (szinteken) érvényesüljön. A futási szinteket számjegyekkel írjuk egymás után, azaz a 234 például a kettes, hármas és négyes futási szintet jelenti.

3. A művelet vagy az indítás módja.

Ez az oszlop megadja, hogy az adott sort hogyan kell értelmezni, a szolgáltatást milyen módon kell az init programnak elindítania.

4. A szolgáltatást megvalósító program.

Az oszlop annak a programnak a nevét és paramétereit adja meg, amely az adott szolgáltatást megvalósítja.

Az init a szolgáltatások indítása előtt beállítja a PATH környezeti változó értékét, a programokat a /usr/local/sbin/, /sbin/, /bin/, /usr/sbin/ és /usr/bin/ könyvtárakban keresi.

Az indítótáblázat harmadik oszlopában szolgáltatások indításakor a következő kulcsszavakat használhatjuk:

respawn Azt jelzi, hogy az adott szolgáltatást megvalósító programot újra el kell indítani, ha kilép.

Ha az init azt tapasztalja, hogy a programot 2 perc alatt legalább tízszer újra kellett indítani, a szolgáltatás indítását 5 percre felfüggeszti, hogy az állandó újraindítások ne emésszék fel az erőforrásokat.

wait Adott futási szinten egyszer indítandó program.

A szolgáltatást megvalósító programot az init elindítja, ha felveszi az adott futási szintet. Az init megvárja, amíg a program befejezi futását, és csak akkor fut tovább.

once Adott futási szinten egyszer indítandó program.

A szolgáltatást megvalósító programot az init a futási szint felvételekor elindítja, majd azonnal továbbfut.

Szintén a harmadik oszloban használhatjuk a következő – rendszerindításra utaló – kulcsszavakat a szolgáltatások leírásánál:

sysinit A rendszerindításkor indítandó szolgáltatások. Az init az ilyen szolgáltatások esetében figyelmen kívül hagyja, hogy a második oszlopba milyen futási szint van írva, a szolgáltatást csak egyszer, a rendszer indításakor indítja el.

A sysinit móddal meghatározott szolgáltatás a boot és a bootwait móddal meghatározott szolgáltatások előtt indul el.

boot A szolgáltatást a rendszerindításkor futtatja a program, függetlenül attól, hogy a második mezőben milyen futási szintet írtunk elő.

bootwait A szolgáltatást a rendszerindításkor indítja el az init, függetlenül attól, hogy a harmadik oszloban milyen futási szintet írtunk elő.

A bootwait móddal meghatározott szolgáltatásokat megvalósító programok lefutását az init megvárja, csak akkor fut tovább, ha azok befejezték futásukat.

Néhány szolgáltatást különleges külső események bekövetkeztekor indít el az init. Ezek esetében a következő kulcsszavakat használhatjuk a harmadik mezőben:

powerwait A szolgáltatást akkor kell végrehajtani, ha a szünetmentes tápegység jelzi, hogy az áramszolgáltatás megszakadt, a számítógép tartalék áramforrásról üzemel. Az init a második oszlopba írt futási szintet figyelmen kívül hagyja az ilyen szolgáltatások esetében.

Az init az ilyen kulcsszóval jelzett szolgáltatást megvalósító program esetében megvárja, amíg az kilép, csak utána fut tovább.

powerokwait A szolgáltatást a második oszloban található futási szintet jelölő bejegyzéstől függetlenül akkor kell indítani, ha a szünetmentes áramforrás jelzi, hogy a vezetékes áramszolgáltatás helyreállt.

Az init az ilyen szolgáltatások esetében a szolgáltatást megvalósító program lefutását megvárja.

powerfailnow Az ilyen kulcsszóval létrehozott szolgáltatásokat akkor kell indítani, amikor a szünetmentes áramforrás jelzi, hogy a telepek rövidesen kiemerülnek. Az ilyen szolgáltatások esetében az init figyelmen kívül hagyja a második oszlopba írt futási szintet.

ctrlaltdel Az ilyen szolgáltatást a második oszlopba írt futási szintektől függetlenül akkor indítja az init, ha a felhasználó megnyomja a **[Ctrl]+[Alt]+[Del]** billentyűkombinációt.

A következő kulcsszó szintén az indítótáblázat harmadik oszlopába írható, de nem szolgáltatást határoz meg, hanem az alapértelmezett futási szintet:

initdefault Az ilyen kulcsszóval létrehozott bejegyzés az init számára a rendszerindítás után beállítandó futási szintet adja meg. Az init az adott sorban, a harmadik mezőben található futási szintre lép a rendszerindításkor, hacsak a rendszerindításkor más utasítást nem kap.

Ha a rendszerindításkor nem határoztuk meg a futási szintet, és az indítótáblázatban sincs alapértelmezett futási szint beállítva, az init az indítást végző felhasználótól várja a futási szint meghatározását.

50. példa A következő állomány egy egészen egyszerű indítótáblázatot mutat be:

```
1  #
2  # /etc/inittab: Indítótáblázat
3  #
4  defr:1:initdefault:
5
6  init::sysinit:/etc/rc.d/rc.sysinit
7
8  1:12345:respawn:/sbin/agetty 115200 /dev/tty1
9  2:12345:respawn:/sbin/agetty 115200 /dev/tty2
10 3:12345:respawn:/sbin/agetty 115200 /dev/tty3
11 4:12345:respawn:/sbin/agetty 115200 /dev/tty4
12 5:12345:respawn:/sbin/agetty 115200 /dev/tty5
13 6:12345:respawn:/sbin/agetty 115200 /dev/tty6
14
15 cad::ctrlaltdel:/sbin/shutdown -r now
```

Az állományban megtalálható az alapértelmezett futási szint meghatározása az initdefault kulcsszó segítségével. Az alapértelmezett futási szint az 1-es.

Könnyen felismerhető a rendszerindítást végző BASH program hívása a sysinit kulcsszóval. Ez a BASH program hivatott az állományrendszer ellenőrzésének és beillesztésének elvégzésére.

6.2. tábla: init, telinit

A program a rendszermag által indítva a rendszerindításban, a felhasználó által indítva a futási szint megváltoztatásában játszik szerepet.

```
init futási_szint
telinit [kapcs] futási_szint
```

Ha a programot a rendszermag indította, a /etc/inittab alapján szolgáltatások indít és állít le, amíg a GNU/Linux fut. Ha a programot a felhasználó indította, a programmal a futási szintet állíthatjuk be.

Kapcsoló

Jelentés

-t idő

A kapcsoló után megadhatjuk, hogy az init a SIGTERM üzenet után hány másodperccel küldjön SIGKILL üzenetet a leállítandó szolgáltatásokat megvalósító folyamatoknak.

A respawn kulcsszó felhasználásával meghatározott 1–6. szolgáltatások hat darab karakteres bejelentkezési felületet biztosítanak az agetty program hívásával. Ezeket a szolgáltatásokat az init újraindítja, ha a felhasználó kilépett, hogy a következő bejelentkezés megtörténhessen.

A beállítóállomány utolsó sorában a **Ctrl + Alt + Del** billentyűkombináció lenyomásakor végrehajtandó újraindítást hozzuk létre.

6.2.2. Az init használata

Amint láttuk, az init a különféle futási szinteken futtatandó programokat, illetve a különféle futási szintekhez és eseményekhez tartozó szolgáltatásokat kezeli.

Érdekes módon viselkedik azonban az init, ha nem a Linux rendszermag indítja el, hanem a rendszergazda. Ha az init azt tapasztalja, hogy a folyamataazonosítója nem 1, akkor nem a szolgáltatások elindítását végzi, hanem a futási szintek beállítását. A program ilyenkor felveszi a rendszeren található 1-es folyamataazonosítójú folyamatot a kapcsolatot a /dev/initctl csővezetéken keresztül, és annak küld üzenetet a parancssorban kapott paraméterek alapján. Ezt a szolgáltatást a futási szint megváltoztatására használjuk.

A parancssorból indítva az init programot, a rendszergazda egyszerűen megadhatja paraméterként, hogy melyik futási szintre szeretne kapcsolni. Az elindított init üzenetet küld az 1-es folyamataazonosítóval futó folyamatnak, és az átkapcsol a megadott futási szintre. Bizonyos szolgáltatások elindulnak, mások esetleg leálnak, attól függően, hogy melyik futási szintről melyik futási szintre kapcsolunk.

51. példa Tegyük fel, hogy a számítógép karbantartása miatt egyfelhasználós üzemmódra szeretnénk átállni. Ekkor a következő parancsot kell kiadnunk:

```
$ init 1
```

A parancs hatására a megfelelő szolgáltatások leállnak, a rendszer felveszi az 1-es futási szintet.

Természetesen más futási szintre is hasonló módon kapcsolhatunk át, egyszerűen a megfelelő számot kell az init számára paraméterként átadni.

6.3. Alrendszer indítása és leállítása

Amint az előző részekben láttuk, a különféle szolgáltatások indításáért az 1-es folyamatazonosítóval futó init program a felelős, a /etc/inittab beállítóállomány alapján.

Nem minden szolgáltatást indít azonban az init közvetlenül, és ennek több oka is van. Nem volna szerencsés, ha egy ilyen fontos program indítaná azokat a szolgáltatásokat, amelyek a rendszer működése szempontjából nem elengedhetetlenek, hiszen akkor a kevésbé fontos szolgáltatások telepítése, törlése, beállítása veszélyeztetné a működőképesség szempontjából létfontosságú alapszolgáltatások meglétét. Nem volna szerencsés, ha az egyes magasszintű szolgáltatásokat megvalósító programcsomagok telepítése során az /etc/inittab állományt módosítanunk kellene, hiszen például nehézkessé válna a programcsomag törlése során az állomány eredeti állapotra való visszaállítása. Sokkal szerencsésebb, ha betartjuk a szabályt, miszerint minden állomány egy programcsomaghoz tartozik.

A GNU/Linux gyűjteményekben az init csak az elengedhetetlenül fontos alapszolgáltatásokat indítja közvetlenül a /etc/inittab bejegyzései alapján. A magasabb szintű szolgáltatások – amelyeket alrendszerünknek (*subsystem*) nevezünk – saját indítóprogramokkal rendelkeznek. Az alrendszer indíthatók és leállíthatók a saját – általában BASH programként megvalósított – indítóprogramjuk segítségével.

GNU/Linux rendszereken az init az alrendszer indítását és leállítását közvetve végzi. minden futási szint-váltásnál elindítja a /etc/rc.d/rc programot, amely eldönti, hogy mely alrendszeret kell leállítani és elindítani az új futási szinten, és a megfelelő indítóprogramok segítségével el is végzi ezt a munkát.

52. példa Készítsük el a /etc/inittab állományban azokat a sorokat, amelyek a /etc/rc.d/rc BASH programot hívják a futási szint változásakor! A 6.2.1. példánál megadott állományhoz a következő sorokat kell hozzáfűznünk:

```
1 7a8,18
2 > #
3 > # Alrendszer indítása és leállítása.
4 > #
5 > cr0:0:wait:/etc/rc.d/rc 0
6 > cr1:1:wait:/etc/rc.d/rc 1
7 > cr2:2:wait:/etc/rc.d/rc 2
```

```

8 | > cr3:3:wait:/etc/rc.d/rc 3
9 | > cr4:4:wait:/etc/rc.d/rc 4
10 | > cr5:5:wait:/etc/rc.d/rc 5
11 | > cr6:6:wait:/etc/rc.d/rc 6
12 | >

```

Amint látjuk, az egyes futási szintekhez egy-egy sort kell elhelyeznünk a /etc/inittab állományban, hogy paraméterként átadhassuk a futási szintet és a program minden futási szinten elinduljon.

Tudnunk kell, hogy az init újabb változatai az általuk indított programokban átadják a RUNLEVEL és PREVLEVEL környezeti változókban az éppen felvett és az előző futási szintet. Ha ezeket a változatokat használjuk, a futási szintet nem kell paraméterként átadnunk.

A /etc/rc.d/rc BASH program megvizsgálja, hogy az új futási szinten mely alrendszereknek kell futniuk, és a megfelelő alrendszereket elindítja, illetve leállítja. A program az alrendszerekhez tartozó indítóprogramokat használja az alrendszerek kezelésére.

Az /etc/rc.d/rc program működése a következő szakaszokra bontható:

1. Az init futásiszint-váltáskor elindítja a /etc/rc.d/rc programot, és átadja neki az új futási szint értékét.
2. Az /etc/rc.d/rc megkeresi a /etc/rcn.d/ könyvtárat, ahol az újonan felvett futási szinthez tartozó alrendszerek indítóprogramjai találhatók. (Az *n* itt az új futási szint száma.)
3. A program először leállítja azokat az alrendszereket, amelyek indítóprogramjainak neve K betűvel kezdődik, és éppen futnak. Az adott futási szinten leállítandó szolgáltatások indítóprogramjainak neve tehát minden K betűvel kezdődik.

A program ezután elindítja azokat az alrendszereket, amelyek indítóprogramjainak neve S betűvel kezdődik, és éppen nem futnak. Az alrendszerek indítóprogramjainak neve tehát, amelyeket az adott futási szinten el kell indítani, minden S betűvel kezdődik.

A /etc/rc.d/rc BASH program onnan tudja, hogy egy adott alrendszer fut-e, hogy megvizsgálja a /var/run/ könyvtárban található állományokat. Az alrendszerek induláskor létrehoznak egy szabályos állományt ebben a könyvtárban, és elhelyezik benne a szolgáltatást megvalósító folyamat folyamatazonosítóját. Így bármikor ki tudjuk deríteni, hogy az alrendszer fut-e, és bármikor le tudjuk állítani a megfelelő folyamatazonosító ismeretében.

Az /etc/rc.d/rc programnak az indítóprogramokat, amelyeknek neve S betűvel kezdődik, a hagyomány szerint a start, azokat pedig, amelyek neve K betűvel kezdődik, a stop paraméterrel kell indítania. Ez lehetővé teszi, hogy ugyanaz

a program végezze az alrendszer indítását és leállítását, hiszen a paraméterből megállapítható, hogy indításra vagy leállításra van-e szükség.

Az indítóprogramokat tároló állomány neve a hagyomány szerint kezdő S és K betű utáni két számjeggyel, majd az alrendszer nevével folytatódik. Ez lehetővé teszi, hogy meghatározzuk az alrendszerek indítási és leállítási sorrendjét – a két számjegy alapján –, és kideríthessük az alrendszer nevét az indítóállomány nevéből – az állománynév végéből.

A /etc/rc.d/rc program tehát az indítandó és leállítandó alrendszerek indítóállományait névsorban hajtja végre, így betartja a rendszert összeállító szakember által előírt sorrendet. Ez fontos, hiszen az egyes alrendszerek egymásra épülhetnek, ezért nem lehet őket tetszőleges sorrendben elindítani.

Az /etc/rc.d/rc állomány működésének ismeretében elkészíthetjük a programot. Ezt mutatja be a következő példa.

53. példa Írunk BASH programot, amely a GNU/Linux rendszereken szokásos feladatokat látja el az alrendszerek indításakor és leállításakor!

```
1 #! /bin/bash
2 #
3 # /etc/rc.d/rc: Alrendszerek indítása és leállítása.
4 #
5 #
6 #
7 # Ez a függvény ellenőrzi, hogy a paraméter program-e.
8 #
9 function check_script() {
10     if [ -f "$1" -a -x "$1" ]; then
11         return 0
12     else
13         return 1
14     fi
15 }
16 #
17 #
18 # Ellenőrizzük, hogy a futási szinthez létezik-e a
19 # megfelelő könyvtár. A futási szint értékét a
20 # RUNLEVEL körny. változó tartalmazza.
21 #
22 if [ ! -d "/etc/rc$RUNLEVEL.d" ]; then
23     exit 1
24 fi
25 #
26 #
27 # Leállítandó alrendszerek.
```

```

28 #
29 for S in /etc/rc$RUNLEVEL.d/K*; do
30   NAME=${S#/etc/rc$RUNLEVEL.d/K??}
31   if check_script /etc/rc$RUNLEVEL.d/S??$NAME; then
32     continue
33   fi
34   if [ ! -f /var/run/$NAME.pid ]; then
35     continue
36   fi
37   if check_script "$S"; then
38     $S stop
39   fi
40 done
41
42 #
43 # Elindítandó alrendszerek.
44 #
45 for S in /etc/rc$RUNLEVEL.d/S*; do
46   PIDFILE=/var/run/${S#/etc/rc$RUNLEVEL.d/S??}.pid
47   if [ -f "$PIDFILE" ]; then
48     continue
49   fi
50   if check_script "$S"; then
51     $S start
52   fi
53 done

```

A bemutatott program a `/var/run/xxxx.pid` állomány alapján dönti el, hogy egy adott alrendszer éppen fut-e. Az állománynévben az `xxxx` helyén az alrendszer neve található.

Az egyes GNU/Linux gyűjteményekben ennél a bemutatott programnál általában sokkal összetettebb BASH program található, de a működési elv megértése e program alapján egyszerűbb.

6.3.1. Az alrendserek indítóprogramjai

A különféle alrendserek indítóprogramjai a `/etc/init.d/` könyvtárban találhatók. GNU/Linux esetén az alrendserek indítóprogramjai általában a BASH héj számára készült programok.

Az indítóprogramok a `start` paraméter hatására elindítják, illetve a `stop` paraméter hatására leállítják az adott alrendszert, hiszen a futási szinteket kezelő program így fogja elindítani és leállítani a megfelelő alrendsereket a különféle futási szinteken.

54. példa Indítsuk el a httpd alrendszer a megfelelő indítóprogram segítségével!

```
$ /etc/init.d/httpd start
A(z) httpd indítása: [ OK ]
$
```

Hasonló módon indíthatók más szolgáltatások is a /etc/init.d/ könyvtárban található indítóprogramok segítségével.

Ha a szolgáltatást le akarjuk állítani, használjuk a stop kulcsszót paraméterként:

```
$ /etc/init.d/httpd stop
httpd leállítása: [ OK ]
$
```

Minden futási szinthez tartozik egy könyvtár a /etc/ könyvtáron belül. Az 1-es futási szinthez tartozik a /etc/rc1.d/ könyvtár, a 2-es futási szinthez a /etc/rc2.d/ könyvtár és így tovább. Az egyes futási szinten elindítandó és leállítandó alrendszerek indítóprogramjaira mutató közvetett hivatkozásokat a megfelelő névvel itt kell elhelyeznünk.

55. példa Gondoskodunk ról, hogy az lpd alrendszer automatikusan elinduljon az 5-ös futási szinten!

Előbb vizsgáljuk meg, hogy az 5-ös futási szinten az lpd alrendszer leállító bejegyzés megtalálható-e!

```
$ cd /etc/rc5.d/
$ ls K??lpd
ls: K??lpd: Nincs ilyen fájl vagy könyvtár
```

Amint látjuk, nincs olyan bejegyzés, ami leállítaná az alrendszeret.

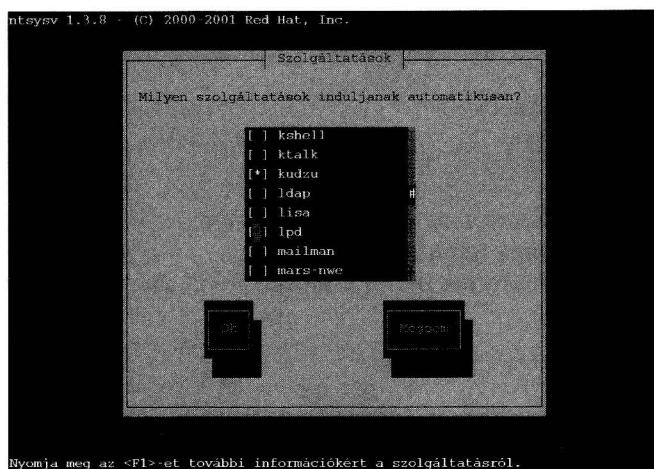
Most vizsgáljuk meg, hogy milyen alrendszerek indulnak ezen a futási szinten, hogy megállapíthassuk, milyen sorrendet célszerű betartani az indítás során!

```
$ ls S*
S08iptables  S24pcmcia  S60gpm      S95atd
S12syslog    S26apmd    S90crond   S99local
S13portmap   S28autofs  S90xfs
S17keytable  S55sshd    S95anacron
$
```

Tegyük fel, hogy úgy döntünk, az lpd alrendszer a gpm után, a crond előtt induljuk el. Ehhez a létrehozandó közvetett hivatkozás nevében elhelyezhetjük például a 70-es számot. Hozzuk létre a hivatkozást!

```
$ ln -s ../init.d/lpd S70lpd
$
```

Ezzel a feladatot el is végeztük, az lpd alrendszer a következő rendszerindításkor automatikusan elindul.



6.1. ábra. Az automatikusan induló szolgáltatások beállítása



Red Hat gyűjteményt használva az automatikusan elindítandó alrendsereket beállíthatjuk a chkconfig, a redhat-config-services vagy az ntsysv programok segítségével. Ez utóbbi a 6.1. ábrán látható.

Mi magunk is készíthetünk olyan BASH programot, amely a /etc/init.d/könyvtárban elhelyezve adott alrendszer indításáról gondoskodik. Ezt mutatja be a következő példa.

56. példa Készítsünk BASH programot, amely a syslogd programot indítja el a megfelelő futási szinten!

```

1 #!/bin/bash
2 #
3 # /etc/init.d/syslogd: A rendszernapló indítása,
4 # illetve leállítása.
5 #
6 #
7 #
8 # Az indítást végző függvény; paramétere a futtatandó
9 # programállomány neve.
10 #
11 function start_daemon() {
12     if ! [ -x "$1" ]; then
13         echo "A $1 állomány nem található."
14         return 1

```

```
15     fi
16
17     echo "Indítás: $1."
18     $1
19 }
20
21 #
22 # A leállítást végző függvény; paramétere a
23 # leállítandó programállomány neve.
24 #
25 function stop_daemon() {
26     local PIFILE
27
28     PIDFILE=/var/run/${basename $1}.pid
29     if ! [ -f "$PIDFILE" ]; then
30         echo "A syslogd PID állomány nem található."
31         exit 1
32     fi
33
34     PID=$(cat $PIDFILE)
35
36     echo "Leállítás: ${basename $1}($PID)."
37     kill -TERM $PID
38     return 0
39 }
40
41 #
42 # Ha 'start' paramétert kapott a program, a syslogd
43 # szolgáltatást elindítjuk, ha 'stop' paramétert,
44 # leállítjuk.
45 #
46 case "$1" in
47     start)
48         start_daemon /sbin/syslogd
49         ;;
50     stop)
51         stop_daemon /sbin/syslogd
52         ;;
53     *)
54         echo "Ismeretlen kapcsoló: '$1'."
55         ;;
56 esac
```

A példaprogram megértésében segít, ha tudjuk, hogy a `/sbin/syslogd` indítás-
kor a saját folyamataazonosítóját elhelyezi a `/var/run/` könyvtárban a megfelelő
állományban. A bemutatott példaprogram ezt az állományt használja a program
kilövésére, ha a `stop` paraméterrel hívjuk.

A program működésének megértéséhez azt is tudnunk kell, hogy a `syslogd`
démon, azaz indulása után a hívó felé úgy viselkedik, mintha kilépne, és a munkát
közvetlenül az `init` gyermekfolyamataként végezzi.

7. fejezet

Felhasználók nyilvántartása

E fejezetben a felhasználók nyilvántartásáról, kezeléséről olvashatunk. A fejezetben tárgyalt ismeretek megismerése igen fontos a GNU/Linux felépítésének és működésének megértése szempontjából, és a minden nap munka során is elengedhetetlen.

A Unix minden könyvtárbejegyzéshez, minden folyamathoz egy-egy felhasználi azonosítót (UID) rendel. A különféle műveletek elvégzése előtt az operációs rendszer minden ellenőrzi, hogy az adott folyamatnak van-e jog a műveletet végrehajtani.

A *felhasználói adatbázis* az operációs rendszer által kezelt nyilvántartás, amelyben minden felhasználói azonosítóhoz felhasználói nevet (*username*) és más, fontos kiegészítő információkat rendelünk.

A felhasználói adatbázis részét képezi egy csoportadatbázis is, amely megkönnyíti a felhasználók kezelését. A csoportnyilvántartás segítségével a nyilvántartott felhasználókat munkacsoportokba rendezhetjük, hogy a különféle erőforrásokat egységesen kezelhessük. A felhasználói csoportok nyilvántartásának alapja a csoportazonosító (GID). Az operációs rendszer minden könyvtárbejegyzéshez és folyamathoz egy csoportazonosítót rendel, ami segíti a jogosultságok megszervezését.

A csoportnyilvántartás szerint minden felhasználó legalább egy csoport tagja (elsődleges csoport), de tetszőlegesen sok csoport tagjai közé felvehető.

A GNU/Linux a vonatkozó szabványok szerint kezeli a felhasználókat és felhasználói csoportokat.

A hagyomány szerint root a neve annak a felhasználónak, akinek a felhasználói azonosítója 0. A felhasználói név természetesen megváltoztatható, de a 0 felhasználói azonosítóval rendelkező felhasználó mindenkorban teljhatalommal rendelkezik. Az a folyamat, amely ennek a felhasználónak a névében fut, minden állományt olvashat, írhat, törlhet, minden folyamat futását megszakíthatja, minden programot futtathat, egyszóval bármit megtehet. Az ilyen, teljhatalommal felruházott felhasználót általában rendszergazdának nevezik.

A GNU/Linux rendszernek szüksége van a rendszergazdára. Nem csak azért van szükség rendszergazdára, mert a számítógép folyamatos felügyeletet igényel, hanem azért is, mert maga a GNU/Linux is indít rendszergazdai jogokat használó folyamatokat. A rendszergazda létező személy, de akkor is futnak a nevében folyamatok, ha nincs a számítógép közelében. A felhasználókat, akiknek jogosultságait a GNU/Linux akkor is használja, amikor azok nincsenek bejelentkezve, rendszerfelhasználóknak hívjuk.

A rendszerfelhasználók közé tartozik a *nobody* (senki) nevű felhasználó, aki bizonyos értelemben a rendszergazda ellentéte. A *nobody* nevű felhasználónak szinte semmire sincs joga.

A *root* és *nobody* felhasználókon kívül még sok rendszerfelhasználó létezhet. Nevük és felhasználói azonosítójuk gyűjteményenként más és más lehet, de a legtöbb esetben könnyen felismerhető a nevük alapján, hogy mi a szerepük. A hagyomány alapján a rendszerfelhasználók alacsony – 500-nál vagy 1000-nél kisebb – felhasználói azonosítót kapnak, így könnyen megkülönböztethetők azoktól a felhasználóktól, akiket a rendszert használó személyek nyilvántartása közben hoztunk létre.

A GNU/Linux a rendszerfelhasználókat és az egyszerű felhasználókat egységesen kezeli, ezért a továbbiakban csak felhasználókként emlegetjük őket.

7.1. Szöveges adatbázisok

A felhasználók nyilvántartása sokféleképpen történhet. Kevés felhasználó esetén a felhasználók alapadatait elhelyezhetjük például néhány szöveges állományban, sok felhasználó esetén használhatunk adatbázis-kiszolgálót, nagyméretű hálózat esetén központi nyilvántartást. A GNU/Linux feladatai közé tartozik, hogy a nyilvántartás technikai részleteit elrejtse a programok, alkalmazások elől, a felhasználói adatbázis kezelése független legyen a tárolás mikéntének gyakorlati részleteitől.

A Unix a hagyomány szerint néhány szöveges állományt használ a felhasználók és felhasználói csoportok nyilvántartására. Ez a módszer a legtöbb esetben kielégítő módja a felhasználók kezelésének, így a mai napig a legtöbb Unixot futtató számítógépen ezt használják. Ebben a fejezetben a szöveges állományokkal megalvósított felhasználói nyilvántartást ismertetjük.

7.1.1. Az adatbázis-állományok

A szöveges állományokat használó módszer esetében a felhasználók nyilvántartása a `/etc/passwd`, a csoportok nyilvántartása pedig a `/etc/group` állományba kerül. Mindkét állomány kettőspontokkal elválasztott oszlopokban tartalmazza az adatokat, soronként egy-egy bejegyzéssel.

A `/etc/passwd` és a `/etc/group` mindenki által olvasható állományok. Ha

az olvasási jogot elvesszük valamelyik felhasználótól vagy felhasználói csoport-tól, a GNU/Linux nem működik helyesen.

Az /etc/passwd a következő mezőket tartalmazza:

1. Felhasználói név. A felhasználói név egy az angol ábécé betűiből álló, legfeljebb 8 karaktert tartalmazó egyedi név, amellyel a rendszer a felhasználót azonosítja.
2. A felhasználó kódolt jelszava. A kódolt jelszó lehetővé teszi a felhasználó azonosítását, de a jelszó előállítására nem alkalmas, így a kódolt jelszó ismeretével nem lehet visszaélni.
3. A felhasználó felhasználói azonosítószáma (UID).
4. A felhasználó elsőleges csoportjának csoportazonosító-száma (GID).
5. A GECOS mező a felhasználó egyéb adataival. Ebben a mezőben a hagyomány szerint vesszővel elválasztva szerepel a felhasználó valódi neve, irodájának száma, irodai és otthoni telefonszáma, de igény szerint eltérhetünk ettől a szokástól.
6. A felhasználó saját könyvtárának abszolút könyvtárleírása.
7. A felhasználó által használt héjprogram. Belépés után természetesen más héjprogramot is elindíthat a felhasználó, de a belépéskor ez a héjprogram fog elindulni.

A felhasználói csoportok nyilvántartására szolgáló /etc/group a következő – kettőspontokkal elválasztott – mezőket tartalmazza:

1. A csoport neve. Az angol ábécé betűiből álló, legfeljebb 8 betűt tartalmazó egyedi azonosító.
2. A csoporthoz tartozó jelszó kódolt formában. Ha ez a jelszó üres, akkor a csoporthoz való csatlakozáskor nem kell jelszót megadni.
3. A csoportazonosító (GID).
4. A csoporthoz tartozó összes felhasználó felhasználói neve, vesszővel elválasztva.

57. példa A következő példaprogram a helyi szöveges felhasználói nyilvántartás alapján küld körlevelet elektronikus levélként.

```
1 #! /bin/bash
2
3 #
4 # Ha nem létezik a küldendő állomány, kilépünk.
```

```

5  #
6 if [ ! -f "$1" ]; then
7   echo "A '$1' állomány nem létezik."
8   exit 1
9 fi
10 #
11 #
12 # A ciklus a felhasználói neveket járja végig, ahol
13 # az UID >= 500.
14 #
15 for NEV in $(awk -F: '$3>=500{print $1}' /etc/passwd)
16 do
17   echo "Küldés '$NEV' címre."
18   mail $NEV -s "Körlevél." <$1
19 done

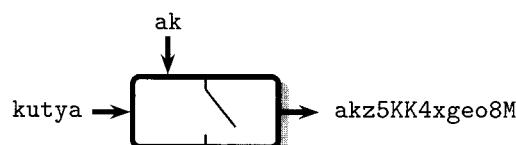
```

A program 15. sorában az awk számára beállítottuk a kettőspontot mezőelválasztó jelként, és kiírtattuk az első mező értékét, ahol a harmadik mező értéke eléri az 500-at.

7.1.2. Az árnyékjelszórendszer

A Unix rendszerek régebben a jelszavakat a /etc/passwd állományban tárolták. Ezt az állományt mindenki olvashatta, ez azonban nem okozott gondot, hiszen itt a jelszavak kódolt formában voltak elhelyezve, azokból az eredeti jelszót visszaállítani nem volt lehetséges.

A számítástechnika fejlődése, az egyre gyorsabb és gyorsabb számítógépek megjelenése lehetővé tette, hogy valaki pusztán próbálgatással is esélyes legyen a jelszavak visszakódolására. Ez a komoly számítási teljesítményt igénylő kódtörő munka kizártlag akkor lehet sikeres, ha a felhasználó jelszava eléggé egyszerű a kódtörő számítógép sebességehez képest, ennek ellenére a nyílt jelszótárolás ma már nem megnyugtató megoldás. A bárki számára elérhető jelszótárolást ezért felváltotta az árnyékjelszó- (*shadow password*) rendszer.



7.1. ábra. A jelszókodolás

Az árnyékjelszó használatával a felhasználók és csoportok alapadatai bárki számára olvashatók maradnak, de a jelszavak más, kizárolag a rendszergazda által olvasható állományokba kerülnek. Az így tárolt jelszavak próbáltatással nem fejtethetők meg, hiszen a jelszó kódolt változata is titokban marad.

Árnyékjelszó esetén a felhasználók adatait tároló /etc/passwd állományból a jelszó átkerül a /etc/shadow állományba. A /etc/passwd állományban a felhasználó jelszava helyén egy x betű találunk, amely jelzi, hogy a felhasználó kódolt jelszavát a /etc/shadow állományban kell keresni.

A /etc/shadow nem csak a kódolt jelszavakat tartalmazza, hanem néhány egyéb bővítést is. Az állomány a következő adatokat tárolja a felhasználókról:

1. A felhasználó felhasználói neve.
2. A kódolt jelszó.
3. Az utolsó jelszóváltoztatás napja (hány nap telt el 1970. január 1. óta a jelszó utolsó változtatásáig).
4. Hány napig lehet még megváltoztatni a jelszót.
5. A jelszó kötelező megváltoztatásának napja (hány nap múlva kell a jelszót megváltoztatni).
6. A jelszó lejárta előtt hány nappal kell a felhasználót figyelmeztetni.
7. A jelszó lejárta után hány nappal kell a felhasználót letiltani.
8. 1970. január 1. után hány nappal kell letiltani a felhasználót.

A /etc/shadow állomány mintájára létezik egy – az /etc/group állományt kiegészítő- /etc/gshadow állomány is, hiszen jelszót nem csak a felhasználóhoz, hanem a csoportokhoz is rendelhetünk.

Az árnyékjelszavak használata esetén a csoportok jelszavait nem a /etc/group állomány tartalmazza, hanem a /etc/gshadow állomány. Ilyenkor a /etc/group állományban a csoport jelszavának helyén egy x karakter találunk, amely jelzi, hogy a jelszót a /etc/gshadow állományban kell keresnünk.

7.2. A felhasználók kezelése

A UNIX rendszerek üzemeltetésének fontos része a felhasználók kezelése, nyilvántartása. Ebben a részben arról olvashatunk, hogy a szöveges állományban tárolt felhasználói nyilvántartás milyen eszközökkel és módszerekkel kezelhető.

7.2.1. Felhasználók létrehozása

Felhasználók létrehozására szolgál a useradd program. A program a megadott adatokkal egy új felhasználói bejegyzést készít a felhasználói adatbázisban.

A useradd program viselkedése az egyes Linux gyűjteményekben más és más lehet, a gyűjtemények készítői a programot a gyűjteményre jellemző beállításokkal látják el.

7.1. tábla: useradd

Új felhasználót vesz nyilvántartásba.

useradd [kapcs.] felhnév

A program segítségével új felhasználó adatait helyezhetjük el a szöveges állományokban tárolt felhasználói nyilvántartásban.

Kapcsoló	Jelentés
-c név	A felhasználóról nyilvántartott kiegészítő információk (például név, iroda száma, irodai telefonszám, otthoni telefonszám) megadása.
-d kvt.	A felhasználó saját könyvtárának megadása.
-g csop.	A felhasználó elsődleges csoportjának megadása névvel vagy csoportazonosítóval.
-G csop.	További csoportok, amelyekbe a felhasználó tartozik (vesszővel elválasztva).
-m	A kapcsoló hatására a program megkísérli létrehozni a felhasználó saját könyvtárát.
-M	A program a kapcsoló hatására nem hozza létre a felhasználó saját könyvtárát (Red Hat).
-n	A kapcsoló hatására a program nem hoz létre külön csoportot minden felhasználó számára (Red Hat).
-p jelszó	A kapcsoló után megadott kódolt jelszó a nyilvántartásba kerül.
-s héj	A felhasználó alapértelmezés szerinti héjprogramjának megadása.

A megfelelő beállítások mellett a useradd kikeresi az első szabad felhasználói azonosítót, és azt rendeli a felhasználóhoz. A felhasználók nyilvántartására általában 500-tól vagy 1000-től kezdődő felhasználói azonosítókat használ a program.

A létrehozott felhasználóhoz a beállításoktól függő módon rendel a useradd program elsődleges csoportot. Bizonyos Linux gyűjtemények alapértelmezés szerint minden felhasználóhoz ugyanazt a felhasználói csoportot rendelik, mások pedig minden felhasználó létrehozásakor létrehoznak egy egyszemélyes csoportot is, és azt használják elsődleges csoportként.

A useradd a megfelelő beállítások mellett létrehozza a felhasználó saját könyvtárát. A program a könyvtár létrehozása után a /etc/skel/ könyvtár teljes tartal-

mát bemásolja a felhasználó könyvtárába, majd a tulajdonos megváltoztatásával a felhasználó könyvtárát és teljes tartalmát a felhasználó számára átadja.

A /etc/skel/ könyvtár másolásával a useradd egy minimális munkakörnyezetet biztosít a létrehozott felhasználó számára, aki már az első bejelentkezés alkalmaival megtalálja a rendszergazda által biztosított állományokat a saját könyvtárában. Mivel sok program használja a saját könyvtárat a beállítások tárolására, a /etc/skel/ használata jól használható az alapbeállítások átadására.



Minden GNU/Linux gyűjteményről elmondhatjuk, hogy alapértelmezés szerint a useradd a felhasználó jelszavát tiltott állapotba kapcsolja a létrehozásakor. A felhasználó csak akkor tud bejelentkezni, ha a létrehozása után a rendszergazda jelszót állít be a számára.

58. példa A következő példa bemutatja, hogyan hozhatunk létre a legegyszerűbb módon új felhasználót, és hogyan állíthatjuk be a jelszavát.

```
$ useradd -c "Horvath Janos" joe
$ passwd joe
Changing password for user joe.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
$
```

A példában szereplő felhasználó felhasználói neve joe lesz, az egyéb adatok közt pedig nyilvántartjuk a teljes nevét is.

A useradd program beállításait a /etc/default/useradd állományban találjuk. Ezeket a beállításokat a program -D kapcsolójával is megváltoztathatjuk.

7.2.2. A jelszó megváltoztatása

A felhasználó a legtöbb rendszeren megváltoztathatja jelszavát a passwd program segítségével. A parancs begépelése után meg kell adnia a jelenlegi jelszavát, majd kétszer az új jelszót. A jelszó természetesen nem jelenik meg a képernyón, hogy ne lehessen titokban leolvasni.

A felhasználó jelszavát a rendszergazda bármikor megváltoztathatja, neki nem kell megadnia a felhasználó jelszavát a változtatáshoz, elegendő az új jelszót kétszer beírnia.

Amikor a rendszergazda új felhasználót vesz nyilvántartásba, a felhasználói jelszót be kell állítania. Ha a useradd programmal létrehozza a rendszergazda a felhasználói bejegyzést, de a jelszót nem állítja be a passwd programmal, a felhasználó nem tud bejelentkezni.

7.2. tábla: passwd

A jelszó megváltoztatását végző program.

passwd [kapcs.] [felhnév]

A program segítségével a felhasználók megváltoztathatják saját jelszavukat, a rendszergazda pedig megváltoztathatja bármely felhasználó jelszavát.

Kapcsoló	Jelentés
-l	A jelszó zárolása.
-u	A jelszó zárolásának feloldása.
-d	A jelszó törlése. A felhasználó jelszó nélkül jelentkezhet be.
-stdin	A kapcsoló hatására a jelszót a program a szabványos bemenetről olvassa.

A passwd program segítségével a felhasználó bejelentkezési joga felfüggeszthető. Ezt a program úgy végzi, hogy egy ! vagy * karaktert helyez el a kódolt jelszó első karaktere előtt. Ilyen esetben a felhasználó nem tud bejelentkezni, amíg a rendszergazda a zárolást fel nem oldja.

A program segítségével a jelszó törölhető is. Törölt jelszó esetén a felhasználó jelszó megadása nélkül jelentkezhet be.

59. példa A következő példaprogram új felhasználót hoz létre, és beállítja a jelszavát. Ha a -titok kapcsolót kapja a program, a felhasználó jelszava a „titok” szó lesz. Ha a kapcsoló az -ures, a felhasználó jelszó nélkül beléphet. Ha a program nem kap kapcsolót, a beállítandó jelszót bekéri a felhasználótól.

```

1 #! /bin/bash
2
3 read -p "Fehl. név: " FNEV
4 read -p "Telj. név: " TNEV
5
6 if ! useradd -c "TNEV" FNEV; then
7   echo "A létrehozás sikertelen." >&2
8   exit 1
9 fi
10
11 if [ "$1" = "-titok" ]; then
12   echo "titok" | passwd --stdin $FNEV
13   exit 0
14 fi
15
16 if [ "$1" = "-ures" ]; then

```

```
17     passwd -d $FNEV
18     exit 0
19 fi
20
21 passwd $FNEV
```

7.2.3. Felhasználók törlése

A felhasználót a `userdel` program segítségével törölhetjük a szöveges állományokat használó nyilvántartásból. A törlés csak akkor sikerülhet, ha az adott pillanatban a felhasználó nem futtat programot. Ha a felhasználó nevében akár egyetlen folyamat is fut, a törlés előtt a folyamatot meg kell szakítani.

7.3. tábla: `userdel`

Felhasználók törlését végzi a felhasználói adatbázisból.

`userdel [kapcs.] felhnév`

Kapcsoló *Jelentés*

`-r` A program a felhasználó saját könyvtárát és a postafiókjának tartalmát is törli.

A `userdel` a `-r` kapcsoló hatására törli a felhasználó saját könyvtárát – teljes tartalmával együtt – és a postafiókjában található leveleket. Ha a felhasználónak másutt is vannak állományai, a program nem törli azokat. A törölt felhasználó állományai „gazdátlanul” maradnak. A következő példa bemutatja, hogyan törölhetünk egy felhasználót, és hogyan kereshetjük meg a gazdátlanul maradt állományokat.

60. példa Töröljük a `joe` nevű felhasználót a nyilvántartásból és töröljük a saját könyvtárát is annak teljes tartalmával:

```
$ userdel -r joe
$
```

Vizsgáljuk meg a teljes könyvtárrendszert, és keressük meg azokat az állományokat, amelyeknek tulajdonosa nem szerepel a felhasználói adatbázisban:

```
$ find / -nouser
$
```

Amint látható, nincs olyan könyvtárbejegyzés a könyvtárszerkezetben, amelynek tulajdonosa hiányozna a felhasználói adatbázisból.

7.2.4. Felhasználó tulajdonságainak módosítása

A rendszergazda bármely felhasználó tulajdonságait megváltoztathatja a felhasználói adatbázisban, de maga a felhasználó is módosíthatja a róla nyilvántartott adatok egy részét.

A héj megváltoztatása

A chsh (*change shell*, héj megváltoztatása) programmal a felhasználó bármikor beállíthatja, hogy a belépés után melyik héj induljon el a számára.

A héjprogram megválasztásakor óvatosan kell eljárnunk, ha ugyanis rossz vagy nem létező programot választunk héjnak, nem tudunk bejelentkezni, programokat futtatni, azaz a chsh programot sem tudjuk futtatni, hogy a használt héjat visszaállítsuk.

A chsh program minden előket annak érdekében, hogy ne „zárjuk ki magunkat” a rossz program beállításával. Mielőtt megváltoztatná a beállítást, a chsh ellenőrzi, hogy létezik-e a program, amelyet héjként használni akarunk, és azt is megvizsgálja, hogy szerepel-e a /etc/shells állományban.

A felhasználók csak olyan programokat állíthatnak be héjprogramként, amelyek szerepelnek a /etc/shells állományban.

61. példa A következő példa bemutatja, hogyan változtathatja meg egy felhasználó a héjprogramot, ami elindul, amikor bejelentkezik.

```
$ chsh -s /bin/ash
Changing shell for joe.
Password:
Shell changed.
$
```

Látható, hogy a program jelszót kért a felhasználótól, mielőtt elvégezte volna a módosítást. Természetesen ez az ellenőrzés kikapcsolható.

A rendszergazda bármely felhasználó héjprogramját beállíthatja. Ez lehetőséget teremt a számára arra is, hogy az interaktív hozzáférést megtöltsa a felhasználó számára. Ha a rendszergazda olyan héjprogramot állít be a felhasználó számára, ami nem működik héjként, a felhasználó nem tud parancsokat kiadni, nem tud a számítógépre bejelentkezni.

Az interaktív bejelentkezés tiltására általában a /sbin/nologin programot szokás használni. Ha a rendszergazda ezt a programot állítja be a felhasználó számára, az nem tud bejelentkezni, de egyéb szolgáltatásokat elérhet, így például letöltheti állományait az ftp program segítségével, olvashatja elektronikus posztfiókját távoli számítógépről és így tovább. Az interaktív bejelentkezés tiltását mutatja be a következő példa.

7.4. tábla: chfn

A felhasználóról tárolt személyes adatok megváltoztatására szolgáló program.

chfn [kapcs.] [felhnév]

Kapcsoló	Jelentés
-f név	A felhasználó nevének megváltoztatása.
-o iroda	A felhasználó irodájának megváltoztatása.
-p telefon	A felhasználó irodai telefonszámának megváltoztatása.
-h telefon	A felhasználó otthoni telefonszámának megváltoztatása.

62. példa A rendszergazda szeretné megtiltani a joe nevű felhasználónak az interaktív bejelentkezést, ezért e felhasználónak a /sbin/nologin programot állítja be héjprogramként:

```
$ chsh -s /sbin/nologin joe
Changing shell for joe.
Shell changed.
$
```

Ha most a felhasználó megpróbál bejelentkezni, a /sbin/nologin program fut le, amely egy üzenetben tájékoztatja a felhasználót, hogy nem jelentkezhet be:

```
$ ssh joe@server
joe@server's password:
This account is currently not available.
Connection to localhost closed.
$
```

Személyes adatok megváltoztatása

A chfn program segítségével a felhasználó megváltoztathatja a róla tárolt személyes adatokat (név, munkahelyi szobaszám, munkahelyi telefonszám és otthoni telefonszám). A rendszergazda bármely felhasználó személyes adatait megváltozthatja.

Egyéb adatok megváltoztatása

További változtatásokat a usermod program segítségével végezhetünk a felhasználói adatbázison. Ezt a programot használhatja a rendszergazda a felhasználó saját könyvtárának, elsődleges csoportjának és néhány egyéb adatának megváltoztatására.

7.5. tábla: usermod

A rendszergazda ezzel a programmal a felhasználóról tárolt adatokat módosíthatja.

usermod [kapcs.] felhnév

Kapcsoló	Jelentés
-c szöveg	Az egyéb adatok (név, szobaszám, telefonszám, otthoni telefonszám) megváltoztatása.
-d könyvtár	A felhasználó saját könyvtárának megváltoztatása.
-g csop.	A felhasználó elsőleges csoportjának megváltoztatása (név vagy csoportazonosító alapján).
-G csop.	További csoportok megadása (vesszővel elválasztott csoportnevek vagy csoportazonosítók).
-l felhnév	A felhasználói név megváltoztatása.
-s héj	A felhasználóhoz rendelt héjprogram megváltoztatása.
-u	A felhasználó felhasználói azonosítójának megváltoztatása.

A rendszergazda a felhasználók minden adatát más programokkal is megváltozhatja, hiszen írású joga van a felhasználói adatbázis állományaira. A legszegényebb módja az állományok megváltoztatásának a szövegszerkesztővel való módosítás. A szöveges adatállományokat megnyithatjuk és átírhatjuk bármelyik szövegszerkesztővel, így az adatbázist tetszésünk szerint módosíthatjuk.



Ügyeljünk, hogy a szövegszerkesztővel való módosítás ideje alatt más az adatbázist ne módosítsa. Ha a szövegszerkesztővel való megnyitás és mentés közben más felhasználó módosítja a felhasználókról nyilvántartott adatokat – például új felhasználót hoz létre –, a módosításai a szövegszerkesztővel való mentés pillanatában elvesznek.

7.2.5. Felhasználók lekérdezése

A felhasználók a saját magukról vagy felhasználótársaikról nyilvántartott nyilvános információkat bármikor lekérdezhetik a finger program segítségével. Ezt mutatja be a következő példa.

63. példa Kérdezzük le a pipas nevű felhasználó adatait a finger program segítségével!

```
$ finger pipas
```

```
Login: pipas
```

```
Directory: /home/pipas
```

```
Name: Pere Laszlo
```

```
Shell: /bin/bash2
```

Office: 208, 4233 Home Phone: (72)xxx-xxx
Last login Tue May 29 17:09 (CEST) on :0
Mail forwarded to pipas@cs.ttk.pte.hu
New mail received Sun Apr 29 12:33 2003 (CEST)
 Unread since Wed Mar 28 09:00 2003 (CEST)
 No Plan.
\$

Amint látjuk, a program a szokásos információkon kívül például azt is megjeleníti, hogy mikor kapott utoljára elektronikus levelet a felhasználó, és mikor olvasta utoljára a leveleit.

Ha a felhasználó a saját könyvtárában létrehoz egy .plan nevű állományt, és azt az egész világ számára olvashatóvá teszi, akkor annak tartalmát is kiírja a finger program.

7.3. Csoportok kezelése

A groups programmal lekérdezhetjük, hogy egy felhasználó mely felhasználói csoportokhoz tartozik. A programnak paraméterül csak a felhasználó felhasználói nevét kell megadnunk. Ha több felhasználói nevet is megadunk, a program által kiírt listában a felhasználói nevek és a hozzájuk rendelt csoportok kétoszlopos listában jelennek meg.

7.3.1. Csoportok létrehozása

Csoportot létrehozni a groupadd parancs segítségével lehetséges. A parancsnak csak a létrehozandó csoport nevére van szüksége, csoportazonosító számot automatikusan keres.

7.3.2. Csoportok törlése

Csoportot törölni a groupdel parancs segítségével lehet; első paraméterként a törlni kívánt csoport nevét kell megadnunk. Olyan csoportot, ami egy felhasználó elsődleges csoportja, nem törölhetünk. Ilyenkor előbb a felhasználót kell törölnünk a nyilvántartásból.

A program nem ellenőrzi, hogy az adott csoporthoz mint csoport tulajdonoshoz tartozik-e állomány a rendszeren, ezt a rendszergazdának a find parancs segítségével magának kell elvégeznie.

7.4. Az árnyékjelszórendszer kezelése

A rendszergazda – ha a megfelelő szoftverelemeket telepítette – bármikor áttérhet az árnyékjelszavak használatára vagy éppen azok mellőzésére. Erre a következő programok állnak rendelkezésre:

pwconv Ez a program létrehozza a /etc/shadow állományt a passwd állomány felhasználásával. Ha már létezik /etc/shadow, akkor az abban tárolt adatokat is figyelembe veszi a program.

pwunconv E parancs visszahelyezi a passwd állományba a jelszavakat, majd törli a shadow állományt, így visszatérve a hagyományos jelszótárolásra.

grpconv A parancs a /etc/group állományból elkészíti a gshadow állományt. Ha a gshadow állomány már létezik, akkor az abban tárolt információkat is figyelembe veszi.

grpunconv A parancs a /etc/gshadow állományból visszaállítja a /etc/group eredeti tartalmát, és törli a gshadow állományt.

Ezek a programok a kézikönyvlapok szerint különös dolgokat képesek művelni, ha az egyes állományok hibásak. Mielőtt használnánk őket, futtassuk le a pwck és a grpck programokat, hogy azokkal ellenőrizzük a passwd és a group állományokat.

8. fejezet

Be- és kijelentkezés

8.1. Beléptetés

A következő fejezetben megvizsgáljuk, miképpen zajlik a bejelentkezés, mi történik a felhasználó bejelentkezése közben.

A felhasználók többféleképpen is bejelentkezhetnek egy GNU/Linuxot futtató számítógépre. A legegyszerűbb esetben a számítógép előtt ülve valamelyik karakteres munkafelületen gépelik be a felhasználói nevüköt és a jelszavukat. Ezt az esetet vizsgáljuk meg alaposabban a következő oldalakon.

A felhasználók bejelentkezhetnek grafikus felületen is. Ezt az esetet a grafikus felület bemutatása során tárgyaljuk, bár a grafikus bejelentkezési folyamat megértéséhez ez a fejezet is hasznos lehet, hiszen néhány programcsomag minden bejelentkezési folyamat esetében szerepet kap.

A bejelentkezés történhet számítógép-hálózaton keresztül is. A hálózaton történő bejelentkezés működésére szintén visszatérünk a későbbi fejezetekben.

8.1.1. A hagyományos bejelentkezés folyamata

A UNIX rendszerek bejelentkezési folyamata többé-kevésbé megegyezik, azaz a bejelentkezés általában hasonlóképpen megy végbe minden UNIX rendszeren. A bejelentkezésről szóló fejezet ezen szakasza a hagyományos, UNIX rendszerekben megszokott elemeket mutatja be.

Üzenetek bejelentkezéskor

A bejelentkezés előtt a UNIX rendszerek a /etc/issue állomány tartalmát írják ki a helyi képernyőre. (Távoli bejelentkezés esetén a felhasználó képernyőjén a /etc/issue.net állomány tartalma jelenik meg.) Ezt az állományt a rendszergazda a bejelentkezéshez szükséges információk kiíratására használhatja.



Szokás a számítógépre, a rajta található operációs rendszerre vonatkozó információkat elhelyezni a /etc/issue.net állományban. Ez biztonsági szempontból igen szerencsétlen, hiszen így az esetleges támadó minden erőfeszítés nélkül információkat szerezhet a számítógépen futtatott szoftverrendszeről, annak gyenge pontjairól.

Szerencsésebb, ha a távoli bejelentkezés előtt a számítógép az üzemeltető személy vagy személyek adatait írja ki, hogy sikertelen bejelentkezés esetén a felhasználók tudják, kihez fordulhatnak segítségért.

Néhány GNU/Linux terjesztés esetében a /etc/issue és a /etc/issue.net rendszerindításkor felülíródik. Ha az állományokat állandó tartalommal kívánjuk feltölteni a /etc/rc.d/rc.local vagy a /etc/init.d/rc.local állományok megváltoztatásával gondoskodnunk kell arról, hogy az állományokat a rendszer a rendszerindításkor ne írja felül.

A /etc/motd állomány tartalma a sikeres bejelentkezés után jelenik meg, akkor, amikor a felhasználó a hiteles bejelentkezési nevét és jelszavát beírta, és annak ellenőrzése megtörtént. A rendszerre jellemző információkat inkább ebbe az állományba tegyük, hiszen ezt csak azok lábják, akik érvényes névvel és jelszóval rendelkeznek. A misztikus MOTD rövidítés mögött a „napi hír” (*message of the day*) kifejezés áll.

A beléptetés

A felhasználó beléptetését UNIX rendszereken a login program végzi. Az, hogy milyen program indítja a login programot, az egyes UNIX rendszereken, sőt az egyes GNU/Linux terjesztésekben is meglehetősen változó, a login azonban általában biztos pont, amelyet érdemes már most megvizsgálnunk.

A login fő feladata a felhasználó azonosítása és a felhasználó alapértelmezett héjprogramjának elindítása az azonosítás után. A login ezt a feladatot a következő lépések elvégzésével látja el:

1. Ha a login program indulásakor még nem ismert a felhasználó felhasználói neve, a login megkérdezi.
2. Ha az adott felhasználó bejelentkezése jelszóhoz kötött, a login megkérdezi a jelszót. A jelszó begépelésekor a képernyőn *semmi* nem jelenik meg, hogy ne lehessen kikémszerni a jelszót és azt sem, hogy az hány karakterből áll. Érdemes megemlítenünk, hogy az egyes login-megvalósítások kínosan ügyelnek arra, hogy a jelszó begépelésekor a már bejelentkezett felhasználók ne hallgathassák le a felhasználóval kiépített kapcsolatot.
3. Ha a bejelentkezés minden feltétele adott, a login elindítja az azonosított felhasználó alapértelmezett héjprogramját. A héjprogram elindítását a program úgy végzi, hogy az már az azonosított felhasználó nevében fusson, az azonosított felhasználó jogaival rendelkezzen.

4. Ezek után a login vár, amíg a héj fut. Amikor a felhasználó kilép a héjból, a login is kilép, hogy az őt indító program tudja, új felhasználó jelentkezhet be.

A login a UNIX rendszereken néhány apró feladatot is elvégez, amelyeket érdemes megismernünk.

Ha a /etc/nologin állomány létezik, akkor csak a rendszergazda léphet be. minden más felhasználó belépési kérelmét elutasítja a login, az elutasítást a /etc/nologin tartalmának kiírásával jelezve. Ha tehát a felhasználók bejelentkezését valamilyen okból kifolyólag meg akarjuk akadályozni, az okot egyszerűen be kell írnunk a /etc/nologin állományba. Ezt mutatja be a következő egyszerű példa.

64. példa A rendszergazda szeretné megakadályozni, hogy a felhasználók belépjenek. Ehhez a következő parancsot adja ki:

```
$ echo "Karbantartas miatt zarva" >/etc/nologin  
$
```

Ha most a felhasználó be akar lépni, a login megakadályozza ebben a megadott kifogást használva:

```
Fedora Core release 2 (Tettnang)  
Kernel 2.6.5-1.358 on an i686
```

```
localhost login: pipas  
Password:  
Karbantartas miatt zarva  
  
Login incorrect
```

```
login:
```

Figyeljük meg, hogy a visszautasítás csak az után jelenik meg, hogy a felhasználó a jelszavával sikeresen azonosította magát. Azok, akik nem rendelkeznek érvényes felhasználói azonosítóval, nem tudhatják meg, hogy karbantartás folyik. Amikor a rendszergazda elvégezte a karbantartást, a következő parancsot adja ki:

```
$ rm -f /etc/nologin  
$
```

Amint törölte az állományt, a felhasználók újra bejelentkezhetnek.

UNIX rendszereken a /etc/securetty állomány tartalmazza azoknak a termináloknak az azonosítóját, amelyeken keresztül a rendszergazda beléphet. Amikor a rendszergazda lép be, a login ellenőrzi, hogy megtalálható-e az általa használt terminál ebben az állományban, és ha nem, akkor magyarázat nélkül elutasítja a belépési kérelmet akkor is, ha egyébként a jelszóval minden rendben van.

A belépést megkísérő személy biztonsági okokból még azt sem fogja látni, hogy a jelszó helyes vagy helytelen volt-e.

Az egyes UNIX rendszerek, sőt az egyes GNU/Linux terjesztések sem mindig egységesen nevezik el azonban a terminálokat. Ha tehát meg akarjuk engedni, hogy a rendszergazda más helyről is beléphessen, ki kell derítenünk, hogy mit kell írni a `/etc/securetty` állományba. Ezt egyszerűen megtehetjük, úgy, hogy az új terminálról bejelentkezünk egyszerű felhasználóként, és a `who` parancs kimenetén megjelenő táblázat második oszlopában található terminálnevet vesszük alapul.



Egyes programok nem veszik figyelembe a /etc/securetty állományt, mert titkosítást használva biztonságos bejelentkezési felületet hoznak létre. Ilyen program például az ssh, amelynek segítségével a rendszergazda számítógép-hálózaton keresztül is elvégezheti munkáját.

A `login` bejelentkezéskor megvizsgálja, hogy a `/var/spool/mail/` könyvtárban van-e a felhasználó felhasználói nevével megegyező nevű állomány, és az 0 hosszúságú-e. Ha nem 0 a hossza az állománynak, akkor kiírja a képernyőre, hogy a felhasználónak olvasatlan levele van.

Nyilvánvaló, hogy a levéltovábbító alrendszer a beérkezett elektronikus leveleket a `/var/spool/mail/` könyvtárban tárolja, minden felhasználó számára egy állományt tartva fenn. Az is magától értetődik, hogy a felhasználói levelezőprogram az olvasott elektronikus leveleket eltávolítja innen, így ha az állomány nem üres, a felhasználónak olvasatlan levele van. Az elektronikus levelek kezelésére a későbbiekben visszatérünk.

Ha a felhasználó a `.hushlogin` (*hush*, csendes, nyugodt) rejtett állományt elhelyezi a saját könyvtárában, akkor a belépéskor a `login` nem ellenőrzi a levelesládat, és nem írja ki az utolsó belépés időpontját – jóllehet a beléptetést a szokott módon elvégzi.

A `login` következő feladata a bejelentkezés naplázása. A Unix rendszerek a `wtmp` és `utmp` állományokban rögzítik a felhasználók bejelentkezéseit, így a `login` is ezekben az állományokban rögzíti a bejelentkezéseket és a kilépéseket.

A rendszerhasználat naplázására a későbbiekben még visszatérünk, de már most érdemes annyit megemlítenünk, hogy a `w` és a `who` programok is ezt a nyilvántartást használják a pillanatnyilag belépett felhasználók listájának kiírására.

A `login` szerencsés esetben túljutva ezeken az adminisztratív feladatokon elindítja a felhasználó héjprogramját, azaz kiolvassa a felhasználói nyilvántartásból, hogy milyen héjat használ a felhasználó, és elindítja azt. Azt gondolhatnánk, hogy ez már nem lehet túlságosan bonyolult, sajnos azonban a héj indítása korántsem egyszerű folyamat.

Tovább bonyolítja a helyzetet az, hogy UNIX rendszereken sokféle héj létezik és ezek viselkedése már induláskor különböző. A következő szakaszban részletesen megvizsgáljuk a GNU/Linux rendszereken elterjedten használt BASH héjat ebből a szempontból, azaz áttekintjük, milyen folyamatok játszódnak le a BASH héj indításakor.

8.1. tábla: login

A felhasználó beléptetését végző program.

```
login [kapcsolók] [felhasználói_név]
```

A beléptetést végző program, amely automatikusan fut a belépéstől a kilépésig helyi karakteres bejelentkezés esetén. A felhasználó jelszavának ellenőrzését, a belépés és kilépés naplázását és az alapértelmezett héj indítását végzi.

Kapcsoló	Jelentés
-p	A meglévő környezeti változók továbbadása.
-h gép-név	A felhasználó távoli gépének nevét átadó kapcsoló (csak rendszergazdaként).

8.1.2. A héj indítása

A BASH héj a legelterjedtebben használt héj GNU/Linux környezetben, és elérhető más UNIX rendszereken is[6, 20, 18]. Igen sok GNU/Linux-felhasználónak a BASH az alapértelmezett héjprogramja, ezért áttekintük, hogy mi történik a BASH indításakor. Igen fontos ez a rövid áttekintés, igen fontos, hogy meg tudjuk, mi történik pontosan, amikor a héj elindul. Valójában minden BASH-felhasználónak pontosan tisztában kellene lennie azzal, hogyan indul az általa használt héj, a rendszer üzemeltetésével megbízott szakembernek azonban létérdés ez az ismeretanyag.

A BASH alapjában véve három üzemmóddal rendelkezik, háromféleképpen indulhat el:

- Beléptető héj üzemmód (*login shell mode*).

Ez az üzemmód kimondottan a belépéskor való indításra készült; akkor szokás használni, amikor a felhasználó belépésekor alapértelmezett héjként indul a BASH.

- Interaktív héj üzemmód (*interactive shell mode*).

Az interaktív kifejezés arra utal, hogy a BASH akkor használja ezt az üzemmódot, amikor közvetlenül a felhasználóval áll kapcsolatban, a felhasználótól kapja a parancsokat.

Ez természetesen nem jelenti azt, hogy a *beléptető üzemmód* nem interaktív jellegű! Az *interaktív üzemmód* egyik jellemzője az, hogy interaktív kapcsolattartást valósít meg, de más üzemmódban is használható interaktív kapcsolattartással a program.

- Nem interaktív héj üzemmód (*non-interactive shell mode*).

Ez az üzemmód szöveges állományban elhelyezett programok futtatására szolgál. Ilyenkor a BASH nem egy felhasználóval tartja a kapcsolatot, hanem egy állományból veszi az utasításokat, és ennek megfelelően kissé másképp viselkedik.

Érdemes néhány szót ejteni arról, hogy miért is van szükség erre a három üzemmódra, miért kellett a BASH viselkedését bonyolítani ezek bevezetéséval.

A beléptető üzemmód bevezetésére elsősorban erőforrás-takarékkossági okokból volt szükség. Amikor a BASH beléptető üzemmódban indul, beállítja magának a munkakörnyezetet. A munkakörnyezet beállítására a további indított BASH-példányok számára azonban már nincs szükség, hiszen a folyamatok a környezeti változókat öröklik a szülőfolyamataiktól. A környezeti változók beállítására tehát csak az első BASH-példány indításakor van szükség, a többi BASH-példány esetében felesleges volna a beállításokat elvégezni. (Nyilvánvaló, hogy a mai számítógépek és programok esetében ilyen csekély pazarlás elfogadható volna, szenencsére azonban ez nem volt mindig így.)

Az interaktív üzemmód esetében a BASH a felhasználóval áll közvetlen kapcsolatban. A felhasználó számára a BASH parancskérő jelet ír a képernyőre, amikor parancsok fogadására kész, sőt a felhasználó számára parancsszerkesztő eszközöket is biztosít, hogy az könnyebben begépelhesse utasításait. Nyilvánvaló, hogy a nem interaktív módban, amikor a BASH a parancsokat állományból veszi, nem kell parancskérő jelet írni a képernyőre. Az interaktív és a nem interaktív üzemmódok megkülönböztetése tehát elsősorban kényelmi okokból szükséges.

A következőkben vizsgáljuk meg, hogy mitől függ, hogy melyik üzemmódban indul a BASH!

A BASH beléptető üzemmódban indul, ha indításkor a nevének az első betűje a „-“ karakter vagy ha a -login kapcsolót kapja. Ha a két feltétel közül legalább az egyik fennáll, a BASH mindenkorban beléptető héj üzemmódban indul el.

A login program a BASH héjat GNU/Linux rendszereken -bash néven indítja, ahogyan ezt a következő példa mutatja:

65. példa A w program segítségével megvizsgáljuk, hogy mivel foglalatoskodnak a belépett felhasználók, és a következő kimenetet kapjuk:

```
$ w
 08:43:47 up  2:03,  4 users,  load average: 0,01, 0,03, 0,06
USER     TTY    FROM      LOGIN@     IDLE     JCPU    PCPU WHAT
pipas    tty1   -          07:15     1:07m   0.27s   0.27s -bash
```

Láthatjuk az utolsó sor utolsó oszlopában, hogy a -bash néven futó BASH-példány neve a „-“ karakterrel kezdődik, amiből tudjuk, hogy ez egy beléptető héj üzemmódban futó BASH.



Ha megpróbáljuk megkeresni a könyvtárszerkezetben a -bash állományt, megdöbbenve tapasztaljuk, hogy ilyen program nem létezik. Kíváncsiak lehetünk rá, hogyan futhat egy program, ami nem létezik?

UNIX rendszereken a folyamat, amely egy új folyamatot hoz létre, megadhatja a folyamat paramétereit, amelyek közül szokás szerint a legelső (valójában a nulladik), annak a programállománynak a neve, amelyet az új folyamat futtat. Ez

azonban nem feltétlenül van így! Az új folyamat legelső (nulladik) paramétere és a programállomány különbözhet egymástól.

Ezt a lehetőséget használja ki a login, amikor a /bin/bash állomány tartalmát futtatja -bash néven.

Ha a BASH nem kapott -login kapcsolót, és indításkor a neve nem „-” karakterrel kezdődött, a héj nem beléptető üzemmódban indul el. Ilyenkor a program megvizsgálja, hogy a bemenete és kimenete terminálhoz kapcsolódik-e. Ha a be- menet és kimenet terminálhoz kapcsolódik, a program feltételezi, hogy a terminált egy felhasználó használja, és interaktív üzemmódban indul el.

Ha a BASH nem beléptető üzemmódban és nem is interaktív üzemmódban indul, akkor az indulás nem interaktív üzemmódban történik.

A BASH induláskor különféle állományokat keres, olvas és hajt végre attól füg- gően, hogy milyen üzemmódban indul el. A rendszergazdának ismernie kell eze- ket az állományokat, hiszen az induláskor betöltött és végrehajtott állományok segítségével rugalmasan testeszabhatja a felhasználók munkakörnyezetét. (Más kérdés persze, hogy ezt a felhasználóknak kellene megtenniük, de legtöbbük ilyen mélységen nem kívánja megtanulni a számítógép kezelését.)

Amikor a BASH beléptető üzemmódban indul el, a következő állományokat ol- vassa és hajta végre induláskor:

/etc/profile Ha a /etc/profile állomány létezik, a BASH először ezt tölti be.

Ezt az állományt a rendszergazda általában a mindenire érvényes beállítá- sok érvényesítésére használja.

\$HOME/.bash_profile Ha létezik a felhasználó saját könyvtárában a .bash_profile állomány, akkor második lépéssben ezt az állományt tölti be és hajta végre a BASH.

A felhasználók ebben az állományban testeszabhatják munkakörnyezetü- ket.

\$HOME/.bash_login Ha a BASH nem találja a .bash_profile állományt a felhasználó saját könyvtárában, akkor megkísérli ugyaninnen betölteni a .bash_login állományt.

\$HOME/.profile Ha a .bash_login állomány sem létezik, akkor a BASH megkí- sérli betölteni a .profile állományt.

\$HOME/.bashrc A BASH ezek után mindenkor megkísérli, hogy betöltse és végrehajtsa a felhasználó saját könyvtárából a .bashrc állományt.

A BASH beléptető üzemmódban kilépéskor megkísérli betölteni és lefuttatni a felhasználó saját könyvtárában található .bash_logout állományt. Ez az állo- mány kitűnően használható arra, hogy kilépéskor ideiglenes állományokat töröl- jünk, letöröljük a képernyőt és így tovább.

Ha a BASH interaktív üzemmódban indult el, a következő állományt olvassa be:

```
$HOME/.bashrc Az interaktív módban induló BASH betölti és végrehajtja a fel-
használó saját könyvtárában található .bashrc állományt.
```

A nem interaktív módban induló BASH a következőképpen viselkedik:

```
$BASH_ENV A nem interaktív üzemmódban induló BASH megvizsgálja
a $BASH_ENV környezeti változót. Ha a környezeti változó létezik és
a benne található néven állomány érhető el, a BASH megkísérli megnyitni
és végrehajtani az állomány tartalmát.
```

Meg kell jegyeznünk, hogy az itt bemutatott viselkedés a BASH --posix kapcsolójának hatására némmiképp módosul, és akkor is, ha a programot a /bin/sh néven indítottuk el. Ezekben az esetekben a BASH szigorúan a szabvány szerint viselkedik.

Érdemes azt is megjegyeznünk, hogy az /etc/profile állományt nem csak a BASH, hanem más héjak is használhatják. A rendszergazdának úgy kell megírnia az állományban található programot, hogy az több héj nyelvén is értelmes legyen, ettől a feladattól azonban az igazi rendszergazda nem riad vissza.

Érdemes megfigyelnünk, hogy a BASH a bemutatott állományokat nem külső programként futtatja, hanem betölti és végrehajtja a bennük található utasításokat. A BASH source és . belső parancsával ezt mi magunk is megtehetjük, azaz mi magunk is beilleszthetünk külső állományokat a héjprogramba. Ezt mutatja be a következő példa.

66. példa Bizonyos GNU/Linux terjesztésekben szokás létrehozni és használni a /etc/bashrc állományt, hogy azt a beléptető és az interaktív üzemmódban is betöltsük.

A /etc/bashrc állomány betöltését végzi el a következő programrészlet:

```
1 if [ -f /etc/bashrc ]; then
2     . /etc/bashrc
3 fi
```

A program az 1. sorban megvizsgálja, hogy létezik-e a /etc/bashrc, és ha igen, a 2. sorban végrehajtja annak tartalmát.

Ha ezt a programrészletet elhelyezzük a \$HOME/.bashrc állományban, akkor a BASH valahányszor betölti a .bashrc állományt, mindenkor ugyanis ez a /etc/bashrc állományt is.

A rendszerszintű felhasználói profil

Amint láttuk a beléptető módban indított BASH betölti és végrehajtja a /etc/profile állományt, amelyet a rendszergazda arra használhat, hogy a minden felhasználó számára érvényes beállításokat elhelyezze benne. Szokás ezt

Rövidítés	Jelentés
\d	a dátum
\h	a gép neve az első pont karaterig
\H	a gép teljes neve
\j	a héjból indított feladatok száma
\l	a terminál neve
\t	az idő 24 órás formában
\T	az idő 12 órás formában
\u	a felhasználó felhasználói neve
\w	a munkakönyvtár neve
\W	a munkakönyvtár nevének utolsó szakasza

8.2. táblázat. A BASH parancskérő jelének rövidítései

az állományt rendszerszintű felhasználói profilnak (*system wide user profile*) is nevezni. A következő oldalakon a gyakorlat szempontjából hasznos eszközöket veszünk sorra, melyeket ebben az állományban szoktunk használni. Olyan mély-séggig próbáljuk meg tárgyalni a rendszerszintű felhasználói profilt, hogy az olvasó az ismeretek birtokában megértse a legtöbb GNU/Linux terjesztéssel telepített profilt, és az igényeknek megfelelően tudja módosítani azt.

A `/etc/profile` főképpen környezeti változók értékét állítja be. A környezeti változókat, amelyeket az `export` parancs segítségével beállítunk, gyakorlatilag minden folyamat megkapja. A beléptető módban indított BASH átadja ezeket a környezeti változókat az összes gyermekfolyamatának.

Szokás ebben az állományban beállítani a `$PATH` környezeti változó értékét, amely meghatározza, hogy a héj minden könyvtárakban keresse a programokat. A `$PATH` változó kettőspontokkal elválasztva tartalmazza azoknak a könyvtáraknak a nevét, amelyekben a héj a felhasználó által begépelet és a héjprogramokban szereplő programneveknek megfelelő állományokat keresi.

A `$PATH` változóban a felsorolt könyvtárak sorrendjének is fontos szerepe van. Ha több könyvtárban is megtalálható ugyanazzal a névvel a program, a héj azt fogja elindítani, amelyiket a `$PATH` változó alapján előbb megtalálja.

A rendszerszintű felhasználói profilban szokás beállítani a parancskérő jelet (*prompt*) is. Parancskérő jelként a BASH a `$PS1` környezeti változóban tárolt értéket írja ki, elegendő tehát a `/etc/profile` állományban ennek a változónak értéket adni, hogy meghatározzuk, mi legyen a parancskérő jel.

A BASH a parancskérő jelben található különleges rövidítéseket figyelembe veszi, a megfelelő értékkel helyettesíti. A `$PS1` környezeti változóban használható rövidítéseket a 8.2. táblázatban olvashatjuk.

A `/etc/profile` állományban, a rendszerszintű felhasználói profilban az `ulimit` parancs segítségével szokás bizonyos erőforrások felhasználását korlátozni a felhasználók számára. Az `ulimit` parancs a BASH beépített parancsa.

8.3. tábla: ulimit

A BASH beépített parancsa, amely a felhasználható erőforrások korlátozására szolgál.

`ulimit [kapcsolók [érték]]`

A parancs segítségével a BASH és az általa indított folyamatok által felhasználható erőforrások korlátozhatók. A korlátok értéke kilobájtból értendő, kivéve a -u kapcsoló esetében, ahol darabszámban.

Kapcsoló Jelentés

- a A pillanatnyilag érvényes korlátok kiíratása.
- c A programhiba esetén készítendő hibakereső állomány (`core`) legnagyobb mérete.
- s A verem legnagyobb mérete.
- u Az egyidőben futtatható folyamatok száma.
- S Az áthágható korlát beállítása és lekérdezése.
- H Az áthághatatlan korlát beállítása és lekérdezése.

Az ulimit parancs segítségével áthágható (*soft*, lágy) és áthághatatlan (*hard*, kemény) korlátokat vezethetünk be a felhasználható erőforrásokra nézve az alkalmazások számára. Nem tárgyaljuk részletesen ennek a parancsnak a használatát, mert a későbbiekben hatékonyabb eszközt mutatunk be az erőforrások felhasználónként és csoportonként való korlátozására.

Ha egy program futása súlyos hiba miatt megszakad, a BASH a futtatást végző folyamat memóriaterületét állományba menti. A létrehozott, hibakeresésre használható állomány a munkakönyvtárban jön létre `core` néven.

A legtöbb felhasználó nem fog neki a hibakeresésnek, ha egy program nem működik, így a felhasználók többsége nem rajong túlságosan a különböző könyvtárakban felhalmozódott hatalmas méretű `core` állományokért, sőt a legtöbben szeretnék kiüzni életükönök ezeket. Habár ma már a legtöbb GNU/Linux terjesztés alapbeállítás szerint tiltja `core` állományok létrehozását, a következő példa hasznos lehet az ulimit beépített parancs bemutatására.

67. példa Szeretnénk letiltani a `core` állományok létrehozását minden felhasználó esetében. Első lépésként megvizsgáljuk az érvényes beállításokat:

```
$ ulimit -H -c
10000
$ ulimit -S -c
10000
```

Láthatjuk, hogy mind az áthágható, mind pedig az áthághatatlan korlát 10 000 kilobájt. Állítsuk a korlátokat 0 értékre, így az állomány egyáltalán nem jön létre:

```
$ ulimit -c 0  
$
```

Ha kipróbáltuk a beállítást, módosíthatjuk a /etc/profile állományt, elhelyezve benne az új beállítást, hogy az minden felhasználó belépésekor lefutva minden felhasználó számára érvényes legyen:

```
1 # A core állományok létrehozásának tiltása  
2 ulimit -c 0 >/dev/null 2>&1
```

A rendszerszintű felhasználói profilban beállíthatjuk a létrehozandó könyvtár-bejegyzések jogosultságait, azaz beállíthatjuk, hogy amikor a felhasználó létrehoz egy új állományt vagy könyvtárat, ahhoz milyen hozzáférési jogok tartozzanak. A beállítást a BASH umask belső parancsával végezhetjük el.

Nagyon hasznos az umask parancs, különösen akkor, ha az általunk üzemeltetett GNU/Linux rendszert olyanok is használják, akik nincsenek tökéletesen tisztában a jogosultságok használatával és fontosságával (azaz mindig). Az ilyen felhasználók általában létrehozzák a könyvtárbejegyzéseket, de a jogosultságok beállításával már nem foglalkoznak, adataikat tehát az umask segítségével beállított maszk védi meg.

Amikor a programok szabályos állományokat hoznak létre, annak jogait alapértelmezés szerint általában 666 értékre állítják be. Ez azt jelenti, hogy mindenkinek írasi és olvasási jogot adnak az állományhoz. Amikor a programok könyvtárat hoznak létre, annak jogait alapértelmezés szerint 777 értékre állítják. Ez azt jelenti, hogy *mindenkinek olvasási, írasi és belépési jogot adnak a könyvtárhoz*. Ezek az alapértelmezés szerint beállított értékek a legtöbb esetben túlságosan elnözők, így a legtöbb esetben használnunk kell az ulimit parancsot az új könyvtár-bejegyzésekhez tartozó jogok korlátozásához. (Szerencsére az összes GNU/Linux terjesztés telepítés után már tartalmaz valamilyen – az ulimit segítségével bevezetett – korlátozást.)

Az ulimit parancssal megadott maszk az alapértelmezett jogokból kivonódik, azaz amely jog a maszkban be van kapcsolva, ott az alapértelmezett jogban tiltás lesz. Minél szigorúbb munkakörnyezetet kívánunk teremteni, annál több jogot kell megadnunk a maszk elkészítésekor.

Az ulimit segítségével megadott maszkot a folyamatok átadják egymásnak, azaz a maszk nem csak a BASH segítségével indított folyamatok viselkedését határozza meg, minden folyamat viselkedését befolyásolja. Nem szabad azonban szem elől tévesztenünk, hogy a folyamatok bármelyike megváltoztathatja a maszkot, és a felhasználók is bármikor átállíthatják a könyvtárbejegyzésekhez tartozó jogot a chmod parancs segítségével.

A következő példa bemutatja, hogyan állíthatunk be szigorú maszkot a felhasználók számára.

8.4. tábla: umask

Az újonan létrehozott könyvtárbejegyzésekhez alapértelmezés szerint rendelt jogosultságok meghatározása.

umask [maszk]

Ha a parancs után a chmod programnál megszokott formában egy maszkot adunk meg, az kivonódik az újonan létrehozott könyvtárbejegyzések jogaiból. Ha a maszkot nem adjuk meg, a program kiírja az érvényben lévő maszk értékét.

68. példa Első lépésként kérdezzük le a maszk értékét, és vizsgáljuk meg, milyen hatással van az újonan létrehozott szabályos állományok és könyvtárak jogaira:

```
$ umask
0006
$ touch fajl; mkdir konyvtar
$ ls -ld fajl konyvtar/
-rw-rw---- 1 pipas pipas 0 jún 12 10:31 fajl
drwxrwx--x 2 pipas pipas 4096 jún 12 10:31 konyvtar/
$
```

Láthatjuk, hogy az új állományok jogai meglehetősen elnézőek. Állítsuk át a maszkot úgy, hogy az csak a tulajdonos számára tegye lehetővé a hozzáférést új állományok létrehozásakor! Vizsgáljuk meg, hogy az új maszk milyen hatással van az új könyvtárbejegyzések létrehozására!

```
$ umask 0077
$ rm -rf fajl konyvtar/
$ touch fajl; mkdir konyvtar
$ ls -ld fajl konyvtar/
-rw----- 1 pipas pipas 0 jún 12 10:36 fajl
drwx----- 2 pipas pipas 4096 jún 12 10:36 konyvtar/
$
```

Amint látjuk, a jogok megnyirbálására irányuló törekvésünket siker koronázta, így a megváltoztatott maszkot elhelyezhetjük a /etc/profile állományban:

```
1 # Szigorú maszkot állítunk be
2 umask 0077
```

8.2. A biztonsági házirend

A GNU/Linux használata közben megfigyelhető, hogy bizonyos műveletek, szolgáltatások működését szigorú biztonsági előírások szabályozzák. A következő oldalakon ezeknek a biztonsági előírásoknak a beállításáról, finomhangolásáról lesz szó.

A legtöbb GNU/Linux rendszer biztonsági előírásai megfelelnek a legtöbb felhasználó és rendszergazda igényeinek, így szerencsés esetben a biztonsági rendszer finomhangolására semmi szükség, az úgy működik, ahogyan a telepítőprogram beállította. A legtöbb GNU/Linux rendszert üzemeltető szakember tehát egyszerűen átugorhatja a biztonsági házirend módosításáról szóló részt, amelyet csak akkor kell elolvasnia, amikor különleges feladat előtt áll.

A következő néhány példa bemutatja, hogy milyen kérdésekre ad választ a biztonsági házirendről szóló szakasz. A példák remélhetőleg segítenek az olvasónak, hogy eldöntse, szüksége van-e ezekre az ismeretekre.

1. A legtöbb GNU/Linux a számítógépre belépett felhasználó számára lehetővé teszi a számítógép leállítását és újraindítását, de csak akkor, ha a felhasználó nem távoli gépről jelentkezett be.

Hogyan lehetne megváltoztatni ezt a viselkedést? Hogyan lehetne elérni, hogy bárhonnan le lehessen állítani a gépet, vagy éppen azt, hogy kizárálag csak a rendszergazda állíthatssá le a számítógépet?

2. A felhasználók a számítógépre általában a nap 24 órájában, a hét 7 napján bejelentkezhetnek.

Hogyan lehetne a bejelentkezést időben korlátozni, azaz meghatározni, hogy a felhasználók csak a nap bizonyos óráiban, a hét bizonyos napjain jelentkezhessenek be (például munkaidőben)?

3. A GNU/Linux rendszereken a felhasználó a jelszavát általában megváltoztathatja. A jelszóváltoztatáshoz meg kell adni a régi jelszót és az új jelszót is, amelyet a rendszer szigorúan ellenőriz, hogy elegedően bonyolult legyen.

Hogyan lehetne megváltoztatni ezt a viselkedést, és megengedni például, hogy a felhasználó egyszerű jelszót is beállíthasson, esetleg egyszerűen megtiltani a jelszó megváltoztatását?

4. Alapbeállítás szerint a GNU/Linux rendszerek a háttértáron tárolják a felhasználók adatait, így ha több számítógépen akarunk hozzáférést biztosítani a felhasználó számára, annak adatait több helyen tárolnunk kell.

Hogyan lehet elérni azt, hogy a felhasználói adatok egy távoli számítógépről, központi nyilvántartásból származzanak, hogy a felhasználók adatait csak egy helyen kelljen tárolni?

Az ilyen jellegű kérdések megválaszolásához a rendszergazdának ismernie kell a Linux-PAM (*Pluggable Authentication Modules for Linux*, csatlakoztatható azonosító modulok Linuxhoz) rendszert, mert a GNU/Linux rendszerek ezt a programcsomagot használják a biztonsági házirend betartatására. A következő oldalakon a Linux-PAM (vagy egyszerűen PAM) felépítéséről, működéséről és beállításáról lesz szó.

8.2.1. A Linux-PAM felépítése

A UNIX rendszerek többfelhasználós munkakörnyezetet biztosítanak, ezért a biztonság fontos szerepet kap az üzemeltetésük során. A GNU/Linux rendszer sok eleme kapcsolódik valamilyen módon a biztonság kérdésköréhez.

Sok program található a GNU/Linux rendszerekben, amelyeknek olyan kérdéseket kell eldöntení, hogy a felhasználó valóban az-e, akinek mondja magát, hogy egy bizonyos szolgáltatás az adott időpillanatban elérhető-e, hogy a felhasználó az adott szolgáltatást jogosult-e igénybe venni és így tovább. Az ilyen kérdéseket gyűjtőfogalomként hitelesítése kérdéseknek, hitelesítési feladatoknak nevezzük.

A Linux-PAM előtti időkben az ilyen feladatokat minden alkalmazásnak magának kellett elvégeznie. Ha a programozó a munkája során hitelesítési feladattal találta magát szemben, akkor a feladatot ellátó programrész magának kellett megírnia. Így aztán hitelesítési feladatokat végző programrészek kerültek a beléptetést végző programba, az elektronikus levelezést megkönnyítő alkalmazásba, a webkiszolgálóba és az alkalmazások garmadájába. Nyilvánvaló, hogy ennél sokkal hatékonyabb, ha a hitelesítést végző programrészeket külön kezeljük, és az alkalmazások számára hozzáférhető módon, szolgáltatásként vezetjük be. Vegyük sorra, milyen előnyöket remélhetünk egy közösen használható, általános hitelesítési rendszertől!

- A hitelesítést végző programrészek fejlesztése egy helyen történik, így a fejlesztéshez használt erőforrásokat jobban ki tudjuk használni. A programozók, ahelyett, hogy sok alkalmazásban újra és újra kifejlesztenék a hitelesítési programrészeket, csak egyszer végzik el a feladatot, és így több energiájuk marad annak finomítására és fejlesztésére.
- A módosításokat csak egy helyen kell elvégezni, az alkalmazásokat nem kell módosítani, ha a hitelesítési programrész módosítjuk. Ha például egy hibát javítunk a hitelesítési programrészben, nem kell lecserélni a levelezőprogramot, webkiszolgálót és adatbázis-kezelő rendszert, csak a hitelesítési csomagot kell frissíteni.
- A hitelesítési rendszer finomhangolását az alkalmazások ismerete nélkül elvégezhetjük, akár külön könyvfejezetben tárgyalhatjuk a témát, hiszen az alkalmazások ismerete nem szükséges az általuk használt hitelesítési beállításához.

A GNU/Linux terjesztések ezekből az okokból használják a Linux-PAM hitelesítési rendszert, amely ma már minden GNU/Linux terjesztés része. Természetesen más UNIX rendszerek is hasonlóan épülnek fel, azaz más UNIX rendszerek is használnak PAM hitelesítési alrendszert, ami felépítésében és beállításában többé-kevésbé hasonlít a Linux-PAM rendszerre. Ha tehát megismerkedünk a Linux-PAM rendszerrel, komoly reménytelent indulhatunk neki annak, hogy más UNIX rendszerek biztonsági házirendjét beállítsuk.



A biztonsági házirend módosításával könnyedén „kizárhatsuk” magunkat.

Ha a rendszergazda nem tud belépni, mert a PAM beállítóállományok helytelenül vannak beállítva, akkor nyilvánvalóan kijavitani sem tudja ezeket a beállításokat, így könnyen reménytelen helyzetbe kerülhet.

Egyszerűbb, kevés felhasználót kiszolgáló rendszerek esetében a rendszergazdának általában nincs szüksége arra, hogy a PAM beállításait, a biztonsági házirendet megváltoztassa, és ha mégis úgy dönt, hogy ilyen módosításokat végez, a legtöbb esetben elegendő valamelyik grafikus beállítóprogramot megismernie. Ma már a legtöbb GNU/Linux terjesztés rendelkezik használható beállítóprogrammal erre a célra. Ennek megfelelően a PAM rendszer működését és beállítását csak vázlatosan ismertetjük, csak a legszükségesebb ismereteket adjuk közre.

8.2.2. A PAM beállítóállományok

A Linux-PAM régebbi változatai a beállításokat a `/etc/pam.conf` állományban tárolták. minden beállítás, minden alkalmazás viselkedése ebben az állományban volt leírva.

A mai Linux-PAM-változatok minden alkalmazás számára külön állományt tartanak fenn. Ezek az alkalmazásoknál külön tárolt beállítások a `/etc/pam.d/` könyvtárban kapnak helyet.

A beállítások tárolásának két módszere nem különbözik túlságosan egymástól. Valójában csak egy különböszég figyelhető meg az egy beállítóállományt használó megoldás és a több beállítóállományra épülő újabb megoldás között. A régebbi, egy állományra épülő beállítás beállítóállományának első oszlopába a szolgáltatást kell írnunk, amelynek a beállítására az adott sor vonatkozik. A több állományra épülő beállítás esetében minden állomány egy szolgáltatást határoz meg, így az első oszlopot elhagyjuk.

A Linux-PAM szöveges beállítóállományai soronként egy biztonsági modul használatára adnak utasítást. Az állományokban `#` jellel készíthetünk megjegyzéket, csakúgy, mint a BASH programokban. A Linux-PAM beállítóállományai a következő oszlopokat tartalmazzák:

1. Az első oszlop határozza meg, hogy az adott sor melyik szolgáltatásra vonatkozik.

A szolgáltatás neve bármi lehet, a hagyomány szerint azonban általában megegyezik a szolgáltatást megvalósító program nevével. Így a szolgáltatás – például login, passwd – könnyen felismerhető.

A modern Linux-PAM-változatok több beállítóállományt használnak, minden szolgáltatáshoz egyet, ezért ez az oszlop nem szerepel az állományokban. A szolgáltatás neve a modern változatokban a beállítóállomány nevével egyezik meg.

Ha egy alkalmazás hitelesítést kér, de az általa használt szolgáltatásnévhez nem található meg a beállítás, a Linux-PAM az other szolgáltatásnevét használja. Ügyelnünk kell, hogy az other szolgáltatás igénybe vétele ne legyen ellenőrizetlen, hogy az ilyen esetek ne jelentsenek biztonsági problémát.

2. A következő oszlop a modul felhasználási módját, a modul típusát adja meg. Ez az oszlop tehát azt adja meg, hogy az adott modult milyen jellegű hitelesítési feladatra szeretnénk használni.

Ebben az oszlopból a következő kulcsszavak egyike szerepelhet:

auth Az ilyen kulcsszóval használt modulok a felhasználó azonosítását végzik, vagyis megállapítják, hogy a felhasználó valóban az-e, akinek mondja magát.

account Az ilyen kulcsszóval használt modulok megállapítják, hogy az adott szolgáltatás az adott körülmények között elérhető-e a felhasználó számára.

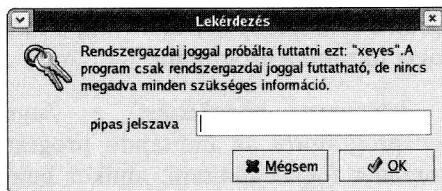
session Az ilyen kulcsszóval használt modulok nem csak a szolgáltatás előtt, hanem a szolgáltatás közben és a szolgáltatás után is szerepet kapnak. A felhasználók belépésének és kilépésének naplázása jó példa az ilyen jellegű modulhasználatra.

password Az ilyen kulcsszóval használt modulok a felhasználó azonosításához használt adatok módosításakor – például jelszóváltoztatáskor – kapnak szerepet.

3. A következő oszlop a modul fontosságát adja meg, vagyis azt, hogy az adott PAM modul sikeres végrehajtása szükséges vagy elégges-e a szolgáltatás igénybe vételéhez.

A modulok végrehajtása sikeres vagy sikertelen kimenetelű lehet. Ez az oszlop azt határozza meg, hogy mi történék az adott modul sikeres és sikertelen végrehajtása esetén. A következő kulcsszavakat használhatjuk itt:

required Az adott szolgáltatás igénybe vételéhez az ilyen típusú modul sikeres lefutása szükséges. Ha egy required modul sikertelenül fut le, a szolgáltatást nem lehet igénybe venni, bár erről a felhasználó csak a többi modul lefutása után értesül.



8.1. ábra. Jelszókérés grafikus felületen

requisite Ezek a modulok ugyanúgy elengedhetetlenül szükségesek a szolgáltatás igénybe vételéhez, mint a **required** modulok, azzal a különbséggel, hogy sikertelen lefutás esetén a felhasználó azonnal értesül arról, hogy a szolgáltatást nem veheti igénybe.

sufficient Az ilyen modulok sikeres lefutása a szolgáltatás igénybe vételéhez elégges. Ha egy ilyen modul sikeresen lefut, akkor más modulok futtatására már nincs szükség. A **sufficient** modul sikertelen lefutása nem jelenti azt, hogy a szolgáltatást nem lehet igénybe venni, azt más modul lehetővé teheti.

optional Az ilyen modulok sikeres lefutása nem szükséges a szolgáltatás igénybe vételéhez, ezek a modulok elhagyható szolgáltatásokat valósítanak meg.

4. A következő oszlopban a modul neve és elérési útja található, amelyet a modul esetleges kapcsolói követnek.

A beállítóállományok formátumának ismertetése után példákon keresztül mutatjuk be a különféle modulok használatát. A példák minden bizonnal hasznosak lehetnek a Linux-PAM beállításának megismerésében.

8.2.3. Rendszergazdai jogok biztosítása

A **consolehelper** program segítségével olyan programokhoz is PAM hitelesítést rendelhetünk, amelyeket szerzőjük nem készített fel erre. Fontos, hogy megértük, mire is való a **consolehelper** valójában!

A **consolehelper** program segítségével a felhasználók rendszergazdai jogkörrel, rendszergazdaként futtathatnak bizonyos programokat a megfelelő hitelesítés után. A programok tehát rendszergazdai jogkört kapnak a futás idejére, ami igen körültekintő munkát követel ennek a szolgáltatásnak a beállítása során.

A **consolehelper** használata viszonylag egyszerű. A következő lista receptként használható a beállítás során:

1. Helyezzük el a rendszergazdaként futtatandó programot a `/sbin/` könyvtárban. A rendszergazda által használt `$PATH` környezeti változóban ez

a könyvtár általában a többi könyvtár előtt van, a rendszergazda tehát közvetlenül innen fogja futtatni a programot.

2. Hozzunk létre közvetett hivatkozást a `/usr/bin/` könyvtárban a `consolehelper` programról az indítandó program nevére. A felhasználók ezt az állományt fogják indítani, amikor beírják a program nevét, a felhasználók tehát valójában a `consolehelper` programot indítják el.
3. Hozzunk létre egy üres állományt a `/etc/security/console.app/` könyvtárban az indítandó állomány nevén.
4. Készítsük el a Linux-PAM beállításokat a szolgáltatás számára, amelynek neve megegyezik az indítandó program nevével.

A következő példa bemutatja ezeket a lépéseket a gyakorlatban is.

69. példa Szeretnénk a `consolehelper` program segítségével futtatni az `xeyes` programot. Ez túl sok haszonnal nem kecsegett, de a segítségével kipróbalhatjuk a Linux-PAM beállítását és egy olyan szolgáltatáshoz jutunk, amelynek beállításával kísérletezhetünk.

Másoljuk át az `xeyes` állományt a `/sbin/` könyvtárba. A felhasználók a legtöbb GNU/Linux rendszeren az itt található programokat is futtathatják, de természetesen nem rendszergazdai jogkörrel.

Hozzunk létre közvetett hivatkozást a `consolehelper` programra, és hozzuk létre a `/etc/security/console.app/` könyvtárban az `xeyes` állományt.

```
$ mv /usr/X11R6/bin/xeyes /sbin/
$ cd /usr/bin/
$ ln -s consolehelper xeyes
$ touch /etc/security/console.app/xeyes
$
```

Most már csak a Linux-PAM beállítóállományt kell elkészítenünk az `xeyes` szolgáltatás számára. Kezdetnek megteszi a következő egyszerű beállítás a `/etc/pam.d/xeyes` állományban:

1	<code>auth required pam_unix.so</code>
2	<code>account required pam_unix.so</code>

Ha most a felhasználó beírja az `xeyes` parancsot, a `/usr/bin/` könyvtárban található program indul el, a `consolehelper` a beállításoknak megfelelően elvégzi a hitelesítést, és rendszergazdaként futtatja a `/sbin/` könyvtárból az `xeyes` programot. A jelszókéréshez megjelenített ablakot a 8.1. ábrán láthatjuk.

8.2.4. A modulok

A következő oldalakon a Linux-PAM moduljainak leírását olvashatjuk a használatuk bemutató példákkal.

Tiltás, engedélyezés és figyelmeztetés

A szakaszban néhány egyszerű PAM modult mutatunk be, amelyekkel a szolgáltatást mindenki számára engedélyezhetjük, mindenki számára letilthatjuk és a szolgáltatás igénybevételeit naplózhatjuk. Ilyen célokra a következő modulokat használhatjuk:

pam_deny.so Ez a modul minden sikertelenül fut le, egyébként azonban nem csinál semmit.

A modul auth, account, session és password feladat ellátására használható; egyik módban sem csinál semmit, és minden sikertelen értékkel tér vissza.

pam_permit.so Ez a modul minden sikeresen fut le, egyébként azonban semmit sem csinál.

A modul auth, account, session és password feladat ellátására használható; egyik módban sem csinál semmit, és minden sikeres értékkel tér vissza.

pam_warn.so A modul naplóbejegyzés elhelyezésére használható. A rendszernaplóba helyez információt a szolgáltatás igénybe vételéről.

A modul auth és password módban használható.

70. példa Szeretnénk egy szolgáltatást elérhetetlennek tenni, és szeretnénk, ha a szolgáltatás igénybe vételére irányuló próbálkozások naplóbejegyzést eredményeznének. Használjuk a következő beállítóállományt:

```
1 #  
2 # Mindent tiltunk, minden feljegyzünk.  
3 #  
4 auth      required pam_warn.so  
5 auth      required pam_deny.so  
6 account   required pam_deny.so  
7 password  required pam_warn.so  
8 password  required pam_deny.so  
9 session   required pam_deny.so
```

Figyeljük meg, hogy a naplózás minden visszautasítás előtt található! Figyeljük meg azt is, hogy minden négy szolgáltatásmódot gondosan letiltottunk!

Hagyományos eszközök

A következő néhány Linux-PAM modul segítségével a hagyományos bejelentkezést valósíthatjuk meg.

pam_unix.so A modul a szokásos unixos hitelesítési eljárást valósítja meg a C programkönyvtár hívásával. A alapértelmezett beállítások mellett ez a /etc/passwd és /etc/shadow állományokon alapuló azonosítást eredményezi, habár a C programkönyvtár figyelembe veszi a /etc/nsswitch.allyban található beállításokat.

A modul auth, account, session és password feladat ellátására használható.

Az auth módban a modul a felhasználó azonosítását végzi. Ebben a módban a modul értelmezi a következő paramétereket:

nullok Lehetővé teszi az üres jelszó használatát. Alapértelmezett esetben a modul sikertelen értékkel tér vissza, ha a felhasználó tárolt jelszavának hossza 0.

try_first_pass Ha a modul azt érzékeli, hogy a felhasználótól valamelyik modul már kért jelszót, a kapcsoló hatására előbb azzal próbálja meg az azonosítást elvégezni.

use_first_pass A kapcsoló hatására a modul semmiképpen nem kér jelszót a felhasználótól, mindenkorban az előző modulok által kért jelszót használja.

nodelay Alapértelmezett esetben, ha a jelszó hibásnak bizonyul, a modul körülbelül 1 másodpercig várakozik, hogy ezzel is nehezítse a próbálkozásos jelszófeltörést. A kapcsoló hatására a modul nem késlekedik, azonnal befejezi a futást hibás jelszó esetén.

A password módban a modul a jelszó megváltoztatására használható, ekkor a modul értelmezi a következő kapcsolókat:

md5 A kapcsoló hatására a modul az új jelszó kódolására az MD5 módszert használja, amely lehetővé teszi, hogy a hatásos jelszó 8 karakternél hosszabb legyen.

Bejelentkezéskor a modul automatikusan érzékeli, ha a felhasználó jelszava MD5 kódolással készült, ezért csak a jelszó megváltoztatásakor van ennek a kapcsolónak szerepe.

nullok A kapcsoló hatására a modul lehetővé teszi, hogy a 0 karakter hosszú jelszót megváltoztassuk.

try_first_pass A kapcsoló hatására a modul megvizsgálja, hogy más modulok beolvasták-e már a felhasználó régi jelszavát. Ha igen, a modul nem kéri újra a régi jelszót a jelszó megváltoztatásához.

use_authok A kapcsoló hatására a modul nem kér új jelszót, hanem új jelszóként a más modul által már beolvasott jelszót használja.

nis A kapcsoló hatására a modul megkíséri a jelszót a NIS kiszolgálón is megváltoztatni. Ez akkor lehetséges, ha a NIS a megfelelő módon be van állítva.

Azonosításkor – a jelszó felhasználásakor – nem kell külön kapcsolónak jeleznie a NIS használatát, mert a /etc/nsswitch állományban beállított NIS-használatot a C programkönyvtár automatikusan figyelembe veszi.

remember=n A kapcsoló hatására a modul az utolsó *n* darab jelszót a /etc/security/opasswd állományba menti. Ez alapján a nyilvántartás alapján tilthatjuk meg a régi jelszavak új jelszóként való felhasználását.

Az account mód használata esetén a modul figyelmezteti a felhasználót, ha a jelszavát meg kell változtatni, session módban pedig naplóbejegyzéseket készít a belépéskor és a kilépéskor.

pam_cracklib.so A modul új jelszót kér a felhasználótól és ellenőrzi, hogy az elég bonyolult-e. A modul nem tárolja az új jelszót, azt a beállítóállomány következő moduljainak kell megtennie.

A modul password módban használható, indításkor a következő paraméreket értelmezi:

retry=n A paraméter megadja, hányszor próbálkozzon a modul új jelszó kérésével. Alapértelmezés szerint a modul csak egyszer kér új jelszót; ha az nem megfelelő, többször nem próbálkozik.

use_authok A kapcsoló hatására a modul nem kér új jelszót, hanem az előző modulok által kért jelszót használja.

pam_mail.so Ez a modul megvizsgálja hogy az adott felhasználónak van-e olvasatlan elektronikus levele, és ha van, akkor arról egy üzenetben értesíti.

A modul auth és session módban használható. A modul minden üzemi módban elfogadja a következő kapcsolót:

dir=könyvtárnév A beérkezett elektronikus levelek tárolására használt könyvtár nevének megadása. A felhasználó olvasatlan levelei ebben a könyvtárban találhatók, egy szöveges állományban, amelynek neve megegyezik a felhasználó nevével.

Az alapértelmezett érték /var/spool/mail/.

pam_lastlog.so A modul naplózza a bejelentkezést és a kijelentkezést a /var/log/lastlog állományban, valamint egy üzenetben értesíti a felhasználót arról, hogy mikor jelentkezett be utoljára.

A modul **session** módban használható (a Linux-PAM jelenlegi dokumentációja több hibát is tartalmaz ennek a modulnak a leírásában).

pam_motd.so Ez a modul a /etc/motd állományt jeleníti meg, amelyet általában arra használunk, hogy sikeres bejelentkezés után üdvözöljük a felhasználót.

A modul **session** módban használható, ahol értelmezi a következő kapcsolót:

motd=állománynév A kiírt állomány neve. Alapértelmezés szerint ez a /etc/motd.

pam_securetty.so A modul a rendszergazda számára csak azokról a terminálokról engedélyezi a bejelentkezést, amelyek fel vannak sorolva a /etc/securetty állományban.

A modul sikeres értékkel tér vissza, ha nem a rendszergazda azonosítását kérte az alkalmazás, és a használt terminál fel van sorolva a /etc/securetty állományban, vagy az alkalmazás nem a rendszergazda azonosítását kérte.

A modul **auth** módban használható.

pam_nologin.so Ez a modul a szokásos módon tiltja a felhasználók számára a bejelentkezést, ha a /etc/nologin állomány létezik, azaz ilyen esetben csak a rendszergazda léphet be.

Ha a /etc/nologin állomány létezik, és az alkalmazás nem a rendszergazda azonosítását kérte, a modul kiírja az állomány tartalmát, és sikertelen értékkel tér vissza.

A modul **auth** és **account** feladatok ellátására használható, és értelmezi a következő kapcsolót:

file=állománynév A /etc/nologin állomány helyett a modul a paraméterként megadott állományt használja.

71. példa Igen egyszerű példaként készítsünk egy PAM beállítóállományt, amely a szolgáltatás használatát a C programkönyvtár által biztosított eszközökkel hajta végre.

```

1 # 
2 # Minden a C programkönyvtárra bízunk.
3 #
4 account    required  pam_unix.so
5 auth        required  pam_unix.so nodelay
6 password   required  pam_unix.so
7 session    required  pam_unix.so

```

Amint látható, a hibás jelszó esetén beiktatott késleltetést kikapcsoltuk, ami üzemelő rendszernél nem szerencsés, de a tesztelés során kissé megyorsítja a használatot.

72. példa A következő részlet bemutatja, hogyan ellenőrizhetjük az új jelszót a pam_cracklib.so modul segítségével.

```
1 password required pam_cracklib.so retry=3  
2 password required pam_unix.so use_authtok
```

A részlet beilleszthető például a 8.2.4. példa beállítóállományába.

Az erőforrások korlátozása

A Linux rendszermag képes korlátozásokat bevezetni, amelyek megóvják a rendszert attól, hogy a felhasználói programok túlterheljék. A Linux-PAM segítségével kifinomult módon tudjuk kezelni ezeket a korlátokat. Erre a feladatra a következő modult használhatjuk:

pam_limits.so A modul segítségével korlátozásokat vezethetünk be az erőforrás felhasználásra. A korlátozásokat a rendszermag kötelező jelleggel érvényesíti. A modul session módban használható. A modul értelmezi a következő kapcsolót:

conf=állománynév A korlátozásokat leíró beállítóállomány megadása.
Az alapértelmezett érték /etc/security/limits.conf.

Fontos megemlítenünk, hogy az erőforrások korlátozása a rendszergazdára nem érvényes. Beállíthatunk ugyan korlátokat a rendszergazda által használható erőforrásokra nézve, a rendszermag azonban nem érvényesíti ezeket.

Az erőforráskorlátok beállítására használt /etc/security/limits.conf négy oszlopos szöveges állomány, amelyben a # karakterrel helyezhetünk el megjegyzéseket a szokásos módon. Az állomány egyes oszlopai a következő információkat hordozzák:

1. Az első oszlop azt határozza meg, hogy az adott sor által leírt korlátozás kire vonatkozik. Az első oszlopból a következő formátumokat használhatjuk:

felhasználónév A felhasználó neve, amelyre a korlátozás vonatkozik.

@csoportnév A felhasználói csoport neve, akire a korlátozás vonatkozik.

***** Helyettesítő karakter, amely minden felhasználóra vonatkozik.

2. A második oszlopból a következő kulcsszavakat használhatjuk:

soft A korlátozás az alkalmazások számára nem kötelező, csak figyelmeztetésként szolgál.

hard A korlátozás az alkalmazások számára kötelező, a rendszermag gondoskodik a betartásáról.

- A - jel egyszerre helyettesíti a **soft** és a **hard** kulcsszavakat.

3. A harmadik oszlopban azt adhatjuk meg, hogy a korlátozás mire vonatkozik. Itt a következő kulcsszavak szerepelhetnek:

core Ha programhiba esetén a rendszermag arra kényszerül, hogy a folyamat futását megszakítsa, a munkakönyvtárban egy állományt hoz létre **core** néven. Ebbe az állományba a rendszermag a folyamat memória-területét menti későbbi hibakeresés céljából.

A **core** kulcsszóval a **core** állományok méretére adhatunk felső korlátot kilobájtban.

data Az egy folyamat által használható adatszegmens méretének felső határa kilobájtban.

fsizze A létrehozható legnagyobb állományméret kilobájtban.

memlock A folyamatok az egyes memóriaterületeket memóriába zárolhatják. A memóriába zárolt memóriablokkot a Linux rendszermag nem távolítja el a fizikai memóriából, nem teszi át a virtuális memóriába.

Ezzel a kulcsszóval a memóriába zárolható terület méretének felső korlátját állíthatjuk be kilobájtban.

nofile Az egy időben nyitva tartható állományok számára adhatunk meg felső korlátot ezzel a kulcsszóval.

rss Az egy időben felhasználható fizikai memória mérete kilobájtban.

stack A végrehajtási verem lehetséges legnagyobb mérete kilobájtban.

cpu A felhasználható processzoridő percben. Ha a folyamat eléri a processzoridő-korlátot, a Linux rendszermag megszakítja a futását.

nproc Az egy időben a felhasználó által futtatható folyamatok számának felső határa. Ha a felhasználó által futtatott valamely folyamat új folyamatot próbál meg létrehozni, azt a rendszermag visszavonja.

maxlogins Az egy időben aktív belépések száma felhasználónként.

locks Az egy időben aktív állományzárolások száma.

4. A negyedik oszlopban találhatjuk a korlát értékét. A mértékegységet nem kell beírnunk, csak az értéket.

73. példa A felhasználható erőforrásokat egy bizonyos felhasználói csoport számára szeretnénk korlátozni.

Első lépésként ellenőrizzük, hogy a megfelelő Linux-PAM modul már megtalálható-e a beállítóállományokban. A következő sor például alkalmás az erőforrások korlátozását végző modul indítására:

```
1 session required pam_limits.so
```

Korlátozzuk most az erőforrások felhasználását a /etc/security/limits.conf állományban. Használhatjuk például a következő sorokat:

```
1 @guests      -      core          0
2 @guests      -      nproc         100
3 @guests      -      maxlogins     2
4 @guests      -      rss           1000
```

Az erőforrások korlátozása a következő bejelentkezéstől érvénybe lép minden felhasználó számára, akik a guests felhasználói csoportba tartoznak.

Próbáljuk ki a korlátozást! Tegyük fel, hogy az egyik felhasználó egy hibás programot ír, ami szerencsétlen módon megpróbálja elfogyasztani az összes erőforrást:

```
1 #! /bin/bash
2 echo -n "$1"
3
4 $0 $(( $1+1 ))
```

A program megkíséri elindítani önmagát, és mivel ehhez a szokásához makacsul ragaszkodik, egy idő után minden erőforrást elfogyaszt. Ha most a korlátozások életbe lépése után elindítjuk a programot, a következő kimenetet látjuk:

```
./rossz
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96
./rossz: fork: Erőforrás ideiglenesen elérhetetlen
```

Amint láthatjuk, a program jónéhány példányban elindította magát, de a folyamat egy idő után megszakadt. Amikor a felhasználó által indított folyamatok száma elérte a 100-at, a rendszermag az újabb folyamat indítására hibaüzenettel reagált. A hiba megtörte az önhivatkozó folyamatot, és a program minden példánya befejezte futását.

8.2.5. A saját könyvtár automatikus létrehozása

Néhány esetben szükséges lehet a felhasználó saját könyvtárának létrehozása a belépéskor. Döntethetünk például úgy, hogy a felhasználók nyilvántartását egy központi NIS kiszolgálón tartjuk, a saját könyvtárakat azonban nem. Ez esetben minden felhasználónak a belépéskor létrehozhatjuk a saját könyvtárát, így az adott számítógépen azoknak lesz saját könyvtáruk, akik már használták a gépet.

Belépéskor a pam_mkhomedir.so nevű PAM modullal ellenőrizhetjük és hozhatjuk létre a felhasználó saját könyvtárát. Használhatjuk például a következő bejegyzést a megfelelő helyen:

```
1 session required /lib/security/pam_mkhomedir.so \
2   skel=/etc/skel/ umask=0077
```

A modul első paramétere azt fogja eredményezni, hogy a saját könyvtár létrehozásakor a /etc/skel/ alkönyvtár tartalma is bemásolódik a saját könyvtárba (ez az alapértelmezés is). A második paraméter azt eredményezi, hogy a saját könyvtár és a benne található könyvtárbejegyzések létrehozásához a 0077 maszk kivonódik a jogokból. Ez azt eredményezi, hogy a csoporttulajdonosnak és a többi felhasználónak semmilyen joga nem lesz az adott könyvtárbejegyzésekre.

8.3. A lemezkorlát

A lemezkorlát (*quota*, kvóta) nem tartozik a Linux-PAM felügyelete alá, azaz a lemezkorlátot nem a Linux-PAM segítségével kezeljük, mégis szerencsés itt, a biztonsági házirend témaköreben tárgyalunk. A lemezkorlát az állományrendszer része, felhasználók által foglalható erőforrások korlátozására használható[26].

A felhasználók által felhasználható háttérterületet korlátoznunk kell, hiszen a háttérteraink kapacitása véges. Ha nem korlátoznánk a felhasználható területet, egyes felhasználók a háttérterület egészét feltöltve elvehetnék a többiek elől a háttérterek használatának lehetőségét.

A lemezkorlát kezelése egyszerű. A rendszermag minden írási művelet előtt ellenőrzi, hogy a felhasználó számára elegendő hely áll-e rendelkezésre a háttérterén, és az írást csak akkor hajtja végre, ha a felhasználó még nem használta fel a rendelkezésre álló helyet. Ahhoz azonban, hogy a rendszermag minden írási műveletnél ellenőrizni tudja, hogy elegendő hely áll-e a felhasználó rendelkezésére, folyamatosan nyilván kell tartania a felhasználó által használt lemezterület nagyságát. Nyilvánvaló, hogy nem olvashatja végig a rendszermag minden írás előtt a teljes lemezterületet, hogy megtudja, makkora területet foglalnak a felhasználó tulajdonában lévő állományok összesen.

A lemezkorlát tehát a következő lépések alapján működik:

1. A rendszermag minden írási művelet előtt ellenőrzi a lemezkorlát-nyilvántartásából, hogy a felhasználó számára mekkora lemezterület áll rendelkezésre, és ebből a területből mennyit használt a felhasználó.
2. Ha az alkalmazás által kért új lemezterület nem áll rendelkezésre, a rendszermag a megfelelő hibakódval visszautasítja az alkalmazás kérését.
3. Ha a kért lemezterület rendelkezésre áll, a rendszermag az írási műveletet elvégzi.
4. A rendszermag ezután a felhasználó által elfoglalt lemezterület nagyságát frissíti a lemezkorlát-nyilvántartásban. Ez jelenthet csökkenést és növelést is annak megfelelően, hogy az írási művelet növelte vagy csökkentette a foglalt lemezterület nagyságát.

Amint látjuk, a lemezkorlát kettős nyilvántartás alapján működik. A felhasználók állományai által valójában elfoglalt lemezterület nagysága mellett a lemezkorlát-nyilvántartás is tartalmazza ezt az értéket. Szerencsés esetben persze a nyilvántartott és a valós lemezfelhasználás megegyezik, bizonyos esetekben azonban előfordulhatnak problémák. A lemezkorlát bevezetésekor – amikor a felhasználók már rendelkeznek állományokkal, de a nyilvántartás még nincs használatban – a lemezkorlát-nyilvántartást létre kell hozni, és a használat során is érdemes néha ellenőrizni az összhangot. Mind a kezdeti nyilvántartás, mind pedig az ellenőrzés időigényes folyamat, hiszen ilyenkor a foglalt lemezterületet végig kell olvasni, és a felhasználók által elfoglalt területösszeget ki kell gyűjteni.

Az XFS állományrendszer a lemezkorlátot különleges módon kezeli, ezért az XFS állományrendszer a későbbiekben külön tárgyaljuk.

8.3.1. A lemezkorlát kezdeti beállítása

A lemezkorlát működéséhez el kell végeznünk a lemezkorlát kezdeti beállítását. A kezdeti beállítás során a szükséges állományokat létrehozzuk, a szükséges beállításokat elvégezzük, és a felhasználók lemezhasználatának nyilvántartását elkezítjük.

Vegyük sorra, hogy mi szükséges a lemezkorlát működéséhez:

1. A lemezhasználat nyilvántartása szabályos állományokban történik, amelyek a védett lemezrész gyökérkönyvtárában helyezkednek el. A szabályos állományok neve:

`quota.user` Lemezhasználat-nyilvántartás felhasználónként régebbi lemezkorlátrendszer esetében. Ma már általában nem használjuk.

`quota.group` Lemezhasználat-nyilvántartás felhasználói csoportonként régebbi lemezkorlátrendszer esetében. Ma már általában nem használjuk.

aquota.user Lemezhasználat-nyilvántartás felhasználónként újabb lemezkorlátrendszer esetében.

aquota.group Lemezhasználat-nyilvántartás felhasználói csoportonként újabb lemezkorlátrendszer esetében.

Akkor van szükségünk erre az állományra, ha a lemezhasználatot nem csak felhasználónként, hanem felhasználói csoportonként is korlátozni kívánjuk, de ha már elkészítjük a lemezkorlát-nyilvántartást, érdemes a felhasználói csoportonkénti nyilvántartást is elkészíteni.

Az állományok jogosultságait úgy kell beállítanunk, hogy azokat csak a rendszergazda érhesse el, ő viszont írásra és olvasásra is. Legyen a tulajdonos a rendszergazda, a jogokat pedig állítsuk 0600-ra.

Ha a lemezkorlátot nyilvántartó szabályos állományokat más felhasználók is elérhetik, a lemezkorlát esetleg nem fog működni, azaz a túlságosan megengedő jogok is problémát jelenthetnek.

2. A lemezkorlát használatát a lemezrész beillesztésekor jelezünk kell. A legegyszerűbb, ha az adott lemezrészhez tartozó állományrendszer-táblázat (/etc/fstab) bejegyzésnél szerepelnek az alábbi állományrendszerkapcsolók:

usrquota A felhasználónkénti lemezkorlát bekapcsolása.

grpquota A felhasználói csoportonkénti lemezkorlát bekapcsolása.

Valójában ezeket a kapcsolókat a mount program figyelmen kívül hagyja, és átadja a rendszermagnak, amely gondoskodik a lemezkorlát kezeléséről.

Ha azonban hibásan gépeljük be a kulcsszavak valamelyikét, akkor a mount nem hagyja figyelmen kívül a kulcsszót, megpróbálja értelmezni (ami valószínűleg nem fog sikerülni), és nem illeszti be az állományrendszer-táblázatot. Ha tehát egyetlen betűt is elrontunk a kulcsszavak begépelésekor és újraindítjuk a számítógépet, azt valószínűleg csak szervizüzembe kapcsolással tudjuk helyreállítani (lásd a 5.4. példát).

3. A lemezkorlát-nyilvántartás kezdeti elkészítése a quotacheck program segítségével történhet.

Gondoskodnunk kell arról, hogy a lemezkorlát-nyilvántartás kezdeti kitöltése közben a felhasználók és az időzített feladatok ne írjanak a lemezrészre, hiszen ha a nyilvántartás elkészítése során megváltozik a lemezrész tartalma, a nyilvántartás nem a valós adatokat fogja tartani. (Valójában a nyilvántartás elkészítésének kezdete és a lemezkorlát kezelésének bekapcsolása között eltelt időben kell megvédenünk a lemezrészről.)

Természetesen túl nagy katasztrófa nem történik akkor sem, ha a nyilvántartás elkészítése közben történik némi írás a lemezrészre, csak nem lesz pontos a nyilvántartás, legalábbis a quotacheck következő futtatásáig.

8.5. tábla: quotacheck

Lemezkorlát-nyilvántartás ellenőrzését, javítását és elkészítését végző program.

`quotacheck [kapcsolók] beillesztési_pont`

A program az állományrendszer gyökérkönyvtárában található `quota.user`, `quota.group` vagy `aquota.user`, `aquota.group` lemezkorlát-nyilvántartás kezelésére használható.

Kapcsoló Jelentés

<code>-v</code>	Folyamatosan kiírja, hol tart a munkában.
<code>-u</code>	Csak a felhasználói lemezkorlátok kezelése (alapértelmezett).
<code>-g</code>	Csak a felhasználói csoportok lemezkorlátainak kezelése.
<code>-c</code>	Ne foglalkozzon a meglévő nyilvántartással.
<code>-M</code>	Írásra beillesztett állományrendszer kezelése.

(A rendszerindító héjprogramok valószínűleg gondoskodnak arról, hogy a quotacheck néha lefusson, például akkor, ha az állományrendszt ellenőrizni kell.)

4. A lemezhasználat folyamatos nyilvántartását és a lemezkorlát érvényesítését a quotaon program segítségével be kell kapcsolnunk. A számítógép bekapcsolásakor ezt a rendszerindító héjprogramok valószínűleg elvégzik, ha nem, akkor magunknak kell erről gondoskodnunk.

Ha valamilyen oknál fogva úgy döntünk, hogy a lemezkorlát kezelését ki szeretnénk kapcsolni, használhatjuk a quotaoff programot, vagy a quotaon -f kapcsolóját, ami valójában ugyanazzal a hatással jár, hiszen a quotaon és a quotaoff ugyanaz a program.

A lemezkorlát kezdeti beállítása után az működni kezd, de az írást egyetlen felhasználó számára sem fogja megtiltani, mert alapértelmezés szerint a felhasználók számára nincsenek korlátok beállítva. Ha tehát elvégeztük a kezdeti beállítást, az egyes felhasználók és felhasználói csoportok által foglalt lemezterület lekérdezhető, de a háttértárat csak akkor tudjuk megvédeni, ha minden felhasználó számára korlátot állítunk be.

A következő példa bemutatja, hogyan végezhetjük el a lemezkorlát kezdeti beállítását a gyakorlatban.

74. példa Szeretnénk az általunk üzemeltetett számítógépen bevezetni a lemezkorlátot. A lemezelek kialakításánál az egyszerűség kedvéért a / és a /home/ könyvtárnak készítettük külön lemezeinket. Mivel az elektronikus levelek a /var/spool/mail/ könyvtárba érkeznek, minden lemezrészet szeretnénk lemezkorláttal védeni.

Első lépésként hozzuk létre az aquota.user és az aquota.group állományokat minden lemezrész gyökérkönyvtárában, és állítsuk be a jogokat, hogy csak a rendszergazda érhesse el őket:

```
$ touch /aquota.user /aquota.group
$ chmod 600 /aquota.user /aquota.group
$ touch /home/aquota.user /home/aquota.group
$ chmod 600 /home/aquota.user /home/aquota.group
$
```

Következő lépésként gondoskodunk róla, hogy a lemezrészek beillesztésénél az usrquota és a grpquota állományrendszer kapcsolók be legyenek kapcsolva. Módosítsuk a /etc/fstab állományrendszer-táblázatot, és helyezzük el a megfelelő sorok negyedik oszlopában a kapcsolókat:

1	/dev/hda1	/	ext3	defaults,usrquota,grpquota	1	1
2	/dev/hda3	/home	ext3	defaults,usrquota,grpquota	1	2

Most újrailleszthetjük a lemezrészeket, hogy az újonan elhelyezett kapcsolók érvénybe lépjenek:

```
$ mount -o,remount /
$ mount -o,remount /home
$ mount
/dev/hda1 on / type ext3 (rw,usrquota,grpquota)
/dev/hda3 on /home type ext3 (rw,usrquota,grpquota)
$
```

A következő lépés a lemezkorlát-nyilvántartás frissítése lesz. Mielőtt ezt elvégeznénk, gondoskodnunk kell arról, hogy a frissítés során ne módosuljon a lemezrész tartalma. Ezt elérhetjük például úgy, hogy egyfelhasználós módba kapcsolunk az init 1 parancs kiadásával.

Miután gondoskodtunk róla, hogy a lemezrészek tartalma ne módosuljon a munka során, végezzük el a lemezkorlát-nyilvántartás frissítését. Ez minden lemezrészben minden felhasználóra, minden pedig a csoportokra vonatkozó nyilvántartás frissítését jelenti egy-egy parancs kiadásával.

```
$ quotacheck -c -M /
$ quotacheck -c -g -M /
$ quotacheck -c -M /home
$ quotacheck -c -g -M /home
```

A lemezkorlát-nyilvántartás frissítése hosszú folyamat lehet, hiszen ilyenkor a quotacheck a teljes állományrendszer végigolvassa, hogy kigyűjtse a felhasználók és a csoportok által használt területek nagyságát.

A következő lépés során be kell kapcsolnunk a lemezkorlát figyelését és frissítését a quotaon program segítségével, de lehet, hogy megtakaríthatunk némi

8.6. tábla: quotaon

A lemezkorlát figyelésének és nyilvántartásának bekapcsolása.

quotaon [kapcsolók] beillesztése_pont

A program az előzőleg létrehozott lemezkorlát-nyilvántartás használatát kapcsolja be. A használat a lemezkorlát-nyilvántartás folyamatos frissítését és betartatását vonja maga után.

Kapcsoló Jelentés

-u	Felhasználókra vonatkozó korlát bekapcsolása (alapértelmezett).
-g	Felhasználói csoportokra vonatkozó korlát bekapcsolása.
-p	Nem kapcsolja be a lemezkorlátot, és kiírja az állapotát.
-f	A lemezkorlát kikapcsolása.

munkát azzal, ha egyszerűen újraindítjuk a számítógépet. Az újraindítással ellenőrizhetjük azt is, hogy a rendszerindító héjprogramok a lemezkorlát figyelését megfelelően bekapsolják-e a számítógép bekapsolása után.

A rendszerindítás után bejelentkezve a quotaon program segítségével elenőrizhetjük, hogy a lemezkorlát figyelése és frissítése minden lemezrészben bekapsolódott-e.

```
$ quotaon -p /
group quota on / (/dev/hda1) is on
user quota on / (/dev/hda1) is on
$ quotaon -p /home
group quota on /home (/dev/hda3) is on
user quota on /home (/dev/hda3) is on
$
```

Ha a bemutatott módon sikerült elindítanunk a lemezkorlátot, már csak arról kell gondoskodnunk, hogy az egyes felhasználóknak (és esetleg az egyes csoportoknak) a megfelelő korlátok be legyenek állítva. A bekapsolt lemezkorlát önmagában még nem akadályoz meg senkit a lemez írásában, ahhoz a megfelelő korlátokat be kell állítanunk minden felhasználó számára.

8.3.2. Lemezkorlát rendelése felhasználóhoz

A következő feladatunk a felhasználók és felhasználói csoportok lemezhasználatának korlátozása. minden felhasználónak és ha kívánjuk minden csoportnak egyenként kell beállítanunk a lemezkorlátot. (Egy ügyes BASH program természetesen mérhetetlenül meggyorsíthatja a folyamatot.)

A lemezkorlátok beállítására az edquota szolgál. Ez a program meglehetősen érdekesen működik. Először létrehoz egy ideiglenes állományt, amelyben elhelyezi az aktuális beállításokat, majd elindít egy szövegszerkesztőt, amellyel szerkeszthetjük az állományt. Amikor kilépünk a szövegszerkesztőből, az edquota megvizsgálja, hogy módosítottunk-e az állományon, és ha igen, érvényesíti a módosításokat. A szövegszerkesztőből való kilépés után tehát nem kell tennünk semmit, az edquota automatikusan kezeli a szövegszerkesztővel mentett állományt.

Az edquota mielőtt elindítaná a szövegszerkesztőt, megvizsgálja az \$EDITOR és a \$VISUAL környezeti változót. Ha valamelyik változó létezik, az edquota feltételezi, hogy a kedvenc szövegszerkesztőnk indítását végző parancsot helyeztük el a változóban.



Léteznek más programok is, amelyek hasonlóképpen viselkednek, azaz feltételezik, hogy az \$EDITOR vagy a \$VISUAL környezeti változók valamelyike a felhasználó kedvenc szövegszerkesztőjének nevét tartalmazza.

A lemezkorlát-rendszer segítségével a felhasználók (és felhasználói csoportok) lemezterület-felhasználását korlátozzák, beállíthatjuk, hogy mekkora lemezterületet és hány mutatópontot használhatnak fel az egyes felhasználók. A programok a lemezterületet a következő két kulcsszóval jelölik:

blocks A felhasználható lemezblokkok száma.

A felhasználható blokkok száma meghatározza, hogy a felhasználók által birtokolt állományok és könyvtárak mekkora adatterületet használhatnak fel. A blokkok valós méretét a használt háttérítő típusa befolyásolhatja.

inodes A felhasználható mutatópontok száma.

Mint azt már láttuk, minden könyvtárbejegyzéshez egy mutatópont tartozik. A mutatópontok számát is korlátozzák a lemezkorlát-rendszerrel, hogy a felhasználók ne tölthessék fel a mutatópont területeket üres állományok létrehozásával.

Az adatblokkok számát és a mutatópontok számát erős és gyenge korláttal láthatjuk el:

soft A gyenge korlátokat a felhasználók átléphetik, de csak időlegesen.

Amikor a felhasználó átlépi valamelyik gyenge korlátot, a méltányossági idő (*grace period*) figyelése bekapsolódik. A felhasználók nem használhatnak újabb lemezterületet a méltányossági idő letelte után, csak akkor, ha az erőforrás-felhasználásukat újra a lágy korlát alá csökkentik.

hard Az erős korlátot senki sem lépheti át. Az alkalmazás, amely újabb lemezterületet próbál meg foglalni az erős korlát felett, kudarcra van ítélezve.

A következő példa bemutatja, hogyan állíthatjuk be egy felhasználó számára a lemezkorlátot.

75. példa A lemezkorlát kezdeti beállítása után szeretnénk a lemezfelhasználást korlátozni egy felhasználó számára. Szeretnénk, ha ezt a vim szövegszerkesztővel tehetnének meg, mert az a kedvenc szövegszerkesztő programunk.

Állítsuk be a szövegszerkesztőt jelölő környezeti változót és indítsuk el az edquota programot a következő parancsokkal:

```
$ export EDITOR=vim
$ edquota joe
```

Ekkor a szövegszerkesztő elindul és a következőhöz hasonló sorokat tartalmazza:

1	Disk quotas for user joe (uid 502):
2	Filesystem blocks soft hard inodes soft hard
3	/dev/hda1 0 0 0 1 0 0
4	/dev/hda3 96 0 0 18 0 0

Könnyedén megérthető az állomány tartalma, láthatjuk az egyes állományrendszereket, a gyenge és erős korlátokat, amelyek a felhasználható adatblokkokra és a mutatópontokra vonatkoznak. Láthatjuk azt is, hogy a felhasználó az adott pillanatban mennyi erőforrást használ az egyes lemezrészeken belül.

Az állomány szerkesztése után mentenünk kell azt és egyszerűen ki kell lépnünk a szövegszerkesztőből.

A lemezkorlát beállításának bemutatott módja kissé nehézkes, semmiképpen nem használható, ha sok felhasználó számára szeretnénk a lemezkorlátot beállítani. Ilyen esetben a setquota -p kapcsolóját használhatjuk, ami lehetővé teszi a lemezkorlát beállításainak másolását egyik felhasználótól a másikig. Ezt mutatja be a következő példa.

76. példa Szeretnénk több felhasználó számára is ugyanazokat a lemezkorlát-beállításokat érvényesíteni. Nyilvántartásba vettük egy felhasználót, és beállítottuk számára a lemezkorlátokat, amelyeket szeretnénk a többi felhasználó számára is érvényesíteni.

A következő egyszerű példaprogram átmásolja a lemezkorlát-beállításokat az összes felhasználó számára, akiknek a nevét paraméterként megkapja:

1	#!/bin/bash
2	
3	FILESYSTEMS="/ /home"
4	TEMPLATE=joe
5	

8.7. tábla: setquota

A felhasználók számára érvényes lemezkorlát beállítását végző program.

```
setquota [kapcsolók] felhasználó
setquota -p mintafelhasználó beillesztési_pont
```

A program elindítja az \$EDITOR vagy \$VISUAL környezeti változóban jeolt szövegszerkesztőt a felhasználó vagy felhasználói csoport lemezkorlát-beállításainak módosítására.

Kapcsoló Jelentés

- | | |
|---------|---|
| -u | Felhasználó lemezkorlátjának beállítása (alapértelmezett). |
| -g | Felhasználói csoport lemezkorlátjának beállítása. |
| -p felh | A felhasználó lemezkorlátjainak mintaként való felhasználása. |

```
6 | for USERS in "$@"; do
7 |   echo "Másolás: $USERS"
8 |   for F in $FILESYSTEMS; do
9 |     setquota -p $TEMPLATE $USERS $F
10 |   done
11 | done
```

A program 3. sorában egy változóban elhelyezi a beillesztési pontok listáját, amelyeken a lemezkorlátot érvényesíteni akarjuk, a 4. sorban pedig elhelyezi egy változóban a mintaként használt felhasználói nevét.

A program további soraiban egy ciklussal végigjárjuk az összes paraméterként kapott felhasználói nevet, és egy belső ciklussal az összes állományrendszert.

A példaprogrammal kapcsolatban meg kell jegyeznünk, hogy bizonyos setquota-változatokkal ezt a feladatot sokkal egyszerűbben is meg lehet oldani.

8.3.3. Jelentések készítése

Fontos, hogy a lemezkorlátokat és a felhasználók által felhasznált lemezterület nagyságát folyamatosan figyeljük. Erre a cérla több eszköz is a rendelkezésünkre áll.

A quota program segítségével a felhasználók megjeleníthetik a saját lemezfelhasználásukat és a rájuk vonatkozó lemezkorlátot, a rendszergazda pedig bármely felhasználó felhasználói nevét megadhatja paraméterként, hogy a felhasználók lemezkorlátait lekérdezze.

A következő egyszerű példa bemutatja, hogyan használhatja a rendszergazda a quota programot.

8.8. tábla: quota

A lemezkorlát és a lemezhasználat lekérdezése.

`quota [kapcsolók] [felhasználói_név]`

A felhasználók a program segítségével lekérdezhették a lemezkorlátjukat, a rendszergazda pedig lekérdezhettek bármelyik felhasználó lemezkorlátját.

Kapcsoló Jelentés

`-g` Felhasználói csoportok lemezkorlátjának lekérdezése.

`-q` Csak azokról a lemezrészkről ad információkat, ahol a felhasználó átlépte a gyenge korlátot.

77. példa Szeretnénk tudni, hogy egy bizonyos felhasználó lemezkorlátja milyen értékeket tartalmaz és mekkora területeket használ ez a felhasználó. Használjuk a quota programot!

```
$ quota joe
Disk quotas for user joe (uid 502):
  Filesystem blocks quota limit grace   files quota limit grace
    /dev/hda1      0   100   250          1   100   250
    /dev/hda3    96  1000  2500         18   100   250
$
```

A repquota programmal a rendszergazda egy-egy lemezrészről készíthet összefoglaló jelentést, amelyben szerepel az összes felhasználó, aki az adott lemezrészben adatokat tárol. Ezt a programot mutatja be a következő példa.

78. példa Szeretnénk részletes jelentést kapni az egyik lemezrészről. Használjuk a repquota programot a megfelelő beillesztési pont megadásával!

```
$ repquota -s /home
*** Report for user quotas on device /dev/hda3
Block grace time: 7days; Inode grace time: 7days
                                         Block limits                               File limits
                                         used   soft   hard   grace     used   soft   hard   grace
User
-----
root    --    127M     0     0           509     0     0
pipas   --   13270M    0     0           114k     0     0
joe     --     96   1000   2500          18   100   250
fred    --     32   1000   2500           8   100   250
$
```

8.9. tábla: repquota

A program segítségével jelentést készíthetünk a lemezkorlát állásáról.

`repquota [kapcsolók] beillesztési_pont`

A program segítségével az adott lemezrészben érvényes lemezkorlátokról és a felhasználók által elfoglalt lemezterületek nagyságáról kaphatunk jelentést szöveges táblázat formájában.

Kapcsoló	Jelentés
-v	Azokról a felhasználókról is jelenjen meg információ, akik nem használnak területet.
-s	Az értékek kerekítve, mértékegységgel jelenjenek meg.

8.4. A karakteres munkafelület beállításai

A karakteres munkafelületen a legfontosabb a karakterek alakjának és a billentyűkiosztásnak a beállítása. Különösen fontosak ezek a kérdések akkor, ha magyar nyelvű szövegeket akarunk kezelni, hiszen a magyar nyelv különleges ékezes karakterek használatát is szükségessé teszi. A következő oldalon arról olvashatunk, miképpen állíthatjuk be a GNU/Linux rendszert úgy, hogy magyar nyelvű szövegeket is kezelhessünk.

8.4.1. Karakterkódolási szabványok

A UNIX rendszereken használt karakterkódolási szabványok az ASCII (*American Standard Code for Information Interchange*, amerikai szabványos kód információátvitelre) szabványra épülnek.

Az ASCII szabvány eredetileg 7 bites karakterkódolást használt, amely 128 féle karakter kódolására tette képessé a szabványt. Ez elegendő volt a szabvány eredeti céljainak megvalósítására, de kevés a többnyelvű Európa számára. A szabványt a nyugat-európai nyelvek számára 8 bites kódolássá bővítették, így az alkalmassá vált 256 féle karakter kódolására. A 8 bites ASCII kódolást – megkülönböztetendő az eredeti ASCII szabványtól – kiterjesztett ASCII kódolásként emlegették.

A kiterjesztett ASCII szabvány alkalmas volt például a francia, német szövegek kódolására, de hiányoztak belőle például a lengyel, magyar, orosz nyelvek karakterei. Érdekes módon a magyar nyelvű szövegek kódolására *majdnem* alkalmas a kiterjesztett ASCII, hiszen a francia és német nyelvekből majdnem az összes ékezes karakter összegyűjthető, amelyet a magyar nyelv használ. Az öÖüüÉ-áÁúÚÓ betűk a kiterjesztett ASCII kódolással kódolhatók. Hiányoznak azonban a kiterjesztett ASCII kódolásból az őŐ és úŰ betűk, így a magyar nyelvű szövegek kódolása lehetetlen a kiterjesztett ASCII kódolás felhasználásával.



A kiterjesztett ASCII kódolás tartalmazza a ô Ô és õ Õ, valamint úÜűÜ karaktereket, amelyekkel az öÖűŰ betűket szokás helyettesíteni, annak ellenére, hogy ez a legnagyobb otrombaság a könyvégetés után.

A számítástechnika terjedésével szükségessé vált a kiterjesztett ASCII továbbfejlesztése a kelet-európai országok nyelveinek irányába is. Erre a célra az első elterjedt szabvány az ISO-9958 kódnevet viselő nemzetközi szabvány volt, amely a kiterjesztett ASCII továbbfejlesztésének tekinthető.

Az ISO-8859 az ASCII szabványú kódolást magába foglalja, a kiterjesztett ASCII kódokat azonban továbbfejleszti. A szabvány alváltozatainak megfelelően többféle betűlak rendelhető a kiterjesztett ASCII kódú karakterekhez, azaz a 128-255 kódú karakterek alakja attól függ, hogy az ISO-8859 melyik alváltozata szerint értelmezük a szöveget.

Az ISO-8859-1 alváltozat a nyugat-európai nyelvek kódolására használható, az ISO-8859-2 pedig többek között alkalmas a magyar és a lengyel nyelvű szövegek kódolására. Az ISO-8859-2 hivatalos elnevezése szerint kelet-európai kódolási szabvány, így Magyarország a szabvány készítői szerint Kelet-Európában van.

A GNU/Linux régebbi változatai is támogatták az ISO-8859 szabványt, és – köszönhetően többek között a lengyel GNU/Linux mozgalomnak – teljes mértékben alkalmasak voltak magyar nyelvű szövegek készítésére.

Az írott szövegek számítógépes kálváriája azonban ezzel nem ért véget. Az ISO-8859 szabvány sok nyelvet támogat ugyan, de nem minden nyelvet! A szabvány egyes alváltozatai között különbségek ráadásul sok problémát okoztak, a szöveg betűinek alakjai, a szöveg jelentése ugyanis attól függ, hogy a szabvány melyik alváltozatát használjuk a megjelenítésre. Nyilvánvaló, hogy előbb-utóbb meg kellett jelennie egy egyesített szabványnak, amely bolygónk minden nyelvét támogatja, a szövegek értelmezését egyértelműen előírja.

A nagy egyesített szabvány Unicode néven vált híressé. A Unicode mára jobbára elterjedtnek tekinthető, a könyv írásakor az alkalmazások, rendszerprogramok egyre népesebb családjára támogatja. A számítógépes szakemberek bizakodva tekintenek a Unicode szabványra, amely a több évtizedes huzavona után a végső megoldást jelentheti a betűkódolás kérdéskörében. Ugyanakkor persze sokan el-lenségesen fogadják, hiszen amíg nem terjed el, minden szabvánnyal csak a gond van.

A Unicode szabvány több kódolási módszert fogad el, amelyeket az UTF (*unicode transformation format*, Unicode átalakítási formátum) rövidítéssel és a legkisebb használt kódszavak hosszával jelölünk:

UTF-32 A kódolás minden egyes karakter kódolására 32 bitet használ.

Egy angol nyelvű szöveg – amelynek karakterei alacsony értékű kódot kapnak – például éppen négyeszer annyi helyet foglal UTF-32 kódolás esetén,

8.10. tábla: loadkeys

A rendszermag által használt billentyűkiosztási táblázat betöltése.

`loadkeys [kapcsolók] [állománynév]`

Kapcsoló Jelentés

-d Az alapértelmezett állomány betöltése.

mint egy ASCII kódolású állományban. Talán nem véletlen, hogy ez az ábrázolási forma nem terjedt el.

UTF-16 A kódolás az alacsonyabb kódú karakterek tárolására 16 bitet, a magasabb kódú karakterek kódolására 32 bitet használ.

Ez a módszer sikeresebb, mint az UTF-32, de nem ez a Unicode szabványú kódok legelterjedtebb ábrázolása.

UTF-8 A kódolás az alacsony kódú karakterek kódolására 8 bitet használ, a magasabb kódú karakterek kódolására pedig többet. A leghosszabb egy karaktert jelző kód 6 bájtos.

A Unicode által használt 8 bites kódok megegyeznek az ASCII kódolású karakterek kódjaival, így egy angol nyelvű ASCII kódolású szöveg egyben UTF-8 kódolású is. minden bizonnal ez az oka annak, hogy a Unicode ábrázolások közül az UTF-8 a legelterjedtebb.

A GNU/Linux terjesztések mai modern változatainak sok rendszerprogramja és alkalmazása támogatja az UTF-8 kódolást, de még nem mondhatjuk, hogy a rendszer minden összetevője Unicode-támogatású volna. Mindenesetre úgy tűnik, a jövő az UTF-8 kódolásé.

8.4.2. A billentyűzet beállítása

A karakteres munkafelületen használt billentyűkiosztást, amely meghatározza, hogy a lenyomott billentyű hatására milyen karaktert kapjon a futó alkalmazás, a Linux rendszermag kezeli. A rendszermagból a billentyűkiosztás táblázatát kiolvashatjuk és oda betölthetjük, így megváltoztatva a billentyűzet viselkedését.

A karakteres felületen érvényes billentyűkiosztást a `dumpkeys` program kiolvassa a rendszermagból, és szöveges formában kiírja a szabványos kimenetére. A `loadkeys` program a paraméterként kapott állományból a billentyűkiosztást beolvassa, és betölti a rendszermagba. A `dumpkeys` és a `loadkeys` ugyanazt a formátumot használja, így a `dumpkeys` segítségével kiolvasott billentyűkiosztást a `loadkeys` programmal beolvashatjuk.

Az egyes GNU/Linux terjesztések előre elkészített billentyűkiosztással vannak felszerelve a legtöbb nyelvhez és a legtöbb billentyűzettipushoz. A loadkeys program a telepített billentyűkiosztások helyét ismeri, így ha a paraméterként megadott állomány nem található, a telepített billentyűkiosztások között keresi a betöltendő állományt.

79. példa Töltsük be a hu101 billentyűkiosztást a loadkeys program segítségével!

```
$ loadkeys hu101
Loading /lib/kbd/keymaps/i386/qwerty/hu101.map.gz
$
```

Ha kipróbáljuk a billentyűzetet, azt tapasztaljuk, hogy a magyar ékezes betűk elérhetők, az „y” betű a billentyűzet közepén a felső sorban található (qwerty kiosztás).

Próbáljuk meg betölteni a hu billentyűkiosztást!

```
$ loadkeys hu
Loading /lib/kbd/keymaps/i386/qwertz/hu.map.gz
$
```

Ha kipróbáljuk a billentyűzetet, azt tapasztaljuk, hogy a billentyűzet közepén a felső sorban most a „z” betű érhető el (qwertz kiosztás).

(Ha az itt bemutatott parancsok hatására a billentyűzetünk furcsa karaktereket produkál az ékezes betű helyett, valószínűleg Unicode üzemmódot használunk, amelynek a beállítására még visszatérünk.)

Nem elegendőan azonban a billentyűkiosztást beállítani, arról is gondoskodnunk kell, hogy a képernyőn a megfelelő alakú karakterek jelenjenek meg.

Karakteres képernyőn a megjelenő betűk alakja a videókártya karaktertáblájából származik, azaz a megjelenő karakterek formáját a videóramkörökbe épített memóriában található betükészlet határozza meg. A modern videókártyák (például a VGA szabvány szerinti kártyák) betükészlete az új betükészlet feltöltésével megváltoztatható. Ha tehát feltölthjük a megfelelő betükészletet, a megfelelő formájú karakterek jelennek meg az egyes billentyűk lenyomásakor.

A karakteres munkafelület által használt betükészlet megváltoztatására, a betükészlet feltöltésére a consolechars program használható. A programnak paraméterként a betöltendő betükészlet nevét kell megadnunk.

80. példa Töltsük be a setfont program segítségével a drdos8x16 betükészletet!

```
$ setfont drdos8x16
$
```

A program futtatása után a hatás azonnal megfigyelhető, a betűk alakja megváltozik.

8.11. tábla: dumpkeys

A billentyűkiosztást írja ki a szabványos kimenetére.

`dumpkeys [kapcsolók]`

A program a karakteres felületen használatos billentyűkiosztást kiolvassa a rendszermagból, és a szabványos kimenetre szöveges formában kiírja.

Kapcsoló	Jelentés
-l	Részletes információt ír ki, amelyben szerepelnek a program által használt karakternevek is.
-f	A program a billentyűkiosztási táblázatot nem egy-szerűsíti, mielőtt kiírná.
--charset=név	A karakterkódolás megadása iso-8859-X formában.

A Unicode kódolás használata karakteres felületen

A modern GNU/Linux terjesztések általában támogatják a Unicode kódolás használatát a karakteres munkafelületen, sőt sokszor ez az üzemmód az alapértelmezett. A különféle Unicode kódolások közül az UTF-8 kódolás a legelterjedtebb, a legtöbb alkalmazás és programkönyvtár ezt támogatja.

A karakteres munkafelületen bekapcsolhatjuk a Unicode kódolást – billentyűkiosztást és betűkészletet – a `unicode_start` program segítségével, az alkalmazások számára pedig beállíthatjuk a Unicode kódolás használatát az `$LCA_ALL` vagy a `$LANG` környezeti változó beállításával. Ezt mutatja be a következő példa.

81. példa Vizsgáljuk meg a `$LANG` környezeti változó értékét, hogy megállapítsuk, milyen kódolást kell az alkalmazásoknak használniuk!

```
$ echo $LANG
hu_HU.UTF-8
$
```

Amint látjuk, a változó az alkalmazások számára a magyar nyelvet és az UTF-8 kódolást írja elő.

Állítsuk be a karakteres munkafelületen a Unicode billentyűkiosztást és a Unicode betűkészletet!

```
$ unicode_start
$
```

A Unicode üzemmódot a `unicode_stop` program segítségével kapcsolhatjuk ki.

Ha figyelmesen megvizsgáljuk, hogy mit csinál a `unicode_start` program, akkor azt tapasztaljuk, hogy bekapcsolja ugyan a Unicode támogatást, de nem változtatja meg a billentyűzet kiosztását. Ha magyar billentyűkiosztást használtunk a program futtatása előtt, a program futása után azt találjuk üzemben.

A Unicode-támogatás bekapcsolása után azonban a `loadkeys` programot a `--unicode` kapcsolóval kell használnunk, és csak olyan betűkészletet használhatunk, ami a Unicode szabvány szerint készült.

A legtöbb GNU/Linux terjesztés többféle billentyűkiosztást és betűkészletet támogat a karakteres munkafelületen. Általában valamelyik beállítóállományban helyezhetjük el, hogy milyen billentyűzetet és betűkészletet akarunk használni, a rendszer indításakor pedig automatikusan beállítódik a billentyűzet és a betűkészlet.



Red Hat alapú rendszereken a `/etc/sysconfig/keyboard` nevű állományban van elhelyezve a billentyűkiosztás neve a `$KEYTABLE` változóban. Az alapértelmezett billentyűkiosztást úgy állíthatjuk be, hogy átírjuk az itt található értéket.

Red Hat rendszereken az alapértelmezett betűkészlet, valamint az alapértelmezett nyelv és kódolás (például Unicode) a `/etc/sysconfig/i18n` állományban található.



Debian alapú rendszereken a `/etc/init.d/keymaps-lct.sh` program a `/etc/console-tools/` alkönyvtárban található `default.kmap` vagy `default.kmap.gz` állományból tölti be a billentyűkiosztást a rendszer indításakor. Az alapértelmezés szerinti billentyűkiosztást úgy állíthatjuk át, hogy a `default.kmap` (vagy `default.kmap.gz`) állományt felülírjuk.

Debian alapú rendszereken a `/usr/share/consolefonts/` alkönyvtárban találhatók a karakteres felületen használt betűk. Itt a `/etc/console-tools/config` állományban található `$SCREEN_FONT` változó adja meg, hogy mi legyen a karakteres felület alapértelmezés szerinti betűtípusa.

Saját billentyűkiosztás létrehozása

A legelterjedtebb számítógépek billentyűzetének működése alapjában véve igen egyszerű. Amikor a billentyűzeten lenyomunk egy gombot, akkor a billentyűzet egy jelet küld a számítógép felé, amelyben jelzi, hogy egy billentyűt nyomtunk le, és elküldi ennek a billentyűnek a letapogatási kódját (*scan code*). Ha elég sokáig tartjuk lenyomva a billentyűt, a billentyűzet ismételgetni kezdi a lenyomást jelző jelet és a letapogatási kódot.

Amikor a billentyű felengedjük, a billentyűzet újra elküldi a billentyű letapogatási kódját és egy üzenetet, amelyben jelzi, hogy felengedtük a billentyűt.

Mindezeket a jeleket a rendszer Linux rendszermag fogadja, és a megfelelő betűkóddá alakítja. A rendszermag gondoskodik arról is, hogy a megfelelő alkalmazás kapja meg a lenyomott billentyű által jelzett betűt.

A letapogatási kód átalakítása betűkóddá igen fontos feladat, hiszen ez határozza meg, hogy az adott billentyű lenyomása milyen betűt jelent, milyen nyelvű billentyűzetet használunk.

Ha meg szeretnénk változtatni a billentyűkiosztásban található valamelyik billentyű jelentését, a következő lépéseket kell elvégeznünk:

1. Első lépésként ki kell olvasnunk a rendszermagból a táblázatot, amely a letapogatási kódokat betűkódokká alakítja, és egy állományban kell tárolnunk azt.
2. Meg kell tudnunk, hogy mi a letapogatási kódja annak a billentyűnek, amelyet módosítani szeretnénk.
3. Módosítanunk kell az állományban az adott letapogatási kód jelentését egy szövegszerkesztő programmal.
4. Végül vissza kell töltenünk a rendszermagba a billentyűkiosztás táblázatát.

A Linux rendszermagból a billentyűzetkiosztást a `dumpkeys` program segítségével olvashatjuk ki. A program a szabványos kimenetre írja a táblázatot szöveges formában.

A `dumpkeys` programnak sok kapcsolója van, számunkra azonban a `--charset` a legfontosabb. Ezzel a kapcsolóval állíthatjuk be, hogy a program a táblázatot milyen szabvány szerint értelmezze. A kapcsoló után megadhatjuk a használt kódolási szabványt `iso-8859-X` formában. (Az X helyén az 1–9 számok állhatnak.)



A `--charset` valójában csak azt állítja be, hogy a rendszermagban elhelyezett kódokat a `dumpkeys` milyen módon fordítsa le szöveges karakternévvé. Ugyanaz a kód ugyanis más-más jelentéssel bír az ISO-8859 szabvány egyes alváltozataiban.

A következő példa a magyar nyelv különleges ékezeteit is magába foglaló szabvány (`iso-8859-2`) szerint olvassa ki a táblázatot:

82. példa Olvassuk ki a rendszermagban található billentyűkiosztást az ISO-8859-2 szabvány szerint, és vizsgáljuk meg az eredményül kapott szöveges táblázat első néhány sorát!

A billentyűkiosztást kiolvashatjuk a `dumpkeys` program segítségével:

```
$ dumpkeys --charset=iso-8859-2 -f >bill.txt
$
```

A `-f` kapcsoló hatására a program a teljes táblázatot kiírta. Ha ezt a kapcsolót nem adjuk meg, az állomány kisebb lesz, egyszerűbb szabályokat tartalmaz, de a megértéséhez ismernünk kell az egyszerűsítési szabályokat, amelyeket a program a táblázat méretének csökkentésére használ.

Az eredményül kapott állomány első néhány sora a következő formájú (az állomány sorait utólag tördeltük rövidebbre a \ karakter segítségével):

```

1  charset "iso-8859-2"
2  keymaps 0-2,4-6,8,10,12
3  keycode 1 = Escape      Escape      Escape \
4          Escape      Escape      Escape \
5          Meta_Escape  Meta_Escape  Meta_Escape
6  keycode 2 = one        apostrophe  asciitilde \
7          VoidSymbol   VoidSymbol  VoidSymbol \
8          Meta_one     Meta_asciitilde  VoidSymbol

```

Amint látjuk, az elkeszült állományban az első sorban olvasható a használt kódolási szabvány neve.

A billentyűkiosztás szöveges táblázatában a következő elemeket találhatjuk meg:

- A `charset` kulcsszó után megtaláljuk a használt betűkódolási szabvány nevét, ami igen fontos a billentyűkiosztás értelmezése szempontjából.
- A `keymaps` kulcsszó után megtaláljuk, hogy a táblázat az eredeti billentyűkiosztás mely oszlopait tartalmazza. Azok az oszlopok, amelyek nem szerepelnek a `keymaps` kulcsszó után, az egyszerűség érdekében nem szerepelnek a táblázatban.

A `keymaps` kulcsszó után vesszővel elválasztva számokat és tartományokat sorolhatunk fel. A tartományokat a „” karakterrel jelöljük.

- A `keycode` kulcsszó után egy-egy billentyű jelentését adhatjuk meg.
A `keycode` kulcsszó után egy szám következik, amely az adott sorban leírt billentyű letapogatási kódja. Az egyenlőségjel után a billentyű által létrehozott betűket soroljuk fel a nevükkel. Tudnunk kell, hogy egy billentyű sokféle betűt hozhat létre attól függően, hogy milyen módosítóbillentyűt nyomunk le a billentyűvel egy időben.

Minden módosítóbillentyű – olyan billentyű, amelyet a Linux rendszermag más billentyűk jelentésének módosítására használ – rendelkezik egy értékkel (8.12). Az egyszerre lenyomott módosítóbillentyűk értékét össze kell adnunk, és így kapjuk meg, hogy a billentyűkiosztás táblázatának hányas oszlopában található a módosított billentyű jelentése.

A billentyűkiosztás táblázatának megértését segíti a következő példa.

83. példa A 8.4.2. példában található billentyűkiosztás-részlet alapján állapítuk meg, hogy az **[Alt]+[AltGr]** módosítóbillentyűk egyszerre történő lenyomása miképpen módosítja a billentyűk hatását! A billentyűkiosztás részlete a következő:

Módosító	Érték
Shift	1
AltGr	2
Ctrl	4
Alt	8
Bal Shift	16
Jobb Shift	32
Bal Ctrl	64
Jobb Ctrl	128

8.12. táblázat. Módosítóbillentyűk és értékeik

```

1 charset "iso-8859-2"
2 keymaps 0-2,4-6,8,10,12
3 keycode 1 = Escape      Escape      Escape \
4           Escape      Escape      Escape \
5           Meta_Escape  Meta_Escape  Meta_Escape
6 keycode 2 = one         apostrophe apostrophe \
7           VoidSymbol   VoidSymbol  VoidSymbol \
8           Meta_one     Meta_asciitilde  VoidSymbol

```

Ha az **[AltGr]** és az **[Alt]** módosítókat egyszerre nyomjuk le, a 8.12. táblázat szerint $8 + 2 = 10$ lesz a módosítóbillentyűk által képviselt összeg.

A billentyűkiosztásban a keymaps kulcsszó után a nyolcadik helyen található a 10-es számjegy, vagyis az állomány nyolcadik oszlopában található meg az **[Alt] + [AltGr]** módosító által módosított érték.

A 2-es kódú billentyű például az **[Alt] + [AltGr]** hatására a **Meta_asciitilde** betűt jelenti.

A billentyűzet letapogatási kódjainak megismerésére igen praktikus eszköz a showkey program, amely folyamatosan figyeli a billentyűzetet, és ha lenyomunk vagy felengedünk egy billentyűt, annak kódját a képernyőre írja.



A showkey minden billentyűnek kiírja az értékét, tehát nincs olyan billentyűkombináció vagy parancs, amellyel kiléphetünk a programból.

A showkeys programból kilépni – igen elmés módon – úgy lehet, hogy kis ideig nem nyomunk le egyetlen billentyűt sem!

8.4.3. Az egér használata karakteres felületen

A karakteres felületen való egérhasználathoz a gpm szolgáltatást kell elindítanunk. A gpm rendszer használatáról bővebben olvashatunk a gpm kézikönyvében.

A tapasztalat azt mutatja, hogy a gpm szolgáltatás a legtöbb esetben minden különösebb fennakadás nélkül működik, néha azonban zavart okoz a grafikus felület egérkezelésében. Ha a grafikus felületen az egérmutató a mozgatása hatására furcsán ugrál a képernyő egyik pontjáról a másikra, érdemes megpróbálnunk, hogy a hiba akkor is jelentkezik-e, ha a gpm rendszert leállítjuk.

9. fejezet

Programcsomagok

A rendszergazda egyik legfontosabb és legbonyolultabb feladata a programcsomagok kezelése, ami általában programcsomagok telepítését és frissítését teszi szükségessé. Ez a feladatkör komoly felkészültséget és tapasztalatot igényel, és némi programozási ismeret is jól jön az elvégzéséhez. Kissé tréfásan azt is mondhatnánk, hogy a jó rendszergazda felkutatja és letölti programot, kijavítja benne a hibákat és telepíti a felhasználók számára, majd munkája végeztével megírja a programozónak, hogy mit is rontott el igazából.

Ebben a fejezetben a programok, programcsomagok fordításáról és telepítéséről lesz szó. Ezt a bonyolult kérdéskört megpróbáljuk alaposan, az elméleti ismeretek és a gyakorlati tapasztalatok hadrendbe állításával tárgyalni. Valószínűleg sokaknak kissé túlzottnak tűnik majd az a lendület, amivel a kérdés tárgyalásának neki kezdünk, de aki már telepített az Internetről letöltött programokat GNU/Linux rendszerére, valószínűleg semmilyen segítséget nem sokall a témaban.

9.1. Programok fordítása

A forrásprogram – más néven forrás – (*source*) a számítógépprogramnak a programozó által készített változata, ami valamelyen programozási nyelven megfogalmazott leírás. A forrásprogram leírja, hogy egy bizonyos feladattípus hogyan kell megoldania a számítógépnek, általában úgy, hogy megadja a feladat megoldásának menetét, más néven algoritmusát.

A számítógépek azonban általában nem a forrásprogramot hajtják végre, hanem annak egy módosított változatát, amelyet gépi kódú programnak vagy bináris programnak szokás nevezni. A forrásprogram átalakítása bináris programmá pusztán a hatékonyság növelésének érdekében történik, azaz kizártlag azért szokás lefordítani a forrásprogramot bináris programmá, hogy a számítógép hatékonyabban tudja végrehajtani a benne található utasításokat. (Jól alátámasztja ezt a tényt az, hogy léteznek olyan programok, amelyeket nem kell lefordítanunk

a futtatás előtt, de ezek végrehajtása sokszor nem elég hatékony az adott feladat megoldására.)

A forrásprogram és a bináris program kérdéskörében egy igen fontos tényre kell felhívunk a figyelmet: *a forrásprogram könnyedén lefordítható bináris programra (hogy futassuk), a bináris program azonban nem fordítható vissza forrásprogrammá (hogy elolvassuk)*. E tény a GNU/Linux történetének és filozófiájának minden bizonnal legfontosabb alapeleme, de nyugodtan mondhatjuk, hogy a napjainkban a számítástechnikában uralkodó viszonyok kulcsa is.

A GNU/Linux alapfilozófiája a szabad forrás (*open source*) elve, ami kimondja, hogy minden felhasználó szabadon hozzáférhet a forrásprogramhoz, azt elolvashatja, és tetszése szerint módosíthatja. Ez a hozzállás természetesen számtalan számítástechnikai céget irritál, főképpen azokat, amelyek programjaikat az olvashatatlan bináris változat terjesztésével titokzatosságba kívánják burkolni. Természetesen innen származik a GNU/Linux használat pikáns forradalmi felhangja.

Sokan kissé értetlenül állnak a szabad forrás elvével szemben, és azt gondolják, hogy a szabad forrás csak arra való, hogy a program lefordításának feladatát a programozó rájuk hárítsa. Ez természetesen nem így van, hiszen a GNU/Linux programok nem csak forrásprogramként, hanem lefordított bináris programként is elérhetők. A legtöbb program esetében választhatjuk az egyszerű utat, és beszerezhetjük a programok lefordított, bináris változatát, de a tapasztalat azt mutatja, hogy az igazán hozzáértő rendszergazda sok esetben a forrásprogramot választja.

A programcsomagok telepítéséről szóló fejezet elején a forrásprogramok fordításának és telepítésének összetett folyamatát ismertetjük.

9.1.1. A GNU C fordító

A legtöbb UNIX alkalmazás és segédprogram C programozási nyelven íródott. A C programozási nyelv és a UNIX története erősen összefonódott; viszonyuk leginkább ahhoz a szoros kapcsolathoz hasonlítható, amelyet egy nemzet és a nemzet által beszélt nyelv között figyelhetünk meg.

A GNU/Linux rendszereken általában a GNU C fordítót használjuk, amely szabványos C nyelvű programokat képes lefordítani szabványos bináris programokká. Természetesen használhatnánk más C fordítót is, de a GNU C fordító szabad forrású...

A C fordító a szabvány szerint a `cc` parancsal indítható, a GNU C fordító azonban indítható `gcc` néven is. Ha beírjuk a `gcc` parancsot a parancssorba, láthatjuk, hogy telepítve van-e a GNU C fordító az adott számítógépen:

```
$ gcc  
gcc: no input files  
$
```

A GNU C fordító hibaüzenete – ami szerint nem adtunk meg bemenőállományt a parancssorban – megmutatja, hogy a fordító telepítve van. Ez természetesen

nem jelenti azt, hogy a programunk lefordításához minden adott, de jelzi, hogy van némi esélyünk egy C nyelvű program lefordítására.

Egyszerű C program fordítása

Nem célunk az olvasót a C programozási nyelv ismertetésével terhelni, fontos azonban, hogy annak néhány alapelemét bemutassuk. Vizsgáljuk meg a következő egyszerű példaprogramot:

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]){
4     printf("OK\n");
5 }
```

A program egy üzenetet ír a szabványos kimenetre, majd kilép. Az üzenetet kiíró utasítás (valójában függvény) a program 4. sorában látható, neve `printf()`. A program többi sora tulajdonképpen csak körítés, a lényegi munkát a `printf()` végzi.

A `printf()` függvény azonban bármilyen furcsa, nem a C programozási nyelv része! A `printf()` függvény egy programrészlet neve, amelyről a C fordító számára az `stdio.h` állományban találhatók információk.

A példaprogram első sorában található egy hivatkozás az `stdio.h` állományra, amelyet a C fordító betölt a fordítás elején, így a 4. sorban nem áll értetlenül a `printf()` név előtt.

Az egyszerű példaprogramunk lefordításához a következő elemekre van szükség:

- A C forrásprogram. A bemutatott példaprogram kicsi, egyszerű és többé-kevésbé hibamentes, de a gyakorlat nem mindig ilyen egyszerű.
- A C fordító. A GNU C fordító egy viszonylag nagy méretű alkalmazáscsomag, amely bármely GNU/Linux rendszerre telepíthető.
- A programkönyvtár, azaz a `printf()` néven közismert programrészlet. A legtöbb GNU/Linux programot C nyelven írják, így ez a programrészlet biztosan telepítve van a számítógépünkre, hiszen nyilvánvaló, hogy más programok is használják.
- Az `stdio.h` fejállomány. Ez az állomány – többek között – a `printf()` használatát magyarázza el a C fordítónak, és csak bizonyos eszközöket használó programok fordításához szükséges.

Ha minden szükséges elem rendelkezésre áll, a program lefordítható. Helyezük el a programsorokat egy `.c` végződésű névvel ellátott állományban (például `main.c`), és futtassuk a GNU C fordítót:

```
$ gcc main.c -o proba
$
```

Amint látjuk, a C fordítónak a lefordítandó állomány nevét adtuk át, valamint a `gcc -o` kapcsolóját használtuk, amely után megadható az elkészítendő bináris állomány neve. Ha a fordítás során nem kapunk hibaüzenetet, a lefordított program a szokásos módon futtatható:

```
$ ./proba
OK
$
```

Amint látható, sok elemnek kell ugyan megfelelően működnie ahhoz, hogy a C program leforduljon, maga a fordítás azonban egyszerű. Legalábbis ilyen egyszerű program esetében.



A *GNU C fordító kapcsolói és a használat leírása* elérhető a `man gcc parancsal`, de a leírás megtekintését csak erős idegenzetűek számára javasolhatjuk, és azoknak is csak akkor, ha kimondottan ráérnek (az adott hónapban). A *gcc* kézikönyve bonyolultságának köszönhetően leginkább tapasztalatlan kollégák megfélmelegítésére használható.

Vizsgálódásainkat folytassuk egy kissé bonyolultabb példaprogrammal, amely egy szám szinuszát írja ki a szabványos kimenetre. A példaprogram legyen a következő:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(int argc, char *argv[]){
5     printf("%f\n", sin(1.2));
6 }
```

A program a 2. sorban betölti a `math.h` fejállományt, amelyben a `sin()` függvényre vonatkozó információk vannak. A fejállománynak köszönhetően a programban az 5. sorban használhatjuk a függvényt az 1,2 szinuszának kiszámítására.

A program látszólag egyszerű, de ha megpróbáljuk lefordítani a már ismertetett módszerrel, kudarcot vallunk:

```
$ gcc main.c -o proba
/tmp/ccQ08hPo.o(.text+0x23): In function `main':
: undefined reference to `sin'
collect2: ld returned 1 exit status
$
```

A hibaüzenet szerint a fordítás során szükség lett volna a `sin()` függvényt megvalósító programrészletre, de az nem volt elérhető. A `printf()` függvényt megvalósító utasítások elérhetők voltak, a `sin()` függvényt megvalósító utasítások azonban hiányoztak.

A `gcc -l` kapcsolója segítségével előírhatjuk a C fordítónak, hogy a fordítás során külső programkönyvtárat is használjon. A sikeres fordítást és a program futtatását mutatják be a következő sorok:

```
$ gcc -lm main.c -o proba
$ ./proba
0.932039
$
```

A példában a `-l` kapcsoló segítségével előírtuk a matematikai programkönyvtár használatát. A matematikai programkönyvtár, amely a `sin()` függvényt is tartalmazza, az egybetűs `m` néven érhető el. A `-lm` kapcsoló segítségével tehát le tudtuk fordítani a programot.

Összetett C programok

A gyakorlatban a C programokat általában több állományra darabolva készítik el. Az egyes állományok külön fordítási egységet alkotnak, azaz a fordítás során állományonként dolgozzuk fel a forrásprogramot, és csak az egyes állományok lefordítása után készítjük el végtermékként a bináris programot.

A programot természetesen alapos okkal készítjük el több állományra darabolva. A külön fordítási egységek használatának több előnye is van:

- A fordítóprogram a fordítás során különféle táblázatokat készít, amelyeket a memoriában tárol.

Ha nagyméretű programokat egyszerre, egészben fordítanánk le, a teljes program táblázatait egy időben kellene a memoriában tárolni, ami túlterhelné a számítógépet, esetleg meghiúsítaná a fordítást.

A fordítási egységekre való bontás egyik célja tehát az, hogy a fordító munkáját megkönnyítsük.

- A programfejlesztés során általában kisebb változtatásokat hajtunk végre a programon. Ha a programot állományokra daraboljuk, nyilvánvalóan csak azokat az állományokat kell újrafordítani, amelyeket módosítottunk.

A módosítások újrafordításakor a megtakarítás hatalmas lehet, hiszen előfordulhat, hogy egy több tízezer, esetleg több millió soros programnak csak néhány száz soros elemét kell újrafordítani.

- A programfejlesztést általában csoportok végezik. Előfordulhat, hogy egy programot több száz programozó készít a feladatok felosztásával.

Szerencsés, ha a feladatot kezelhető részekre osztjuk, így minden programozó minden feladata külön állományba kerül, hogy áttekinthető legyen a program.

Vizsgáljuk meg egy példán keresztül, hogy miképpen készülnek a több fordítási egységből álló C programok, és miképpen fordíthatjuk le őket.

A következő program az előző példaprogramokkal összevetve könnyedén megérthető:

```

1 #include "proba.h"
2 #include <stdio.h>
3
4 int main(int argc, char *argv[]){
5     printf("%d\n", osszeg(1, 2));
6 }
```

A program az 5. sorban az `osszeg()` nevű függvényt hívja, és kiírja a függvény által visszaadott értéket. A különbség csak annyi az előző példaprogramokkal összevetve, hogy az `osszeg()` nevű függvényt nem a számítógépre telepítve kell megkeresnünk, hanem magunknak kell megírnunk.

Az `osszeg()` függvény megvalósítását a `proba.c` állományban találjuk:

```

1 #include "proba.h"
2
3 int osszeg(int a, int b){
4     return a + b;
5 }
```

A programunkat tehát két állományra bontva készítettük el. A `main()` függvény a `main.c` állományban, az `osszeg()` függvény pedig a `proba.c` állományban található. A `main()` függvény hívja az `osszeg()` nevű függvényt, azaz a `main()` függvény készítője felhasználja az `osszeg()` függvény szerzőjének munkáját.

A két állomány között a kapcsolatot a `proba.h` állomány teremti meg. Ebben az állományban írjuk le a `proba.c` állományban megvalósított függvényeket:

```

1 #ifndef PROBA_H
2 #define PROBA_H
3
4 int osszeg(int a, int b);
5
6 #endif
```

A `proba.h` fejállományt a `main.c` állományban, annak 1. sorában töltjük be a fordítás során. Figyeljük meg, hogy a `proba.h` betöltésekor nem <> jeleket, hanem "" jeleket használunk. A kettős idézőjelek hatására a C fordító a fejállományt a munkakönyvtárban keresi.

9.1. tábla: gcc

A GNU C fordító elsősorban C és C++ programok fordítására használható.

gcc [kapcsolók] állománynév

A GNU C fordító kifinomult, összetett eszköz, amely több száz kapcsolót értelmez.

Kapcsoló

Jelentés

-o fájl	Kimenő állomány nevének megadása.
-c	Csak a fordítást végzi el, a szerkesztést nem.
-lprg	Programkönyvtár megadása.
-Ikönyvtár	Fejállományokat tartalmazó könyvtár nevének megadása.

Ha ezt a három állományban elhelyezett C nyelvű programot le akarjuk fordítani, a következő parancsokat kell kiadnunk:

```
$ gcc -c main.c
$ gcc -c proba.c
$ gcc main.o proba.o -o proba
$ ./proba
3
$
```

A gcc -c kapcsolója arra ad utasítást a fordítónak, hogy a kész bináris programot ne készítse el, csak fordítsa le a programot. A C fordító a kapcsoló hatására úgynevezett tárgykódú programot (*object code*) hoz létre, amelynek neve alapértelmezés szerint a .o karakterekkel végződik.

A példa első két sora tehát létrehozza a main.c forrásállományból a main.o tárgykódú állományt, majd a proba.c állományból a proba.o állományt. A harmadik parancs a két tárgykódú programállományból készíti el a bináris programállományt, amely már programként futtatható.

A bináris program tárgykódú állományokból való elkészítését szerkesztésnek (*linking*) nevezzük. Ez a munkafázis a tárgykódú programokban megkeresi a hivatkozásokat, egymáshoz rendeli a különböző tárgykódú állományokban található programdarabokat, és összeszerkeszti a kész programot.

Valójában a szerkesztés munkafázisát nem is a C fordító végzi el, hanem az ld, a szerkesztő. Az egyszerűbb használat érdekében a C fordító vállalja, hogy megírja a szerkesztőt, ha paraméterként tárgykódú programállományokat kap. Így a felhasználóknak nem kell megtanulniuk az ld használatát.

9.1.2. Projektek kezelése

Ha figyelmesen áttanulmányozzuk az előző szakaszban bemutatott példát, akkor látjuk, hogy a több állományban elhelyezett C nyelvű program meglehetősen sok

parancs kiadásával fordítható le. minden egyes forrásállományt egyenként le kell fordítanunk, majd el kell végeznünk a szerkesztést.

Ha tehát egy alkalmazás 100 C nyelvű forrásállományból áll, legalább 101 parancsot kell kiadnunk a fordítás során. A fordítást tehát mindenkorban automatizálnunk kell, különben a rendszergazdai munkakörre jelentkezők tábora igen hamar elnéptelenedik.

Az automatizálást végezhetjük úgy, hogy a fordításhoz használt parancsokat egy szöveges állományba írjuk, és a héj segítségével futtatjuk ezt az állományt. Van azonban jobb megoldás is!

A make program kimondottan olyan feladatok kezelésére készült, ahol állományokból készítünk állományokat, bonyolult összefüggések figyelembe vételével. A make számára tulajdonképpen mellékles, hogy C nyelvű programokról vagy más állományokról van szó, hiszen a tényleges fordítást nem ő végzi. A make az állományok közti összefüggéseket kezeli, azt határozza meg, hogy milyen parancsokat kell végrehajtani a végeredményként elkészítendő állományok előállítása során[19].

A projektállomány alapszerkezete

A make az indítása után a projektállományt keresi, amely leírja az állományok közti viszonyokat. A make alapértelmezés szerint a `makefile`, majd a `Makefile` állományokat keresi a munkakönyvtárban, hogy a projekt leírását és a teendőit beolvassa.



A UNIX rendszerek számára készült projektállományok neve általában `Makefile`, így mi is ezt használjuk a példákban. A `makefile` inkább olyan egyszerűbb rendszereken divatos, amelyek nem képesek különbséget tenni a kis- és nagybetűk között.

A projektállomány a make saját nyelvén készül, amely egyszerű szabályokat tartalmaz az állományok közti viszonyok leírására. A szabályok alakja a következő:

```

1 részcél: előfeltételi előfeltétel2
2     parancs1
3     parancs2

```

A make számára roppant fontos, hogy a részcél után, az előfeltétel előtt egy ketőspont legyen, és minden parancs előtt egy tabulátor karakter szerepeljen. Ha a szövegszerkesztőnk a tabulátor karakterek helyett szóközöt helyez az állományba, a make vissza fogja utasítani a projektállományt!

A szabály egyes elemeinek szerepe a következő:

részcel A cél, amelynek elérésére a szabály vonatkozik.

A cél a legtöbb esetben valamely állomány előállítására vonatkozik, a részcél pedig általában annak az állománynak a neve, amelyet az adott szabály alapján el lehet készíteni.

előfeltétel A részcél eléréséhez szükséges előfeltétel vagy előfeltételek.

Előfeltételként általában állománynevet adunk meg, a szabály első sorában a kettőspont után tehát azoknak az állományoknak a neve szerepel szóközzel elválasztva, amelyek az adott részcél eléréséhez, az adott állomány előállításához szükségesek.

parancs A parancs az adott részcélt megvalósító utasítás, amelyet a make a héj számára futtatásra átad.

Egy részcélhoz tetszőlegesen sok utasítás tartozhat, amelyeket a make sorban hajt végre, de minden parancs előtt egy tabulátor karakternek kell állnia!

A make elindítása után megkeresi a projektállományt, betölti, és kikeresi az első részcélt, majd megkísérli elérni a részcélt a parancsok futtatásával. Ha az első részcél előfeltételei nem adottak, a command megpróbálja megkeresni a projektállomány részcéljai között, hogyan lehet lehet az előfeltételeket előállítani.

A következő példa bemutatja, hogyan készíthetünk projektállományt, hogyan kell a részcélokat elrendeznünk a make számára.

84. példa Vegyük alapul a már bemutatott programot, amelyet a következő parancsok kiadásával fordítottunk le:

```
$ gcc -c main.c  
$ gcc -c proba.c  
$ gcc main.o proba.o -o proba
```

Az eddigi ismeretek alapján a projektállomány könnyen elkészíthető, csak arra kell ügyelnünk, hogy a projekt célja – a proba állomány – első részcélként szerepeljen az állományban.

```
1 proba: proba.o main.o  
2         gcc main.o proba.o -o proba  
3  
4 main.o: main.c proba.h  
5         gcc -c main.c  
6  
7 proba.o: proba.c proba.h  
8         gcc -c proba.c
```

Figyeljük meg, hogy a projektállomány a fordításhoz használt parancsokat és az állományok közti függőségeket tükrözi! A main.o állomány előfeltételeként

azért szerepel a proba.h, mert a main.c állományban az #include parancs segítségével felhasználjuk a proba.h állományt.

Ha most indítjuk a make programot, az beolvassa a Makefile tartalmát, és létrehozza az első részcélét:

```
$ make
gcc -c proba.c
gcc -c main.c
gcc main.o proba.o -o proba
$
```

Amint látjuk, az első részcél megvalósításához szükség volt a második és a harmadik részcélcíként megadott két állományra is.

Ha a make csak a projekt mechanikus megvalósítására volna képes, nem sok értelme volna a használatának. Ami igazán hasznossá teszi a programot – és a projektállományok használatát – az az, hogy a make csak a szükséges lépéseket végzi el, ki tudja hagyni a felesleges fordítási fázisokat.

A make igen egyszerű módszert használ a felesleges lépések kihagyására. minden részcél esetében megvizsgálja a célállomány és az előfeltételként adott összes állomány módosítási dátumát, és ha a célállomány frissebb, mint az előfeltételként adott összes állomány, akkor a részcélt kihagyja. Ha ugyanis a célállomány újabb, mint azok az állományok, amelyekből készült, akkor felesleges újra előállítani. Ilyen egyszerű!

A következő példa bemutatja, hogy mi történik, ha valamelyik állományt módosítjuk, és a programot újra le akarjuk fordítani.

85. példa Módosítsuk a main.c állományt, és futtassuk újra a make programot!

```
$ make
gcc -c main.c
gcc main.o proba.o -o proba
$
```

Amint látjuk, a make csak a szükséges lépéseket hajtotta végre, hogy erőforrást takarítson meg.

Ha nem módosítunk egyetlen állományt sem, és újra futtatjuk a make programot, az egyetlen lépést sem hajt végre, csak egy üzenetben jelzi, hogy a projekt a lehető legnagyobb rendben van.

```
$ make
make: 'proba' is up to date.
$
```

Ez az üzenet azt jelenti, hogy a program már le van fordítva.

Ne becsüljük alá a make által elért megtakarítást! A legtöbb programcsomag több száz állományból áll, a fejlesztés során pedig általában csak néhány állományt módosítunk egyszerre. Ha egy 100 állományból álló programcsomagban 2 állományt módosítunk, a make a fordítási időt akár 98%-kal csökkentheti!

A projektállomány makrói

A make a projektállományban makrókat is képes kezelni. A makrókat a projektállomány egyszerűsítése érdekében használhatjuk, körülbelül úgy, ahogyan a változókat szokás kezeln a BASH programokban.

A makrók használata hasonló ugyan a BASH programok változóihoz, van azonban néhány külöbség is. Vegyük sorra az azonosságokat és a különbségeket!

- A projektállományban is elhelyezhetünk megjegyzéseket a # karakter után a sor végéig.
- A projektállomány makróinak a BASH változókhoz hasonlóan az = jellet adhatunk értéket, de a BASH programokkal ellentétben az = előtt és után állhat szóköz.
- A projektállomány makróit nem kell " " jelek közé tennünk, ha az érték szóközt is tartalmaz.
- A BASH változókat a \$ jellet helyettesíthettük be értékükkel, a projektállomány makróinak esetében azonban () jeleket is kell használnunk. A projektállományban a MAKE makró értékét a \$(MAKE) kifejezéssel helyettesíthetjük be.

A következő példa bemutatja, hogyan helyezhetjük el a projektállomány ismétlődő részeit makróban annak érdekében, hogy az állományt később könnyebben módosíthassuk.

86. példa A 9.1.2. példában használt Makefile állomány ismétlődő részeit helyezzük el makrókban, és hivatkozzunk a makrókra a szükséges helyeken!

```
1 CC = gcc
2 TARGET = proba
3
4 proba: proba.o main.o
5     $(CC) main.o proba.o -o $(TARGET)
6
7 main.o: main.c proba.h
8     $(CC) -c main.c
9
10 proba.o: proba.c proba.h
11     $(CC) -c proba.c
```

A példában már sokkal könnyebb a használt C fordító nevét megváltoztatni, hiszen ha azt átírjuk az első sorban, a teljes projektben megváltozik.

A makrók bevezetése látszólag nem tette egyszerűbbé a projektállományt, de nyilvánvaló, hogy összetettebb feladat esetén a nyereség nagyobb lett volna.

Név	Jelentés
\$@	az előállítandó állomány neve (réscél)
\$<	az első előfeltétel
\$?	az összes előfeltétel neve, amelyek újabbak az előállítandó állománynál
\$+	az összes előfeltétel
\$^	az összes előfeltétel az ismétlődések eltávolításával
\$(@D)	a \$@ könyvtár része
\$(@F)	a \$@ állomány része
\$(<D)	a \$< könyvtár része
\$(<F)	a \$< állomány része
\$(^D)	a \$^< könyvtár része
\$(^F)	a \$^< állomány része
\$(+D)	a \$+ könyvtár része
\$(+F)	a \$+ állomány része
\$(?D)	a \$? könyvtár része
\$(?F)	a \$? állomány része

9.2. táblázat. A make beépített makrói

A make jónéhány beépített makróval is rendelkezik, amelyek igen hasznosak lehetnek kifinomult projektállomány készítésáhez. A beépített makrók egy részének neve egybetűs, ezek értékére a () jelek nélkül hivatkozhatunk. A legfontosabb beépített makrókat a 9.2. táblázat tartalmazza.

Szabályok állománynév-végződés alapján

A szabályokat egyszerűíthetjük, ha megadjuk, hogy bizonyos állománynév-végződések mit jelentenek. Olyan szabályokat készíthetünk, amelyek megadják, hogy az egyes állományok hogyan alakíthatók át más állományokká.

Az állománynév-végződéseket először meg kell adnunk a .SUFFIXES kulcsszóval. A make azokat a karakterláncokat állománynév-végződésként kezeli, amelyeket a .SUFFIXES kulcsszó után megadtunk.

A .SUFFIXES kulcsszó használata után meg kell adnunk, hogy milyen utasítással alakítható az egyik állománytípus a másik típusú állománnyá. Ehhez olyan szabályt kell készítenünk, amelynek részcélja a két állománynév-végződést tartalmazza, előbb a kiindulási, majd a célként létrehozandó állománynév-végződést.

A .SUFFIXES kulcsszó használatát mutatja be a következő példa.

87. példa A Makefile állományban általánosan használható szabályokat szeretnénk létrehozni. Készítsük el a következő állományt:

```

1 CC = gcc
2 TARGET = proba
3 .SUFFIXES: .c .o
4
5 .c.o:
6     $(CC) -c $<
7
8 proba: proba.o main.o
9     $(CC) main.o proba.o -o $(TARGET)
10
11 main.o: main.c proba.h
12 proba.o: proba.c proba.h

```

Figyeljük meg, hogy a 3. sorban a .SUFFIXES kulcsszó segítségével megadtuk, hogy a .c és a .o állománynév-végződéseket kezelje a make, és az 5–6. sorban megadtuk, miképpen készíthetünk .c végződésű forrásállományból .o végződésű tárgykódú állományt.

A 11. és a 12. sorban csak azért kellett megadnunk információkat a main.o és a proba.o állományokról, mert ezek függnek a proba.h állománytól.

A make rendelkezik beépített szabályokkal is a különféle állománynév-végződések kezelésére, azaz a make sok állománytípusról tudja, hogyan alakíthatók egymásba anélkül, hogy azt a projektállományban leírnánk. Ezt mutatja be a következő példa.

88. példa Hagyjuk el a .o és .c végződések kezelését leíró szabályokat a projektállományból, hogy megvizsgáljuk, miképpen kezeli a make ezeket az állományokat a beépített szabályok alapján! A szabályok elhagyásával a következő állományt kapjuk:

```

1 CC = gcc
2 TARGET = proba
3
4 proba: proba.o main.o
5     $(CC) main.o proba.o -o $(TARGET)
6
7 main.o: main.c proba.h
8 proba.o: proba.c proba.h

```

Ha most megkíséreljük lefordítani a programot, az a beépített szabályok alapján problémamentesen lefordul:

```
$ make
gcc    -c -o proba.o proba.c
gcc    -c -o main.o main.c
gcc main.o proba.o -o proba
$
```

Amint láthatjuk, a beépített szabályok nem egyeznek meg tökéletesen azokkal, amelyeket eddig használtunk, de a hatásuk ugyanaz.

Hamis állományok használata

Az eddigi projektállományokban minden szabály egy állomány előállítására vonatkozott, minden részcél egy állomány neve volt. A szabályokhoz tartozó parancsok pedig elő is állították a részcélként megadott állományt.

Készíthetünk azonban olyan szabályokat, amelyek nem állítják elő a részcélként megadott állományokat. Mondhatjuk, hogy az ilyen szabályok hamis – soha létre nem hozott – állományokra vonatkoznak.

Az eddig bemutatott szabályakra az is igaz, hogy minden összefüggő fát alkottak. A `make` megkísérlelte létrehozni az első szabályban meghatározott részcélt, és – ha szükséges volt – ehhez felhasználta a projektállományban található további szabályokat is. Ez azt jelenti, hogy minden egyes szabály elérhető volt az első szabály előfeltételein keresztül.

Készíthetünk olyan szabályokat is, amelyek nem elérhetők az első részcélből kiindulva. Az ilyen szabályokat csak akkor próbálja meg megvalósítani a `make`, ha erre külön utasítást kap. Az ilyen szabályokban megvalósított részcélok megvalósítására egyszerűen úgy adhatunk utasítást a `make` számára, hogy beírjuk a megfelelő előfertételel nevét a parancsorba.

A hamis állományok és előírándó részcélok használatát mutatja be a következő példa.

89. példa Készítsünk projektállományt, amely segíti a pótolható állományok törlését! A következő állományt ilyen céllal hoztuk létre:

```
1 CC = gcc
2 TARGET = proba
3
4 proba: proba.o main.o
5     $(CC) main.o proba.o -o $(TARGET)
6
7 main.o: main.c proba.h
8 proba.o: proba.c proba.h
9
10 clean:
11     rm -f *.o
12     rm -f $(TARGET)
```

Az állományban található clean részcél nem érhető el az első részcélból, ezért ha meg akarjuk valósítani, külön elő kell írnunk a make indításakor. Ez a részcél törli azokat az állományokat, amelyeket újra elő lehet állítani a forrásállományokból:

```
$ make clean
rm -f *.o
rm -f proba
$
```

Látható, hogy a make által kiadott parancsok nem állították elő a clean nevű állományt, hiszen a végrehajtott szabály hamis állományra vonatkozott.

Összetett parancsok használata

Az eddigi projektállományokban is használtunk parancsokat a szabályokban, ezek azonban meglehetősen egyszerű parancsok voltak. Néhány apróságot tudnunk kell ezekkel a parancsokkal kapcsolatban, ha kifinomultabb projektállományokat akarunk készíteni.

Fontos tudnunk, hogy a make a szabályokban megadott parancsokat soronként hajtja végre, minden sort új folyamatként. Ez azt eredményezi, hogy az egyes sorok önálló életet élnek. Ha egy többsoros parancsot egy folyamatként szeretnénk végrehajtani, a sorok végére \ jelet kell helyeznünk az újsor karakter figyelmen kívül hagyására. Fontos azonban, hogy a \ közvetlenül az újsor karakter előtt legyen, még egy szóköz se kerüljön a \ és az újsor karakter közé. Ezt a módszert mutatja be a következő példa.

90. példa A következő példa egy projektállomány részeként készült, ami nem működik:

```
1  clean:
2      for F in *.o; do
3          rm -f $F
4      done
```

A projektállomány nem működik, mert a parancs három sorát három folyamat – a héj három példánya – kapja meg, és természetesen egyik sem tud mit kezdeni a töredékparancsokkal.

A hibás részlet a következőképpen javítható:

```
1  clean:
2      for F in *.o; do \
3          rm -f $F; \
4      done
```

Figyeljük meg, hogy az utolsó sor kivételével minden újsor karaktert hatástartanítottunk a parancsban. Az utolsó újsor karaktert természetesen nem kellett „kikapcsolnunk”, hiszen a harmadik sor végén vége a parancsnak.

Fontos, hogy megfigyeljük, hogy a parancs második sorában el kellett helyeznünk egy ; karaktert. Nyilvánvalóan elengedhetetlenül fontos ez a karakter, hiszen az egy sorban megadott parancsokat el kell választanunk egymástól.

Talán érdemes megemlíteni, hogy a példa javított programrészlete is hibás a \$ karakter helytelen használata miatt.

A következő apróság, amit meg kell ismernünk a parancsokkal kapcsolatban, a \$ karaktert érinti.

A make számára a \$ karakter a makrók értékének behelyettesítésére ad utasítást, ahol tehát ez a karakter található, ott a make makróbehelyettesítést végez, és a \$ karaktert eltávolítja.

Ha a héjnak átadandó parancsban \$ jelet akarunk használni, a \$\$ kifejezést kell a parancsba írnunk. A \$\$ kifejezést a make kicseréli a \$ karakterre, és átadja a héjnak. Ezt mutatja be a következő példa.

91. példa Az előzőleg bemutatott hibás projektállomány-részlet a következő volt:

```

1 clean:
2     for F in *.o; do \
3         rm -f $F; \
4     done

```

Javítsuk ki a hibát a \$ karakter \$\$ kifejezással való felcserélésével:

```

1 clean:
2     for F in *.o; do \
3         rm -f $$F; \
4     done

```

Ez a részlet már tökéletesen működik, hibátlan!

9.1.3. A forrásprogram foltozása

UNIX rendszereken a szöveges állományok közti különbségek felderítésére használható a diff program. Ez a program a parancssorban megadott két szöveges állományt – vagy két könyvtár összes állományát – összehasonlítja, és a szabványos kimenetre írja a különbségeket.

A programok szerzői a program új változata mellett gyakran a régebbi változattal készített különbségi állományt is terjesztik. A hozzáértő a különbségi állományból azonnal látja, mi módosult a programban, és a régi állományok birtokában előállíthatja az új programváltozatot. Ez a módszer gyorsíthatja a program letöltését is, hiszen a különbségi állomány csak a módosult és új sorokat tartalmazza.

A különbségi állományban található módosításokat a `patch` programmal érvényesíthetjük. Ezt a folyamatot a szakirodalom a „foltozás” szóval emlegeti, amely jól leírja, mi is történik ilyenkor.

A `diff` program fontos kapcsolója a `-r`, amelynek segítségével arra utasíthatjuk a programot, hogy teljes könyvtárakat, azok minden állományát hasonlítsa össze. Ha használjuk a `-r` kapcsolót, hasznos lehet a `-N` kapcsoló is, amelynek hatására a program azokat az állományokat is figyelembe veszi, amelyek az egyik könyvtárban léteznek, a másikban azonban nem. Ha a munka során a könyvtárszerkezetben új állományt hoztunk létre, nyilvánvalóan használnunk kell a `-N` kapcsolót.

Szintén hasznos a `-u` kapcsoló, amely a létrehozott különbségi állomány formáját befolyásolja. A `diff` többféle állományformátumot tud létrehozni, amelyek közül – az elterjedt vélemény szerint – a `-u` segítségével létrehozott formátum a legolvashatóbb. Ha használjuk e kapcsolót, kevesebb haragosunk lesz!

Összetett állományszerkezetekben általában vannak olyan állományok, amelyek változásait nem akarjuk terjeszteni. Az ideiglenes, automatikusan létrehozott állományok változásait felesleges terjesztenünk, ezért ezek összehasonlítását tiltanunk kell. Ha ezt nem tesszük meg, a különbségi állomány igen nagyjá és nagyon olvashatatlantá válhat. A `diff` a `-X` kapcsoló után megadott állományban található állományneveket a munka során nem használja, ezeknek az állományoknak az összehasonlítását nem végzi el. Össze kell gyűjtenünk tehát az elkerülendő állományok nevét egy állományban és megadni ennek az állománynak a nevét a `-X` kapcsoló után, ha szép tiszta különbségi állományt akarunk készíteni.

92. példa Két könyvtárban megtalálható a programunk régebbi és újabb változata. Készítsünk különbségi állományt, amelyben a két állományszerkezet különbsége megtalálható!

```
$ diff -urN minysql-1.3/ minysql-1.4/ >patch.txt
```

Amint látjuk, a különbségi állomány könnyedén elkészíthető a két könyvtár nevének megadásával.

A `patch` program használata meglehetősen egyszerű. A program a szabványos bemenetről olvassa a különbségi állományt és azt érvényesíti a megfelelő állományokon. Ha azonban a különbségi állomány érvényesítése – a foltozás – nem abban a könyvtárban történik, amelyben a foltot létrehoztuk, a program nem találja meg az állományokat. Ilyen esetben szükség lehet a `patch -p` kapcsolójára. A kapcsoló után megadhatjuk, hogy a foltállományban található állománynevek

```

mplayerplug-in.patch (~/Programs) - GVIM1
Ejő Szerkesztés Eszközök Szintaxis Pufferök Ablak Súgó
diff -urN -X dontdiff.mplayerplug-in mplayerplug-in-2.11/Source/nsIScriptableMplayerPlugin.h mplayerplug-in-hacked/Source/nsIScriptableMplayerPlugin.h
ugina.h
1 --- mplayerplug-in-2.11/Source/nsIScriptableMplayerPlugin.h 2004-04-22 1
2 4:44:01.000000000 +0200
3 +++ mplayerplug-in-hacked/Source/nsIScriptableMplayerPlugin.h 2004-04-22
4 14:43:48.000000000 +0200
5 @@ -16,52 +16,55 @@
6 #endif
7 /* starting interface: nsIScriptableMplayerPlugin */
8 #define NS_ISCRIPTABLEMPLAYERPLUGIN_IID_STR "f728830f-1dd1-4444-6666-fb9f414f2465"
9 #define NS_ISCRIPTABLEMPLAYERPLUGIN_IID "{f728830f-1dd1-4444-6666-fb9f414f2465"
10
11 #define NS_ISCRIPTABLEMPLAYERPLUGIN_IID \
12 {0xf728830f, 0x1dd1, 0x4444, \
13 {0xf728830f, 0x1dd1, 0x4244, \
14 {0x66, 0x66, 0xfb, 0x9f, 0x41, 0x4f, 0x24, 0x65}}
15
16 -class NS_NO_VTABLE nsIScriptableMplayerPlugin:public nsISupports {
17 public:
2,1 Top

```

9.1. ábra. A különbségi állomány a vim szövegszerkesztőben

elejéről hány könyvtárnevet távolítson el a program, amikor a foltozandó állományt keresi.

93. példa A 9.1.3. példában elkészített különbségi állomány segítségével foltozzuk az állományokat az újabb változatra!

```
$ cd minisql-1.3
$ patch <patch.txt
patching file calculator.y
patching file test1.sql
$
```

A példában a foltozás egyszerű volt, mert az állományok helye – a könyvtár teljes elérési útja – nem változott. Ha az állományok más könyvtárban lennének, a patch programnak a megfelelő kapcsolókat kellett volna megadnunk.

Szerencsés, ha egy pillantást vétünk a különbségi állományra, mielőtt másoknak adnánk vagy magunk érvényesítenénk. A vim szövegszerkesztő kiszínezi a különbségi állományt, így könnyen olvasható (9.1. ábra).

A különbségi állomány egymástól független különbségek sorozata. A különbségek elején minden parancs olvasható, ami a különbségi állományt létrehozta, utána pedig az összehasonlított állományok neve. Előbb a régebbi állomány állománynevét olvashatjuk – --- jelekkel a sor elején –, majd az újabb állomány neve következik – +++ jelekkel bevezetve. Ezek után néhány számmal az állományok érintett sorainak sorszáma következik a @@ jelekkel a sor elején.

A szakaszt a tulajdonképpeni különbség zárja. A különbségekkel jelölt sorok elején a - és a + jeleket látjuk. A folt eltávolítása során a - jelrelt sorokat el

kell távolítani, a + jelekkel jelölt sorokat pedig be kell illeszteni. A különbségi állományban a teljes sor szerepel akkor is, ha a módosítás csak egyetlen karaktert érint, a diff a sorokon belüli különbségeket nem vizsgálja.

Az eltávolítandó és beszúrandó sorok előtt és után minden olvashatunk néhány sort, amely a módosítandó sorok előtt és után volt az állományokban. Ezek a régi és új állományban megegyező sorok segítik a módosítások elolvasását és ellenőrzésre használhatók a különbségi állomány érvényesítésekor.

A patch nem érvényesíti azokat a különbségeket, amelyek előtt és után nem a megfelelő sorok találhatók a célállományban, hiszen ez egyértelműen azt jelzi, hogy az állományt a foltozás előtt módosítottuk. A visszautasított különbségeket az adott állomány nevével megegyező nevű, .rej végződésű állományokba menti a program.

9.1.4. Hordozható programok

Az előző oldalakon részletesen megvizsgáltuk a make programot és a számára készített projektállományokat. A program igen hasznos bármilyen számítógéppel támogatott projekt esetében, és komoly szerepet kap a C nyelven megírt alkalmasok fordításakor és telepítésekor.

Az eddigi eszközök azonban a fordítás során minden ugyanazokat a parancsokat használják. Ha az adott számítógépen a fordítást más parancsok kiadásával kell elvégezni, a projektállományt át kell írnunk.

A következő oldalakon olyan eszközökről olvashatunk, amelyek segítségével a projektállomány automatikusan előállítható és módosítható az adott számítógépen uralmodó viszonyoknak megfelelően. Ezek az eszközök a programot hordozhatóvá, a telepítést könnyebbé és hatékonyabbá teszik, és ezért általánosan elterjedtek GNU/Linux és más UNIX rendszereken.

Hordozható programok készítésére GNU/Linux rendszereken általában az automake és az autoconf programokat használjuk. Mindkét program lehetősen bonyolult, ezért a részletes ismertetésük nem célunk. A következő oldalakon olvasható vázlatos leírás célja, hogy a rendszerelő számára hasznos ismereteket összefoglalja.

Hordozható program készítése

A következőkben megvizsgáljuk, miképpen lehet az autoconf és az automake segítségével hordozható programot készíteni. A munka menetét néhány pontba szedjük, hogy könnyebben lehessen követni az eseményeket.

1. Készítsünk egy könyvtárat, amely a programfejlesztési munkánk alapkönyvtára lesz. Ide kerülnek mindenek az állományok, amelyek a szoftverfejlesztés során keletkeznek.

Hozzunk létre egy alkönyvtárat src néven, és helyezzük el ebben az alkönyvtárban a C nyelvű forrásprogramokat. Az egyszerűség kedvéért most

csak egy állományt hozunk létre `main.c` néven, melynek tartalma a következő:

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[]){
4     printf("Hello world.\n");
5 }
```

2. A következő lépés a `configure.in` állomány létrehozása és kitöltése az alapkönyvtárban. Ezt az állományt létrehozhatnánk az `autoscans` programmal is, de az egyszerűség kedvéért most írjuk be a következő sorokat az állományba:

```

1 AC_INIT(proba, 0.1, proba@linux)
2 AM_INIT_AUTOMAKE(proba, 0.1)
3
4 AC_PROG_CC
5
6 AC_OUTPUT(Makefile src/Makefile)
```

Az állományba az `automake` és az `autoconf` számára a következő makrókat helyeztük el:

`AC_INIT()` Az `autoconf` indítómakrójá, amelynek paraméterei a programunk neve, a programunk változatának száma és a program szerzőjének e-mail címe, ahol a felhasználók panaszkodhatnak a hibás program miatt.

`AM_INIT_AUTOMAKE()` Az `automake()` indítómakrójá, amelynek paraméterei a program neve és a változat száma. Ezeket az információkat tehát két helyre is beírtuk.

`AC_PROG_CC` Az `autoconf` makrójá, amely utasítást ad a fordítás előtt a C fordító felkutatására. Valójában még sok ilyen makró áll a rendelkezésünkre, de az egyszerűség kedvéért most csak ezt az egyet használjuk.

`AC_OUTPUT()` Az `autoconf` és az `automake` számára is fontos makró, amely azt határozza meg, hogy mely állományokat kell létrehozni a program fordítása előtt. A felsorolt állományok neve közé csak egy szóközt kell tennünk. Ha az állománynevek közé vessző karaktereket teszünk, a fordítási kísérleteink kudarcot vallanak.

A példában a `Makefile` és az `src/Makefile` állományneveket írtuk a létrehozandó állományok közé. A programunk attól lesz hordozható, hogy a projektállományokat a fordítás színhelyén automatikusan hozzuk létre.

A make alapvetően egykönyvtáras program, azaz minden könyvtárban, ahol a fordítás során valamilyen munkát kell végezni, létre kell hoznunk egy projektállományt.

3. A következő lépés a `Makefile.am` és az `src/Makefile.am` állomány létrehozása és kitöltése lesz. Ezek az állományok az automake számára írják le, hogy miképpen kell előállítani a `Makefile.in` állományokat.

Az alapkönyvtár `Makefile.am` állománya egyszerű, a következő sort tartalmazza:

```
1 SUBDIRS = src
```

A sor azt jelenti az automake számára, hogy az `src/` alkönyvtárban is el kell végeznie a munkát. Ha több alkönyvtár is lenne, azok nevét szóközzel elválasztva be kellene ide írnunk.

Az `src/Makefile.am` állomány kissé bonyolultabb, de szintén könnyen megérthető:

```
1 bin_PROGRAMS = proba  
2 proba_SOURCES = main.c
```

A sorok azt írják le az automake számára, hogy egy bináris programot szeretnék létrehozni, amely a `main.c` állományból készül. Hogy miképpen kell C forrásállományból bináris programot készíteni, azt az automake jól tudja a segítségünk nélkül is.

4. A következő lépésekben hozzunk létre néhány állományt az alapkönyvtárban. Ezeknek az állományoknak létezniük kell, de akár üresek is lehetnek, tehát majd később kitölthük őket:

`NEWS` A programfejlesztés során adódó újdonságok, hírek.

`README` A legfontosabb információk, amelyeket a programról érdemes elmondani.

`AUTHORS` A program szerzőinek névsora.

`ChangeLog` A program fejlesztésének naplója.

5. A következő lépés az `aclocal` program futtatása az alapkönyvtárban.

Ez a program beolvassa a `configure.in` állományt, előállítja az `aclocal.m4` állományt, amely azokat a makrókat tartalmazza, amelyekre szükség lesz a fordítás előtt. A programunk ezeket a makrókat magával viszi a telepítés helyére, hátha ott nem állnak rendelkezésre.

6. A következő lépés az automake futtatása az alapkönyvtárban a --add-missing kapcsolóval. A kapcsoló hatására a program létrehoz néhány szöveges állományt, amely segíti a telepítést és a program megértését, és amelyeket lusták voltunk megírni.

Az automake ezen kívül létrehozza a Makefile.in és az src/Makefile.in állományokat. Ez hatalmas munka, hiszen ezek az állományok leírják, miképpen kell lefordítani a programunkat és telepíteni azt, már csak néhány változót kell behelyettesítenünk bennük.

7. A következő lépés az autoconf futtatása az alapkönyvtárban.

Az autoconf legfontosabb feladata az, hogy létrehozza a configure nevű programot az alapkönyvtárban, amely képes felderíteni a fordításkor a környezetet, és a fellelt elemeknek megfelelően képes behelyettesíteni a hiányzó változókat, a Makefile.in állományban létrehozva a Makefile állományokat.

A programunk ezzel tulajdonképpen kész, lefordíthatjuk és telepíthetjük akár erre a számítógépre, akár más számítógépre.

Hordozható program fordítása

A programunk alapkönyvtárában található configure egy héjprogram, amely felderíti a környezetet, és előállítja a félkész Makefile.in állományokból a Makefile állományokat a hiányzó részek kitöltésével. Ezt a folyamatot fordítás előtti beállításnak is nevezhetjük.

94. példa Végezzük el a fordítás előtti beállítást az előző oldalakon elkészített programon a configure program futtatásával!

```
$ ./configure
checking for a^BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
configure: creating ./config.status
```

```
config.status: creating Makefile
config.status: creating src/Makefile
config.status: executing depfiles commands
$
```

Figyeljük meg, hogy a program megkereste a szükséges eszközöket, és elkészítette a fordításhoz szükséges állományokat!

Miután elvégeztük a fordítás előtti beállítást, és a szükséges Makefile állományok elkészültek, a fordítást egyszerűen elvégezhetjük a make futtatásával.

95. példa Futtassuk a make programot az előzőekben előállított program alapkönyvtárában, hogy lefordítsuk a programot!

```
$ make
Making all in src
make[1]: Entering directory '/home/pipas/teszt/src'
if gcc -DPACKAGE_NAME=\"proba\" -DPACKAGE_TARNAME=\"proba\"
-DPACKAGE_VERSION=\"0.1\" -DPACKAGE_STRING=\"proba\ 0.1\
-DPACKAGE_BUGREPORT=\"proba@linux\" -DPACKAGE=\"proba\
-DVERSION=0.1\" -I. -I. -g -O2 -MT main.o -MD -MP -MF
\".deps/main.Tpo" -c -o main.o main.c; \
then mv -f \" deps/main.Tpo\" \" deps/main.Po"; else rm -f
\".deps/main.Tpo"; exit 1; fi
gcc -g -O2 -o proba main.o
make[1]: Leaving directory '/home/pipas/teszt/src'
make[1]: Entering directory '/home/pipas/teszt'
make[1]: Nothing to be done for 'all-am'.
make[1]: Leaving directory '/home/pipas/teszt'
$
```

Amint látjuk, a program fordítása talán kissé bonyolult, de sikeres.

Az automake és autoconf páros által létrehozott állományok az egyszerű beállításnál és telepítésnél összetettebb feladatokra is alkalmasak. A configure program például elfogadja a következő paramétereket:

--help A configure program használatáról kapunk egy rövid leírást ennek a kapcsolónak a segítségével. Ez igen fontos lehet, hiszen a beállítóprogramok nem egységesek.

--prefix=könyvtárnév Az architektúrától független állományok telepítési alapkönyvtárnak megváltoztatása. Alapértelmezés szerint ez a könyvtár a /usr/local/.

--exec-prefix=könyvtárnév Az architektúrafüggő bináris állományok telepítési alapkönnyvtárnak megváltoztatása. Alapértelmezés szerint ez a könyvtár a /usr/local/.

--enable-XXX Bizonyos programok esetében különféle elemek programba való fordítását lehet engedélyezni. A konkrét elemekről a --help kapcsolóval kérhetünk információt.

--disable-XXX Bizonyos programok esetében különféle elemek programba fordítását tilthatjuk. A konkrét elemekről a --help kapcsoló segítségével kérhetünk információt.

A configure program által létrehozott projektállomány többféle hasznos részcél is megvalósít. Ezek közül a legfontosabbak a következők:

all Az első részcél az all, amely lefordítja a programot.

install A program telepítését, a program állományainak felhasználási helyükre való másolását végző részcél.

uninstall Az install által felmásolt állományok törlését végző részcél.

clean A pótolható, újra előállítható állományok törlését végző részcél.

dist A programcsomagból terjeszthető tömörített állományt készít ez a részcél. A terjeszthető állomány tartalmazza a program lefodításához, telepítéséhez és módosításához szükséges állományokat.

A következő példa bemutatja, miképpen készíthetünk terjeszthető változatot a programunkból.

96. példa Készítsük el a bemutatott programunk terjeszthető változatát a dist részcél végrehajtásával (az egyszerűség érdekében néhány sort elhagytunk a példából):

```
$ make dist
test ! -d proba-0.1 || find proba-0.1 -type d ! -perm
-200 -exec chmod u+w ';' && rm -fr proba-0.1; ;
mkdir proba-0.1
list='src'; for subdir in $list; do \
    if test "$subdir" = .; then :; else \
        test -d "proba-0.1/$subdir" \
        || mkdir "proba-0.1/$subdir" \
        || exit 1; \
        (cd $subdir && \
        make \
            top_distdir="../proba-0.1" \
            distdir="../proba-0.1/$subdir" \
            distdir) \
        || exit 1; \
    fi; \
done
$
```

A programunk terjeszthető változata elkészült:

```
$ ls -lh proba-0.1.tar.gz
-rw-rw-r-- 1 pipas pipas 66K jún 20 17:30 proba-0.1.tar.gz
$
```

Az állományt más számítógépekre másolhatjuk, és ott lefordíthatjuk és telepíthetjük a programunkat.

9.2. Programkönyvtárak

A számítógépek programozásának igen elterjedten használt fogása a programkönyvtárak (*software library*, programkönyvtár) készítése. minden operációs rendszer, így a GNU/Linux is lehetővé teszi programkönyvtárak készítését és használatát. A következő oldalakon arról olvashatunk, miképpen kezelhetjük a programkönyvtáratokat GNU/Linux rendszereken.

A programkönyvtárat nem önállóan futtatjuk, a szolgáltatásaiat a felhasználók nem közvetlenül veszik igénybe, hanem az alkalmazásokat készítő programozók segítségével. A programkönyvtárak általában általános feladatok megoldásaként készített programok, amelyeket több alkalmazás is felhasznál feladatainak ellátására. Ezek az alkalmazások csak akkor használhatók, ha a megfelelő programkönyvtár a rendelkezésünkre áll, és a megfelelő módon be van állítva. A rendszergazda feladata, hogy a megfelelő eszközökkel elérhetővé tegye a programkönyvtáratokat az alkalmazások számára[29].

A programkönyvtárak használata igen hasznos, hiszen a segítségükkel bizonyos részfeladatokat önálló programként valósíthatunk meg, így a programrészeket nem kell többszörösen tárolnunk a memoriában és a merevlemezen, valamint a karbantartás is egyszerűsödik. Ha például a JPEG képeket kezelő programrész a külön programkönyvtárban valósítjuk meg, frissíthetjük anélkül, hogy a képeket kezelő összes programot frissítenünk kellene.

9.2.1. Programformátumok

A Unix rendszereken a bináris – lefordított – programok futtatását a Linux rendszermag végzi. A programot a mag betölti a memória egy tetszőleges pontjára – valahova, ahol az adott pillanatban szabad memóriaterület található –, majd a futtatásra előkészíti és futtatja.

A futtatható bináris állományok formátuma független a számítógépben található processzor típusától, sőt valójában a használt operációs rendszer típusától is. A UNIX rendszerek legtöbbje szabványos állományformátumot használ a bináris programok tárolására.

Az állományformátum azt írja le, hogy a program futtatásának előkészítéséhez szükséges információk milyen formában állnak rendelkezésre az állományban.

Természetesen ez nem jelenti azt, hogy a UNIX bináris programok bármilyen processzoron futtathatók, hiszen maga a gépi kódú program az adott architektúrára jellemző gépi kódú utasításokat tartalmaz. A szabványos formátum azt jelenti, hogy a futtatásra előkészítő rész a rendszermagban egységes lehet bármely architektúrán és operációs rendszeren.

A Linux rendszer kezdetben az `a.out` formátumot használta. Habár még ma is tartalmazza a Linux rendszermag az `a.out` formátum támogatására szolgáló programrészeket, ez a bináris formátum ma már elavultnak tekinthető.



A UNIX C fordítóprogramok alapértelmezés szerint `a.out` nevű állományt hoznak létre a fordítás során, ez azonban nem jelenti azt, hogy a létrehozott állomány az `a.out` formátumot használja! Az `a.out` ma is a UNIX assembler alapértelmezett állományneve a kimenet elnevezésére.

A UNIX későbbi fejlődése során egy újabb bináris formátum terjedt el, melynek neve ELF (*executable and linking format*, végrehajtható és összekapcsolható formátum), amelyet a Linux rendszermag is átvett. Ma már szinte kizárolag az ELF formátumot használjuk a bináris programok tárolására.

Az ELF állományok – ahogyan nevük is mutatja – összekapcsolhatók programkönyvtárrakkal. A programozó tehát, aki az ELF formátumot választja a bináris programja tárolására, könnyedén és hatékonyan felhasználhatja a rendelkezésére álló programkönyvtárak szolgáltatásait.

9.2.2. Statikus programkönyvtárak

A statikus programkönyvtárak (*static libraries*) olyan programkönyvtárak, amelyeket a bináris programokba a fordítás utolsó lépéseként bemásol a szerkesztőprogram. A statikus programkönyvtárakat a következők miatt használjuk:

- A program fordításakor a statikus könyvtárat nem kell újrafordítanunk, így a fordítás gyorsabb, és a statikus könyvtár forrását sem kell tárolnunk.
- A programkönyvtárat és az azt használó alkalmazást más-más programozók készítik, így szerencsés azokat egymástól elkülönítve kezelní.

A statikus programkönyvtárak, GNU/Linux rendszereken . a végződésű állományokban tároljuk. A . a végződésű állományok archívumok, amelyeket az `ar` programmal kezelhetünk. A statikus programkönyvtárakat tartalmazó . a archívumok tárgykódú programokat tartalmaznak. (A tárgykódú programokról a 205. oldalon olvashattunk.)

A telepített statikus könyvtárakat az `ld` program használja a program fordításának utolsó szakaszában, a szerkesztés során. Mivel az `ld` a használt statikus programkönyvtárak tartalmát az elkészített programba bemásolja, a . a állományoknak nem kell elérhetőknek lenniük a program futtatásához.

A statikus programkönyvtárakat használó alkalmazások tehát viszonylag nagy méretűek, hiszen a használt programkönyvtárakat is tartalmazzák, és ugyanezen okból könnyen telepíthetők.

9.2.3. Megosztott programkönyvtárak

A megosztott programkönyvtárak (*shared objects*, megosztott tárgykódú elemek) nevüket onnan kapták, hogy a bennük tárolt programrészleteket fizikailag megosztva több alkalmazás is használhatja.

A megosztott programkönyvtárakat külön állományokban tároljuk, és csak akkor töltjük be a memoriába, ha szükség van rájuk. Amint egy olyan alkalmazást indítunk, amely az adott megosztott könyvtárat használja, a program indítása során a szükséges megosztott könyvtár is betöltődik, és az alkalmazással automatikusan összekapcsolódik. Ha egy időben több alkalmazás is fut a megosztott programkönyvtárat használva, maga a programkönyvtár csak egyetlen példányban foglal helyet a memoriában.

A megosztott programkönyvtárak használata tehát helytakarékkosság szempontjából igen előnyös, hiszen sem a merevlemezen, sem pedig a memoriában nem kell többszörösen tárolnunk a megosztott programkönyvtár utasításait.

A megosztott programkönyvtárak használatának ugyanakkor vannak hátrányai is! Mivel a megosztott programkönyvtár utasításai nem az alkalmazást tároló állományokban vannak, azok telepítéséről és karbantartásáról külön kell gondosknunk. A megosztott programkönyvtárak betöltése és összekapcsolása automatikus ugyan, de csak a megfelelő előkészítés után működik.

Napjainkban gyakorlatilag az összes program használ megosztott programkönyvtárakat, így azok kezelését mindenki által meg kell ismernünk.

A dinamikus szerkesztő

Az alkalmazások indításakor a megosztott programkönyvtárak betöltését és a programmal való összekapcsolását az `ld.so` program, a dinamikus szerkesztő végzi. A Linux rendszermag szempontjából a dinamikus szerkesztő az értelmezőprogram, amely az ELF formátumú állományok futtatását lehetővé teszi.

Az `ld.so` a következő lépésekben keresi meg az alkalmazás által igényelt megosztott programkönyvtárakat:

1. A dinamikus szerkesztő megvizsgálja az `$LD_PRELOAD` nevű környezeti változót, és ha abban szóközökkel elválasztott állományneveket talál, a megosztott programkönyvtárakat ezekből az állományokból kíséri meg először betölteni. (A régebbi, `a.out` formátumú állományok esetében a dinamikus szerkesztő az `$LD_AOUT_PRELOAD` változót veszi figyelembe.)

Az `$LD_PRELOAD` környezeti változót általában arra használjuk, hogy az alkalmazás számára betöltendő megosztott könyvtárat megváltoztassuk. Ha

egy alkalmazásnak a többi alkalmazástól eltérő megosztott programkönyvtárra van szüksége, ennek a megosztott programkönyvtárnak az állománynevét elhelyezzük az \$LD_PRELOAD környezeti változóban.

Amennyiben a futtatható program SUID vagy SGID bitje be van kapcsolva, a szerkesztő az \$LD_PRELOAD (\$LD_AOUT_PRELOAD) környezeti változó értékét nem veszi figyelembe. Erre biztonsági okokból van szükség, hogy a felhasználók ne vehessék rá a dinamikus szerkesztő módosított, kiskapukkal ellátott megosztott programkönyvtárak futtatására más felhasználók nevében.

2. Az ld.so az \$LD_LIBRARY_PATH nevű környezeti változóban ketőspontokkal elválasztva megadott könyvtárakban keres. (A régebbi, a.out formátumú állományok esetében a dinamikus szerkesztő az \$LD_AOUT_LIBRARY_PATH változót veszi figyelembe.)
3. A dinamikus szerkesztő a /etc/ld.so.cache nevű állományban is keres. Ez az állomány gyorsítótárként használatos a telepített megosztott programkönyvtárak kezelésének meggyorsítására.

A /etc/ld.so.cache állományt az ldconfig program készíti el. Az ldconfig program használatára a későbbiekbén még visszatérünk.

A legtöbb megosztott programkönyvtárat a dinamikus szerkesztő a /etc/ld.so.cache alapján találja meg és tölti be.

4. Ha a keresett megosztott programkönyvtárat a bemutatott helyeken nem találja a dinamikus szerkesztő, megkíséri azokat betölteni a /lib/ és a /usr/lib/ könyvtárakban.

Ez az utolsó lépés garantálja, hogy a legtöbb megosztott programkönyvtárat hiba esetén is megtalálja a dinamikus szerkesztő. Ha valamilyen oknál fogva például nem érhető el a /etc/ld.so.cache állomány, a dinamikus szerkesztő a legfontosabb programkönyvtárakat megtalálja a /lib/ könyvtárban, így a rendszer helyreállítható.

A bemutatott keresési sorrend szerint az ld.so dinamikus szerkesztő megkeresi az alkalmazás futtatásához szükséges megosztott programkönyvtárakat. Azt, hogy milyen megosztott programkönyvtárára van szükség az adott program futtatásához, a programot tároló bináris állomány tartalmazza. Ezt az információt lekérdezhetjük az ldd program segítségével, ahogyan ezt a következő példa is bemutatja.

97. példa Kérdezzük le, hogy milyen megosztott programkönyvtárakra van szükség a /bin/ls program futtatásához!

```
$ ldd /bin/ls
 linux-gate.so.1 => (0x0022d000)
 librt.so.1 => /lib/tls/librt.so.1 (0x00dea000)
```

```
libacl.so.1 => /lib/libacl.so.1 (0x0083b000)
libselinux.so.1 => /lib/libselinux.so.1 (0x003b4000)
libc.so.6 => /lib/tls/libc.so.6 (0x00436000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x00684000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x0041d000)
libattr.so.1 => /lib/libattr.so.1 (0x00835000)
$
```

A program kiírta, hogy milyen programkönyvtárra van szüksége az alkalmazásnak, és azt is, hogy ezek a programkönyvtárak hol, melyik állományban találhatók meg.

A programkönyvtárak neve

A megosztott programkönyvtárak neve igen fontos szerepet tölt be a kezelésük során. Amint láttuk, az alkalmazások név szerint kérik a megosztott programkönyvtárak betöltését, és a programkönyvtárak különféle változatainak kezelése is a név alapján történik.

A megosztott könyvtárak neve szabványos. Az ilyen állományok nevének első három betűje a lib – a *library* (könyvtár) szóból –, majd azt követi a könyvtár neve és az .so (*shared object*, megosztott objektum) betűcsoporthoz. Ezen betűcsoportok után pontokkal elválasztva következik a fő- és alváltozat-szám.

A megosztott könyvtárak változatszámozása kötött. Az azonos főváltozatszámu könyvtárak általában csereszabatosak, vagyis például az 5.2.11 változatú könyvtár helyett használhatjuk az 5.3.0 változatot, hiszen a programozói felület azonos. A megosztott programkönyvtárakat készítő programozók akkor adnak új főváltozatszámot, ha olyan nagymértékű a fejlődés, hogy már nem vállalható a csereszabatosság. Az 5.2.11 verzójú könyvtár helyett például már nem használható a 4.0.12.

A változatszámok kezelését támogatja a könyvtárakat tartalmazó állományok elnevezése. A programok betöltésénél az ld.so csak a főváltozatszámot keresi, számára mellékes a könyvtár alváltozata. A főváltozatszámon található a megfelelő alváltozatú programra mutató kötés, így a betöltés abból a változatból végezhető el, amelyik éppen a rendszeren van:

```
libresolv-2.1.3.so
libresolv.so.2 -> libresolv-2.1.3.so
```

A megosztott könyvtárak változatszámozása igen fontos szolgáltatás, a segítségével a rendszer a megosztott könyvtárak többfélé változatát is tartalmazhatja.

Tudnunk kell, hogy a megosztott könyvtárakra mutató közvetett hivatkozásokat az ldconfig automatikusan elkészíti.

9.2.4. A dinamikus könyvtárak nyilvántartása

Az ldconfig nevű program a rendszerre másolt megosztott könyvtárak nyilvántartásba vételét végzi, és általában akkor használja a rendszergazda, amikor új megosztott könyvtárakat másol a rendszerre.

Az ldconfig két fő feladatot végez el: egyrészt a megosztott könyvtárat tartalmazó alkönyvtárakban létrehozza a szükséges csatolásokat, másrészt a /etc/ld.so.cache nevű állományba kigyűjti, hogy milyen megosztott könyvtárak vannak a rendszeren. Az ldconfig által létrehozott közvetett hivatkozások a megosztott könyvtárak változatszám szerinti nyilvántartását teszik kezelhetőbbé, a /etc/ld.so.cache pedig a dinamikus szerkesztő munkáját teszi gyorsabbá.

A program mindenötöt a könyvtárakat végignézi, amelyeket a /etc/ld.so.conf állományban talál, amikor a /etc/ld.so.cache állományt felépíti, ha tehát egy új könyvtárba kívánunk megosztott könyvtárakat telepíteni, akkor ennek az új könyvtárnak a nevét be kell írnunk az /etc/ld.so.conf állományba és ki kell adnunk az ldconfig parancsot.

Az ldconfig képes arra, hogy az általa ismert megosztott könyvtárakat kiírja. Erre a -p kapcsoló szolgál. Igen jól használható ez a lehetőség hibakereséskor.

```
$ ldconfig -p | grep sql
libpsqlodbc.so.0.25 (libc6) => /usr/lib/libpsqlodbc.so.0.25
libpsqlodbc.so.0 (libc6) => /usr/lib/libpsqlodbc.so.0
libpsqlodbc.so (libc6) => /usr/lib/libpsqlodbc.so
libmysqlclient.so.8 (libc6) => /usr/lib/libmysqlclient.so.8
libmysqlclient.so (libc6) => /usr/lib/libmysqlclient.so
$
```

Nagyon fontos, hogy a dinamikus könyvtárakat alkotó állományokra mutató közvetett hivatkozásokat ne a szokásos módon az ln parancssal hozzuk létre, hanem az ldconfig parancssal. Az ln ugyanis maga is használja a dinamikus könyvtárakat, még az ldconfig nem használ egyet sem. Az ldconfig tehát akkor is működőképes, ha a dinamikus könyvtárak kezelése nem működik.



Az nm parancs segítségével kiíratjuk milyen szimbólumok (például függvény és változónevek) vannak egy megosztott könyvtárban, az objdump parancssal pedig igen sokféle információt olvashatunk ki az ilyen állományokból. A readelf programmal az ELF binárisokból olvashatunk ki információkat. Mindhárom program részletes kézikönyvoldallal rendelkezik, és igen sokféleképpen használható.

9.2.5. Dinamikusan betöltött programkönyvtárak

A dinamikusan betöltött programkönyvtárak (*dynamically loaded libraries*) olyan programszegmensek, amelyeket az alkalmazások futás közben töltenek be, ami-

kor azokra szükség van. Ezek a könyvtárak tehát nem töltődnek be a program indításakor.

A dinamikusan betöltött programkönyvtárak formátumukban nem különböznek a már megismert megosztott könyvtáraktól, különbség csak a betöltésükben fedezhető fel.

9.3. A Red Hat csomagkezelő rendszere

A programokat nem csak a forrásprogram letöltésével és fordításával telepíthetjük, hanem a már lefordított bináris állományok felmásolásával is. A különféle GNU/Linux terjesztések általában különféle programokat, úgynevezett csomagkezelőket használnak a programcsomagok lefordított bináris formában való telepítésére. A következő oldalakon az rpm (*Red Hat package manager*) csomagkezelőről olvashatunk, amely eredetileg a Red Hat GNU/Linux terjesztésekhez készült, és amelyet mára több terjesztés is használ[3].

9.3.1. A csomagok

Az rpm csomagkezelővel telepíthető programcsomagok tömörített állományok, amelyek mind a telepítendő állományokat, mind pedig a telepítéshez szükséges információkat tartalmazzák. Általában egy állományban egy teljes programcsomag állományait helyezzük el.

Az rpm számára készített programcsomag-állományok neve .rpm végződésű. Az ilyen állományok neve általában szabályos, a következő részekre tagolható:

programnév A programcsomagot tartalmazó állomány nevének az első - karakterig terjedő része a program neve.

Ezt a nevet a program szerzője adja a programnak, így a név nem az rpm csomag jellemzője, hanem a programcsomag általánosan használt elnevezése.

változat száma Az állománynév következő része a programcsomag változatszáma (*version number*), amelynek segítségével a program fejlesztői a változásokat nyilvántartják.

Ez a rész a program jellemzője, azaz nem az rpm állomány készítője határozza meg a változat számát.

csomagváltozat A következő rész általában a csomag változatszáma.

Ezt a részt a program változatszámától a - karakter választja el, és a . karakter zárja le. A csomag változatszámát az rpm csomag készítője határozza meg, általában azt jelölve, hogy az adott programból hányadik rpm csomagot készítí.

architektúra Ez a szakasz azt mutatja meg, hogy az rpm csomag milyen hardver-környezetben használható.

Az architektúrát az előtte és utána sorakozó szakaszoktól a . karakter választja el. A következő rövidítéseket szokás használni:

src A csomag forrásprogramot tartalmaz.

noarch A konkrét hardvertől független csomag. Az ilyen állományok minden számítógép-architektúrán használható programokat – például héjprogramokat, adatállományokat – tartalmaznak.

i386 Intel processzorcsaládon – annak 386-os vagy fejlettebb tagjain – használható programcsomag.

i586 Intel processzorcsaládon – 586-os vagy fejlettebb tagjain – használható programcsomag.

sparc A SPARC processzorcsaládon használható csomag.

sparc64 A SPARC processzorcsalád 64 bites változatain használható programcsomag.

alpha Az Alpha processzorcsaládon használható programcsomag.

végződés A névben található utolsó szakasz a .rpm végződés, amely jelzi, hogy az rpm csomagkezelő számára készült állományról van szó.

98. példa A következő két állomány az rpm csomagkezelő számára készült programcsomag-állomány:

```
$ ls
prosper-1.00.4-alt1.noarch.rpm
acl-2.2.7-5.i386.rpm
$
```

Figyeljük meg az állományok neveinek egyes részeit, amelyekből a legfontosabb információk kiolvashatók!

9.3.2. A csomagkezelő szolgáltatásai

Az rpm csomagkezelő fel van készítve a programcsomagokat alkotó állományok felmásolására. Amikor egy programcsomagot telepítünk, a csomagkezelő az állományokat a megfelelő könyvtárakba másolja.

Néhány állományt nem elég egyszerűen felmásolni. A beállítóállományokat néha a telepítés közben az adott körülményekhez kell igazítani. Az rpm csomagkezelő ezt úgy oldja meg, hogy a csomag készítője számára lehetővé teszi olyan héjprogram készítését, amely az állományok felmásolása után fut le. Ezek a programok általában nem veszik fel a kapcsolatot a felhasználóval; automatikusan futnak le és végzik el a szükséges beállításokat.

A csomagkezelő a telepített csomagokról és állományokról nyilvántartást vezet. A nyilvántartás bármikor lekérdezhető, az állományok a nyilvántartás alapján bármikor törlhetők a könyvtárszerkezetből. Az `rpm` csomagkezelő a nyilvántartásban tárolt adatok alapján lehetővé teszi a telepített programcsomagok ellenőrzését és eltávolítását, valamint a telepített programcsomagok új változattal való frissítését.

A csomagkezelő a programcsomagok között fennálló összefüggések kezelésére is képes. Az `rpm` csomagkezelő a telepített programcsomagok más csomagokkal fennálló kapcsolatait nyilvántartja, a telepítés és az eltávolítás során ellenőrzi. Ez gyakorlatilag azt jelenti, hogy a csomagkezelő alapértelmezés szerint nem engedi a programcsomagokat telepíteni vagy eltávolítani, ha a programcsomagok között fennálló függőségek ezt megkövetelik.

9.3.3. Csomagok lekérdezése

Az `rpm -q` kapcsolója a telepített programcsomagok lekérdezésére használható. Az `rpm` mérhetetlen számú egyéb kapcsolójával a programcsomagok adatai változatos formában lekérdezhetők.

A lekérdezés esetében meg kell adnunk, hogy melyik csomagról szeretnénk információt szerezni és azt, hogy milyen információra van szükségünk. Először vegyük sorra, miképpen határozhatjuk meg, hogy melyik csomagról szeretnénk információt szerezni:

programnév Ha a lekérdezés során megadjuk a program nevét, amellyel a program készítői a programot elnevezték, a telepített programcsomagok adatait kérdezhetjük le.

-f állománynév Ha a `-f` kapcsoló után megadjuk egy állomány nevét, lekérdezhetjük annak a programcsomagnak az adatait, amelyhez az adott állomány tartozik.

Ez igen hasznos lehetőség, hiszen a segítségével kinyomozhatjuk, hogy egy állomány miképpen került a számítógépünkre, és mi a feladata.

-a Az összes csomag lekérdezése. Ennek a kapcsolónak a segítségével az összes telepített programcsomagról kérhetünk egy lépésben adatokat, ami meglehetősen időigényes folyamat, de hasznos lehet.

-p állománynév A `-p` kapcsoló után megadhatjuk egy `rpm` csomagot tartalmazó állomány nevét, és így még nem telepített állomány adatait kérdezhetjük le.

--whatrequires programnév A kapcsoló segítségével lekérdezhetjük, hogy az adott programcsomagra mely más csomagoknak van szüksége a működéshez.

--whatprovides *szolgáltatás* A kapcsoló segítségével lekérdezhetjük, hogy melyik programcsomag biztosítja az adott szolgáltatást (például webkiszolgáló).

Még izgalmasabb az a kérdés, hogy mit kérdezhetünk le egy programcsomagról a csomagkezelő segítségével. Ha nem határozzuk meg, hogy mit szeretnénk tudni a programcsomagról, az rpm csomagkezelő a programcsomag nevét és teljes változatszámát írja ki, de a következő kapcsolókkal megadhatjuk, hogy mit szeretnénk tudni:

- i A kapcsoló hatására a csomagkezelő a csomagról tárolt legfontosabb információkat táblázatos formában összefoglalja és kiírja.
 - l A csomag állományainak listája, amelyeket a csomagkezelő a telepítés során felszámolt.
 - c A csomag állományai közül a beállítóállományok, amelyek a programcsomag viselkedésének módosítására használatosak.
 - d A csomag állományai közül a dokumentációt tartalmazó állományok, amelyekben a programcsomag leírása található.
- scripts A kapcsoló hatására a csomagkezelő kiírja azokat a héjprogramokat, amelyeket a csomag telepítése előtt, telepítése után, eltávolítása előtt és eltávolítása után végre kell hajtania.

--provides A kapcsoló segítségével lekérdezhetjük, hogy a programcsomag milyen programokat, programkönyvtárakat és szolgáltatásokat biztosít.

--requires A kapcsoló segítségével lekérdezhetjük, hogy a csomag telepítéséhez és működéséhez milyen más csomagok és programkönyvtárak telepítése szükséges.

--queryformat *formátum* A kapcsoló segítségével a csomagról nyilvántartott információkat a saját igényeinknek megfelelő formában írathatjuk ki. A formátumot leíró karakterláncban megadhatunk egyszerű szöveget – amely változatlan formában jelenik meg a kimeneten –, különleges jeleket – amelyek a printf() C nyelvű függvény dokumentációjában megadott módon viselkednek –, és megadhatunk mezőneveket.

A mezőnevek leírják, hogy milyen információk jelenjenek meg a kimeneten. Azt, hogy milyen mezőneveket használhatunk, az rpm a -querytags kapcsoló hatására kiírja a szabványos kimenetre. Mi csak a legfontosabb mezőneveket soroljuk fel a következő listában:

%{NAME} A programcsomag neve.

%{VERSION} A programcsomagban található program változatszáma.

%{RELEASE} Az rpm csomag változatszáma.

%{SUMMARY} A programcsomag egysoros leírása, amely roppant hasznos azok számára, akik nem tudják, hogy az adott programcsomagnak mi a feladata.

%{DESCRIPTION} A programcsomag néhány soros, részletesebb leírása.

%{BUILDTIME} A dátum, amely megmutatja, mikor készült az rpm csomag.

%{BUILDHOST} Annak a számítógépnek a neve, ahol az rpm csomagot készítették.

%{INSTALLTIME} A telepítés időpontja.

%{SIZE} A csomagot alkotó állományok összmérete telepítés után.

%{DISTRIBUTION} Megadja, hogy az adott rpm csomagot melyik GNU/Linux terjesztéshez készítették. A csomag telepíthető más terjesztésre is, de ekkor problémák léphetnek fel a telepítés és a használat során.

%{VENDOR} Az rpm csomagot szállító vállalat neve.

%{LICENSE} A program felhasználói engedélyének rövid neve.

%{PACKAGER} Az rpm csomagot készítő személy vagy vállalat neve.

%{GROUP} Megadja, hogy a programcsomag milyen kategóriába sorolható, milyen jellegű feladatok ellátására készült.

%{URL} A programcsomag honlapjának címe.

%{OS} Megadja, hogy az rpm csomag milyen operációs rendszerre készült.

%{ARCH} Az architektúra.

A következő oldalakon az rpm csomagok lekérdezésére láthatunk néhány példát.

99. példa Szeretnénk tudni, hogy kedvenc szövegszerkesztőnk melyik programcsomag részeként került fel a számítógépre, és szeretnénk megtudni, hogy milyen leírást adott a csomag készítője erről a programcsomagról.

A programcsomag nevét könnyedén lekérdezhetjük, ha a program egyik állományát ismerjük:

```
$ rpm -qf $(which vim)
vim-enhanced-6.2.457-1
$
```

Láthatjuk, hogy a vim parancs kiadásakor elinduló program melyik programcsomag részeként került fel a számítógépre. Most vizsgáljuk meg, mit írt a csomagról a csomag készítője:

```
$ rpm -qf $(which vim) --queryformat "%description\n"
VIM (Vi IMproved) is an updated and improved version of the vi
editor. Vi was the first real screen-based editor for UNIX, and
```

is still very popular. VIM improves on vi by adding new features: multiple windows, multi-level undo, block highlighting, and more. The vim-enhanced package contains a~version of VIM with extra, recently introduced features like Python and Perl interpreters.

Install the vim-enhanced package if you want to use a~version of the VIM editor which includes recently added enhancements like interpreters for the Python and Perl scripting languages. You will also need to install the vim-common package.

\$

100. példa Szeretnénk megtudni minden programcsomag nevét, amelyeknek nevében szerepel a „perl” részlet. Használjuk a következő módszert:

```
$ rpm -qa | grep perl
mod_perl-1.99_12-2.1
perl-HTML-Parser-3.35-5
perl-libxml-perl-0.07-29
perl-DBD-MySQL-2.9003-4
perl-5.8.3-18
$
```

Amint látjuk, az összes programcsomag nevét lekérdeztük, és kiszűrtük a listából a minket érdeklő részleteket a grep program segítségével.

101. példa Szeretnénk megtudni, hogy egy bizonyos rpm csomag, amelyet még nem telepítettünk, milyen programokat tartalmaz.

```
$ rpm -ql -p acl-2.2.7-5.i386.rpm | grep bin
/usr/bin/chacl
/usr/bin/getfacl
/usr/bin/setfacl
$
```

Amint látjuk, feltételeztük, hogy a csomagban szereplő bináris programállományok valamelyik bin nevű könyvtárba kerülnek a telepítés során, és ezt felhasználtuk az állományok listájának szűrésére.

9.3.4. Csomagok telepítése

Az rpm -i kapcsolójával programcsomagokat telepíthetünk. A telepítés során meg kell adnunk, hogy melyik állományt vagy állományokat szeretnénk telepíteni, és jónéhány kapcsoló segítségével befolyásolhatjuk a telepítés menetét.

A telepítés során az `rpm` alapértelmezés szerint ellenőrzi, hogy a csomag számára szükséges egyéb csomagok telepítve vannak-e és ha nem, a telepítést megtámadja. (Ezt az ellenőrzést természetesen ki lehet kapcsolni.) Fontos tudnunk, hogy ha egyszerre több csomagot telepítünk, a csomagkezelő a függőségeket együttesen ellenőrzi, és olyan sorrendben telepíti a csomagokat, hogy a függőségek minden teljesüljenek.

A telepítés során a telepítendő csomagokat megadhatjuk állománynevekkel és a hálózati elérhetőségükkel is. Ha azt akarjuk, hogy az `rpm` csomagkezelő letöltsse az állományt a telepítés előtt, az `ftp://` vagy a `http://` előtaggal kell megadnunk az állomány pontos helyét.

A telepítés során használható kapcsolók közül feltétlenül érdemes megismernedünk a következőkkel:

- v Részletes információk kiírása; a telepítés során üzenetek jelenniekn meg a telepítés folyamatáról.
- h A telepítés menetét a program # jelek kiírásával jelzi.
- force A csomagkezelő az alapértelmezett beállítások mellett nem írja felül a létező állományokat. Ha ezt a kapcsolót használjuk, a létező állományokat a program a csomagban található állományokkal felülírja.
- nodeps A csomagkezelő alapértelmezés szerint nem telepít olyan programcsomagokat, amelyeknek olyan csomagokra van szüksége, amelyek nincsenek telepítve. Ha a csomagkezelő ezt a kapcsolót kapja, a csomagok közti függőségeket nem veszi figyelembe.

A következő példa bemutatja, miképpen telepíthetünk programcsomagokat az `rpm` csomagkezelő segítségével.

102. példa Egy programcsomagot szeretnénk telepíteni az `rpm` csomagkezelő segítségével. A következő recept alapján könnyen megtehetjük ezt:

```
$ rpm -ivh acl-2.2.7-5.i386.rpm
Preparing... ##### [100%]
 1:acl      ##### [100%]
$
```

Figyeljük meg, hogy a telepítés folyamatát a használt kapcsolóknak köszönhetően kiírathattuk a képernyőre.

9.3.5. Csomagok eltávolítása

Ha egy programcsomagot el akarunk távolítani a rendszerről, akkor az `rpm -e` kapcsolóját kell használnunk. A kapcsoló hatására a csomagkezelő törli azokat az állományokat, amelyeket a program telepítésekor másolt fel. Ha a program

tartalmazott utasításokat arra nézve, hogy milyen utasításokat kell végrehajtani a csomag eltávolításakor, a csomagkezelő azokat is végrehajtja.

A csomagkezelő nem távolítja el azokat a beállítóállományokat, amelyek a csomag telepítése után módosultak. Ezeket az állományokat a csomagkezelő .rpmsave névvel gyűjti össze a csomag telepítésével.

A csomagkezelő a csomag eltávolítása előtt megvizsgálja, hogy nincsenek-e olyan csomagok, amelyeknek szükségük van az eltávolítandó csomagra. Ha valamelyik telepített csomag működésképtelenné válna az eltávolítandó csomag eltávolításával, a csomagkezelő alapértelmezés szerint megtagadja a csomag eltávolítását.

- v Részletes információk kiírása.
- nodeps A csomagfüggőségek vizsgálatának tiltása.
- test A tulajdonképpeni törlést nem végzi el, csak a vizsgálatokat. A -v kapcsoló segítségével kaphatunk részletes leírást a vizsgálatokról.

103. példa Az a gyanúnk, hogy soha az életünkben nem lesz szükségünk a docbook-utils nevű programcsomagra, ezért megkíséreljük eltávolítani:

```
$ rpm -e docbook-utils
error: Failed dependencies:
      docbook-utils = 0.6.14 is needed by (installed) docbook-uti
ls-pdf-0.6.14-2
      docbook-utils is needed by (installed) gtk-doc-1.2-1
$
```

Amint látjuk, néhány más programcsomagnak azonban szüksége van az eltávolítandó csomagra, ezért a csomagkezelő nem hajlandó eltávolítani azt. Próbáljuk meg törölni azokat a csomagokat is, amelyeknek szükségük van a docbook-utils csomagra!

```
$ rpm -e docbook-utils docbook-utils-pdf gtk-doc
$
```

Amint látjuk, a törlés sikeres volt, a csomagkezelő eltávolította mind a három csomagot. Ha szerencsések vagyunk, valóban nem lesz szükségünk az eltávolított programokra.

9.3.6. Csomagok ellenőrzése

Az rpm program segítségével könnyedén ellenőrizhetjük, hogy a csomagok telepítésekor felmásolt állományok nem változtak-e meg.

Az rpm a -V kapcsoló hatására beolvassa a kért programcsomag adatait a nyilvántartásából, és összehasonlítja az állományokat a nyilvántartott adatokkal. Ha az állományok módosultak a telepítés óta, a program figyelmeztetést küld.

Az ellenőrzés során ugyanúgy lehet az ellenőrzendő csomagokat megadni, ahogyan a lekérdezés során a lekérdezendő csomagokat.

Az állományok ellenőrzése során a csomagkezelő egy listát ír a szabványos kiemenetre, amelyben felsorol minden módosított állományt. A listában a következő oszlopok találhatók:

1. A program az első oszlopban azt írja le, hogy milyen jellegű változásokat eszlelt. Az itt megjelenő karakterek jelentése a következő:
 - 5 Az állomány MD5 ellenőrzőösszege megváltozott, tehát az állomány tartalma megváltozott.
 - S Az állomány mérete megváltozott.
 - T Az időbényeg megváltozott.
 - U A tulajdonos megváltozott.
 - G Az állomány tulajdonoscsoportja megváltozott.
 - M A jogok vagy az állomány típusa megváltozott.
2. A második oszlopban egy c betű jelenik meg, ha az adott sor beállítóállományra vonatkozik. (A beállítóállományok életének a változás természetes része.)

Ha az adott sor nem beállítóállományra vonatkozik, a második oszlop elmarad.
3. A harmadik oszlopban az állomány neve olvasható.

104. példa Ellenőrizzük, hogy a vim-common programcsomag valamelyik állománya nem változott-e meg!

```
$ rpm -V vim-common
S.5....T /usr/share/vim/vim56/syntax/tex.vim
$
```

Amint látjuk, az egyik állomány megváltozott, és ha szerencsénk van, akkor azonban felrémlik bennünk, hogy mi változtattuk meg az állományt, nem egy idegen behatoló.

9.3.7. Csomagok frissítése

Ha valamely programcsomagból újabb változat áll rendelkezésünkre, mint ami a rendszerre telepítve van, akkor ajánlatos lehet a csomag frissítése. Erre az `rpm -U` kapcsolója használható.

A csomag frissítésére a csomag telepítéséről szóló részben felsorolt megjegyzések és kapcsolók érvényesek.

105. példa *Szeretnénk frissíteni egy csomagot az Internetről letöltött újabb változattal. Használjuk a -U kapcsolót!*

```
$ rpm -U mswordview-0.5.14-bw6.i386.rpm  
$
```

Amint látjuk, a program problémamentesen frissítette a programcsomagot.

9.3.8. Kicsomagolás

Előfordulhat, hogy a programcsomagot ki szeretnénk csomagolni, de nem akarjuk telepíteni. Ha az állományokat ki szeretnénk nyerni a csomagból, az `rpm2cpio` programot használhatjuk, amely az `rpm` állományokból a `cpio` programmal kezelhető archívumot készít.

106. példa *Ki szeretnénk csomagolni az összes rpm csomagot, amely a munkakönyvtárban található. Használjuk a következő parancsot:*

```
$ for q in *.rpm; do echo $q; rpm2cpio $q | cpio --extract  
--make-directories; done  
$
```

A példában szereplő `cpio` program a kapcsolók hatására kicsomagolja a szabványos bemenetére érkező archívumot, és az állományokat bemásolja a munkakönyvtárban létrehozott alkönyvtárakba.

A példában a több állomány kicsomagolására ciklust hoztunk létre, amely az összes állománnyal végrehajtja a ciklusmagként szereplő parancsot.

9.3.9. Csomagok készítése

A következő oldalakon arról olvashatunk, miképpen lehet az `rpm` csomagkezelő számára programcsomagokat készíteni. A bemutatott módszerekkel és eszközökkel az általunk készített vagy az Internetről letöltött programokból és adatállományokból olyan `rpm` állományokat készíthetünk, amelyeket az `rpm` csomagkezelővel telepíthetünk.

Az `rpm` csomagok létrehozására a csomagban elhelyezendő program forrásán kívül az `rpm` csomagkezelőre és a csomagolás menetét leíró állományra van szükségünk. Ez utóbbi állományt valószínűleg nekünk kell megírnunk a megfelelő adatok összegyűjtésével, ha új `rpm` csomagot szeretnénk létrehozni.

A csomag elkészítése

A .src.rpm csomagok az adott program forrását és az rpm csomag előállításához szükséges leíróállományt tartalmazzák. Amikor az .src.rpm csomagot telepítjük, a csomagkezelő a forrást és a leíróállományt a megfelelő könyvtárakba másolja.

A programforrások helye Red Hat GNU/Linux terjesztések esetében a /usr/src/redhat/SOURCES/ könyvtár, a leíróállományok helye pedig a /usr/src/redhat/SPECS/ könyvtár. A leíróállományok neve általában a .spec végződést kapja.

Ha a programforrás és a leíróállomány megtalálható a megfelelő könyvtárakban, létrehozhatjuk az rpm csomagot. Erre a műveletre régebbi rpm csomagkezelő esetében az rpm program -ba kapcsolóját használhatjuk, újabb változatok esetében pedig az rpmbuild program -ba kapcsolóját. A következő példa ezt mutatja be.

107. példa Telepítettük a prosper nevű programcsomag .src.rpm változatát. A /usr/src/redhat/SPECS/ könyvtárban megtalálható a prosper.spec állomány. Állítsuk elő a programcsomag telepíthető változatát az állomány segítségével!

```
$ rpmbuild -ba prosper.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.47178
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ cd /usr/src/redhat/BUILD
+ rm -rf prosper
+ /usr/bin/gzip -dc /usr/src/redhat/SOURCES/prosper-1.00.4.tar.gz
+ tar -xvf -
```

Amint látjuk, a programcsomag előállítása megkezdődött. Jónéhány sornyi üzenet után – ha minden jól megy – azt is láthatjuk, hogy a csomag létrehozása befejeződött:

```
Requires(preun): /bin/sh
Requires: tetex-latex
Checking for unpackaged file(s): /usr/lib/rpm/check-files %buildroot
Wrote: /usr/src/redhat/SRPMS/prosper-1.00.4-alt1.src.rpm
Wrote: /usr/src/redhat/RPMS/noarch/prosper-1.00.4-alt1.noarch.rpm
$
```

A csomag létrehozásának végén láthatjuk, hogy az rpmbuild létrehozta a programcsomag telepíthető változatát és a forrás telepítésére használható src.rpm csomagot is.

Amint látjuk, a csomag létrehozása igazán egyszerű, csak ki kell adnunk egy egyszerű parancsot és a program lefutása után a megfelelő könyvtárakban megtaláljuk az elkészített programcsomagokat. Az rpm csomag létrehozásában egyedül a leíróállomány elkészítése az összetett. A leíróállomány létrehozását mutatják be a következő oldalak.

A csomagleíró állomány alapelemei

A csomagot leíró állomány első néhány sora a csomagról tartalmaz fontos információkat. A következő kifejezésekkel kell használnunk a csomag adatainak megadására:

Name: *név* A programcsomag neve. Ez a név lesz a csomagot tartalmazó állomány első része. Ennek a kifejezésnek mindenkorábban szerepelnie kell a csomag leíróállományban.

Summary: *leírás* A csomag tartalmának egysoros leírása. Ennek a kifejezésnek a leíróállományban mindenkorábban szerepelnie kell.

Version: *változatszám* A programcsomag változatának száma, ami általában megegyezik a program változatának számával, amelyből a csomagot készítettük. A programcsomag változatának száma az elkészített .rpm állomány nevében a csomagnév után fog szerepelni.

Ennek a kifejezésnek a leíróállományban mindenkorábban szerepelnie kell.

Release: *változatszám* Az elkészített .rpm csomag változatszáma, ami segít különbséget tenni az rpm csomagok között, amelyeket ugyanannak a programnak ugyanabból a változatából készítettünk. Az elkészített .rpm állomány nevében a változatot jelölik szám után a csomag változatának száma fog következni.

Ennek a kifejezésnek mindenkorábban szerepelnie kell a csomagleíró állományban.

License: *engedély* A csomag másolását, módosítását és felhasználását szabályozó felhasználói engedély neve. Itt általában valamilyen betűszóval, rövidítéssel utalunk az engedély közismert elnevezésére.

Ennek a kifejezésnek mindenkorábban szerepelnie kell a csomagleíró állományban.

Group: *csoport* Ennek a kifejezésnek a segítségével a programcsomagokat csoportokba rendezhetjük. A kifejezésben szereplő csoportot a csomagkezelő programok arra használhatják, hogy megkönnyítsék a csomagok közti eligazodást.

A Interneten ajánlásokat találhatunk arra nézve, hogy milyen csoportokba rendezzük a programcsomagjainkat. A szokásos csoportosítás alapcsoportjait mutatja be a 9.3. táblázat. A 9.4. táblázat az alkalmazások csoport csoportjait mutatja be.

Source: URL A kifejezés segítségével megadhatjuk a programcsomag előállítására használható forrásprogramot tartalmazó állomány pontos helyét az Interneten.

A kifejezésben található URL állománynév részének pontosan meg kell egyeznie a /usr/src/redhat/SOURCES/ könyvtárban található tömörített állomány nevével, amelyben a forrásállományokat tároljuk. A programcsomag előállításakor a forrásállományoknak a kifejezésben szereplő állománynevével elérhetőnek kell lenniük.

A kifejezésnek mindenkorban szerepelnie kell a csomagleíró állományban, de nem feltétlenül kell teljes internetcímet megadnunk. Ha nem tudjuk, hogy a forrásállomány pontosan honnan tölthető le, elegendő egyszerűen egy állománynevét beírnunk a Source: kulcsszó után.

Ha a programcsomag forrásállományai több tömörített állományból származnak, a kifejezést többször is megismételhetjük. Ilyen esetben a Source0, Source1 stb. kulcsszavakat használjuk.

Patch: URL A kifejezés segítségével megadhatjuk a programcsomag fordítása előtt érvényesítendő foltállományok pontos helyét az Interneten.

A kifejezésben található URL állománynév részének pontosan meg kell egyeznie a /usr/src/redhat/SOURCES/ könyvtárban található foltállomány nevével. Ha nem akarjuk foltállománnyal módosítani a forrásállományokat, ezt a kifejezést nem kell használnunk a csomagleíró állományban. Ha használjuk, akkor sem kell okvetlenül a teljes címet megadnunk a kifejezésben, elegendő, ha csak az állománynevét adjuk meg.

Ha a forrás módosítására több foltállományt is használunk, azokat sorra megadhatjuk a Patch0:, Patch1: stb. kulcsszavakkal.

BuildArch: típus A kifejezés segítségével megadhatjuk, hogy milyen architektúrájú programcsomagot akarunk létrehozni. Ha a kifejezés nem szerepel a csomagleíró állományban, az rpmbuild az alapbeállításainak megfelelő architektúrájú csomagot hozza létre.

A kifejezést a legtöbb esetben arra használjuk, hogy architektúrafüggetlen (noarch) csomagot hozzunk létre.

%description: Ez után a kulcsszó után többsoros, részletesebb leírást adhatunk a csomagról.

Ennek a kulcsszónak és az utána található néhány soros leírásnak mindenkorban szerepelnie kell a csomagleíró állományban.

Csoport	Leírás
Amusements	szórakoztatásra használt programok
Applications	alkalmazások
Desktop	ablakkezelők, munkafelületek
Development	fejlesztőeszközök
Documentation	dokumentáció
Libraries	programkönyvtárak
Office	irodai programcsomagok
Public Keys	nyilvános kulcsok a titkosításhoz
System Environment	rendszereszközök
User Interface	felhasználói felületek
Hardware Support	hardvereszköz-kezelők
X11	az X Window grafikus felület részei

9.3. táblázat. Az RPM csomagok alapcsoportjai

A csomagleíró állomány következő soraiban a csomag előállítását végző utasítások szerepelnek. A programcsomagot előállító rpmbuild program (régebbi rendszereken maga az rpm program) a csomagleíró állományban leírt utasítások alapján kicsomagolja a forrásprogramot, elvégzi a fordításhoz szükséges beállításokat, lefordítja a programot szimulált környezetben, telepíti a programot, és az eredményül kapott állományokat .rpm állományba csomagolja.

A csomagleíró állományban tehát azokat a parancsokat kell leírnunk, amelyek segítségével a programot „kézzel” kicsomagolnánk, beállítanánk, fordítanánk és így tovább. A csomagkezelő néhány makrót tesz elérhetővé számunkra a munka megkönnyítésére. A makrókat a következő lista tartalmazza:

%prep A csomag leírását tartalmazó első rész után a csomagleíró állományban általában a %prep makró és az utána írt parancsaink találhatók. A csomagleíró állomány ezen szakasza a fordításra való előkészítésre (*prepare*, előkészít) használatos.

A %prep szakaszban a megszokott állománykezelő programokon és héjparancsokon kívül használhatjuk a következő makrókat is:

%setup Ennek a makrónak a hatására a program kicsomagolja a forrásállományt tartalmazó tömörített állományt, és az így létrejött könyvtárba – a forrásállományok gyökérkönyvtárába – lép.

Az rpmbuild feltételezi, hogy a forrásprogramot tartalmazó állomány (Source: kifejezés) kicsomagolásakor létrejön egy könyvtár, amelynek neve a csomag nevét (Name: kifejezés), egy - karaktert és a program változatának számát (Version: kifejezés) tartalmazza.

A %prep makrónak a programokhoz hasonlóan paramétereit is adhatunk. A legfontosabb paraméterek a következők:

Csoport	Leírás
Applications/Archiving	archiválóprogramok
Applications/Communications	a kapcsolattartás eszközei
Applications/Databases	adatbázis-kezelő eszközök
Applications/Editors	szövegszerkesztők
Applications/Emulators	utánzó programok
Applications/Engineering	mérnöki tervezőprogramok
Applications/File	állománykezelő programok
Applications/Internet	internethasználati programok
Applications/Multimedia	multimédia-programok
Applications/Publishing	kiadványszerkesztő programok
Applications/System	rendszeralkalmazások
Applications/Text	szövegkezelő programok
Applications/Networking	hálózati segédprogramok

9.4. táblázat. Az RPM csomagok Applications csoportjai

- n könyvtárnev A könyvtár neve, amely a forrásállományokat tartalmazza. Ahogyan már láttuk, az alapértelmezett érték a program nevéből és a változatának számából áll. E könyvtárnak a forrásállományokat tartalmazó állomány kicsomagolásakor kell létrejönnie.
- b szám A megadott számú forrásarchívum (Source0:, Source1: stb.) kicsomagolása, mielőtt a forrásállományokat tartalmazó könyvtárba belépnénk. Ez a kapcsolót általában több állományból készült programcsomag esetén használjuk.
- a szám A megadott számú forrásállomány kicsomagolása, miután a program belépett a forrásállományokat tartalmazó állományba.
- T Ez a kapcsoló kikapcsolja az alapértelmezett viselkedést, ami az első számú forrásarchívum kicsomagolása és a keletkezett könyvtárba való belépés.
- D A forrásarchívum kicsomagolásakor keletkező könyvtár törlésének kikapcsolása.
Az rpmbuild a forrás kicsomagolása előtt a könyvtár nevének és változatszámának megfelelő nevű könyvtárat törli, ha létezne. Ez a kapcsoló tiltja a könyvtár törlését. Általában akkor használjuk, ha több %setup makró van a csomagleíró állományban.

`%patch` Ezzel a makróval a forrásállományok foltozását végezhetjük el.
A makrónak a következő paramétereit adhatjuk:

- szám A makrónak egy számot adva paraméterként megadhatjuk, hogy hányadik foltállományt akarjuk érvényesíteni.

- p szám Ezt a kapcsolót és a hozzá tartozó számot a makró átadja a patch programnak. (A programnak ezzel a kapcsolóval adhatjuk meg, hogy a forrásállományban található állománynevekből hány könyvtárnevet távolítsan el.) A patch program használatáról a 214. oldalon kezdődő 9.1.3. részben olvashatunk bővebben.
- P A makró alapértelmezés szerint az első foltállományt érvényesít a forrásállományokon. Ezt az alapértelmezett viselkedést tilthatjuk le a -P kapcsolóval, amelyre a második, harmadik stb. foltállomány érvényesítésekor van szükségünk.
- b végződés A kapcsoló hatására a folttal módosítandó forrásállományokat a program a kapcsolóval megadott névvégződéssel menti a foltozás előtt.

%build A csomagleíró állomány következő szakaszát a %build makróval jelezhetjük. Ez a szakasz a programcsomag lefordítására szolgál.

A makró hatására az rpmbuild belép a forrásállományokat tartalmazó könyvtárba (ha a makró használata előtt megváltoztattuk volna a munkakönyvtárat).

A %build makró után – éppen úgy, mint a %prep makró után – héjparancsokat és az rpmbuild makróit használhatjuk.

A %build makró után a következő makrót használhatjuk:

%configure A makró a programcsomag fordítás előtti beállítására használható. A makró hatására az rpmbuild végrehajtja a beállítóprogramot a ./configure parancs kiadásával.

Az rpmbuild a %configure makró paramétereit átadja a beállítóprogramnak.

%install A program telepítésére használt szakasz elejét a %install makró jelzi.

A makró hatására az rpmbuild belép a forrásállományokat tartalmazó könyvtárba (ha a makró használata előtt elhagytuk volna a könyvtárat).

A %install makró után a szokásos módon használhatjuk a héjparancsokat a programcsomag telepítésére.

%clean A makróval jelzett szakasz az átmeneti állományok törlésére szolgál.

A %clean makró után elhelyezett héjparancsokat az rpmbuild akkor hajtja végre, amikor a leíróállomány alapján már elkészítette az rpm csomagokat, ebben a szakaszban tehát már törölhetjük a csomag előállítása közben létrehozott átmeneti állományokat.

A forrásállományok tárolására használt könyvtár törlésére azonban nem érdemes héjparancsokat készítenünk. Az rpmbuild a -clean kapcsolója hatására a programcsomag létrehozása után automatikusan töri a forrásállományok tárolására használt könyvtárat, és annak teljes tartalmát.

A `%clean` szakasz használatára a későbbiekben még visszatérünk.

`%files` A programcsomagot leíró állomány egyik legfontosabb szakasza a programcsomag tartalmának, a benne megtalálható állományoknak a fel-sorolása. A `%files` makró után fel kell sorolnunk az összes állományt, amelyet el akarunk helyezni az elkészített `rpm` csomagban.

A `%files` makróval jelzett szakaszban állományokat, könyvtárakat sorolha-tunk fel, minden sorban egyet. A könyvtárbejegyzések nevében használhat-juk az állománynév-helyettesítő karaktereket.

A könyvtárbejegyzések neve előtt, a sor elején a következő makrókat hasz-nálhatjuk:

`%doc` Az `rpm` csomagban elhelyezendő dokumentációs állományokat jelez-hetjük ezzel a makróval.

Az ilyen állományokat a dokumentáció számára fenntartott könyvtár-ban – például `/usr/share/doc/` vagy `/usr/doc/` – a programcsomag nevéről, változatának számából és az `rpm` csomag változatának számá-ból képzett nevű alkönyvtárban helyezi el az `rpmbuild` program.

A legtöbb esetben a program felhasználását szabályozó felhasználói engedélyt, a program módosításainak listáját, a telepítés menetét leíró dokumentációt szokás elhelyezni az ilyen makróval jelzett állományok-ban. A programcsomaghoz tartozó „igazi” dokumentációt általában kü-lön `rpm` csomagban szokás elhelyezni.

`%config` Azoknak az állományoknak a neve előtt használhatjuk ezt a mak-rót, amelyek a csomag programjainak beállításait tartalmazzák.

Az ilyen állományokat az `rpm` csomagkezelő különlegesen kezeli. Alap-értelmezett beállítások mellett például a programcsomag eltávolítása-kor nem törölnek azok a beállítóállományok, amelyeket módosítot-tunk a telepítés óta.

`%dir` Ezzel a makróval jelezhetjük azoknak a könyvtáraknak a nevét, ame-lyek a programcsomaghoz tartoznak.

A makróval megjelölt könyvtárakat az `rpm` csomagkezelő megkísérli eltávolítani, amikor a csomagot eltávolítjuk.

Fontos tudnunk, hogy ha a `%files` szakaszban a `%dir` makró nélkül adunk meg egy könyvtárnevet, az `rpmbuild` a könyvtárban található összes állományt elhelyezi az `rpm` csomagban.

Már a bemutatott eszközök segítségével is létrehozhatunk egyszerűbb `rpm` cso-magokat. Ezt mutatja be a következő példa.

108. példa Készítsünk egyszerű `rpm` csomagot BASH program terjesztésére! A programcsomag neve legyen `script`, a változatának száma pedig 1.0!

Első lépésként készítsük el a programcsomag nevével és változatszámaival jelzett könyvtárat, és helyezzük el benne a BASH programot! A programcsomag egyetlen BASH programból áll:

```
$ tree
.
'-- script-1.0
    '-- script.sh

1 directory, 1 file
$
```

Készítsük el a programcsomagot tartalmazó tömörített állományt, és másoljuk a /usr/src/redhat/SOURCES/ könyvtárba:

```
$ tar cvzf script-1.0.tgz script-1.0/
script-1.0/
script-1.0/script.sh
$ cp script-1.0.tgz /usr/src/redhat/SOURCES/
$
```

Most már csak el kell készítenünk a programcsomagot leíró állományt. Helyezzük a következő sorokat a /usr/src/redhat/SPECS/script.spec állományba!

```
1 Name: script
2 Summary: Simple demonstration rpm package.
3 Version: 1.0
4 Release: 1
5 License: GPL
6 Group: Applications/File
7 Source: script-1.0.tgz
8 BuildArch: noarch
9
10 %description
11 This simple rpm package demonstrates how to make
12 packages creating a~specification file and processing
13 it with the rpmbuild program.
14
15 %prep
16 %setup
17
18 %install
19 cp script.sh /usr/bin/
20 chmod 755 /usr/bin/script.sh
21
22 %files
23 /usr/bin/script.sh
```

A programcsomag leíróállománya meglehetősen egyszerű, hiszen magát a programcsomagot nem kellett beállítani vagy lefordítani, és a programcsomag csak egyetlen állományból áll.

Figyeljük meg, hogy az állomány az alapadatokon kívül néhány parancsot tartalmaz. Az állomány 16. sorában található %setup makró a forrásprogramot kicsomagolja, és a kicsomagoláskor létrejött könyvtárba lép. A program telepítése így már egyszerű, csak be kell másolnunk az állományt a megfelelő könyvtárba (19. sor).

Figyeljük meg, hogy a 8. sorban a leíróállomány alapján létrehozott csomagot architektúrától függetlenre állítottuk a noarch kulcsszóval. Ez nyilvánvalóan helyes így, hiszen a BASH programok minden processzortípuson egyformán futthatók.

A programcsomag előállítása az rpmbuild -ba kapcsolójával a szokott módon történhet (néhány sort az egyszerűség kedvéért eltávolítottunk):

```
$ rpmbuild -ba script.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.82181
+ umask 022
+ cd /usr/src/redhat/BUILD
+ LANG=C
+ export LANG
+ unset DISPLAY
+ cd /usr/src/redhat/BUILD
+ rm -rf script-1.0
+ /usr/bin/gzip -dc /usr/src/redhat/SOURCES/script-1.0.tgz
+ tar -xvf -
drwxrwxr-x pipas/pipas 0 2004-07-29 16:06:21 script-1.0/
-rwxrwxr-x pipas/pipas 51 2004-07-29 15:28:14 script-1.0/script.sh
+ cd script-1.0
Executing(%install): /bin/sh -e /var/tmp/rpm-tmp.82181
+ cd /usr/src/redhat/BUILD
+ cd script-1.0
+ cp script.sh /usr/bin/
+ chmod 755 /usr/bin/script.sh
Processing files: script-1.0-1
Prefix) <= 4.0-1
Requires: /bin/bash
Wrote: /usr/src/redhat/SRPMS/script-1.0-1.src.rpm
Wrote: /usr/src/redhat/RPMS/noarch/script-1.0-1.noarch.rpm
```

A program kimenetén megjelenő üzenetekből viszonylag jól követhetők az események. Láthatjuk a forrásarchívum kicsomagolását, a BASH program felmásolását és az eredményként kapott rpm csomag elkészítését.

Figyeljük meg, hogy a program az rpm csomagok létrehozása előtt feltérképezte a csomag igényeit és megállapította, hogy a programcsomagunk telepítéséhez a BASH héjra van szükség!

Csomagkészítés szimulált telepítéssel

A 9.3.9. példában bemutatott csomagleíró állománynak van egy igen kellemetlen mellékhatása: a csomag létrehozása közben a csomagleíró állomány alapján az rpmbuild telepítí is a programot a számítógépre!

Ez a mellékhatás több okból is meglehetősen szerencsétlen. A csomag készítése közben az rpmbuild a telepített csomagok nyilvántartását nem módosítja (ami logikus, hiszen nem csomag telepítését végzi), így a csomag létrehozása után a csomagleíró állomány alapján felszámolt állományok miatt a telepített csomagok nyilvántartása hibás adatokat tartalmaz. Különösen kínos a mellékhatásként jelentkező telepítés, ha a csomag létrehozása valami miatt nem sikerül. Ekkor előfordulhat, hogy a telepített GNU/Linux rendszeren egy félretelepített csomagot találunk a meghiúsult kísérlet után, ami – különösen egy alapfontosságú programcsomag esetében – több mint kellemetlen.

A következőkben megvizsgáljuk, miképpen írhatunk olyan csomagleíró állományt, amelynek alapján a csomagkezelő a csomag létrehozása közben nem telepít a működő GNU/Linux rendszerre a programot, amelyből az rpm csomagot szeretnénk készíteni. Erre a feladatra a csomagleíró állományban a BuildRoot:kulcsszót használhatjuk. Mivel az Interneten fellelhető dokumentációk nagy része meglehetősen zavaros képet fest erről a fontos eszközről, lényeges, hogy pontosan megértsük, hogyan működik és miképpen használható.

BuildRoot: könyvtárneve A csomagot alkotó állományok helyének megadása.

A kifejezésben megadott könyvtárnevet az rpmbuild automatikusan beilleszti a %files szakaszban felsorolt könyvtárbejegyzések nevei elő, azaz a csomag létrehozásakor minden állományt ebben a könyvtárban keres.

Az rpmbuild létrehozza az \$RPM_BUILD_ROOT környezeti változót, és elhelyezi benne a BuildRoot: kulcsszó után megadott könyvtár nevét.

A BuildRoot: kifejezésben megadott könyvtárneve felülbírálható az rpmbuild -buildroot kapcsolójával.

Az \$RPM_BUILD_ROOT változó neve azt sugallja, hogy a csomag létrehozásakor használt könyvtárat hordozza. Felületes szemlélő azt gondolhatja, hogy ebbe a könyvtárba fogja a program kicsomagolni a forrásállományokat. Erről azonban szó sincs!



Az rpmbuild egy környezeti változó értékének megadására ad módot a csomagleíró állományban és a parancssorban, majd felhasználja a változó értékét a telepített állományok megkeresésekor. Ha megjegyezzük, hogy ilyen egyszerű eszközről van szó, akkor könnyen megérthetjük a csomagkészítés fejlettebb módját.

Ha olyan csomagleíró állományt akarunk készíteni, amely a csomag létrehozása közben a csomag állományait külön könyvtárban tárolja, és onnan a csomag létrehozása után eltávolítja, a következő lépésekkel kell elvégeznünk:

1. A csomagleíró állomány elején adjuk meg a BuildRoot: kulcsszóval az \$RPM_BUILD_ROOT környezeti változó értékét. Ne feledjük, hogy ezt az értéket a csomag létrehozásakor a parancssorban felül lehet bírálni.
2. A %install makró után, a csomag telepítésekor az állományokat ne a gyökérkönyvtárból induló, hanem az \$RPM_BUILD_ROOT könyvtárból induló néven másoljuk fel. (Ha tehát az állományt a /bin/ könyvtárban szeretnénk látni, tegyük az \$RPM_BUILD_ROOT/bin/ könyvtárba!)

A szakaszban ügyeljünk arra, hogy egyetlen könyvtár meglétét sem feltételezzük, hiszen amíg a /bin/ könyvtár létezik, addig az \$RPM_BUILD_ROOT/bin/ valóság nincs.

A későbbiekbén visszatérünk arra, miképpen lehet a programokat rávenni, hogy a csomag előállításakor az \$RPM_BUILD_ROOT könyvtárba települjenek, de az rpm csomag telepítése után ne lepődjene meg, ha másutt találják magukat.

3. A %clean szakaszban gondoskodunk arról, hogy a csomag létrehozása után az \$RPM_BUILD_ROOT ideiglenes könyvtár teljes tartalmával együtt töröljön.

Ezt azonban nem tehetjük az egyszerű megoldásnak tűnő rm -rf \$RPM_BUILD_ROOT parancs segítségével, mert ha a csomag létrehozása során a rendszergazda megadja a --buildroot / kapcsolót, a rendszergazdaként futó rpmbuild öngyilkosságba hajszolja a teljes rendszert!

Az Interneten fellelhető leírások legtöbbje azt javasolja, hogy vizsgáljuk meg az \$RPM_BUILD_ROOT értékét, és ha az nem a gyökérkönyvtár (az értéke nem /), csak akkor töröljük. Erre persze csak azt mondhatjuk, hogy --buildroot /, és kész a katasztrófa.

Felmerülhet persze a kérdés, hogy miért adná a rendszergazda a csomag létrehozásakor a --buildroot / paramétereket az rpmbuild programnak. A válasz egyszerű: azért, mert nem gondolja, hogy ez az rm -rf / parancsal egyenértékű a hanyagul elkészített csomagleíró állomány alapján!

4. A csomagleíró állományban a %files szakaszban található állomány és könyvtárnevekhez nem kell nyúlnunk, azok neve elé a rpmbuild automatikusan beilleszti az \$RPM_BUILD_ROOT környezeti változó értékét.

A fenti lépések elvégzésével elérhetjük, hogy az rpm csomag létrehozásakor a használt GNU/Linux rendszer programjai érintetlenek maradjanak, a program telepített változata megmaradjon, létfontosságú állományok ne változzanak, és ne sérüljenek meg. Ráadásul egy fontos szabályt is megfogalmazhatunk a mások által készített csomagleíró állományok használatával kapcsolatban:



Ha mások által készített csomagleíró állomány alapján készítünk rpm csomagot, lehetőleg kerüljük a rpmbuild program --buildroot kapcsolójának használatát. Ha elengedhetetlenül fontos, hogy a csomag létrehozá-

sakor használt ideiglenes könyvtárat megváltoztassuk – mert például nincs ele-gendő hely az alapértelmezés szerint használt állományrendszeren –, minden-képpen óvatosan járunk el, és semmiképpen ne adjunk meg olyan könyvtárat a kapcsoló után, amelynek tartalmáról nem szívesen mondanánk le.

A következő példa bemutatja, miképpen készíthetünk a rendszer összezavarása nélkül használható csomagleíró állományt.

109. példa Módosítsuk a 9.3.9. példa csomagleíró állományát úgy, hogy a cso-mag létrehozása közben a meglévő /usr/bin/script.sh ne módosuljon! Az új csomagleíró állomány a következő:

```
1 Name: script
2 Summary: Simple demonstration rpm package.
3 Version: 1.0
4 Release: 1
5 License: GPL
6 Group: Applications/File
7 Source: script-1.0.tgz
8 BuildArch: noarch
9 BuildRoot: /var/tmp/script-root
10
11 %description
12 This simple rpm package demonstrates how to make
13 packages creating a~specification file and processing
14 it with the rpmbuild program.
15
16 %prep
17 %setup
18
19 %install
20 mkdir -p $RPM_BUILD_ROOT/usr/bin/
21 cp script.sh $RPM_BUILD_ROOT/usr/bin/
22 chmod 755 $RPM_BUILD_ROOT/usr/bin/script.sh
23
24 %clean
25 if [ "$RPM_BUILD_ROOT" = "/var/tmp/script-root" ]
26 then
27   echo "Deleting $RPM_BUILD_ROOT..."
28   rm -rf $RPM_BUILD_ROOT
29 else
30   echo "The $RPM_BUILD_ROOT left back..."
31 fi
32
33 %files
34 /usr/bin/script.sh
```

A csomagleíró állományon néhány apró változtatást hajtottunk végre. A 9. sorban a BuildRoot: kulcsszó segítségével megadtuk a csomag szimulált telepítésére használt ideiglenes állomány alapértelmezés szerinti nevét.

A csomag felmásolását végző 20–22. sorokban az \$RPM_BUILD_ROOT környezeti változó értékét is figyelembe vettük. A 20. sorban gondoskodtunk a bin/ könyvtár létrehozásáról, hiszen a /bin/ könyvtárral ellentétben az \$RPM_BUILD_ROOT/bin/ nem valószínű, hogy létezik.

A 25–31. sorban az ideiglenes könyvtárat és teljes tartalmát töröljük, de csak akkor, ha a felhasználó a parancssorban nem változtatta meg az alapértelmezett értéket. Ha a felhasználó más ideiglenes könyvtárat jelölt ki, egy üzenetben figyelmeztetjük, hogy az ideiglenes könyvtárat nem töröltük.

Figyeljük meg, hogy a %files szakaszban nem hajtottunk végre módosításokat!

A példát tanulmányozva talán egy kicsit furcsának és körülményesnek tűnik a csomag összeállításakor végzett munka. A csomag tartalmaként használt BASH programot előbb egy tömörített állományban helyezzük el, majd kicsomagoljuk, aztán egy ideiglenes könyvtárba másoljuk, és végül egy rpm csomagba tömörítjük. Az eljárás azért ilyen összetett, mert a legtöbb esetben összetettebb programcsomagok kezelésére használjuk, azaz nem a módszer a bonyolult, hanem a példa volt egyszerű.

Felmerülhet még a kérdés, hogy miképpen vegyük rá az alkalmazást, amelyből az rpm csomagot szeretnénk csinálni, hogy a megfelelő helyre települjön. Mivel minden alkalmazás más, nem igazán lehet erre a kérdésre választ adni.

Szerencsére a legtöbb alkalmazás az automake programmal készült Makefile állományt használja a fordítás és a telepítés vezérlésére. Az automake segítségével készített Makefile esetében a következő kifejezéseket használhatjuk a csomagleíró állományban a csomag telepítésekor:

```
make prefix="$RPM_BUILD_ROOT" install Sok csomagleíró állományban  
használják ezt a kifejezést a telepítésre. Valójában nem ez a legszerencsesebb módszer.
```

```
%makeinstall Néhány rpm-változat fel van szerelve a %makeinstall makróval,  
amely a csomag telepítését végzi az $RPM_BUILD_ROOT környezeti változó  
figyelembevételével. Ez a makró szintén a Makefile állomány alapján végzi  
a telepítést.
```

```
make DESTDIR="$RPM_BUILD_ROOT" install Az automake által létrehozott  
Makefile fel van készítve az állományok telepítésének befolyásolására. Ha  
a bemutatott módon megadjuk a telepítés célkönyvtárát, a telepítés során  
ide kerülnek a könyvtárak és az állományok, egyébként semmi más nem  
változik.
```

Úgy tűnik, hogy ez a legjobb megoldás a telepítés célkönyvtárának kijelölésére.

Telepítéskor és eltávolításkor futtatandó programok

Nem minden programcsomag telepítésekor elegendő egyszerűen felmásolni a csomagot alkotó állományokat a megfelelő könyvtárakba. Előfordulhat, hogy a program telepítésekor létre kell hozni az alapbeállításokat valamilyen beállítóprogrammal, lehet, hogy létre kell hozni egy rendszerfelhasználót, vagy le kell futtatni egy már telepített programot az újonan telepített program nyilvántartásba vételéhez. Az ilyen feladatok elvégzésére is biztosít eszközt az `rpm` csomagleíró állomány.

A csomagleíró állományban elhelyezhetünk olyan héjparancsokat, amelyeket az `rpm` csomagkezelő a programcsomag telepítésekor, illetve eltávolításakor futtat. A hagyomány szerint ezek a parancsok automatikusan végzik a feladatukat, hogy a csomag telepítése teljesen automatikus legyen.

A következő makrókat a `%files` szakasz előtt szokás használni a telepítéskor végrehajtandó héjparancsok megadására:

```
%pre A héjparancsokat, amelyek ebben a szakaszban vannak elhelyezve, az rpm csomagkezelő az rpm csomag telepítésekor, a telepítés előtt hajtja végre.
```

```
%post A héjparancsokat, amelyek ebben a szakaszban vannak elhelyezve, az rpm csomagkezelő az rpm csomag telepítésekor, a telepítés után hajtja végre.
```

```
%preun Azokat a héjparancsokat, amelyeket a %preun makró által jelzett szakaszban helyezünk el, az rpm csomagkezelő az rpm csomag eltávolításakor, az eltávolítás előtt hajtja végre.
```

```
%postun Ezeket a héjparancsokat a csomagkezelő a csomag eltávolítása után hajtja végre.
```

110. példa Ha megosztott programkönyvtárat telepítünk, az `ldconfig` program segítségével nyilvántartásba kell vennünk az új állományokat. Ennek legegyszerűbb módját mutatja be a következő beállítóállomány-részlet:

```
1 %post -n librsvg2-gtk
2 /sbin/ldconfig
3
4 %postun -n librsvg2-gtk
5 /sbin/ldconfig
```

Amint látjuk, egyszerűen lefuttatjuk az `ldconfig` programot a csomag telepítése és eltávolítása után.

A csomagok közti függőségek

Az rpm csomagkezelő gondosan ügyel, hogy a csomagok közti függőségek mindenkorban érvényesüljenek. Ha a programcsomag telepítéséhez valamilyen eszközre van szükség, a csomagot nem engedi telepíteni, amíg az adott eszköz nem áll rendelkezésre. A védelem fordított irányban is működik. Ha valamelyik csomag olyan eszközt biztosít, ami szükséges egy másik csomag használatához, az eszközt biztosító csomagot nem engedi eltávolítani, amíg az összes azt használó csomag telepítve van.

A csomagok által biztosított és igényelt eszközök azonosításához az rpm csomagkezelő rendszer nevet rendel minden eszközhöz. A név általában – de nem mindenkorban – megegyezik a programcsomag nevével. A BASH program például a bash nevű csomagban található, amely biztosítja a bash eszközt. Az összes BASH programot tartalmazó rpm csomagnak szüksége van a bash eszközre, így egyik sem telepíthető, amíg a bash nincsen telepítve.

Nem minden csomag esetében ilyen egyszerű a biztosított eszköz elnevezése, ahogyan azt a következő példa bemutatja.

111. példa Kérdezzük le az rpm csomagkezelő segítségével, hogy a telepített csomagok közül melyik biztosítja a webserver és a libnss3.so nevű eszközöket!

```
$ rpm -q --whatprovides webserver  
httpd-2.0.50-2.1  
$ rpm -q --whatprovides libnss3.so  
mozilla-nss-1.6-8  
$
```

Amint láthatjuk, ezeket az eszközöket nem olyan programcsomagok biztosítják, amelyek neve megegyezik az eszköz nevével.

A példában használt webserver (webkiszolgáló) eszközt a httpd csomag adott változata biztosítja, de ez nem jelenti azt, hogy ez az eszköz csak a httpd csomag telepítésével használható. Nyilvánvaló, hogy többféle webkiszolgáló is létezik, többféle csomag telepítésével biztosíthatjuk a webkiszolgálói szolgáltatást. Azok a programcsomagok, amelyek működésükhez a webserver szolgáltatást igénylik, nem csak a httpd csomaggal együtt használhatók, hanem bármely csomaggal, amely webkiszolgálóként képes működni – azaz a webserver szolgáltatást nyújtja.

A példa másik csomagja egy megosztott programkönyvtárat biztosít. Nyilvánvaló, hogy egy programcsomag több megosztott programkönyvtárat is biztosíthat, egy csomag több szolgáltatást is nyújthat. A többi támogatást használó programcsomag használatának itt sem az az előfeltétele, hogy a támogatást nyújtó programcsomag telepíthető legyen, hanem az, hogy az adott programkönyvtár elérhető legyen.

Amikor programcsomagot készítünk a csomagleíró állomány elkészítésével, a legtöbb esetben nem kell azzal foglalkoznunk, hogy a csomagfüggőségeket beállítsuk. Az rpmbuild a csomag elkészítése közben futtatja az rpm rendszer részéktel telepített find-requires és find-provides programokat, hogy megállapítsa, milyen eszközökre van szüksége, és milyen új eszközöket biztosít az elkészítendő csomag. A legtöbb csomagfüggőség ezekkel az eszközökkel automatikusan kezelhető. Hálá a megosztott programkönyvtárak kifinomult kezelésének, a programkönyvtárak használatából eredő csomagfüggőségeket a rpm csomagkezelő szinte minden képes automatikusan kezelni.

Néha azonban előfordulhat, hogy finomítanunk kell a csomagok közti függőségek rendszerén. Ilyen esetben a csomagleíró állományban a következő kifejezéseket használhatjuk az automatikusan kezelt csomagfüggőségek módosítására:

AutoReqProv: no Ezzel a kifejezéssel a csomagleíró állomány első szakaszában tilthatjuk a csomagfüggőségek automatikus felderítését, azaz megakadályozhatjuk a find-requires és a find-provides programok futtatását.

Ilyen drasztikus lépésre meglehetősen ritkán van szükségünk; a legtöbb esetben elegendő, ha finomítjuk az automatikusan felderített csomagfüggőségeket.

Requires: eszköz1, eszköz2 A kifejezés segítségével a csomag számára szükséges eszközök körét bővíthetjük egy vagy több újabb eszközzel.

A kifejezés segítségével egy sorban több eszközöt is megadhatunk, de a kifejezés többször is szerepelhet a csomagleíró állományban, így szerencsésebb eszközönként egy-egy új sort létrehozni.

A kifejezésben szereplő eszközöket a következő formákban adhatjuk meg:

név A kifejezés előírja, hogy az adott nevű eszközre a csomagnak szüksége van.

név = változatszám A kifejezés szerint az eszköz adott változatszámú változatára van szükség.

név >= változatszám A kifejezés szerint az eszköz adott változatszámú vagy újabb változatára van szükség.

A gyakorlatban ezt a kifejezést használjuk legtöbbször. A legtöbb esetben az automatikusan felderített csomagfüggőségeket a változatszám szigorításával módosítjuk.

név > változatszám A kifejezés magától értetődő módon a változatszámánál újabb változat meglétét írja elő.

név <= változatszám A kifejezés az eszköz adott változatszámú vagy régebbi változatának használatát írja elő. Ez a kifejezés azt jelenti, hogy a programcsomag nem képes együttműködni a szolgáltatás újabb változataival.

név < változatszám A kifejezés magától értetődően a szolgáltatás adott változatszámnál régebbi változatait írja elő feltételként.

BuildRequires: *eszköz1, eszköz2* A kifejezés segítségével azt jelölhetjük, hogy a csomag létrehozásához, a program lefordításához milyen szolgáltatások szükségesek.

Ezekre a szolgáltatásokra nincs szükségünk a programcsomag telepítéséhez és használatához, csak a programcsomag létrehozásakor szükségesek.

A kifejezésben használható eszközök formája és jelentése gyakorlatilag meggyezik a **Requires:** kulcsszóval létrehozott kifejezésekkel bemutatott formával és jelentéssel.

Provides: *név1, név2* A kifejezés segítségével megadhatjuk, hogy a programcsomag milyen eszközöket biztosít.

A legtöbb esetben akkor használjuk ezt a kifejezést, ha egy bizonyos szolgáltatás több csomag segítségével is megvalósítható. Ilyen esetben kiválasztunk egy alkalmas, használaton kívüli nevet a szolgáltatás jelölésére, és elhelyezzük azt az összes programcsomagban, amely a szolgáltatást megvalósítja.

A Red Hat alapú rendszereken szokás a programcsomag támogatásával végzett programfordítás eszközeit külön rpm csomagokban elhelyezni. Ezeknek az rpm csomagoknak a neve általában a -devel (*development, fejlesztés*) kifejezésre végződik. A következő példa bemutatja, hogyan írhatjuk elő többek között az ilyen programcsomagok meglétét a csomag fordításához.

112. példa A következő állományrészletből megtudhatjuk miképpen helyezhetünk el a szolgáltatások meglétére vonatkozó megszorításokat a csomagleíró állományban.

```
1 Requires:      libart_lgpl >= 2.3.10
2 Requires:      libxml2 >= 2.4.7
3 Requires:      pango >= 1.2.0
4 BuildRequires: libart_lgpl-devel >= 2.3.10
5 BuildRequires: libxml2-devel >= 2.4.7
6 BuildRequires: pango-devel >= 1.2.0
```

Figyeljük meg, hogy az egyes programkönyvtárak meglétét a csomag telepítéséhez, a programkönyvtárak fejlesztőeszközeinek meglétét pedig a csomag fordításához írtuk elő!

9.4. A Debian csomagkezelő rendszere

A Debian csomagkezelése egyértelműen a legfejlettebb, legkifinomultabb az összes GNU/Linux terjesztés közül.

A Debian GNU/Linux terjesztőinek egyik legfontosabb alapelve a ki-próbált, stabilnak tekintett és a fejlesztés alatt álló, kísérleti Debian-változatok különválasztása. Egy időben minden három Debian-változat fejlesztése folyik, amelyek a következők:

stable A **stable** változat a végfelhasználók számára készített kidolgozott változat, amely bizonyítottan működőképes, megbízható.

A **stable** változat programcsomagjain változtatásokat kizárolag biztonsági okokból végeznek. Ez a változat késznek tekinthető.

testing A **testing** változat programcsomagai – és a köztük fennálló kapcsolatok – még nem estek át olyan alapos vizsgálatokon, mint a **stable** változat alkotóelemei, de a fejlesztés ebbe az irányba halad.

A **testing** változatból alapos ellenőrzés és széles körű kipróbálás után **stable** változat lesz, de ez éveket is igénybe vehet.

Sok végfelhasználó nem a **stable** változatot használja, hanem a **testing** változatot, mivel ez utóbbiban a programcsomagok újabb változatai találhatók meg. A **testing** változat tehát elég megbízható a végfelhasználók többsége számára.

unstable Az **unstable** változat a fejlesztés, a programcsomagok kipróbálásának helye. A Debian dokumentáció szerint az **unstable** változatot a fejlesztők és a veszélyeket kedvelő végfelhasználók használják.

Láthatjuk tehát, hogy végfelhasználóként a Debian **stable** vagy **testing** változatát érdemes használnunk. Nyilvánvaló, hogy ha központi feladatokat ellátó kiszolgálót telepítünk, elsősorban a **stable** változat használata javasolható, ha munkaállomást telepítünk, inkább a **testing** változatot használjuk.



*Azok számára, akik nem, vagy alig ismerik a GNU/Linux világát, és nem használtak még Debian terjesztést, mindenkiéppen érdemes megjegyezni, hogy a **testing** változatok megbízhatósága nem mérhető össze a kereskedelben kapható operációs rendszerek többségével. A kereskedelmi forgalomban kapható operációs rendszerek többsége Debian mércével mérve soha nem érte el a **testing** állapotot!*

A Debian terjesztések és a csomagkezelés kapcsán egy fontos apróságra fel kell hívnunk a figyelmet.

A legtöbb felhasználó a telepítőprogramot a telepített GNU/Linux terjesztés-sel azonosítja. A GNU/Linux terjesztés azonban nem azonos a használt telepítő-programmal, mert a telepítés után, a telepítőprogram futtatása nélkül átérhetünk a használt terjesztés másik változatára. A legtöbb rendszergazda a **stable** változat telepítésével kezdi a számítógép üzembe helyezését, és később, ha úgy dönt,

hogy a testing változat jobban megfelel az igényeinek, áttér arra. Ehhez egyszerűen a stable változatot alkotó programcsomagokat le kell cserálni a testing változat programcsomagjaira. Ezt megtehetjük anélkül, hogy a telepítőprogramot újrafuttatnánk.

Egyre több GNU/Linux terjesztés támogatja az újratelepítés nélküli teljes rendszerfrissítést, de a Debian volt az első, amely olyan fejlett csomagkezeléssel rendelkezett, hogy ez a módszer nem csak lehetséges, de elterjedt is lett.

A Debian terjesztésben használt csomagkezelés kétszintű. Az alacsonyabb szintet a `dpkg` program neve fémjelzi, a magasszintű csomagkezelést pedig az `apt-get` program nevével szokták azonosítani. A Debian rendszereket üzemeltető rendszergazdák munkájuk során szinte kizárolag az `apt-get` programot használják a csomagkezelésre, mert az automatikusan megkeresi és letölti a telepítendő csomagokat a számítógépre, nyilvántartja a programok tárolására használt CD-ROM lemezeket, és képes a terjesztés újabb változatára való átállásra. Ezért a `dpkg` csomagkezelőt csak felületesen – a hangsúlyt az elterjedtebben használt magasszintű csomagkezelőre fektetve – ismertetjük.

A következő oldalakon előbb a `dpkg` által megvalósított alacsonyszintű csomagkezelést mutatjuk be, majd a 271. oldalon kezdődő 9.5.2. szakaszban az `apt-get` által megvalósított magasszintű csomagkezelést. (Az `apt-get` csomagkezelőt külön tárgyaljuk, mivel ezt a programot nem csak a `dpkg` programmal együtt használhatjuk.)

9.4.1. A telepítőlemezek

Mielőtt a csomagkezelésről szólnánk, meg kell említenünk a Debian rendszerek telepítésére használható CD-ROM lemezek letöltését.

A jelenleg elterjedten használt Debian-változatok 6, illetve 12 CD-ROM lemezre másolhatók. A lemezek a világ sok kiszolgálójáról letölthetők, így mindenki kiválaszthatja a hozzá legközelebbi, leggyorsabb kiszolgálót a <http://www.debian.org> weblapokon fenntartott listából.

Mivel a világszerte elhelyezett Debian letöltőhelyek terhelése igen magas, a fejlesztők úgy döntötték, hogy a CD-ROM lemeztérképek letöltését csak az erőforrássokkal takarékosabban bánó `jigdo` programmal teszik lehetővé.

A `jigdo` igen elmés módon lehetővé teszi, hogy a számítógépünkön található CD-ROM lemeztérképeket frissítsük, azaz a már meglévő Debian telepítőlemezekből és a letöltött állományokból előállítsuk a telepítőlemezek újabb változatait.

A program induláskor egy `.jigdo` állomány nevét várja paraméterként. Ez az állomány az adott Debian telepítőlemez pontos leírása, amelyet a Debian letöltőhelyéről tölthetünk le. A program megvizsgálja, hogy a lemezen elhelyezendő állományok közül melyek vannak már letöltve, és csak azokat az állományokat tölti le, amelyeket okvetlenül szükséges. A program a letöltés közben a letöltött állományokat elhelyezi a CD-ROM lemeztérképben, amelyet a letöltés után CD-ROM lemezre írhatunk.

A Debian telepítőlemez letöltését mutatja be a következő példa.

113. példa Az egyik Debian letöltőhelyről letöltöttük a Woody kódnevű Debian-változat SPARC processzorcsaládra készült telepítőlemezeinek .jigdo állományát. Töltsük le a teljes CD-ROM lemeztérképet, amelyet a CD-ROM írására használhatunk!

A letöltéshez használjuk a jigdo-lite programot, ahogyan a következő sorok bemutatják!

```
$ jigdo-lite woody-sparc-5.jigdo
```

```
Jigsaw Download "lite"
Copyright (C) 2001-2004 | jigdo@
Richard Atterer | atterer.net
Loading settings from '/home/pipas/.jigdo-lite'
```

```
-----
Images offered by 'woody-sparc-5.jigdo':
1: 'Debian GNU/Linux 3.0 r2 "Woody" - Official sparc Binary-5
CD' (debian-30r2-sparc-binary-5.iso)
```

```
Further information about 'debian-30r2-sparc-binary-5.iso':
Generated on Tue, 02 Dec 2003 05:34:58 +0000
```

```
-----
If you already have a~previous version of the CD you are
downloading, jigdo can re-use files on the old CD that are also
present in the new image, and you do not need to download them
again. Mount the old CD ROM and enter the path it is mounted
under (e.g. '/mnt/cdrom').
```

Alternatively, just press enter if you want to start
downloading the remaining files.

Files to scan:

```
-----
The jigdo file refers to files stored on Debian mirrors. Please
choose a~Debian mirror as follows: Either enter a~complete URL
pointing to a~mirror (in the form
'ftp://ftp.debian.org/debian/'), or enter any regular
expression for searching through the list of mirrors: Try a~two
-letter country code such as 'de', or a~country name like
'United States', or a~server name like 'sunsite'.
Debian mirror [ftp://debian.inf.elte.hu/]: 
```

```
Downloading .template file
--10:44:40-- http://us.cddimage.debian.org/jigdo-area/3.0_r2/ji
gdo/sparc/woody-sparc-5.template
      => 'woody-sparc-5.template'
IP keresés us.cddimage.debian.org... 195.224.53.39
Csatlakozás us.cddimage.debian.org[195.224.53.39]:80-hoz...
```

kapcsolódva.

```
HTTP kérés elküldve, várom a választ... 200 OK
Hossz: 4,270,182 [application/binary]
```

```
10% [====>] 468,798 108.98K/s ETA 00:32
```

Amint látjuk, a *jigdo-lite* programnak paraméterként megadtuk annak a *.jigdo* állománynak a nevét, amelyet előzőleg a Debian letöltőhelyről mi magunk töltöttünk le.

A program indulás után előbb kiírt néhány információt, amelyet a *.jigdo* állományból olvasott ki, majd felajánlotta, hogy a már meglévő állományokat megvizsgálja. Mivel az adott CD-ROM lemeztérkép állományainak előző változata nem állt a rendelkezésünkre, egyszerűen lenyomtuk az **Enter** billentyűt, ezzel jelezve, hogy minden állományt le kell tölteni.

A program a következő lépésekben megkérdezte, hogy melyik Debian letöltőhelyről kívánjuk az állományokat letölteni. Mivel már előzőleg is használtuk a programot, elegendő volt újra lenyomni az **Enter** billentyűt. (A Debian letöltőhelyek listáját a <http://www.debian.org> oldalon találjuk.)

A program ezek után vad letöltésbe kezd, elemenként letölti a CD-ROM lemeztérkép darabjait, és összeilleszti őket egy egésszé. A letöltés végén a CD-ROM lemezre írható lemeztérkép a rendelkezésünkre áll.

A Debian CD-ROM lemeztérképek letöltése kapcsán meg kell említenünk, hogy általában nincs szükség az összes CD-ROM lemez elkészítésére! A Debian alaprendszer az első CD-ROM lemezről telepíthető, a további programcsomagok pedig már a hálózatról közvetlenül is telepíthetők.

Ha viszont hazára akarjuk vinni a Debian lemezeket, hogy az otthoni számítógéünkre is telepítsünk programokat, el kell döntenünk, hogy mennyi lemezt kell színünk. A legfontosabb programok megtalálhatók az első néhány lemezen, így lehetséges, hogy az utolsó lemezekre soha nem lesz szükségünk. Az apt-get programcsomagot ismertető 9.5.2. szakaszban részletesen szólunk az internetcsatlakozással nem rendelkező számítógépekre való telepítésről.

9.4.2. A programcsomagok lekérdezése

A Debian csomagkezelő rendszere nem csak a telepített, hanem a telepíthető csomagokról is nyilvántartást vezet, így azon csomagok adatait is lekérdezhetjük, amelyeket még nem telepítettünk.

A telepített csomagok legfontosabb adatainak lekérdezésére a *dpkg -s* kapcsolója használható. A kapcsoló hatására a program részletes információkat ír ki, ha az adott programcsomag telepítve van, illetve rövid összefoglalót, ha még nincs. A következő példa bemutatja a program működését egy elérhető, de nem telepített programcsomag esetében.

114. példa Kérdezzük le a zsh programcsomag legfontosabb adatait a dpkg program segítségével!

```
$ dpkg -s zsh
Package: zsh
Status: unknown ok not-installed
Priority: optional
Section: shells
$
```

A programcsomagról meglehetősen kevés információt írt ki a dpkg. Ez azért van így, mert ez a programcsomag nincs telepítve.

Kérdezzük most le hasonló módon egy telepített programcsomag adatait!

```
$ dpkg -s vim-gtk
Package: vim-gtk
Status: install ok installed
Priority: extra
Section: editors
Installed-Size: 1296
Maintainer: Wichert Akkerman <wakkerma@debian.org>
Source: vim
Version: 6.1.018-1
Replaces: vim-tiny, vim-perl, vim-python, vim-ruby, vim-tcl, vim-tty
Depends: vim (= 6.1.018-1), libc6 (>= 2.2.4-4), libglib1.2 (>=
1.2.0), libgpmg1 (>= 1.19.6-1), libgtk1.2 (>= 1.2.10-4), libncu
rses5 (>= 5.2.20020112a-1), xlibs (>> 4.1.0)
Suggests: ctags, cscope
Conflicts: vim-tiny, vim-perl, vim-python, vim-ruby, vim-tcl, vim-tty
Description: Vi IMproved - GTK version
Vim is an almost compatible version of the UNIX editor Vi.
Many new features have been added: multi level undo, syntax
highlighting, command line history, on-line help, filename
completion, block operations, folding, unicode support, etc..
This package contains a~version of vim compiled with support
for the GTK frontend.
$
```

Amint látjuk, erről a programcsomagról részletesebb információkat kaphattunk, mert ez a programcsomag már telepítve van.

A dpkg -l kapcsolójával listát készíthetünk a telepített és a rendelkezésre álló programcsomagokról. A lista elkészítését mutatja be a következő példa.

115. példa Készítsünk listát azokról a programcsomagokról, amelyek neve a vim- kifejezéssel kezdődik (a példa bizonyos sorainak végét levágunk)!

```
$ dpkg -l 'vim-*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err
||/ Name          Version       Description
+++-+-----+-----+-----+
ii  vim-gtk        6.1.018-1    Vi IMproved - GTK version
un  vim-perl       <none>        (no description available)
un  vim-python     <none>        (no description available)
un  vim-rt         <none>        (no description available)
un  vim-ruby       <none>        (no description available)
ii  vim-scripts    4-1          plugins for vim, adding bell
rc  vim-tcl        6.1.018-1    Vi IMproved, with tcl script
un  vim-tiny       <none>        (no description available)
un  vim-tty         <none>        (no description available)
$
```

Amint látjuk, az elkészített listában szerepel a programcsomagok állapota, neve, a változat száma és a programcsomag rövid leírása. Azon csomagok esetében, amelyek nincsenek telepítve, bizonyos információk hiányoznak.

Amint a példákból is látható, a dpkg néhány fontos információt tart nyilván a csomagok állapotáról. A program többek közt nyilvántartja a programcsomagok állapotát, ami a következők egyike lehet:

installed A csomag telepítve van.

Azokat a csomagokat, amelyeknek az állapota **installed**, a dpkg kicsomagolta, a megfelelő alapbeállításokkal ellátta, és az állományaikat a megfelelő könyvtárakba másolta.

half-installed A csomag telepítése megszakadt.

Ezeknek a csomagoknak a telepítése elkezdődött, de valamilyen okból megszakadt, így ezek a csomagok nem használhatók.

unpacked A csomag kicsomagolása megtörtént, de a telepítés még nem kezdődött el.

Ezeknek a csomagoknak a kicsomagolása problémamentesen megtörtént, de a telepítés következő lépése – a csomagok alapbeállítása – még nem kezdődött el.

half-configured A csomag alapbeállítása elkezdődött, de nem fejeződött be.

Ez az állapot azt jelzi, hogy a programcsomag alapbeállítása során valamilyen hiba lépett fel.

config-files A programcsomag nincs telepítve, a beállítóállományai azonban jelen vannak.

Ez az állapot azokra a csomagokra jellemző, amelyeket úgy távolítottunk el, hogy a beállítóállományokat megtartottuk. A Debian csomagkezelő rendszere ez az elmés szolgáltatása lehetővé teszi, hogy a programcsomagokat később újratelepítsük, és azok a régi beállításokat használják.

Ugyanakkor érdemes már most megemlíteni, hogy a programcsomag eltávolítása és újratelepítése – ami a kétsége besett rendszergazda helyreállítási kísérleteinek egyik igen jellemző szakasza – nem minden váltja be a hozzá fűzött reményeket.

A `dpkg -L` kapcsolójával lekérdezhetjük, hogy az adott programcsomag telepítésekor mely állományokat másolta fel a program, a `-S` kapcsolóval pedig meg tudhatjuk, melyik programcsomag részeként települt az adott állomány. Ennek a két kapcsolónak a segítségével felkutathatjuk, hogy miképpen épül fel a teljes rendszer a programcsomagokból.

9.4.3. Csomagok telepítése

A `dpkg -i` kapcsolója segítségével olyan programcsomagokat telepíthetünk, amelyek `.deb` állományokban állnak a rendelkezésünkre. A következő példa ezt a módszert mutatja be:

116. példa A munkakönyvtárunkban két Debian programcsomag található, amelyeket telepíteni szeretnénk. Használjuk a következő parancsot:

```
$ dpkg -i *.deb
Selecting previously deselected package mc.
(Reading database ... 51888 files and directories currently installed.)
Unpacking mc (from mc_4.5.55-1.2_sparc.deb) ...
Selecting previously deselected package mc-common.
Unpacking mc-common (from mc-common_4.5.55-1.2_sparc.deb) ...
Setting up mc-common (4.5.55-1.2) ...

Setting up mc (4.5.55-1.2) ...

Amint látjuk, a dpkg mindenkit csomagot telepítette.
```

A telepítés kapcsán meg kell jegyeznünk, hogy sok csomag esetében akkor is érdemes lehet az `apt-get` programot használni, ha a programcsomagokat már letöltöttük.

9.4.4. Csomagok eltávolítása

A `dpkg -r` kapcsolójával eltávolíthatjuk a telepített csomagokat. A program megtagadja a csomagok eltávolítását, ha a csomagok közti összefüggések a csomag

eltávolításával sérülnek. Az összefüggések figyelembe vételét kikapcsolhatjuk a `--force-depends` kapcsoló segítségével. Ezt mutatja be a következő példa.

117. példa *El szeretnénk távolítani a sendmail programcsomagot, de arra sok más programcsomagnak is szüksége van. Títsuk le a programcsomagok figyelembe vételét a --force-depends kapcsolóval!*

```
$ dpkg -r --force-depends sendmail
dpkg: sendmail: dependency problems, but removing anyway as you
request:
  at depends on mail-transport-agent; however:
    Package mail-transport-agent is not installed.
    Package sendmail which provides mail-transport-agent is to be
removed.
    Package exim which provides mail-transport-agent is not
installed.
  mailx depends on mail-transport-agent; however:
    Package mail-transport-agent is not installed.
    Package sendmail which provides mail-transport-agent is to be
removed.
    Package exim which provides mail-transport-agent is not
installed.
(Reading database ... 51935 files and directories currently
installed.)
Removing sendmail ...
Stopping Mail Transport Agent: sendmail.
Cleaning up the queues...done.
```

Figyeljük meg, a program minden részletesen kiírta a csomagok közti függőségeket, amelyekkel kapcsolatos problémákat, de a csomagot mégis eltávolította.

A csomag eltávolítása során a `dpkg` a futó `sendmail` programot leállította, a csomagot alkotó állományokat pedig törölte.

Nagyon fontos tudnunk, hogy a `dpkg` a csomagok eltávolításakor nem töri el a csomagok beállítóállományait ha külön nem utasítjuk erre. Ez persze nagyon hasznos, hiszen a beállítóállományok amúgy is kevés helyet foglalnak, és a csomag újratelepítésekor hasznosak lehetnek a már meglévő beállítások, de megzavarhatja a munkánkat, ha nem számítunk rá.

Ha a csomag állományainak eltávolításakor a beállítóállományokat is el akarjuk távolítani, a `dpkg -P` kapcsolóját kell használnunk.



Ha a `dpkg -r` kapcsolójával eltávolítunk egy csomagot, és aztán újratölöljük, a programcsomag az eredeti beállítóállományokkal működik tovább. Ha a beállítóállományokat is el akarjuk távolítani, a `dpkg -P` kapcsolóját használjuk a csomag eltávolítására.

A következő példa bemutatja a `dpkg -r (--remove)` és `-P (--purge)` kapcsolója közti különbséget.

118. példa Vizsgáljuk meg a különbséget a csomag kétféle eltávolítási módja között!

Először távolítsuk el az Apache webkiszolgálót a beállítóállományok megtartásával! Használjuk a `dpkg -r` kapcsolóját!

```
$ dpkg -r apache
(Reading database ... 51935 files and directories currently
installed.)
Removing apache ...
Stopping web server: apache.
dpkg - warning: while removing apache, directory '/var/log/apache'
not empty so not removed.
dpkg - warning: while removing apache, directory '/etc/apache'
not empty so not removed.
$
```

Figyeljük meg, hogy a programcsomag eltávolítása megtörtént ugyan, de a naplóállományok megmaradtak. A naplóállományokból akkor is kideríthetjük, mi történt a csomag használata közben, ha a csomagot már eltávolítottuk.

Ha most megvizsgáljuk a csomag állapotát, azt találjuk, hogy el van távolítva, de a beállítóállományok továbbra is jelen vannak.

```
$ dpkg -s apache | grep Status
Status: deinstall ok config-files
$
```

Most telepítük újra a programcsomagot a `dpkg` program segítségével, és figyelmesen vizsgáljuk meg, mi történik a telepítés közben!

```
$ dpkg -i apache_1.3.26-0woody3_sparc.deb
Selecting previously deselected package apache.
(Reading database ... 51907 files and directories currently
installed.)
Unpacking apache (from apache_1.3.26-0woody3_sparc.deb) ...
Setting up apache (1.3.26-0woody3) ...
Reloading apache modulesNo process in pidfile '/var/run/apache.
pid' found running; none killed.
$
```

Figyeljük meg, hogy a csomag telepítése közben semmilyen üzenet nem utal arra, hogy a csomag alapbeállítása megtörtént volna. Ez nyilvánvalóan azért van így, mert a programcsomag beállítóállományai a régi telepítésből megmaradtak, azokat nem kellett újra előállítani.

Most távolítsuk el a programcsomagot úgy, hogy a beállítóállományokat is törljük!

```
$ dpkg -P apache
(Reading database ... 51935 files and directories currently
installed.)
Removing apache ...
Stopping web server: apacheNo /usr/sbin/apache found running;
none killed.
dpkg - warning: while removing apache, directory '/var/log/apac
he' not empty so not removed.
Purging configuration files for apache ...
Website at /var/www has NOT been deleted.
$
```

Figyeljük meg, hogy a csomag eltávolítása során a dpkg a beállítóállományokat is törölte. Erre utal a program által kiírt utolsó előtti sor.

Ha most megvizsgáljuk a csomag állapotát, láthatjuk, hogy valaha telepítve volt, de a beállítóállományokkal együtt eltávolítottuk.

```
$ dpkg -s apache | grep Status
Status: purge ok not-installed
$
```

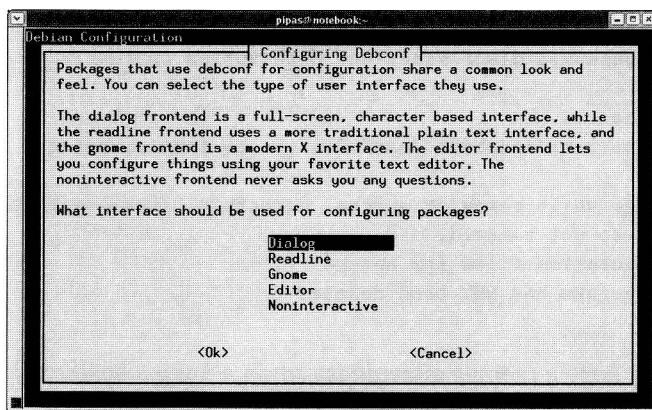
Ha most a programcsomagot újratelepítjük, a dpkg a kezdeti beállításokat újra elvégzi (a példából néhány sort eltávolítottunk):

```
dpkg -i apache_1.3.26-Woody3_sparc.deb
Selecting previously deselected package apache.
(Reading database ... 51904 files and directories currently
installed.)
Unpacking apache (from apache_1.3.26-Woody3_sparc.deb) ...
Setting up apache (1.3.26-Woody3) ...
Initializing apache config for immediate operation.
Installing new configuration file /etc/apache/httpd.conf ...
Installing new configuration file /etc/apache/access.conf ...
Installing new configuration file /etc/apache/srm.conf ...
The ServerAdmin is set to webmaster@sun.lkk.
The DocumentRoot is set to /var/www.
```

A program kimenetéből láthatjuk, hogy a programcsomagban található beállítóállományokat felmásolja, majd a beállításokat a programcsomagban található eszközök segítségével pontosítja.

9.4.5. A programcsomagok kezdeti beállítása

A legtöbb programcsomag alapbeállításai a telepítés során automatikusan megadhatók, néha azonban szükséges a rendszergazda véleményét is megkérdezni. A dpkg programcsomag ilyen esetben a debconf programcsomag segítségével tesszi fel a kérdéseket, amelyeket meg kell válaszolni a telepítéshez.



9.2. ábra. A debconf karakteres felülete

Ha a csomag telepítése után újra el szeretnénk végezni a programcsomag kezdeti beállítását, a `dpkg-reconfigure` program segítségével bármikor kérhetjük a programcsomagok alapbeállításának elvégzését. A programnak csak a beállítandó programcsomag nevét kell megadnunk.

Maga a debconf programcsomag is beállítható a `dpkg-reconfigure debconf` parancs segítségével. A debconf beállításával meghatározhatjuk, hogy ezentúl a csomagok beállítását karakteres felületen (9.2. ábra) vagy grafikus felületen (9.3. ábra) szeretnénk elvégezni.

A debconf automatikusan átter a karakteres felület használatára, ha valamilyen okból nem indítható el a grafikus beállítófelület.

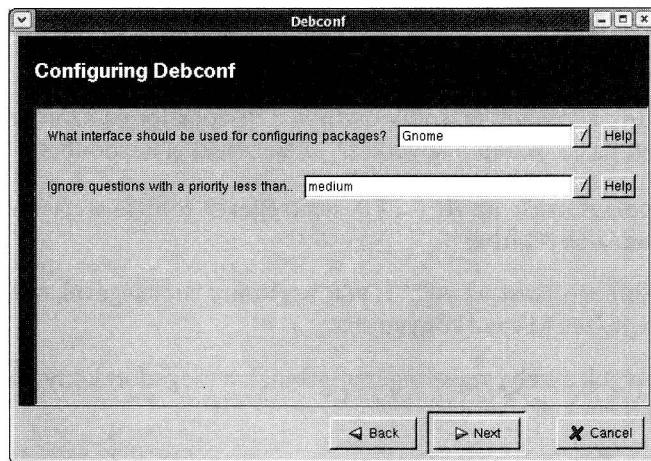
9.5. Programcsomagok automatikus letöltése

A csomagok kezelését igen kényelmesen és gyorsan végezhetjük az APT (*advanced package tool*, továbbfejlesztett csomagkezelő eszköz) segítségével. Az APT eszközökkel az Internetre kötött GNU/Linux egy pillanat alatt képes megkeresni és telepíteni a kérő programcsomagokat vagy frissíteni a teljes rendszert.

Ráadásul az APT rendszer programjai ma már képesek együttműködni a `dpkg` és az `rpm` csomagkezelő rendszerrel is. A program telepítésekor esetleg nem is kell tudnunk, hogy melyik csomagkezelő van telepítve a rendszerre, elég annyit tudnunk, hogy az APT programrendszer telepítve van!

9.5.1. A programcsomagok forrása

Az APT rendszer a `sources.list` állományban tartja nyilván a programcsomagok forrásait. Ebbe a szöveges beállítóállományban állíthatjuk be, hogy honnan szeretnénk letölteni a programokat.



9.3. ábra. A debconf Gnome felülete

A `sources.list` állomány több csomagforrás leírását is tartalmazhatja, az APT programrendszer pedig összegyűjti a programcsomagokat az egyes forrásokból. Lényeges, hogy a programcsomagok keresése a csomagforrások sorrendjében történik, az állományban az előbb található forrás előnyt élvez.

A `sources.list` állományban megjegyzésekkel szokásos módon, a # karakterrel helyezhetünk el.

A csomagforrásokat szóközökkel elválasztott oszlopok segítségével írjuk le, minden csomagforrásról megadva a következő adatokat:

1. Az első oszlop adja meg, hogy milyen csomagformátum esetén használjuk a forrást.
2. A második oszlop tartalmazza a csomagforrás pontos címét, azt, hogy a csomagokat honnan lehet letölteni.
3. A további oszlopok kapcsolók, amelyeket az APT az első oszlop alapján értelmez.

Az APT csomagforrások típusa, azaz a `sources.list` állomány első oszlopa a következő kulcsszavak egyike lehet:

`deb` A Debian rendszereken – és leszármazottain – használható csomagforrásokat a `deb` típussal jelöljük.

`deb-src` A `deb` csomagforrásokat használó rendszereken a `deb-src` csomagforrások a forrásprogramok letöltéséhez szükségesek.

`rpm` A Red Hat alapú rendszereken használt `rpm` csomagforrások típusa.

rpm -src Az rpm csomagforrásokat használó rendszereken az **rpm -src** csomagforrások a forrásprogramok letöltésére használatosak.

A **sources.list** állomány második oszlopa a kiszolgálót és azon belül a könyvtárat adja meg. Ez az oszlop a következő kifejezésekkel kezdődhet:

ftp:// A csomagforrást az APT FTP segítségével tölti le a **/var/cache/apt/archives/** könyvtárba.

http:// A csomagforrást az APT HTTP szabvány segítségével tölti le a **/var/cache/apt/archives/** könyvtárba.

file: A csomagok a helyi számítógépen vannak a megadott könyvtárban, azokat letölteni nem kell.

copy: Az ilyen források használata megegyezik a **file:** kulcsszóval megadott források használatával, azzal az apró különbséggel, hogy a **copy:** kulcsszó előírja az programcsomagokat tároló állományok felszerelését a megszokott könyvtárba (**/var/cache/apt/archives/**).

cdrom: Ezzel a kulcsszóval a CD-ROM lemezeken elhelyezett programcsomagokat kezelhetjük hatékonyan.

Ha az APT ilyen forrást használ, egy üzenetben kiírja, hogy melyik lemezre van szüksége, majd megvárja, amíg a felhasználó az üzenetre válaszolva jelzi, hogy a megfelelő lemezet behelyezte a meghajtóba. A **cdrom:** kulcsszó után **[]** jelek között megadhatjuk a CD-ROM lemez nevét.

A harmadik oszlop megadja, hogy a második oszlopan megadott könyvtáron belül melyik részt kívánjuk használni. Ez az oszlop Red Hat rendszerek esetében szinte bármilyen könyvtárnevet tartalmazhat, Debian csomagforrások esetén azonban kötött könyvtárszerkezetet használunk.

Debian csomagforrások esetén a harmadik oszlopan a használt Debian-változat fejlesztési szintjét (**stable**, **testing**, **unstable**) adjuk meg. A következő oszlopokban az adott szinten elérhető programcsoporthoz alkönyvtárai találhatók. Itt a következő kulcsszavakat szokás használni:

main A fő csoport, amely az alapprogramokat is tartalmazza.

contrib Kiegészítő programcsomagok gyűjteménye.

non-free Azok a programcsomagok tartoznak ebbe a csoportba, amelyek a szó szigorú értelmében nem szabad programok, de szabadon felhasználhatók.

A Debian fejlesztői meglehetősen szigorú értelemben használják a szabad program fogalmát, így meglehetősen sok elterjedten használt GNU/Linux alkalmazás tartozik ebbe a csoportba.

non-us Azok az alkalmazások tartoznak ebbe a csoportba, amelyek az Egyesült Államok területén belül nem használhatók.

119. példa Felmásoltuk a számítógépünkre a Debian Sarge kódnevű változatának telepítőlemezeit a /var/ftp/ könyvtárba, lemezenként külön-külön alkönyvtárat használva. Szeretnénk a könyvtárak tartalmát nyilvántartásba venni, hogy az APT rendszerrel telepíthessük az itt található programokat.

Vizsgáljuk meg a felmásolt könyvtárszerkezet debian/dists/ alkönyvtárának tartalmát! (Valójában a debian egy közvetett hivatkozás a .. könyvtárra, ezért a debian/dists ugyanaz, mint a dists.)

```
$ tree sarge-cdrom-1/debian/dists/
sarge-cdrom-1/debian/dists/
|-- frozen -> sarge
|-- sarge
|   |-- Release
|   |-- contrib
|       |-- binary-sparc
|           |-- Packages
|           |-- Packages.gz
|           '-- Release
|       '-- source
|           |-- binary-sparc
|               |-- Packages
|               |-- Packages.gz
|               '-- Release
|           '-- debian-installer
|               '-- binary-sparc
|                   |-- Packages
|                   '-- Packages.gz
|               '-- source
|-- stable -> sarge
|-- testing -> sarge
`-- unstable -> sarge

13 directories, 9 files
$
```

Amint a könyvtárszerkezetből is láthatjuk, a lemezek a Sarge kódnevű Debian-változatot tartalmazzák, de ugyanerre a tartalomra hivatkozhatunk a frozen, stable, testing és unstable szavakkal is.

Azt is láthatjuk a példából, hogy a lemez a main és a contrib csomaggyűjteményeket tartalmazza. Mivel az összes CD-ROM lemezen ezt találjuk, a /etc/apt/sources.list állományban a következő sorokat helyezzük el:

1	deb file:/var/ftp/sarge-cdrom-1 testing contrib main
2	deb file:/var/ftp/sarge-cdrom-2 testing contrib main

```

3 deb file:/var/ftp/sarge-cdrom-3 testing contrib main
4 deb file:/var/ftp/sarge-cdrom-4 testing contrib main
5 deb file:/var/ftp/sarge-cdrom-5 testing contrib main
6 deb file:/var/ftp/sarge-cdrom-6 testing contrib main
7 deb file:/var/ftp/sarge-cdrom-7 testing contrib main
8 deb file:/var/ftp/sarge-cdrom-8 testing contrib main
9 deb file:/var/ftp/sarge-cdrom-9 testing contrib main

```

A módosítás után futtatnunk kell az apt-get programot az update paraméterrel, hogy a nyilvántartott csomagok listáját frissítsük. (Erre a műveletre a későbbiekben visszatérünk.)

A csomagforrások kezelése

A Debian GNU/Linux terjesztések telepítésekor a telepítőprogram automatikusan kezeli a csomagforrások nyilvántartását.

Amikor az alaprendszert telepítjük, a telepítőprogram megkérdezi, hogy a csomagokat ezentúl szeretnénk-e letölteni az Internetről. Ha a számítógépünknek szélessávú internetcsatlakozása van, mindenkor érdemes megadni a csomagok letöltésére használható kiszolgálók nevét, hiszen a csomagfrissítések így végezhetők el a leg könnyebben.

A telepítőprogram megkérdezi azt is, hogy vannak-e a birtokunkban olyan CD-ROM lemezek, amelyekről a későbbiekben programcsomagokat kívánunk telepíteni. Ha ilyenek lemezeink vannak, azokat sorra be kell helyeznünk, hogy a telepítőprogram nyilvántartásba vehesse a rajta található programcsomagokat.

A programcsomagok forrását később mi magunk is megváltoztathatjuk. Az apt-cdrom program az add kapcsoló hatására megvárja, míg behelyezünk egy CD-ROM lemezt a meghajtóba, majd nyilvántartásba veszi a lemezen található programcsomagokat. A következő példa az apt-cdrom használatát mutatja be.

120. példa Vegyük nyilvántartásba a birtokunkba került Debian telepítő CD-ROM lemezt az apt-cdrom program segítségével! Használjuk az add paramétert!

```

$ apt-cdrom add
Using CD-ROM mount point /cdrom/
Unmounting CD-ROM
Please insert a~Disc in the drive and press enter 
Mounting CD-ROM
Identifying.. [3f63b0ba06025c15fc80534b9ffc4c2-2]
Scanning Disc for index files.. Found 4 package indexes and 0
source indexes.
This Disc is called:
'Debian GNU/Linux 3.0 r2 _Woody_ - Official sparc Binary-1
(20031202)'

```

```
Reading Package Indexes... Done
Wrote 988 records.
Writing new source list
Source List entries for this Disc are:
deb cdrom:[Debian GNU/Linux 3.0 r2 _Woody_ - Official sparc Bi
nary-1 (20031202)]/ unstable contrib main non-US/contrib non-U
S/main
Repeat this process for the rest of the CDs in your set.
$
```

Figyeljük meg, hogy a program kiírta, milyen sort helyezett el a sources.list állományban!

A netselect programcsomag részeként telepített netselect program segítségével programcsomagokat letölthetünk le az Internetről és telepíthetünk a számítógépünkre anélkül, hogy a programcsomag keresésére időt fecsérelnének.

9.5.2. Programcsomagok kezelése

Az APT rendszer részeként telepített apt-get program segítségével programcsomagokat tölthetünk le az Internetről és telepíthetünk a számítógépünkre anélkül, hogy a programcsomag keresésére időt fecsérelnének.

Az apt-get segítségével nem csak telepíteni, de frissíteni és törlni is egyszerűen lehet a csomagokat. A telepítés, frissítés és eltávolítás szerencsés esetben különösebb előtanulmányok nélkül elvégezhető.

Az apt-get számára mindenkorban meg kell adnunk egy paramétert, ami meghatározza, hogy milyen műveletet várunk a programtól. A következő kulcsszavakat használhatjuk a művelet előírására:

update A kulcsszó hatására a program frissíti az általa nyilvántartott programcsomagok listáját.

A program felveszi a kapcsolatot a megfelelő kiszolgálókkal, és letölti a rajtuk tárolt programcsomagok listáját. Nyilvánvaló, hogy az új csomagokat nem telepíthetjük az apt-get segítségével, amíg az update kulcsszóval nem szerünk róluk tudomást, így a listát rendszeresen frissítenünk kell.

upgrade A kulcsszó segítségével a telepített csomagokat frissíthetjük. Ha megadjuk valamelyik telepített csomag nevét, a program azt kíséri meg frissíteni, ha nem, a program megkísérel frissíteni minden csomagot.

A program a programcsomagok frissítése közben tekintettel van a csomagok közötti függőségekre.

Nyilvánvaló, hogy az upgrade használata előtt használnunk kell az update kulcsszót, hogy a programcsomagok listáját frissítsük.

dist-upgrade A kulcsszó hatása többé-kevésbé megegyezik az upgrade hatásával, vagyis programcsomagok frissítését végzi.

A dist-upgrade hatására a program a fontosabb programcsomagok függőségeit akár a kevésbé fontos csomagok kárára is megkíséri kielégíteni. Ez az üzemmód rendkívül hasznos, ha az adott GNU/Linux gyűjtemény újabb változatára akarunk áttérni, és gyakorlatilag az összes programcsomagot lecseréljük.

A dist-upgrade használatakor előbb átállítjuk a csomagok forrását a gyűjtemény új változatának címére, majd az update kulcsszó segítségével letölthetjük a csomagok listáját, és a dist-upgrade segítségével frissítjük a csomagokat.

install A kulcsszó segítségével programcsomagokat telepíthetünk. A kulcsszó után mindenkorban meg kell adnunk legalább egy programcsomag nevét, amelyet a program telepíteni fog.

A telepítés során a program figyelembe veszi a programcsomagok közti függőségeket, azaz ha a telepítendő csomag vagy csomagok más csomagok telepítését írják elő, a program azokat is telepíti.

remove A kulcsszó segítségével programcsomagot vagy programcsomagokat távolíthatunk el.

source A kapcsoló segítségével programcsomagok forrásprogramját tölthetjük le a számítógépunkre.

Ha a programnak a source kulcsszón kívül a --compile kapcsolót is megadjuk, a program nem csak letölti a forrásprogramot, hanem le is fordítja, és előállítja a telepíthető programcsomagot.

build-dep A kulcsszó segítségével frissíthetjük a programcsomagokat annak érdekében, hogy a forrásprogram által előírt függőségeket kielégítsük.

check A kulcsszó segítségével ellenőrizhetjük a nyilvántartásban szereplő – telepített és nem telepített – programcsomagok közti függőségeket, és megvizsgálhatjuk, hogy a függőségeket egyáltalán ki lehet-e elégíteni a kiszolgálókról letöltött programcsomagokkal.

clean A kulcsszó segítségével a letöltött programcsomagok tárolására használt területet szabadíthatjuk fel.

A apt-get a letöltött és telepített programcsomagokat eredeti formájukban is megtartja, amíg a clean kulcsszóval fel nem szabadítjuk a területet a törlésükkel.

autoclean A kulcsszó hatása nagyjából megegyezik a clean kulcsszó hatásával, azaz a letöltött programok telepíthető változatát törli.

Az autoclean azonban csak azokat a programcsomagokat törli, amelyek már nem érhetők el a kiszolgálókon, azaz amelyek már elavultnak számítanak.

A továbbiakban bemutatjuk, miképpen használhatjuk az apt-get programot programcsomagok telepítésére, törlésére, és letöltésére.

9.5.3. Programcsomagok telepítése

Az apt-get program az install paraméter hatására a paraméterként kapott programcsomagokat letölti, és telepíti a számítógépünkre. A program eközben gondosan ügyel a csomagok közti függőségekre, vagyis ha beleegyezünk, a telepített csomagok számára szükséges csomagokat is telepíti.

Ha a számítógépre telepítve van a bash_completion programcsomag is, a telepíthető programcsomagok listájának megfelelően egészíthetjük ki a parancsot a **Tab** billentyű segítségével. (Valójában a bash_completion programállomány ma már a BASH része.) A programok telepítését mutatja be a következő példa.

121. példa Ha telepítettük a BASH programozható parancskiegészítését (bash_completion), vagy más héjat használunk, amely támogatja a programozható kiegészítést, a telepítés parancsát a telepíthető programcsomagok nevével egészíthetjük ki:

```
$ apt-get install vimTabTab
vim          vim-common    vim-minimal
vim-color    vim-enhanced  vim-X11
$ apt-get install vim
```

Amint látjuk, a **Tab** billentyű kétszeri lenyomására a héj kiírta képernyőre, hogy a telepíthető csomagok közül melyek neve kezdődik a megfelelő karakterekkel.

Az apt-get az install paraméter hatására megkeresi a nyilvántartásában az adott programcsomagot, majd letölti a számítógép /var/cache/apt/archives/ könyvtárába az esetlegesen szükséges egyéb programcsomagokkal együtt. Ezek után a program kicsomagolja a programcsomag állományait, a megfelelő könyvtárakba másolja azokat, és elvégzi a programcsomag alapbeállítását.



Nyilvánvaló, hogy ezek közül a műveletek közül az apt-get program néhányat az alacsonyszintű csomagkezelő rendszerrel végeztet el. Az alacsonyszintű csomagkezelő szerepét Red Hat alapú rendszereken az rpm, Debian alapú rendszereken a dpkg program tölti be.

Programcsomagok telepítését mutatja be a következő példa.

122. példa Telepítsük az xqf programcsomagot az apt-get program segítségével!

```
$ apt-get install xqf
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  qstat
The following NEW packages will be installed:
  qstat xqf
0 packages upgraded, 2 newly installed, 0 to remove and 0 not
upgraded.
Need to get 0B/246kB of archives. After unpacking 762kB will
be used.
Do you want to continue? [Y/n] 
Selecting previously deselected package qstat.
(Reading database ... 52142 files and directories currently
installed.)
Unpacking qstat (from .../q/qstat/qstat_2.4e-1_sparc.deb) ...
Selecting previously deselected package xqf.
Unpacking xqf (from .../x/xqf/xqf_0.9.8-1_sparc.deb) ...
Setting up qstat (2.4e-1) ...

Setting up xqf (0.9.8-1) ...
$
```

Figyeljük meg, hogy a program megkérdezte, hogy akarjuk-e telepíteni a qstat programcsomagot is. A kérdésre alapértelmezés szerint adott válasz szerint a program automatikusan letöltötte és telepítette ezt a programcsomagot is a két programcsomag közti összefüggések alapján.

Ha a telepítéskor a csomag neve után egy / után a stable, testing vagy az unstable kulcsszót írjuk, előírhatjuk, hogy a program melyik változatát szereznénk használni.

A programok telepítésekor több programcsomag nevét is megadhatjuk egy paraméterként, ha a * karaktert használjuk.

9.5.4. Programcsomagok frissítése

A telepített programcsomagokat frissíthetjük az apt-get program segítségével. A frissítéshez az upgrade kulcsszót és a frissíteni kívánt programcsomag nevét kell megadnunk. Ha nem adunk meg frissítendő programcsomagot, az apt-get minden programcsomagot megkísérel frissíteni, amiből újabb változat áll a rendelkezésünkre.

Az apt-get képes arra is, hogy az elérhető programcsomagok listáját frissítse. Erre az update paraméter használható. Nyilvánvaló, hogy mielőtt frissítenénk valamelyik programcsomagot, az elérhető programcsomagok listáját kell frissíteni.

A programcsomagok frissítését mutatja be a következő példa, ahol alacsonyszintű csomagkezelőként az rpm programot használtuk.

123. példa Szeretnénk frissíteni a számítógépünkre telepített programcsomagok közül azokat, amelyekből van újabb változat. A munka megkezdéséhez előbb frissítünk a programcsomagok listáját (a példából néhány sort a rövidség érdekében eltávolítottunk):

```
$ apt-get update
Get:1 http://ayo.freshrpms.net fedora/linux/2/i386 release [181
5B]
Fetched 1815B in 0s (4223B/s)
Hit http://ayo.freshrpms.net fedora/linux/2/i386/core pkglist
Hit http://ayo.freshrpms.net fedora/linux/2/i386/core release
Get:1 http://ayo.freshrpms.net fedora/linux/2/i386/updates pkg
list [394kB]
Fetched 394kB in 23s (16,4kB/s)
Reading Package Lists... Done
Building Dependency Tree... Done
```

Most már frissíthetjük a programcsomagokat (ismét eltávolítottunk néhány sort):

```
$ apt-get upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be upgraded
  cdda2wav cdrecord finger gaim
4 upgraded, 0 newly installed, 0 removed and 0 not upgraded.
Need to get 4372kB of archives.
After unpacking 764kB of additional disk space will be used.
Do you want to continue? [Y/n] 
Get:1 http://ayo.freshrpms.net fedora/linux/2/i386/updates gaim
1:0.79-0.FC2 [31 62kB]
Get:2 http://ayo.freshrpms.net fedora/linux/2/i386/updates cdda
2wav 8:2.01-0.a27 .4.FC2.1 [150kB]
Get:3 http://ayo.freshrpms.net fedora/linux/2/i386/updates cdre
cord 8:2.01-0.a27 .4.FC2.1 [378kB]
Get:4 http://ayo.freshrpms.net fedora/linux/2/i386/updates fing
er 0.17-24 [18,8k B]
Fetched 4372kB in 4m7s (17,7kB/s)
Committing changes...
Preparing...          ##### [100%]
 1:gaim               ##### [  9%]
 2:cdda2wav           ##### [ 38%]
 3:cdrecord            ##### [ 87%]
 4:finger              ##### [100%]
Done.
```

Amint látjuk, a programcsomagok frissítése problémamentes volt.

9.5.5. Programcsomagok eltávolítása

Az apt-get segítségével eltávolíthatjuk a telepített programcsomagokat, ha a remove kulcsszót használjuk paraméterként.

A program a csomagok eltávolítása közben figyelembe veszi a programcsomagok köztő függőségeket. Ha az eltávolítandó programcsomagra más programcsomagoknak is szükségük van, felajánlja, hogy azokat is eltávolítja.

124. példa Távolítsuk az mc-common programcsomagot!

```
$ apt-get remove mc-common
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  mc mc-common
0 packages upgraded, 0 newly installed, 2 to remove and 0
not upgraded.
Need to get 0B of archives. After unpacking 5128kB will be
freed.
Do you want to continue? [Y/n]
```

Figyeljük meg, hogy az apt-get az mc programcsomag eltávolítását is felajánlotta, mivel annak használatához az mc-common programcsomagra is szükség van. Nincs mód arra, hogy az mc-common programcsomagot eltávolítsuk és az mc csomagot megtartsuk.

A Debian GNU/Linux terjesztést használva a csomagok eltávolítása során az apt-get megtartja a programok beállítóállományait. A programot a --purge kapcsolóval utasíthatjuk arra, hogy a beállítóállományokat is törölje.



Nem szabad összekevernünk a dpkg --purge kapcsolóját az apt-get --purge kapcsolójával, habár a hatásuk tulajdonképpen megegyezik.

A különbség csak annyi, hogy amíg a dpkg esetében a kapcsoló parancsot ad a programcsomag törlésére, addig az apt-get esetében csak módosítja a törlés menetét.

9.5.6. A terjesztés frissítése

Az APT rendszert használva meglepően egyszerűen és hatékonyan tudjuk frissíteni a használt GNU/Linux terjesztést. Mivel maga a GNU/Linux terjesztés nem más, mint a használt programcsomagok összessége, egyszerűen csak frissítenünk kell az összes általunk használt programcsomag változatát.

Ehhez első lépésként a csomagforrásokat kell módosítanunk, és be kell állítanunk, hogy az APT rendszer a terjesztés új változatát használja. Debian terjesztést használva a legtöbb esetben ez egyszerűen annyit jelent, hogy a /etc/apt/

`sources.list` állományban átírjuk az összes `stable` kulcsszót a `testing` kulcsszóra, hogy a szokott helyről az új változatot használjuk.

A következő lépésként az `apt-get` programot kell futtatnunk az `update` paraméterrel, hogy a nyilvántartott programcsomagok listáját frissítse. Ez néhány másodperc alatt meg is történik.

Az utolsó lépés maga a frissítés, amelyet az `apt-get` program `dist-upgrade` paraméterével kérhetünk. Valójában elvégezhető a frissítés az upgrade kulcsszóval is, de a `dist-upgrade` kifinomult függőségkezelő rendszere alkalmasabb a terjesztés frissítésére.

Mint látható, a terjesztés frissítése egyszerűen elvégezhető feladat, nem szabad azonban szem elől tévesztenünk, hogy milyen összetett és kényes műveletről van szó. A terjesztés frissítése közben több száz, nem ritkán több ezer programcsomagot frissítünk, ezért nem árt, ha figyelmesen elolvassuk a programcsomagok frissítése közben megjelenő üzeneteket. Az APT rendszer roppant hatékony, percek alatt romba lehet dönteni vele egy működő számítógépet...

9.6. Programcsomagok lekérdezése

Az APT rendszer részeként használható `apt-cache` program segítségével információkat kérhetünk a telepíthető és telepített programcsomagokról. A program nem módosítja a rendszer beállításait, nem telepít és nem távolít el programcsomagokat, csak információt szolgáltat. Tekintve a telepíthető programcsomagok széles körét, az `apt-cache` által szolgáltatott információk fontosságát nem lehet túlbecsülni.

Az `apt-cache` program indításakor paraméterként meg kell adnunk, hogy milyen műveletet kérünk. A legfontosabb kulcsszavak a következők:

`showpkg` Adatok kérése a parancssorban felsorolt csomagokról.

`stats` Statisztikai adatok kérése a nyilvántartott programcsomagokról.

`showssrc` Adatok kérése a parancssorban felsorolt programcsomagok forrásprogramjait tartalmazó programcsomagokról.

`dumpavail` Adatok kérése az összes nyilvántartott programcsomagról. Ez kissé hosszadalmas lehet, a program rengeteg sort ír a szabványos kimenetére.

`unmet` Adatok kérése a kielégíthetetlen függőségekről.

`show` Adatok kérése az adott programcsomagról.

`search` A parancssorban megadott kulcsszó keresése a programcsomagok nevében és leírásában.

9.6.1. Adatok kérése az összes csomagról

Az apt-cache program segítségével összesítő statisztikai adatokat kaphatunk a nyilvántartott (a letölthető és a telepített) programcsomagokról. Ehhez a programnak a stats kulcsszót kell paraméterként átadni.

125. példa Kérjünk adatokat a nyilvántartásban szereplő programcsomagokról!

```
$ apt-cache stats
Total Package Names : 7783 (311k)
  Normal Packages: 5681
  Pure Virtual Packages: 140
  Single Virtual Packages: 319
  Mixed Virtual Packages: 66
  Missing: 1577
Total Distinct Versions: 5788 (278k)
Total Dependencies: 30555 (856k)
Total Ver/File relations: 6127 (98.0k)
Total Provides Mappings: 1094 (21.9k)
Total Globbed Strings: 57 (533)
Total Dependency Version space: 138k
Total Slack space: 91.9k
Total Space Accounted for: 1657k
```

Amint látjuk a stats kulcsszó után egyéb paramétereket nem kell megadnunk.

A bemutatott példában szereplő egyes sorok jelentése a következő:

Total Package Names: A nyilvántartott programcsomag-nevek száma. Ez nem feltétlenül egyezik meg a telepíthető csomagok számával, létezhetnek a nyilvántartásban előfeltételként szereplő, de nem telepíthető – hiányzó – csomagok is.

A nyilvántartott csomagnevek számát jelző első sor után felsorolt adatok jelentése a következő:

Normal Packages: A valódi csomagok száma.

A valódi csomagok azok, amelyek valóban az adott néven elérhető programot tartalmazó állományra utalnak.

Pure Virtual Packages: A látszólagos csomagok száma.

Az ilyen csomagok általában szolgáltatásokra hivatkoznak, amelyeket többféle programcsomag segítségével megvalósíthatunk. A csomagok, amelyek előfeltételként az ilyen csomagokra hivatkoznak, akkor telepíthetők, ha valamelyik valódi csomag megvalósítja az adott szolgáltatást.

Single Virtual Packages: Az olyan látszólagos csomagok száma, amelyeket csak egyetlen valódi csomag valósít meg.

Mixed Virtual Packages: Az olyan csomagok száma, amelyeket látszólagos és valódi csomagként is telepíthetünk.

Missing: A hiányzó csomagok száma.

A hiányzó csomagok olyan csomagok, amelyekre más csomagoknak szükségük van, de nem elérhetők, nem telepíthetők. Ha a hiányzó csomagok száma nem 0, akkor vagy a csomagforrásal, vagy a rajta keresztül elérhető programgyűjteménnyel van probléma (például nem vettünk nyilvántartásba minden CD-ROM lemezt).

Total Distinct Versions: A rendelkezésre álló csomagváltozatok száma. Ez nagyobb is lehet, mint az első sorban olvasható programcsomag-nevek száma, hiszen egy programcsomag több változatban is elérhető lehet.

Total Dependencies: A csomagok közti függőségek száma.

Ha az apt-cache programnak a dump kulcsszót adjuk át paraméterként, a nyilvántartásban szereplő összes programcsomagról információkat ad a szabványos kimenetére írt listában. A listában szerepel a csomagok neve, a változatszáma, és az is, hogy milyen más csomagok szükségesek a telepítésükhez.

Az apt-cache a dumpvail paraméter hatására szintén olyan listát készít, amelyben minden elérhető csomag szerepel. Ebben a listában azonban részletes adatok szerepelnek minden csomagról, köztük a csomag néhány soros leírásával, ami segít eldöntenni, hogy szükségünk van-e a programcsomagra.

Az apt-cache az unmet kapcsoló hatására a szabványos kimenetre írja azokat a függőségeket, amelyeket a nyilvántartás alapján nem tud kielégíteni. A kapott listából kideríthetjük, hogy mely csomagok nem érhetők el, és azt is, hogy ez milyen más csomagok telepítését teszi lehetetlenné.

9.6.2. Programcsomagok keresése

Az apt-get segítségével csomagokat kereshetünk a nevükben és leírásukban szereplő szavak alapján. A keresett szót vagy keresett szavakat a search kulcsszó után kell megadnunk paraméterként. Ezt mutatja be a következő példa.

126. példa Keressük meg, hogy a nyilvántartásban szereplő programok közül melyiknek szerepel a nevében és a leírásában a bison szó!

```
$ apt-cache search "bison"
bison - A~parser generator that is compatible with YACC.
kimwitu++ - A~(syntax-)tree-handling tool (term processor)
btyacc - Backtracking parser generator based on byacc
jikespg - Jikes Parser Generator
$
```

Amint látjuk, a program négy programcsomag nevét és rövid leírását írta ki.

Az apt-get a depends kulcsszó hatására kiírja, hogy az adott programcsomag milyen más programcsomagtól függ, azaz milyen más programcsomagot kell telepítenünk a programcsomag telepítéséhez. Szintén kiírja a program, hogy mely csomagokkal nem fér össze a programcsomag, azaz milyen más programcsomagokat kell eltávolítanunk, ha az adott programcsomagot telepíteni akarjuk.

A program által kiírt listában az egyes sorok jelentése a következő:

Conflicts: Ezekkel a csomagokkal a lekérdezett csomag nem fér meg együtt, ezekkel a csomagokkal nem lehet egyszerre telepítve.

Depends: Ezektől a csomaguktól a lekérdezett csomag függ, a telepítéséhez ezeknek a csomagoknak is telepítve kell lenniük.

Recommends: Ezek a csomagok a lekérdezett csomagokkal együtt használhatók. A lekérdezett csomag használatához ezen csomagok telepítése ajánlott.

Replaces: Ezeket a csomagokat a lekérdezett csomag helyettesíti.

Suggests: Ezeknek a csomagoknak a telepítése szintén javasolt.

Ha a listában valamelyik csomag neve <> jelek között olvasható, látszólagos cso-magról van szó.

A program kimenetének értelmezését mutatja be a következő példa.

127. példa Mentsük egy állományba a sendmail programcsomag függőségeit, és vizsgáljuk meg annak részleteit!

```
$ apt-cache depends sendmail >lista.text
$
```

A program által készített állomány első néhány sora a következő:

1	sendmail
2	<i>Depends: adduser</i>
3	<i>Depends: m4</i>
4	<i>Depends: libc6</i>
5	<i>Depends: libdb3</i>
6	<i>Depends: libldap2</i>

Az első sorban a lekérdezett programcsomag neve található, utána pedig azoknak a csomagoknak a listáját olvashatjuk, amelyek szükségesek a programcsomag telepítéséhez.

Az állomány néhány további sora a következő:

1	<i>Suggests: sendmail-doc</i>
2	<i>Suggests: <mail-reader></i>
3	<i> mailx</i>

4	mutt
5	af
6	balsa

A sorok azoknak a csomagoknak a felsorolását tartalmazzák, amelyek telepítése javasolt a sendmail csomaghoz. Az első sor javasolja a programcsomaghoz tartozó dokumentáció telepítését is. A dokumentáció külön programcsomagban található, amelyet nem feltétlenül szükséges, de javasolt szintén telepíteni.

Az állományrészlet második sora szerint javasolt telepítenünk a smail-reader látszólagos csomagot. A sendmail levéltovábbító rendszer, nyilvánvaló, hogy ha használjuk, olyan programra is szükségünk lesz, amellyel a leveleket el tudjuk olvasni. A levélolvasásra sokféle programot használhatunk. A levélolvasást (mail-reader) megvalósító programcsomagok listáját az állományrészlet 3–6. sora tartalmazza, azaz az állományrészletből azt is megtudhatjuk, hogy a látszólagos csomag mely valós csomagok telepítésével valósítható meg. (Valójában ennél jóval több levélolvasó programot sorol fel az állomány, de ez csak egy részlet.)

Az állomány egy másik részlete az összeférhetetlen csomagok listáját adja meg:

1	Conflicts: <mail-transport-agent>
2	exim
3	nullmailer
4	courier-mta
5	masqmail
6	postfix

Amint látjuk, a sendmail nem fér össze a mail-transport-agent látszólagos csomagot megvalósító egyéb csomagokkal. Az állományrészlet 2–6. sora azokat a valódi csomagokat sorolja fel, amelyek megvalósítják a mail-transport-agent látszólagos csomagot.

Valójában a sendmail egy levéltovábbító rendszer, és mivel egy számítógépen egyszerre csak egy levéltovábbító rendszer használható, kölcsönösen összeférhetetlen az összes levéltovabbító rendszerrel.

9.6.3. Csomagok adatainak lekérdezése

Az apt-cache program segítségével a nyilvántartásban szereplő programcsomagokról lekérdezhetjük a legfontosabb adatokat. Nem csak azokról a csomagokról kaphatunk tehát információkat, amelyek már telepítve vannak, hanem azokról is, amelyek szerepelnek a nyilvántartásban, de még nincsenek telepítve.

Az apt-cache segítségével lekérdezhetjük az egyes programcsomagokról nyilvántartott legfontosabb adatokat, ha paraméterként a showpkg kulcsszót adjuk át. A showpkg kulcsszó után annak a programcsomagnak a nevét kell megadnunk, amelynek az adataira kíváncsiak vagyunk.

Részletesebb adatokat kaphatunk a nyilvántartott csomagokról, ha a show kulcsszót használjuk paraméterként. A kulcsszó után azoknak a csomagoknak a nevét kell megadnunk, amelyekről információt szeretnénk kapni.

128. példa Kérjünk információkat az electric programcsomagról az apt-get program show kulcsszavával!

```
$ apt-cache show electric
Package: electric
Priority: optional
Section: electronics
Installed-Size: 10092
Maintainer: Chris Ruffin <cmruffin@debian.org>
Architecture: sparc
Version: 6.05-1
Depends: lessstif1, libc6 (>= 2.2.4-4), xlibs (>> 4.1.0)
Filename: pool/main/e/electric/electric_6.05-1_sparc.deb
Size: 2783044
MD5sum: 2698b7e42c7ae6f633fdd6e2f751adc1
Description: electrical CAD system
Electric is a~sophisticated electrical CAD system that can
handle many forms of circuit design, including custom IC
layout (ASICs), schematic drawing, hardware description
language specifications, and electro-mechanical hybrid
layout.
```

Amint láthatjuk, az apt-cache kiírta a programcsomag legfontosabb adatait.

9.6.4. Telepítés grafikus felületen

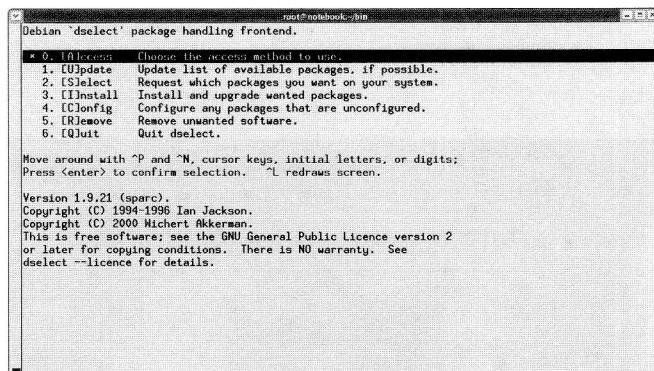
A Debian GNU/Linux telepítést használó számítógépeken a programcsomagok kezelésére sokan a dselect programot használják. A dselect segítségével a programcsomagok kiválasztása és telepítése menük segítségével végezhető el.

A dselect program indítása után a 9.4. ábrán látható főmenüt jeleníti meg. A menüpontok jelentése a következő:

Access Ezzel a menüponttal tudjuk beállítani, hogy a program honnan szerezze be a telepítendő programcsomagokat. Ha az APT rendszer telepítve van, a program az APT által kezelt csomagforrásokat is képes használni.

Update A menüpont segítségével frissíthetjük az elérhető programcsomagok listáját.

Select E menüpont segítségével egy részletes listából kiválaszthatjuk azokat a programcsomagokat, amelyeket telepíteni kívánunk. Szintén itt jelölhetjük ki a törlendő programcsomagokat.



9.4. ábra. A dselect telepítőprogram főmenüje

Install A menüpont segítségével telepíthetjük az előzőleg kiválasztott programcsomagokat.

Config A menüpont segítségével elvégezhetjük azoknak a programcsomagoknak az alapbeállítását, amelyeknél ez még nem történt meg.

Remove Ennek a menüpontnak a segítségével eltávolíthatjuk a programcsomagokat, amelyeket előzőleg eltávolításra jelöltünk ki.

Quit A programból való kilépésre szolgáló menüpont.

A kurzormozgató nyilakkal kiválaszthatjuk a kívánt pontot, majd az **[Enter]** billentyű lenyomásával aktiválhatjuk.

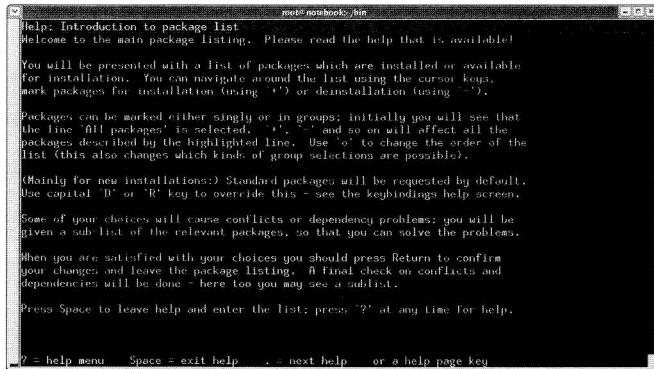
Ha programcsomagokat akarunk telepíteni, a főmenü **Select** menüpontját választjuk. Ekkor a 9.5. ábrán látható bejelentkező képernyő jelenik meg. Innen továbblépni a szóköz billentyű lenyomásával lehet. Ekkor a 9.6. ábrán látható lista jelenik meg.

A **dselect** listáképernyőjének felső részén a programcsomagok listája, alsó részén pedig a kiválasztott programcsomag rövid leírása található. A felső és alsó részt elválasztó vonalon a kiválasztott csomag állapotáról találhatunk fontos információkat. A képernyőn a 9.5. táblázatban található billentyűkombinációk segítségével adhatjuk ki utasításainkat.

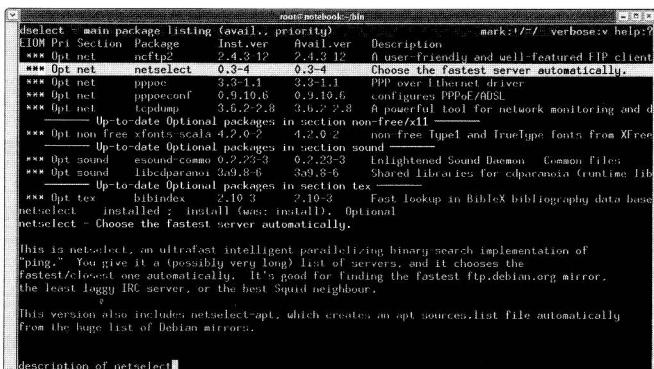
Ha a listából a programcsomagokat kiválasztottuk, és beállítottuk, hogy telepíteni vagy eltávolítani akarjuk őket, a **[Q]** (nagybetű) billentyű lenyomásával kiléphetünk a listából.

Ha a listából kiléptünk, a főmenüben kiválaszthatjuk az **Install** vagy a **Remove** menüpontot, hogy az előzőleg kiválasztott csomagokat telepítsük vagy eltávolítsuk.

Debian és Red Hat GNU/Linux rendszereken egyaránt használható a **synaptic** csomagkezelő program, amellyel az APT rendszer grafikus felületen keresztül használható.



9.5. ábra. A dselect telepítőprogram bejelentkező képernyője



9.6. ábra. A dselect telepítőprogram listaképernyője

<i>Billentyű</i>	<i>Jelentés</i>
J	mozgás lefelé
p	mozgás felfelé
t	ugrás a lista elejére
e	ugrás a lista végére
d	a kép alsó részének lapozása lefelé
u	a kép alsó részének lapozása felfelé
Ctrl + f	mozgás jobbra
Ctrl + b	mozgás balra
Ins	kijelölés telepítésre vagy frissítésre
Del	kijelölés törlésre
/	keresés
X	kilépés mentés nélkül
Q	kilépés a változtatások mentésével

9.5. táblázat. A dselect billentyűkombinációi

A synaptic program indítása után beolvassa az elérhető programcsomagok adatait, majd a 9.7. ábrán látható képernyő jelenik meg. (A 9.7. ábrán látható program a synaptic rpm csomagkezelőn alapuló változata. Más változatok induló képernyője ettől eltérő lehet.)

A főablak felső részén a szokásos menüsor, alatta pedig az eszköztár található. Az eszköztár alatt egy szűrő találunk, amellyel beállíthatjuk, hogy mely programcsomagokat láthatjuk a szűrő alatt található listában.

A szűrő alatt található listában programcsomagok adatait láthatjuk faszerkezetbe rendezve. A lista egyes elemeit kiválaszthatjuk, és az egér jobb gombjával egy menüt is megjeleníthetünk az egyes listaelemek kezelésére.

A programcsomagokat tartalmazó lista alatt bal oldalon a kijelölt programcsomag fontosabb adatait olvashatjuk, jobb oldalon pedig néhány műveletet kezdeményezhetünk ugyanezen a programcsomagon.

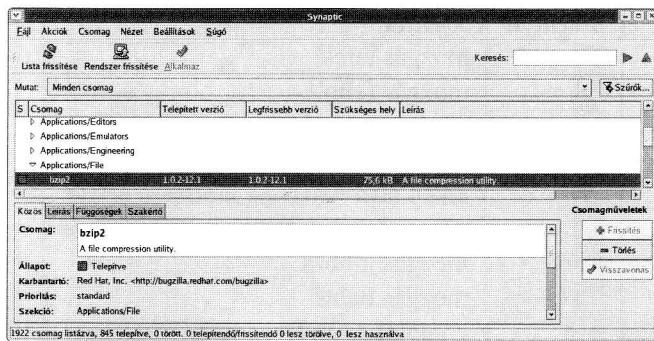
A képernyő főbb elemeinek szerepe és használata a következő:

Lista frissítése Ezzel a nyomógombbal az elérhető csomagok listáját frissíthetjük.

A nyomógomb hatása tehát megegyezik az apt-get update parancs hatásával.

Rendszer frissítése Ez a nyomógomb frissítésre jelöli ki az összes programcsomagot, amelynek újabb változata is a rendelkezésünkre áll.

A nyomógomb megnyomása után feltűnő beállítóablakban megadhatjuk, hogy a későbbi frissítéskor az apt-get upgrade vagy dist-upgrade kulcsszavának megfelelően történjen-e a frissítés. (A két frissítési mód közti különbségről a 271. oldalon olvashatunk részletesebben.)



9.7. ábra. A synaptic grafikus telepítőprogram

Alkalmaz A nyomógomb lenyomása után megkezdődik a telepítendő csomagok telepítése, az eltávolítandó csomagok eltávolítása. Mindazok a beállítások, amelyeket a képernyő közepén található programcsomagokat tartalmazó listán módosítottunk, ezzel a nyomógombbal érvényesíthetők.

Mutat Az itt található listából kiválaszthatjuk, hogy milyen csomagokat szeretnék látni a csomagok listájában.

A **Mutat** lenyiló lista valójában a csomagokra érvényesíthető szűrőket sorolja fel. Új szűrőket a lenyiló lista mellett található **Szűrők...** nyomógombbal hozhatunk létre.

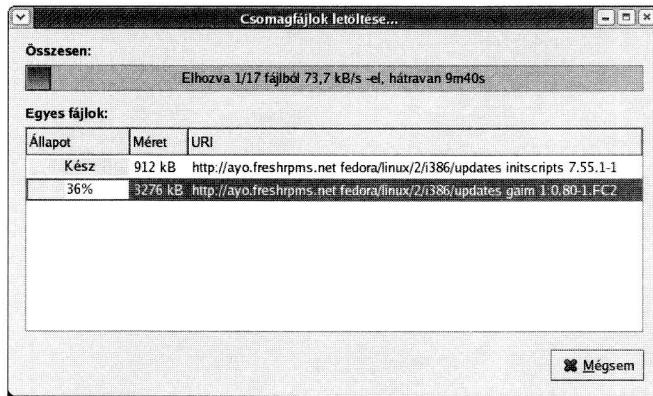
Csomagok listája Az indulóképernyőn található nagyméretű lista a nyilvántartásban szereplő programcsomagokat sorolja fel. Ha egy programcsomagot kiválasztunk a listából, az ablak más területein láthatjuk annak fontosabb adatait.

A programcsomagokat tartalmazó lista elemeire az egér jobb gombjával kattintva egy menüt kapunk, amelyből kiválaszthatjuk, hogy mit szeretnénk a csomaggal csinálni. A legfontosabb műveletek lehetővé teszik a csomag telepítését (ha az nincs telepítve) és eltávolítását (ha már telepítve van). Ha egy vagy több csomagot ilyen módon telepítésre vagy eltávolításra jelöltünk ki, a képernyő felső részén található **Alkalmaz** nyomógombbal indíthatjuk a tényleges munkát.

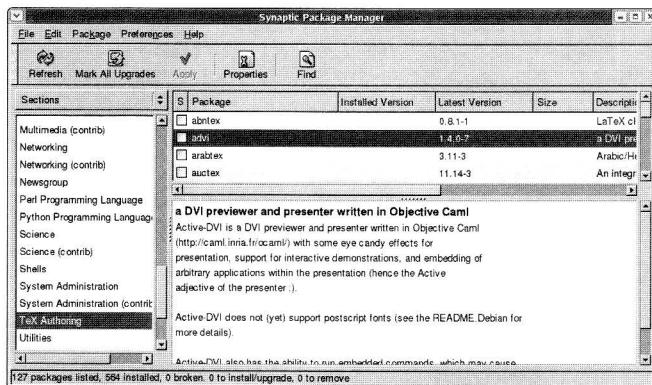
Közös A képernyő alján található **Közös**, **Leírás**, **Függőségek**, illetve **Szakértő** fülek a csomaglistából kiválasztott programcsomag legfontosabb adatait tartalmazzák.

Ha ez a rész nem jelenik meg a nyitóképernyőn, külön ablakban kérhetjük a csomaglistán a jobb egérgomb hatására megjelenő menüből.

Amikor a telepítendő és eltávolítandó csomagokat kiválasztottuk, és a telepítést megkezdtük, a 9.8. ábrán látható ablakban követhetjük nyomon a telepítés



9.8. ábra. A synaptic telepítés közben



9.9. ábra. A synaptic újabb változata

menetét. Az ablakban jól látható, hogy előre láthatóan mennyi időre van szükség a telepítés befejezéséhez.

A synaptic egyik újabb változatának indulóképernyője látható a 9.9. ábrán. Ez a változat a dpkg alacsonyszintű csomagkezelőt használja, és angol nyelvet használ a felhasználóval való kapcsolattartásra. Látható, hogy a legtöbb elemet ítt is könnyen megtaláljuk.

10. fejezet

A grafikus munkafelület

A GNU/Linux rendszereket általában az X Window System grafikus felülettel használjuk. Az X Window System rendszert eredetileg az X Consortium Inc. dolgozta ki. Mára az X Window Systemet hálózati szabvánnyé[21] sok program támogatja. A felhasználók az X Window rendszerrel általában nem mint hálózati szabvánnyal, hanem mint munkaeszközzel találkoznak[7, 8, 17].

Az X Window System – más néven X Window vagy X – tehát egy hálózati szabvány, nem pedig egy program neve! A szabványnak sok megvalósítása ismeretes, a GNU/Linux rendszerek általában a szabad forrású XFree86 programot használják. Az XFree86 tehát egy program, amely X Window System szabványú grafikus felületet biztosít.

Fontos tudnunk, hogy nemrégiben az XFree86 fejlesztésének egy új ága jelent meg Xorg néven. Az új ág létrejöttének elsősorban jogi okai vannak, jelen pillanatban az XFree86 és az Xorg között nincs túl nagy különbség. Ez bizonyára változni fog, az XFree86 és az Xorg valószínűleg eltávolodik egymástól.

Az Xorg tehát szintén egy olyan program, amely X Window System szabványú grafikus felületet biztosít.

Az X Window legelterjedtebb megvalósításai a szabvány 11. változatának 6. kiadását követik, ezért szokás X11R6-ként (*X version 11. release 6.*) is nevezni.



Figyeljük meg, hogy a szabvány neve X Window System, X Window, esetleg X, de semmiképpen nem X Windows! Nem csak hibás és illetlen az X Window rendszert ezzel a névvel illetni, de kissé morbid is.

A következő oldalakon arról olvashatunk, miképpen épül fel az X Window rendszer, hogyan használhatjuk, és miképpen állíthatjuk be számítógépünkön.

10.1. A grafikus felület indítása és leállítása

Az X Window GNU/Linux rendszeren sokféleképpen indítható, de minden indítási mód végső soron az X nevű bináris program indításához vezet. Az X az X kiszolgáló, a program, amely a grafikus felületet mint szolgáltatást biztosítja. Az X kiszolgáló által biztosított eszközök a grafikus képernyő, a betűkészletek, a grafikus felületen használható billentyűzet és a mutatóeszköz (egér).

Nem az X kiszolgáló biztosítja a grafikus felületen látható menüket, nyomógombokat és ablakkereteket. Az X kiszolgáló csak egy alacsonyszintű programozói felületet ad a grafikus alkalmazások számára, amelyek a grafikus felület megjelenítésére használják azt.

10.2. A betűk

Az X Window grafikus rendszer sokféle betűtípusat használ, amelyeket viszonylag bonyolult nevük segítségével érhetünk el. Mivel az X el sem indul, ha a megfelelő betűtípusokat nem találja meg, fontos, hogy már most megismерkedjünk a betűk kezelésével.

A betűket ún. betűkészletekbe – vagy betűtípusokba – rendezve kezeljük. A betűtípusok nem mások, mint hangulatuk, stílusuk, típusuk alapján megkülönböztethető betűkészletek, amelyek valamely nyelv, nyelvek vagy jelkészletek összes elemét megvalósítják. A számítástechnikában megkülönböztetünk bitképes (*bitmap*) és méretezhető (*scalable*) betűtípusokat. Amíg a bitképes betűk képpontonként tárolják a betűk alakját, addig a méretezhető betűtípusok azoknak a görbéknek az egyenleteit írják le, amelyek a betűket határolják. A méretezhető betűtípusok nagy előnye, hogy nagyításkor a betűk alakja „sima”, jól olvasható marad, ellentétben a bitképes betűkkel, amelyek nagyításkor elveszítik alakjukat, olvashatatlanokká válnak.

A betűtípusokat GNU/Linux rendszereken egy-egy állományban tároljuk, családjaik szerint alkönyvtárakba rendezve. Az X képes használni az állományokban elhelyezett betűtípusokat, és képes azokat helyi vagy más gépeken található betűtípus-kiszolgáló (*font server*) segítségével használni. A betűtípus-kiszolgáló olyan program, amely a gépen található betűket más gépek számára szolgáltatja.

10.2.1. A betűk nevei

X alatt minden betűtípusnak egyedi neve van, amely tartalmazza a betűtípusok legfontosabb tulajdonságait. Az alkalmazások beállítóállományában a betűtípusokat általában ezekkel a szabványos nevekkel kell azonosítanunk, ezért igen fontos, hogy tisztában legyünk a nevek jelentésével.

A betűtípus neve – jelekkel elválasztott mezőkből áll, melyek elnevezése és jelentése a következő:

fndry A betűtípus készítő vállalat vagy személy megnevezése.

fmlly A betűtípus családja, általánosan használt elnevezése.

wght A betűtípus vastagsága, súlya, ami többek közt lehet **bold** (kövér) vagy **medium** (félkövér).

slant A betűtípus ferdesége, ami lehet például **i** (*italic* kurzív) vagy **n** (*normal*, normál).

sWdth A betűtípus szélessége, például **normal** (normál) vagy **condensed** (összenyomott).

adstyl További stílusjegy, például **sans serif** (talp nélküli) vagy **serif** (talpas).

pxlsz A betűtípus magassága képpontban.

ptSz A betű nagyítása százalékban.

resx A betűtípushat milyen vízszintes felbontású képernyőn javasolt használni.
A képernyőfelbontást a hüvelykenkénti képpontok számával adjuk meg
(*dpi, dot per inch*).

resy A javasolt függőleges képernyőfelbontás.

spc Az elhelyezésre (térközre) vonatkozó információ.

avgWidth A betűk átlagos szélessége képpontban.

rgstry A betűk kódolására használt szabvány neve.

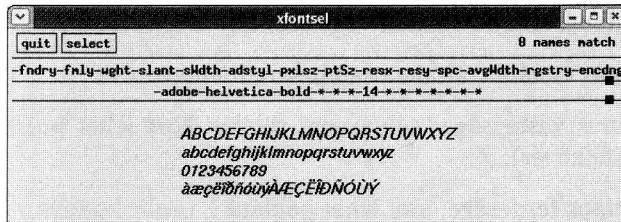
encdng A betűk kódolására használt szabvány változatának száma.

Látható, hogy a betűk elnevezése nem túl egyszerű, ezért hasznos lehet az X betűmegjelenítő segédprogramja, az **xfontsel**, amely képes megjeleníteni az összes telepített betűtípushat. A program indítás után egy grafikus felületet biztosít, amelyen a betűtípusok neveinek mezői egyenként állíthatók, és a kiválasztott betűtípushat azonnal megtekinthető (10.1. ábra).

Fontos segítséget jelent, hogy az X számára nem kell minden mezőt meghatározunk ahhoz, hogy egy betűtípushat kiválasszunk, a számunkra érdektelen mezőket * jellel helyettesíthetjük, a név végén pedig egyszerűen elhagyhatjuk. Ha ilyen helyettesítő karaktereket – névelhagyást – használunk, a névben tulajdonképpen betűtípusok egy csoportjára hivatkozunk, és így a választást a rendszerre bízzuk.

Az **xlsfonts** nevű program segítségével kiírhatjuk azon betűtípusok nevét, amelyek megfelelnek a helyettesítőkarakterekkel megadott névmintának. A névmintát a -fn kapcsoló után kell megadnunk. Ha a -fn kapcsolót elhagyjuk, a program az összes telepített betűtípushat nevét kiírja a szabványos kimenetre.

129. példa Írassuk ki a képernyőre, hogy egy adott mintának milyen betűtípusok felelnek meg!



10.1. ábra. Betűk kiválasztása az xfontsel program segítségével

```
$ xlsfonts -fn -adobe-courier-**-normal-*-*-
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-2
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-2
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-2
-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-2
$
```

A kapott listából kiválaszthatjuk a konkrét betűtípusat, amelyet használni szeretnénk.

A betűtípusok kiválasztásával kapcsolatban meg kell jegyeznünk, hogy külön segédprogramok kezelik a nyomtatók betűtípusait, amelyekre a későbbiekben, a nyomtatás kapcsán visszatérünk.

10.2.2. A betűtípus-kiszolgáló

Ma már a legtöbb GNU/Linux gyűjtemény betűtípus-kiszolgálót használ a betűk kezelésére, még akkor is, ha a számítógép nincs hálózatba kötve. A betűtípus-kiszolgálót az `xfs` program valósítja meg, amely a szokásos módon indítható és leállítható szolgáltatásként az `/etc/init.d/xfs` programmal (a 6.3.1. példa min-tájára).

A betűtípus-kiszolgáló viselkedését a `/etc/X11/fs/config` beállítóállomány határozza meg. Ez a beállítóállomány tartalmazza azoknak a könyvtáraknak a nevét, amelyekben a kiszolgáló a betűtípusokat keresi. Ha új betűtípusokat szeretnénk telepíteni, ezek közül a könyvtárak közül kell egyet választanunk és a betűtípusokat oda kell bemásolnunk.

Nem elegendő azonban egyszerűen felmásolni a betűtípusokat tartalmazó állományokat! A betűtípus-könyvtárakban található listaállományokat is újra létre kell hozni. A listaállományokra – `fonts.dir` néven – akkor is szükség van, ha a betűtípusok kezelését a betűtípus-kiszolgáló végzi, és akkor is, ha a betűtípusokat közvetlenül az X kiszolgáló használja. Az állományokat az adott könyvtárban

kiadott `mkfontdir` programmal állíthatjuk elő, ha azonban TrueType formátumban tároljuk a betűket, a `ttmkfdir` programot kell használnunk.



Az `mkfontdir` és az `ttmkfdir` parancsot szokás automatikusan lefuttatni a betűtípus-kiszolgáló indításakor. Ha a számítógépünkön ilyen indítóprogram található, elegendő újraindítani a betűtípus-kiszolgálót és az új betűtípusok már használhatók. Szerencsétlen dolog azonban, hogy ily módon a betűtípusokat tartalmazó könyvtárak nyilvántartása két helyen található meg, egyrészt a betűtípus-kiszolgáló beállítóállományában, másrészt pedig a betűtípus-kiszolgáló indítóprogramjában.

A betűtípus-kiszolgáló alapjában véve hálózati szolgáltatást nyújt akkor is, ha a számítógépünk nincs számítógép-hálózathoz kapcsolva.

A program alapértelmezés szerint a 7100-as hálózati kapun várja a kéréseket. Néhány gyűjteményben ettől eltérően a -1 kaput használják annak érdekében, hogy a hálózat távoli gépei ne használhassák a számítógépen futó betűtípus-kiszolgálót – csak a helyi X legyen képes azt elérni. Ha távoli gépnek kívánunk betűtípusokat szolgáltatni, vissza kell állítanunk a szabványos kapuszámot a betűtípus-kiszolgálót indító héjprogramban, és a helyi gépen futó X kiszolgálót is át kell állítanunk, hogy a szabványos kapun megtalálja a betűtípus-kiszolgálót.

10.3. Az X kiszolgáló beállítása

Az X kiszolgáló szöveges beállítóállomány alapján működik. A kiszolgáló beállítása igen fontos és összetett feladat. Fontos, mert a grafikus felületet csak a helyes beállítás után használhatjuk és összetett, mert az X kiszolgáló igen bonyolult program, amelynek beállítása az egyes videókártyák esetében más és más lehet.

Az X kiszolgáló beállítóállományában kell beállítanunk többek között a használt videókártya típusát, a használt mutatóeszköz (egér) típusát, a képernyő felbontását (vagy felbontásait), és azt, hogy milyen frekvenciával képes üzemelni a számítógéphez kapcsolt monitor. Szintén az X kiszolgáló beállítóállományában kapott helyet a betűtípusok helye (általában a betűtípus-kiszolgáló elérhetősége), a grafikus felületen érvényes billentyűkiosztás és jónéhány egyéb apróság.



Az X kiszolgáló beállítása meglehetősen bonyolult, így több alkalmazás is készült a megkönnyítésére. Az `xf86config`, `xf86cfg`, `Xconfigurator` régebbi rendszereken megtalálható beállítóprogramok, amelyek az X beállítását segítik.

Az XFree86 és Xorg mai változatai képesek önmagukat automatikusan beállítani, így a beállítóállományon általában már csak kisebb változtatásokat kell végrehajtanunk.

A következő oldalakon arról olvashatunk, hogy miképpen állíthatjuk be az X kiszolgálót a beállítóállomány szerkesztésével. Az X kiszolgáló indulásakor a beállítóállományt több helyen is keresi.

A következő lista nem teljes, de egyfajta keresési sorrendben megmutatja, melyek azok a helyek, ahol az XFree86 X kiszolgálója a beállítóállományt először keresi.

/etc/X11/állománynév Ha az X kiszolgálónak a parancssorban megadtuk a beállítóállomány nevét, a program az állományt a /etc/X11/ könyvtárban keresi először.

/usr/X11R6/etc/X11/állománynév Ha az X kiszolgálónak a parancssorban megadtuk az beállítóállomány nevét, a program a beállítóállományt a /usr/X11R6/etc/X11/ könyvtárban is megtalálja.

/etc/X11/\$XF86CONFIG Ha az X kiszolgálónak az indításkor nem adtuk meg a beállítóállomány nevét, de az \$XF86CONFIG környezeti változót létrehoztuk, a program először a /etc/X11/\$XF86CONFIG állományban keresi a beállításokat.

/usr/X11R6/etc/X11/\$XF86CONFIG Az XFree86 X kiszolgálója a /usr/X11R6/etc/X11/\$XF86CONFIG állományban is megtalálja a beállításokat.

/etc/X11/XF86Config-4 Az XFree86 újabb változatai, amelyeknek fő változatszáma 4, következő lépésként a /etc/X11/XF86Config-4 állományban keresik a beállításokat.

/etc/X11/XF86Config Az XFree86 régebbi és újabb (3-as és 4-es főváltozatszámú) változatai egyaránt megtalálják beállításaikat a /etc/X11/XF86Config állományban.

Az Xorg X kiszolgálója hasonlóan keresi a beállítóállományt, először a következő helyeket vizsgálva meg:

/etc/X11/állománynév Ha a kiszolgálónak indításkor megadtuk a parancssorban a beállítóállomány nevét, a program azt először a /etc/X11/ könyvtárban keresi.

/usr/X11R6/etc/X11/állománynév Ha a kiszolgálónak a parancssorban megadtuk a beállítóállomány nevét, azt másodiként a /usr/X11R6/etc/X11/ könyvtárban keresi.

/etc/X11/\$XORGCONFIG Ha az \$XORGCONFIG környezeti változó létezik, a program megkeresi az értékének megfelelő nevű állományt a /etc/X11/ könyvtárban.

`/usr/X11R6/etc/X11/$XORGCONFIG` Ha az `$XORGCONFIG` környezeti változó létezik, a program a beállítóállományt a `/usr/X11R6/etc/X11` könyvtárban a változó értékének megfelelő nevű állományban is keresi.

`/etc/X11/xorg.conf-4` Az Xorg X kiszolgálója a beállítóállományt a `/etc/X11/xorg.conf-4` állományban is keresi.

`/etc/X11/xorg.conf` A legtöbb GNU/Linux terjesztés az Xorg X kiszolgálójának beállítóállományát a `/etc/X11/xorg.conf` állományban tartja.

`/etc/xorg.conf` A program a beállítóállományát a `/etc/xorg.conf` állományban is keresi.

Az X kiszolgáló újabb változatai képesek arra, hogy önmaguk számára előállításak a beállítóállományt. Jóllehet az ilyen automatikus beállítóállományokat általában kézzel még állítani kell, és nem is minden esetben működnek, minden-képpen hasznos lehet ez a szolgáltatás. Az automatikusan előállított beállítóállományt sokkal könnyebb szerkeszteni, mint egy új állományt felépíteni. A következő példa bennmutatja, hogyan hozhatjuk létre a beállítóállományt.

130. példa Indítsuk el az X kiszolgálót és a `-configure` kapcsolóval utasítsuk arra, hogy hozza létre a beállítóállományát!

```
$ X -configure 2>&1 | tail -n 4  
Your xorg.conf file is /root/xorg.conf.new
```

To test the server, run '`X -xf86config /root/xorg.conf.new`'

\$

A példában a kiszolgáló által kiírt sorokat helytakarékkosságból elhagytuk, a gyakorlatban azonban mindenkorban érdemes ezeket figyelmesen elolvasni.

Amint látjuk, az X kiszolgáló a beállítóállományt a saját könyvtárunkban helyezte el. Innen az állományt – tanulmányozás és szerkesztés után – a végső felhasználási helyére (általában a `/etc/X11/ könyvtárba`) kell másolnunk és át kell neveznünk.

Az X kiszolgáló beállítóállománya szakaszokra oszlik. minden szakasznak neve van, az egyes szakaszok pedig egymásra épülve hivatkoznak ezekre a nevekre. A szakaszokat az állományban a `Section` kulcsszó vezeti be, és a `EndSection` kulcsszó zárja le.

131. példa Vizsgáljuk meg az X kiszolgáló által létrehozott állomány első néhány sorát!

```
$ head -n 6 /root/xorg.conf.new  
Section "ServerLayout"
```

```

Identifier      "X.org Configured"
Screen         0  "Screen0" 0 0
InputDevice    "Mouse0"  "CorePointer"
InputDevice    "Keyboard0" "CoreKeyboard"
EndSection
$
```

Amint látjuk, az automatikusan létrehozott beállítóállomány első szakasza a ServerLayout névre hallgat.

Az X kiszolgáló beállítóállományában kötelezően megtalálhatók a következő szakaszok:

Files A kiszolgáló működéséhez szükséges állományok. Ez a szakasz elengedhetetlenül fontos, mert az X kiszolgáló nem működhet az itt megadott színdatbázis és betűtípusok nélkül.

Module A kiszolgáló által betöltendő bővítőmodulokat meghatározó szakasz.

A modern X kiszolgálók több olyan betölthető programmodult is használnak, amelyek a grafikus megjelenítés során külön szolgáltatásokat biztosítanak.

InputDevice A bemeneti perifériák (billentyűzet, egér stb.) leírására használt szakasz. minden periféria leírására létre kell hoznunk egy-egy InputDevice szakaszt, ami a legtöbb esetben azt jelenti, hogy egy szakaszban leírjuk a billentyűzet, egy másik szakaszban pedig az egér tulajdonságait.

Device A grafikus kimeneti eszköz (videókártya) leírására szolgáló szakasz. minden videókártyához egy szakasz kell létrehoznunk, ami általában azt jelenti, hogy egy ilyen szakasz található az állományban, hiszen a legtöbb számítógépben egy videókártya található.

Monitor A monitort leíró szakasz. minden monitorhoz egy szakaszt kell létrehoznunk, ami a legtöbb esetben azt jelenti, hogy a beállítóállományban egy Monitor szakasz lesz.

Screen A megjelenítést leíró szakasz. Ez a szakasz hivatkozik a Monitor és Device szakaszokra, azaz a megjelenítést leíró szakasz videókártyához és monitorokhoz rendel megjelenést.

ServerLayout Az X kiszolgáló egészét leíró szakasz, amely hivatkozik a Screen és az InputDevice szakaszokra. A ServerLayout szakasz tehát egymáshoz rendeli a megjelenítésre és az adatbevitelre használt eszközöket, legalább egy videókártyát, egy monitort, egy billentyűzetet és egy mutatóeszközt.

A modern lehetőségeket biztosító számítógépeken további szakaszok is előfordulhatnak, amelyek igen fontosak lehetnek a számítógép adta lehetőségek kihasználásában, de az itt felsorolt szakaszok elegendőek a grafikus felület használatához, ezért először ezeket vesszük sorra.

10.3.1. Az állományokat leíró szakasz

Az állományokat leíró Files szakaszban a színadatbázis és a betűtípusok fellelési helyét kell megadnunk. Az X kiszolgáló a színadatbázis és az alapértelmezett betűtípus nélkül nem indul el, ezért ez a szakasz elengedhetetlenül fontos. A Files szakaszban a következő kifejezéseket szokás használni:

RgbPath "állománynév" A szöveges színadatbázis, amely megadja, hogy az egyes színek hogyan állíthatók elő az alapszínekből. A színadatbázist tartalmazó állomány az `rgb.txt` nevű állományban van az X valamelyik könyvtárában. Az RgbPath kulcsszó után az állomány elérési útját és nevét kell megadnunk, de a `.txt` végződés elhagyható.

FontPath "állománynév" A kifejezés segítségével megadhatjuk, hogy az X kiszolgáló hol találja a betűtípusokat. A kiszolgáló több helyről is képes összegyűjteni a betűtípusokat, ezért a szakaszon belül a FontPath kulcsszó többször is használható.

A kulcsszó után a betűtípusok elérési útját kell megadnunk. Ezt a következő formák egyikében tehetjük meg:

könyvtárnév Megadhatjuk a betűtípusokat tartalmazó könyvtár nevét abszolút elérési úttal (az első karakter a `/` jel). A kiszolgáló ekkor a könyvtárban található leíróállományt keresi (például `fonts.dir`), és a könyvtárban található betűtípusok adatait onnan olvassa be.

`unix/:port` A kifejezéssel a helyi számítógépen futó betűtípus-kiszolgálót írhatjuk le. A kifejezésben szereplő szám azt adja meg, hogy hányas hálózati kapun érhető el a kiszolgáló. A szokásos érték 7100.

`tcp/gépnév:port` A kifejezés segítségével távoli gépen futó betűtípus-kiszolgálót adhatunk meg. A kifejezésben a távoli gép nevét és a távoli gépen használt hálózati kapu számát kell megadnunk. A betűtípus-kiszolgálók általában a 7100-as hálózati kaput használják kérések fogadására.

ModulePath "könyvtárnév" A bővítmódulok tárolására használt könyvtár neve. Általában nincs szükség erre a kifejezésre, mert a modulok az alapértelmezés szerinti könyvtárban vannak elhelyezve

A következő példa a Files szakasz felépítésének egy lehetséges módját mutatja be.

132. példa Vizsgáljuk meg az X kiszolgáló beállítóállományának következő részletét:

```
1 #  
2 # Az állományok elhelyezkedésének lirása.  
3 #
```

```

4 Section "Files"
5   RgbPath  "/usr/X11R6/lib/X11/rgb"
6   FontPath "/usr/X11R6/lib/X11/fonts/TrueType"
7   FontPath "unix/:7100"
8   FontPath "tcp:/fontserv.ak.hu:7100"
9 EndSection

```

Amint látjuk, a szokásos módon – a # karakter után – helyezhetünk el megjegyzéseket az állományban.

10.3.2. A bemeneti eszközöket leíró szakasz

Az InputDevice szakaszokban a bemeneti perifériákat (egér, billentyűzet stb.) kell leírnunk, minden szakaszban egy eszközt. Mivel általában legalább egy billentyűzetet és egy mutatóeszközt használunk, a beállítóállományban legalább két InputDevice szakasz található. Az InputDevice szakaszban a legtöbb esetben a következő kifejezéseket használjuk:

Identifier "azonosító" A bemeneti eszköz egyedi, szabadon megválasztható azonosítója, ami a beállítóállományon belül az eszközt jelöli. A beállítóállomány más szakaszaiban ezzel a névvel hivatkozhatunk az itt leírt eszközre.

Az azonosítót minden eszköz esetében használnunk kell, így ennek a kifejezésnek minden InputDevice szakaszban szerepelnie kell.

Ha az azonosító név CorePointer (mutatóeszközök esetében) vagy CoreKeyboard (billentyűzetek esetében), mindenképpen az eszköz lesz az elsődleges mutatóeszköz, illetve billentyűzet.

Driver "meghajtónév" Az eszköz meghajtóprogramja, amely az eszközt vezéri. A meghajtóprogramok az X kiszolgáló moduljaiként töltődnek be a kiszolgáló indításakor. A használható meghajtónevekre, a rendelkezésre álló modulok listájára a későbbiekbén visszatérünk.

Mivel minden egyes bemeneti eszköznek meghajtóprogramra van szüksége, ennek a kifejezésnek minden InputDevice szakaszban szerepelnie kell.

A bemeneti eszközök meghajtóprogramjai modulok, amelyek a megfelelő könyvtárban (például /usr/X11R6/lib/modules/input/) találhatók meg, minden meghajtóprogram külön állományban. A legtöbb meghajtóprogram saját dokumentációval rendelkezik, amelyet a man program segítségével érhetünk el. A legfontosabb meghajtóprogramok a következők:

keyboard A billentyűzet kezelését végző meghajtóprogram. Ez a meghajtóprogram nem modul, hanem az X kiszolgáló beépített része.

A szakaszban, ahol ezt a meghajtóprogramot használjuk, hasznosak lehetnek a következő kifejezések:

Option "XkbModel" "típus" A billentyűzet típusa, ami a legtöbb személyi számítógép esetében pc101 vagy pc102, esetleg microsoft (a Microsoft Natural billentyűzet esetében).

Option "XkbLayout" "billentyűkiosztás" A használt billentyűkiosztás, ami lehet például us, hu vagy hu_qwerty.

Option "AutoRepeat" "szám1 szám2" A billentyűlenyomás automatikus ismétlését lehet beállítani ezzel a kifejezéssel.

Az idézőjelek között megadott első szám megadja, hogy hány ezredmásodperc után indul a billentyűk ismétlése, a második szám pedig megadja, hogy másodpercenként hányszor ismétlődjön a lenyomott billentyű.

kbd A **kbd** meghajtóprogram modulként megvalósított billentyűzetkezelő program az X kiszolgáló számára. A dokumentáció szerint ez a modul fogja felváltani az X kiszolgálóba épített meghajtóprogramot (**keyboard**).

A modul beillesztésére használt szakaszban használhatjuk ugyanazokat a kifejezéseket, amelyeket a **keyboard** belső meghajtóprogram leírására használt szakaszban, és használhatjuk a következő kifejezést is:

Option "Device" "karaktereszköz-meghajtó" A billentyűzettel való kapcsolattartásra használt karaktereszköz meghajtó állomány.

mouse Az egér kezelését végző meghajtóprogram.

A szakaszban, ahol a **mouse** meghajtóprogramot leírjuk, a következő kifejezések használhatók:

Option "Protocol" "típus" A kifejezés segítségével megadhatjuk, hogy milyen szabvány betartásával tartsa a meghajtóprogram a kapcsolatot az eszközzel, vagyis tulajdonképpen azt, hogy mi az eszköz típusa (habár köztudott, hogy sok egér többféle kapcsolattartási módot is támogat).

A **típus** helyén a következő szavakat használhatjuk: Auto, Microsoft, MouseSystems, MMSeries, Logitech, MouseMan, MMHitTab, GlidePoint, IntelliMouse, ThinkingMouse, ValuMouseScroll, AceCad, PS/2, ImPS/2, ExplorerPS/2, ThinkingMousePS/2, MouseManPlusPS/2, GlidePointPS/2, NetMousePS/2, NetScrollPS/2, BusMouse, SysMouse, WSMouse, USB, Xqueue.

Option "Device" "karaktereszköz-meghajtó" Az egérrel való kapcsolattartásra használt karaktereszköz-meghajtó állományt adhatjuk meg ezzel a kifejezéssel.

(A karaktereszköz-meghajtó állomány neve /dev/psaux PS/2 egér-csatlakozó esetén, illetve /dev/cua0, /dev/cua1 stb. a soros csatlakozók esetében.)

Option "Buttons" "szám" A kifejezéssel megadhatjuk, hogy hány gomb van az egéren. Csak akkor kell használnunk ezt a kifejezést, ha az automatikus felismerés valamelyen okból nem működik.

Option "Emulate3Buttons" yes | no A kifejezés segítségével be- és kikapcsolhatjuk a harmadik egérgomb utánzását. Ha ez be van kapcsolva és a felhasználó egyszerre nyomja le a két egérgombot, annak olyan hatása van, mintha a középső (harmadik) gombot nyomta volna le.

A mouse meghajtómodul sok egyéb – kevésbé fontos – kapcsolóval is rendelkezik.

void A void meghajtóprogram nem kapcsolódik fizikai eszközhöz, és így soha nem küld adatokat.

További meghajtóprogramok is a rendelkezésünkre állnak, amelyeket nem tárgyalunk részletesen: acecad, aiptek, calcomp, citron, digitaledge, dmc, dynapro, elographics, fpit, hyperpen, js_x, ur98, magellan, microtouch, mutouch, palmax, pennmount, spaceorb, summa, tek4957, wacom.

A következő példa bemutatja, miképpen hozhatjuk létre a billentyűzet és az egér kezelésére alkalmas szakaszokat:

133. példa A következő részlet két InputDevice szakaszt tartalmaz, egyet a billentyűzet, egyet pedig az egér számára:

```

1 Section "InputDevice"
2   Identifier  "CoreKeyboard"
3   Driver      "keyboard"
4   Option      "XkbModel" "pc105"
5   Option      "XkbLayout" "hu"
6 EndSection
7
8 Section "InputDevice"
9   Identifier  "CorePointer"
10  Driver      "mouse"
11  Option      "Protocol" "IMPS/2"
12  Option      "Device"   "/dev/input/mice"
13  Option      "Emulate3Buttons" "yes"
14 EndSection

```

Figyeljük meg, hogy a két szakasz csak a legfontosabb kifejezéseket tartalmazza!

10.3.3. A videókártyát leíró szakasz

A videókártya beállítására szolgáló Device szakasz minden bizonnal a legfontosabb és legösszetettebb szakasz az X kiszolgáló beállítóállományában.

A Device szakaszban a következő kifejezéseket használhatjuk:

Identifier "azonosító" A szakasz egyedi, szabadon megválasztható azonosítója, ami a beállítóállományon belül a videókártyát jelöli. A többi szakaszban ezzel a névvel hivatkozhatunk a videókártyára. Ez a kifejezés kötelező.

Driver "meghajtónév" Az eszköz meghajtóprogramjának neve. Az X meghajtóprogram modulként megvalósított meghajtóprogramot használ az elterjedt videókártya-családok kezelésére. A meghajtóprogramok nevére és a videókártyák típusára a későbbiekben visszatérünk. Ez a kifejezés kötelező.

VendorName "név" A kifejezés felhasználásával megadható a videókártya gyártójának (forgalmazójának) neve. A nevet csak arra használjuk, hogy a beállítóállományban és a hibaüzenetek között könnyebben eligazodjunk, ezért nem kötelező megadni.

BoardName "név" A videókártya típusát adhatjuk meg ezzel a kifejezéssel. A megadott nevet az X kiszolgáló a hibaüzenetekben megjeleníti, a névnek egyéb jelentősége nincs.

BusID "azonosító" Ha a számítógépben több, azonos típuscsaládba tartozó videókártya is található, a BusID kulcsszó segítségével azonosíthatjuk őket.

Ha a videókártyák a PCI felületre csatlakoznak, azonosítóként az X -scanpci kapcsolója hatására az első oszlopban kiírt azonosítószámot kell megadnunk a PCI kulcsszó után (például PCI:00:02:0).

Screen szám Ha a számítógépbe olyan videókártya van beépítve, amely egy PCI eszközöként jelenik meg, de több képernyőt is képes kezelni, a Screen kulcsszó után megadott sorszámmal különböztethetjük meg az egyes képernyőket. A képernyők számozása 0-tól indul.

Chipset "áramkör neve" Ha a meghajtóprogram nem tudja megállapítani, hogy milyen integrált áramkör van a videókártyába építve, a Chipset kulcsszó után megadhatjuk.

VideoRam szám A kifejezés segítségével megadhatjuk, hogy a videókártyára hány kilobájt memória van építve. Csak akkor használjuk ezt a kifejezést, ha a meghajtóprogram nem képes a memória méretét megállapítani.

A modern X kiszolgálók esetében a videókártyák meghajtóprogramjai modulként vannak megvalósítva. A meghajtóprogramokat megtalálhatjuk a megfelelő könyvtárban (például /usr/X11R6/lib/modules/drivers), vagy letölthetjük

az Internetről. A legtöbb meghajtóprogram saját dokumentációval rendelkezik, amely megtekinthető a `man` programmal.

Az X kiszolgáló következő modulai videókártyákhoz használható meghajtóprogramot valósítanak meg:

`apm` Meghajtóprogram az Alliance ProMotion videókártyákhoz, amelyek a következő áramkörökre épülnek: ProMotion 6420, ProMotion 6422, AT24, AT3D, AT25.

A meghajtóprogram a videókártyák által megvalósított gyorsító áramkörök használatát bizonyos üzemmódokban támogatja.

`chips` Meghajtóprogram a Chips and Technologies videókártyákhoz, amelyek a következő áramkörökre épülnek: ct64200, ct64300, ct65520, ct65525, ct65530, ct65535, ct6554, ct65540, ct65545, ct65546, ct65550, ct65554, ct65555, ct68554, ct69000, ct69030.

A meghajtóprogram bizonyos áramkörök esetében támogatja a beépített gyorsító áramköröket.

`fbdev` Meghajtóprogram a Linux rendszermag által támogatott, *frame buffer* üzeműben használható videókártyák számára.

Ez a meghajtóprogram nem közvetlenül kezeli a videókártya áramköreit, hanem a rendszermag által biztosított `fb` (*frame buffer*) felületet használja. A meghajtóprogram a gyorsító áramköröket nem használja.

Ha a számítógépünkbe épített videókártyát az X kiszolgáló egyik meghajtóprogramja sem támogatja, a rendszermagba megkísérhetjük bekapcsolni a *frame buffer* támogatást és az `fbdev` meghajtóprogrammal indítani a grafikus felületet.

A meghajtóprogram leírására használt szakaszban elhelyezhetjük a következő kifejezéseket:

Option "fbdev" "karaktereszköz-meghajtó" A kifejezés segítségével megadhatjuk, hogy a rendszermag által biztosított `fb` eszközök közül melyikhez kapcsolódjon a meghajtóprogram. Az eszközöket a kapcsolódásra használt karaktereszköz-meghajtók nevével adhatjuk meg (például `/dev/fb0`, `/dev/fb1` stb.).

Option "Rotate" "irány" A kifejezés segítségével elforgathatjuk a képet. A forgatás irányát szövegesen adhatjuk meg: a CW az óramutató járásával megegyező irány, UD a fejjel lefelé állított kép, a CCW az óramutató járásával ellentétes irány jelzésére szolgál.

Fontos megjegyeznünk, hogy ha a videókártya vezérléséhez az `fbdev` meghajtóprogramot használjuk, az X kiszolgáló beállítóállományában nem kell (és nem is szerencsés) külön megadni a képernyő felbontását és színmélységét. Ilyen esetben a grafikus felület felbontása és színmélysége attól függ,

hogy indításkor (a rendszertöltés során) milyen felbontást és színmélysséget adtunk meg.

glint Meghajtóprogram a 3Dlabs és a Texas Instruments által gyártott GLINT és Permedia áramkörkészletre épülő videókártyák számára. A meghajtóprogram a következő áramköröket támogatja: GLINT 300SX, GLINT 500TX, GLINT DELTA, GLINT GAMMA, GLINT GAMMA2, GLINT MX, Permedia, Permedia 2, Permedia 2v, Permedia 3, Permedia R3, Permedia R4.

A meghajtóprogram a videókártyákba épített gyorsító áramkörök használatát támogatja.

i128 Meghajtóprogram a Number 9 I128 áramkörökre épülő videókártyákhoz. A meghajtóprogram a következő áramköröket támogatja: I128-II, I128 rev 1, I128-T2R, I128-T2R4.

A meghajtóprogram támogatja a videókártyákra épített gyorsító áramkörök használatát.

i810 Meghajtóprogram az Intel i810 áramkörsorozatára épülő videókártyák számára. A meghajtó a következő áramköröket támogatja: 830M, 845G, 852GM, 855GM, 865G, i810, i810-DC100, i810e, i815.

mga Meghajtóprogram Matrox videókártyákhoz. A meghajtó a következő áramköröket támogatja: G100, G200, G400, G450, G550, MGA1064SG, MGA2064W, MGA2164W.

A meghajtóprogram támogatja a videókártya gyorsító áramköreinek használatát.

neomagic Meghajtóprogram a Neomagic áramkörökkel épített videókártyák számára. A meghajtó a következő áramkörök használatát támogatja: NM2070, NM2090, NM2093, NM2097, NM2160, NM2200, NM2230, NM2360, NM2380.

A meghajtóprogram támogatja a videókártyákra épített gyorsító áramkörök használatát (kvíve 24 bites színmélység esetén).

nv Meghajtóprogram az NVIDIA áramkörkészletre épülő videókártyák számára. A meghajtóprogram a következő áramköröket támogatja: NV3, NV4, NV5, NV10, NV11, NV15, NV20, NV1A, NV1F, NV17, NV18, NV25, NV28, NV30, NV31, NV34, NV35, NV36, NV38.

Az áramkörök közismert nevei a következők: RIVA 128, RIVA TNT, RIVA TNT2, GeForce 256/QUADRO, GeForce2/QUADRO2, GeForce3/QUADRO DCC, nForce/nForce2, GeForce4/QUADRO4, GeForce FX/QUADRO FX.

A meghajtóprogram támogatja a kétdimenziós számításokat végző gyorsító áramkörök használatát.

radeon Meghajtóprogram az ATI RADEON videókártya-család elemeihez. A meghajtóprogram a következő áramkörök használatát támogatja: R100, R200, R300, R350, R360, RS100, RS200, RS250, RS300, RV100, RV200, RV250, RV280, RV350, RV360.

A meghajtóprogram a legtöb áramkör esetében támogatja a kétdimenziós számításokat végző gyorsító áramkörök használatát és néhány áramkör esetében a háromdimenziós gyorsítást is.

s3virge Meghajtóprogram az S3 VIRGE áramkörcsaládra épülő videókártyához.

A meghajtóprogram a következő áramköröket támogatja: 86C260, 86C280, 86C325, 86C357, 86C362, 86C365, 86C368, 86C375, 86C385, 86C988.

Az áramkörök közismert elnevezése a következő: Trio 3D, Trio 3D/2X, ViRGE, ViRGE DX, ViRGE GX, ViRGE GX2, ViRGE MX, ViRGE MX+, ViRGE VX.

A meghajtóprogram támogatja a videóáramkörökbe épített gyorsító áramkörök használatát.

savage Meghajtóprogram az S3 Savage áramkörcsaládra épülő videókártyákhoz.

A meghajtóprogram a következő áramköröket támogatja: 8a20, 8a21, 8a22, 8a25, 8a26, 8c10, 8c11, 8c12, 8c13, 8d01, 8d02, 8d03, 8d04, 9102.

Az áramkörök közismert elnevezése: Savage3D, Savage4, Savage2000, Savage/MX, Savage/IX, ProSavage PM133, ProSavage KM133, Twister (Pro-Savage PN133), TwisterK (ProSavage KN133), ProSavage DDR, ProSavage DDR-K.

A meghajtóprogram támogatja a gyorsító áramkörök használatát.

siliconmotion Meghajtóprogram a Silicon Motion videókártyák számára.

A meghajtóprogram a következő áramköröket támogatja: SM710, SM712, SM720, SM730, SM810, SM820, SM910.

A meghajtóprogram támogatja a gyorsító áramkörök használatát.

sis Meghajtóprogram a Silicon Integrated Systems videóáramkör-családot használó videókártyákhoz. A meghajtóprogram a következő áramköröket támogatja: SiS300, SiS305, SiS315, SiS315H, SiS315PRO, SiS330, SiS530, SiS540, SiS550, SiS551, SiS552, SiS5597, SiS5598, SiS620, SiS630, SiS6326, SiS6326AGP, SiS6326DVD, SiS650, SiS651, SiS661FX, SiS730, SiS740, SiS741, SiS760, SiSM650, SiSM661FX, SiSM661MX.

A meghajtóprogram támogatja a gyorsító áramkörök használatát.

vesa Meghajtóprogram VESA szabványú videókártyák számára.

A VESA szabvány lehetővé teszi, hogy a különféle gyártók különböző áramköreivel felépített videókártyákat egyazon felületen keresztül használja az X

kiszolgáló, így a vesa meghajtóprogram minden VESA szabványú videókártyát képes kezelni.

Tudnunk kell azonban, hogy ha a VESA felületen kezeljük a videókártyát, az sebességen és a nyújtott szolgáltatások terén is rosszabbul teljesít, ezért ezt a meghajtóprogramot csak akkor érdemes használnunk, ha más, az adott videókártyát közvetlenül kezelő meghajtóprogram nem áll rendelkezésre.

A vesa maghajtóprogram támogatja a 8, 15, 16 és 24 bites színmélység használatát is.

A meghajtóprogram nem támogatja a videókártyán elhelyezett gyorsító áramkörök használatát.

vga Meghajtóprogram általános VGA videókártyák számára. Ez a meghajtóprogram a VGA szabványt használja a videókártyák vezérlésére, így minden VGA szabványú videókártyával működik, de igen rossz teljesítményt nyújt.

A vga meghajtóprogram csak az 1, 4 és 8 bites színmélyiséget támogatja.

A meghajtóprogram nem támogatja a gyorsító áramkörök használatát.

via Meghajtóprogram a VIA Technologies áramkörcsaládjára épülő videókártyákhoz. A meghajtóprogram a következő áramköröket támogatja: CLE266, CLE3022, CLE3122, K8M800, KM400, VT3204, VT32045, VT7204, VT7205.

A meghajtóprogram támogatja a videókártyára épített gyorsító áramköröket.

vmware A VMWare olyan szimulátorprogram, amely lehetővé teszi, hogy a GNU/Linux rendszert más operációs rendszereken („ablakban”) futtassuk. A vmware meghajtóprogram a VMWare által utánzott videókártya meghajtóprogramja.

Az X kiszolgálóhoz további meghajtóprogramokat is találhatunk az Interneten vagy a GNU/Linux terjesztések újabb változataiban, és a leírt meghajtóprogramok is támogathatnak újabb áramköröket, hiszen folyamatosan fejlődnek. Néhány példa további meghajtóprogramokra: ark, ati, atimisc, cirrus, cirrus_alpine, cirrus_laguna, cyrix, gamma, i740, nsc, r128, r200, rendition, riva128, s3, tdfx, tga, trident, tseng, voodoo.

134. példa Vizsgáljuk meg, hogy az X kiszolgáló a beállítóállomány automatikus létrehozásakor milyen formában készítette el a Device szakaszt!

```
1 Section "Device"
2     ### Available Driver options are:-
3     ### Values: <i>: integer, <f>: float,
4     ### <bool>: "True"/"False",
5     ### <string>: "String", <freq>: "<f> Hz/kHz/MHz"
6     ### [arg]: arg optional
```

```

7   #Option      "NoAccel"          "# [<bool>]
8   #Option      "SWcursor"         "# [<bool>]
9   #Option      "ColorKey"        "# <i>
10  #Option      "CacheLines"      "# <i>
11  #Option      "Dac6Bit"         "# [<bool>]
12  #Option      "DRI"             "# [<bool>]
13  #Option      "NoDDC"           "# [<bool>]
14  #Option      "ShowCache"       "# [<bool>]
15  #Option      "XvMCSSurfaces"  "# <i>
16  #Option      "PageFlip"        "# [<bool>]
17  Identifier  "Card0"
18  Driver      "i810"
19  VendorName "Intel Corp."
20  BoardName  "82852/855GM Integrated Graphics Device"
21  BusID      "PCI:0:2:0"
22 EndSection

```

Amint látjuk, a program alaposan előkészítette a terepet a munkára! A videókártyát a program felismerte, a PCI nyilvántartásból a gyártót és az áramkör nevét kiolvasta, és a beállítóállományba beírta. Szintén beírta a program a beállítóállományba a videókártya-meghajtóprogram által kezelt különleges kapcsolókat, hogy azokat könnyebben megleljük.

135. példa Alakítsuk át a 10.3.3. példában található szakaszt úgy, hogy abban csak a legszükségesebbek maradjanak!

```

1 Section "Device"
2   Identifier  "VideoCard0"
3   Driver      "i810"
4   VendorName "Intel Corporation"
5   BoardName  "82852/855GM Integrated Graphics Device"
6 EndSection

```

Amint látjuk, a Device szakasz viszonylag egyszerűen is felépíthető. A példában a videókártya leírására mindenkor négy sort használtunk, és abból is elhagyható volna további kettő (a Vendorname és a BoardName nem kötelező).

10.3.4. A monitort leíró szakasz

Az X kiszolgáló beállítóállományának Monitor nevű szakasza a monitor tulajdon-ságainak leírására szolgál. Ebben a szakaszban a következő kifejezéseket használ-hatjuk:

Identifier "azonosító" A monitor egyedi, szabadon megválasztható azonosítója, ami a beállítóállományon belül a monitort jelöli. A beállítóállomány más szakaszaiban ezt a nevet használhatjuk a monitorra való hivatkozásul. Ez a kifejezés kötelező.

VendorName "gyártó" A kifejezés segítségével megadható a monitor gyártójának (forgalmazójának) neve. Ezt a nevet csak arra használjuk, hogy könnyebben azonosíthassuk a monitort, nem kötelező megadni.

ModelName "típus" A monitor típusának neve. Ezt a nevet is csak a beállítóállományban és a hibaüzenetek között való könnyebb eligazodás miatt használjuk, nem kötelező megadni.

HorizSync frekvenciák A monitor vízszintes eltérítésének frekvenciája vagy frekvenciái. Az X kiszolgáló arra használja ezt a kifejezést, hogy ellenőrizze, az egyes üzemmódok megfelelőek-e a monitor számára.

A kifejezésben egy vagy több frekvenciaértéket adhatunk meg. A számokat és tartományokat vesszővel elválasztva kell egymás után írnunk. Ha tartományt akarunk jelezni, a tartomány alsó és felső határa közé kötőjelet kell tennünk.

A beírt számok alapértelmezés szerint kHz-ben értendők, de ettől eltérhetünk, ha a szám után a Hz vagy a MHz mértékegységet írunk.

Ha a HorizSync kifejezést nem adjuk meg, az X kiszolgáló az alapértelmezés szerinti értéket használja, ami 28–33kHz. Amint látjuk tehát, ezt a kifejezést nem kötelező ugyan használni, de ha a monitor jobb képességeit ki akarjuk használni, vagy az átlagosnál gyengébb képességű monitorunk van, a vízszintes eltérítés frekvenciáját megadó kifejezés nem maradhat el. Ilyen esetben a monitor dokumentációjából ki kell keresnünk az értékeket, és a beállítóállományban el kell helyeznünk azokat.

VertRefresh frekvenciák A monitor függőleges eltérítésének (képfrissítésének) frekvenciája vagy frekvenciái. Az X kiszolgáló arra használja ezeket az értékeket, hogy ellenőrizze, az egyes üzemmódok megfelelőek-e a monitor számára.

A kifejezésbe több frekvenciát és frekvenciatartományt is írhatunk egymás után, vesszővel elválasztva. Ha tartományt akarunk beírni, a tartomány alsó és felső határa közé kötőjelet kell írnunk.

A beírt számok alapértelmezés szerint (a HorizSync kifejezéssel ellentétben) Hz-ben értendők, de ettől eltérhetünk, ha a szám után a kHz vagy a MHz mértékegységet írunk.

Ha a kifejezést nem adjuk meg, az X kiszolgáló a 43–72 Hz tartományt használja.

DisplaySize szélesség magasság A képernyő magassága és szélessége milliméterben megadva. A legtöbb esetben nincs jelentősége ennek a kifejezésnek.

Gamma vörös zöld kék A gammakorrekciónak értékei a három alapszínre. Az egyes színekre érvényes értékeket számmal kell megadnunk 0,1 és 10 közt. (A számok megadásakor tizedespontot kell használnunk, nem pedig tizedevesszőt!)

A kifejezés hatására a színösszetevőket a megadott értékkal erősíti vagy gyengíti a meghajtóprogram, így ellensúlyozza a monitor színhibát. Ennek a színhelyességnak a kritikus alkalmazások esetében van szerepe, illetve esetleg akkor, ha a számítógépünkhez több monitor csatlakozik. Ilyen esetben az egymás mellett álló monitorok színhőmérésének különbsége zavaró lehet.

Sajnos a gammakorrekciónak nem minden videókártya meghajtóprogramja kezeli.

Option "dpms" A monitor energiatakarékos üzemmódjának (DPMS, *display power management services*) engedélyezése, amelynek köszönhetően a monitor kikapcsol, amikor éppen visszatértünk az asztalunkhoz, és leültünk a székünkre.

A monitort leíró szakaszban a monitor üzemmódjait a Mode kulcsszó után számmokkal és kifejezésekkel egészben pontosan leírhatjuk. Az üzemmód ilyen megadása meglehetősen bonyolult és általában szükségtelen, ezért eltekintünk az ismertetésétől.

136. példa Vizsgáljuk meg a következő részletet, ami egy jellemző monitorleíró szakasz az X kiszolgáló beállítóállományából!

```

1 Section "Monitor"
2   Identifier      "Monitor0"
3   VendorName     "Acer"
4   ModelName      "LCD Panel 1280x1024"
5   HorizSync       31.5 - 67.0
6   VertRefresh     50.0 - 75.0
7 EndSection

```

Amint látjuk, a kötelező azonosítón kívül két fontos kifejezés található a szakaszban, a vízsintes és a függőleges frekvenciatartományok, amelyeken belül a monitor működőképes.

10.3.5. A monitor és a videókártya egymáshoz rendelése

Az X kiszolgáló beállítóállományának Screen (képernyő) szakaszában – vagy szakaszai – a monitort és a videókártyát rendeljük egymáshoz. Ha a számítógéphez több monitor és több videókártya is csatlakozik, ez a szakasz többször is szerepelhet a beállítóállományban.

A monitort és a videókártyát egymáshoz rendelő Screen – azaz képernyő – szakaszban a következő kifejezéseket használhatjuk:

Identifier "azonosító" A szakasz egyedi, szabadon megválasztható azonosítója. Az itt található azonosítót a beállítóállomány más szakaszai hivatkozásul használhatjuk.

Ennek a kifejezésnek a használata kötelező.

Device "kártyanév" A Screen szakaszhoz tartozó videókártya azonosítása. A Device kulcsszó után található kártyanév a videókártya egyértelmű azonosítására szolgál. A beállítóállományban tehát kell lennie egy Device szakasznak, amelyben az Identifier kulcsszó után ezt a nevet adtuk meg.

Ennek a kifejezésnek a megadása kötelező.

Monitor "monitornév" A Screen szakaszhoz tartozó Monitor szakasz megadása. A kifejezésben szereplő monitornévnek valamelyik Monitor szakasz Identifier kifejezésében szerepelnie kell, azaz kell ilyen nevű monitornak lennie.

DefaultDepth szám Az alapértelmezés szerinti színmélység megadása bpp-ben (*bit per pixel*, képpontonkénti bitek száma). Ha a Screen szakaszban leírt videókártyát és monitort több színmélységen is lehet használni, a DefaultDepth kulcsszó után adhatjuk meg, melyik színmélység legyen az alapértelmezett.

Ha az X kiszolgálót elindítjuk, a Screen szakaszban leírt képernyő az itt megadott színmélységen fog elindulni. (Feltéve, hogy a kiszolgáló indításakor a -depth kapcsoló után nem adunk meg más értéket.)

A legtöbb esetben a Screen szakaszon belül egy vagy több Display alszakaszt is használunk a különféle színmélységű üzemmódok leírására. A Screen szakaszban leírt képernyő annyi színmélységen képes működni, ahány Display alszakasz található különféle színmélység-megadással.

A Display alszakaszban a következő kifejezéseket használhatjuk:

Depth szám A színmélység megadása. A kifejezés azt adja meg, hogy ez az alszakasz milyen színmélységű üzemmódban érvényes.

A kifejezésben megadott szám bpp-ben értendő, és azt adja meg, hogy egy képpont színének megadása hány bittel történik. A különféle videókártyák különféle színmélységű üzemmódokra képesek, az általánosan elter-

jedt üzemmódok a 8bpp, 16bpp és 24bpp színmélységet támogatják. Néhány videókártya támogatja az 1bpp, 4bpp, 15bpp, és 30bpp üzemmódokat is.

Modes "név1" "név2" Az adott színmélységen használható üzemmódok listája az üzemmódok neveinek felsorolásával.

Az üzemmódokat a Monitor szakaszban kellene megadnunk a Mode kulcsszóval, de szerencsére ma már ez a legtöbb esetben szükségtelen. Az X kiszolgáló beépített üzemmódokkal rendelkezik, amelyekre itt hivatkozhatunk a nevükkel.

A beépített üzemmódok nevei az elterjedt képernyőfelbontásokra utalnak, így van 640x480, 800x600, 1024x768 stb. nevű üzemmód.

A következő példa bemutatja, miképpen hozhatunk létre Screen szakaszt és benne az üzemmódok felsorolására Display szakaszokat.

137. példa A következő sorok az X beállítóállományának Screen szakaszát mutatják be:

```

1 Section "Screen"
2   Identifier "Screen0"
3   Device      "Videocard0"
4   Monitor     "Monitor0"
5   DefaultDepth    16
6
7   # Nagy felbontás, kis színmélység
8   SubSection "Display"
9     Depth     8
10    Modes    "1024x768" "800x600" "640x480"
11  EndSubSection
12
13  # Kis felbontás, nagy színmélység
14  SubSection "Display"
15    Depth    16
16    Modes    "800x600" "640x480"
17  EndSubSection
18 EndSection

```

A részlet magától értehetődő. Figyeljük meg, hogy kétféle színmélységet használhatunk, amelyek közül a 16bpp színmélység az alapértelmezett.

10.3.6. Az elrendezés

Az X kiszolgáló beállítóállományának ServerLayout szakasza az összes eddig bemutatott szakaszok összefogására szolgál, a munkaasztalunk elrendezését írja le. A ServerLayout megadja, hogy az egyes képernyők hogyan helyezkednek el (egymás felett, alatt, mellett), és hogy a grafikus felületet milyen beállításokkal kívánjuk használni.

A legtöbb esetben a munkaasztal elrendezése roppant egyszerű, egy billentyűzet, egy egér és egy képernyő alkotja a teljes grafikus munkahelyet, így maga a ServerLayout szakasz is egyszerű. Ilyen egyszerű elrendezést mutat be a következő példa:

138. példa A következő részlet egyszerű elrendezést mutat be egy billentyűzettel, egy egérrel és egy képernyővel:

```
1 Section "ServerLayout"
2   Identifier      "Sima"
3   Screen          0  "Screen0" 0 0
4   InputDevice    "Keyboard0" "CoreKeyboard"
5   InputDevice    "Mouse0"   "CorePointer"
6 EndSection
```

Figyeljük meg, hogy a ServerLayout szakasz 2. sorában egy azonosítót kellett megadnunk, a 3–5. sorában pedig egy képernyőt, egy billentyűzetet és egy mutatószköt!

A példában vannak ugyan magyarázatra szoruló részek, de a részlet különösebb magyarázat nélkül is felhasználható. Nem kell másat tennünk, mint a Screen0, Keyboard0 és a Mouse0 nevek helyére az általunk használt azonosítókat bemásolni, és a szakaszt felhasználhatjuk egyszerű receptként.

A ServerLayout szakaszban a következő kifejezéseket használhatjuk az elrendezés leírására:

Identifier "azonosító" Az elrendezés egyedi, szabadon megválasztható azonosítója.

Ennek a kifejezésnek a használata kötelező.

Screen sorszám "azonosító" elhelyezés Az elrendezéshez tartozó képernyő megadása. minden elrendezéshez tartoznia kell legalább egy képernyőnek, de az elrendezéshez több képernyő is tartozhat.

A kifejezésben szereplő sorszám a képernyő sorszáma. minden képernyőnek, amely az elrendezéshez tartozik, egyedi sorszámot kell adnunk. A képernyők számozása 0-tól indul.

A kifejezésben található *azonosító* a képernyőt azonosító, a beállítóállomány valamelyik Screen szakaszának Identifier kifejezésében létrehozott egyedi név.

A kifejezésben szereplő elrendezés a képernyők elhelyezkedését adja meg (leírja, hogy hova tettük a monitorokat az asztalunkon). Itt a következő részkifejezéseket használhatjuk:

x y A képernyő bal felső koordinátájának megadása. Ha itt például a 0, 0 érték áll, a képernyőnk a grafikus felület bal felső sarkát fogja mutatni. (Ne feledjük, hogy a számítógépes grafika által használt koordinátarendszerben az *x* tengely lefelé mutat!)

RightOf "azonosító" Ez a képernyő a RightOf kulcsszó után megadott azonosítójú képernyő mellett, attól jobbra helyezkedik el.

LeftOf "azonosító" Ez a képernyő a LeftOf kulcsszó után megadott azonosítójú képernyő mellett, attól balra helyezkedik el.

Above "azonosító" Ez a képernyő az Above kulcsszó után megadott azonosítójú képernyő felett helyezkedik el.

Below "azonosító" Ez a képernyő a Below kulcsszó után megadott azonosítójú képernyő alatt helyezkedik el.

Relative "azonosító" *x y* Ez a képernyő a Relative kulcsszó után megadott azonosítójú képernyőtől *x*, *y* távolságban helyezkedik el a koordinátarendszerben.

Az elrendezés kapcsán meg kell jegyeznünk, hogy a legtöbb grafikus alkalmazás feltételezi, hogy a grafikus felület téglalap alakú, és ha ez nem így van, problémák merülhetnek fel. Az ablakkezelő például automatikusan elhelyezheti a grafikus felület olyan pontján a megnyitott ablakot, amelyet nem ábrázol egyetlen monitorunk sem, így esélyünk sem lesz látni az adott alkalmazást.

InputDevice "azonosító" "kapcsoló" A kifejezés segítségével a bemeneti eszközöket (egér, billentyűzet stb.) adhatjuk meg az adott elrendezéshez.

A kifejezésben szereplő *azonosító* a bemeneti eszköz leírására használt szakaszban létrehozott azonosító, ami az eszközöt egyértelműen meghatározza.

A kifejezés végén található *kapcsoló* elhagyható, a legtöbb esetben az alapértelmezett mutatóeszköz (CorePointer) és billentyűzet (CoreKeyboard) azonosítására szolgál.

Option "DontVTSwitch" "yes | no" Ha a kifejezésben a yes érték szerepel, nem lehet a karakteres képernyőkre kapcsolni a **Ctrl**+**Alt**+**Fn** billentyűkombinációkkal.

Option "DontZap" "yes | no" Az X kiszolgáló kilövésének tiltása. Ha a kifejezésben a yes érték szerepel, az X kiszolgálóból nem lehet erőszakosan kilépni a **[Ctrl] + [Alt] + [Backspace]** billentyűkombinációval.

Option "DontZoom" "yes | no" Ha a kifejezésben a yes érték szerepel, nem lehet képernyőfelbontást váltani a **[Ctrl] + [Alt] + [+]** és **[Ctrl] + [Alt] + [-]** billentyűkombinációval.

Option "AllowMouseOpenFail" "yes | no" Az egér nélküli üzemmód engedélyezése. Ha a kifejezésben a yes érték szerepel, az X kiszolgáló akkor is elindul, ha az egér nem elérhető. Alapértelmezés szerint a mutatóeszköz nélkül nem indítható el a grafikus felület!

Option "BlankTime" szám A kifejezés segítségével megadhatjuk, hogy hány perc után sötétedjen el a képernyő, ha nem használjuk a számítógépet.

Option "StandbyTime" szám A kifejezés segítségével meghatározhatjuk, hogy a monitor hány perc után kapcsoljon készenléti üzemmódra, ha nem használjuk a számítógépet.

Option "SuspendTime" szám A kifejezés segítségével megadhatjuk, hogy a monitor hány perc után kapcsoljon felfüggesztett üzemmódra, ha nem használjuk a számítógépet.

Option "OffTime" szám A kifejezés segítségével megadhatjuk, hogy a monitor hány perc után kapcsoljon ki, ha nem használjuk a számítógépet.

Option "NoPM" "yes | no" Az energiatakarékos üzemmódok tiltása. Ha a kifejezésben a yes érték szerepel, az összes energiatakarékkosságot célzó szolgáltatás kikapcsolódik.

Option "Xinerama" "yes | no" A többképernyős üzemmód bekapcsolása.

A Xinerama üzemmód lényege, hogy több monitoron dolgozhassunk egyszerre, és az ablakokat az egyes képernyők között mozgathassuk. Ez a szolgáltatás alapértelmezés szerint ki van kapcsolva, csak akkor kapcsoljuk be, ha több képernyőn szeretnénk egyszerre dolgozni.

139. példa A következő példa bemutatja, hogyan állíthatunk be két videókártyát és két monitort használó grafikus felületet.

```

1 Section "ServerLayout"
2 Identifier      "dual head configuration"
3 Screen          0 "Screen0" 0 0
4 Screen          1 "Screen1" RightOf "Screen0"
5 InputDevice     "Mouse0" "CorePointer"
6 InputDevice     "Keyboard0" "CoreKeyboard"
7 Option          "Xinerama" "on"
8 EndSection

```

Szerencsére ha több videókártya van a számítógépünkben, és elindítjuk az X kiszolgálót a -configure kapcsolóval, automatikusan xinerama beállítóállományt készít. Azt persze nem tudja kideríteni a kiszolgáló, hogy a minotorokat hogyan helyeztük el az asztalunkon, ezért a beállítóállományt nagy valószínűség szerint módosítanunk kell. Esetleg persze átrendezhetjük az asztalunkat is...

Az eddig bemutatott eszközök már elegendők ahhoz, hogy használható beállítóállományt készitsünk velük az X kiszolgáló számára. A következő példa egy ilyen egyszerű beállítóállományt mutat be.

140. példa A következő példa jól használható receptként az X kiszolgáló beállítóállományának elkészítéséhez.

```

1  Section "ServerLayout"
2    Identifier      "Simple Layout"
3    Screen         0  "Screen0" 0 0
4    InputDevice    "Mouse0"  "CorePointer"
5    InputDevice    "Keyboard0" "CoreKeyboard"
6  EndSection
7
8  Section "Files"
9    RgbPath        "/usr/X11R6/lib/X11/rgb"
10   ModulePath     "/usr/X11R6/lib/modules"
11   FontPath       "/usr/X11R6/lib/X11/fonts/misc/"
12   FontPath       "/usr/X11R6/lib/X11/fonts/Speedo/"
13   FontPath       "/usr/X11R6/lib/X11/fonts/Type1/"
14   FontPath       "/usr/X11R6/lib/X11/fonts/CID/"
15   FontPath       "/usr/X11R6/lib/X11/fonts/75dpi/"
16   FontPath       "/usr/X11R6/lib/X11/fonts/100dpi/"
17 EndSection
18
19 Section "InputDevice"
20   Identifier     "Keyboard0"
21   Driver         "keyboard"
22   Option         "XkbModel" "pc102"
23   Option         "XkbLayout" "hu"
24 EndSection
25
26 Section "InputDevice"
27   Identifier     "Mouse0"
28   Driver         "mouse"
29   Option         "Device"          "/dev/mouse"
30   Option         "Protocol"       "Microsoft"
31   Option         "Emulate3Buttons" "yes"
```

```
32 EndSection
33
34 Section "Monitor"
35   Identifier "Monitor0"
36   VendorName "SAM"
37   ModelName "SyncMaster"
38 EndSection
39
40 Section "Device"
41   Identifier "Card0"
42   Driver      "ati"
43   VendorName "ATI"
44   BoardName   "Radeon QD"
45 EndSection
46
47 Section "Screen"
48   Identifier "Screen0"
49   Device     "Card0"
50   Monitor    "Monitor0"
51   DefaultDepth 24
52   SubSection "Display"
53     Depth    16
54     Modes   "1280x1024" "1024x768"
55   EndSubSection
56   SubSection "Display"
57     Depth    24
58     Modes   "1280x1024" "1024x768"
59   EndSubSection
60 EndSection
```

A példában minden beállítás szerepel, ami a grafikus felület elindításához szükséges, bár nyilvánvaló, hogy a gyakorlatban ennél összetettebb beállítóállományt használunk.

Ha mintaként használjuk e sorokat, nyilván módosítanunk kell a videókártya meghajtóprogramját.

10.3.7. Az X kiszolgáló különleges szolgáltatásai

Az eddig bemutatott eszközök segítségével beállíthatjuk az X kiszolgálót, ami a legtöbb esetben elegendő ahhoz, hogy működőképes grafikus felületet hozzunk létre. Az X kiszolgáló azonban rendelkezik jónéhány különleges szolgáltatással, amelyekkel külön kell foglalkoznunk. A különleges szolgáltatások egy részéhez az X kiszolgáló megfelelő bővítőmodulját kell betöltenünk, más szolgáltatásokhoz

a megfelelő videókártya-meghajtóprogramot kell használnunk, és vannak olyan különleges szolgáltatások is, amelyeket az X kiszolgáló automatikusan kezel.

Az X kiszolgáló beállítóállományában a `Module` szakasz szolgál a betöltendő modulok megadására. Ebben a szakaszban használhatjuk a következő kifejezést:

`Load "modulnév"` A kifejezés az adott néven elérhető modul betöltésére ad utasítást az X kiszolgálónak.

A modulokat tartalmazó könyvtár nevét a `Files` szakaszban a `ModulePath` kulcsszóval adhatjuk meg. Ezt a kulcsszót általában nem kell használnunk a beállítóállományban, mert a modulok az alapértelmezés szerinti helyükön vannak. (A modulok betöltésekor az X kiszolgáló a modulokat a modulok tárolására szolgáló könyvtár bizonyos alkönyvtáraiban keresi.)

A modulokat tartalmazó állományok neve a `lib` kifejezéssel kezdődik és általában `.a` kifejezéssel végződik. A `bitmap` modult tároló állomány neve tehát `libbitmap.a`.

A következő példa bemutatja a `Module` szakasz felépítését.

141. példa A következő részlet bemutatja, milyen egyszerűen tölthetünk be modulokat az X kiszolgáló számára.

```

1 Section "Module"
2   Load  "dbe"
3   Load  "extmod"
4   Load  "fbdevhw"
5   Load  "glx"
6   Load  "record"
7   Load  "freetype"
8   Load  "type1"
9   Load  "dri"
10 EndSection

```

Amint látjuk, a legtöbb esetben egyszerűen csak fel kell sorolnunk a betöltendő modulok nevét.

Az X bővítmóduljainak egy része a betűtípusok tárolására használt állományformátumokat kezeli. Jelenleg a következő modulok tartoznak ebbe a csoportba:

`bitmap` A modul bitképes formában tárolt betűtípusok kezelését valósítja meg.

Az X kiszolgáló mindenéppen betölти ezt a modult, akkor is, ha a beállítóállományban nem adunk erre parancsot.

`freetype` A modul a TrueType betűtípusok kezelését teszi lehetővé. Ha a modult betöltyük, az X kiszolgáló képessé válik TrueType betűket tartalmazó könyvtárak kezelésére.

speedo A modul a Speedo betűtípusok kezelését teszi lehetővé. Ha a modult betöljük, az X kiszolgáló képessé válik Speedo betűket tartalmazó könyvtárak kezelésére.

type1 A modul Type1 betűtípusok kezelését teszi lehetővé. Ha a modult betöljük, az X kiszolgáló képessé válik Type1 betűket tartalmazó könyvtárak kezelésére.

E modulok használata igen könnyű. Egyszerűen csak be kell töltenünk a modulokat, és az X kiszolgáló képessé válik az adott betűtípus-állományok kezelésére.

Tudnunk kell azonban, hogy ezekre a modulokra csak akkor van szükség, ha a betűtípus-állományokat közvetlenül az X kiszolgáló kezeli, ha nem használunk betűtípus-kiszolgálót. Ha a betűtípusok elérésére kizárolag betűtípus-kiszolgálót használunk, akkor ezeket a modulokat felesleges betöltenünk.

Nem különálló modulként van megvalósítva, de fontos lehet a következő szolgáltatás:

XVideo Az Xvideo (*Xv, X Window System Video*) lehetővé teszi, hogy programok videóanyagot helyezzenek a képernyő adott területére. Ezt a szolgáltatást nyilvánvalóan olyan programok használják, amelyek videófelvételek lejátszára használatosak (például az *mplayer*, amely igen kitűnően használható erre a célra).

Az Xvideo szolgáltatás nélkül is lehetséges videófelvétel lejátszása, de az XVideo nagymértékben csökkentheti a szükséges erőforrások mennyiségét, különösen teljes képernyős módban. Azok a meghajtóprogramok, amelyek az Xvideo kiegészítést támogatják, általában képesek a videókártya áramköreit felhasználva méretezni a lejátszás során kapott képet, így a lejátszás zökkenőmentes lehet kisebb erőforrásokkal rendelkező számítógépen is.

Az XVideo szolgáltatás használatához két dologra van szükség. A videókártya meghajtóprogramjának (és természetesen a videókártyának) támogatnia kell ezt a szolgáltatást, és az *extmod* modult be kell töltenünk.

Ha szeretnénk tudni, hogy a számítógépünkön elérhető-e az Xvideo szolgáltatás, az *xqryinfo* programot használhatjuk. A program a *-ext all* kapcsoló hatására részletes jelentést ír a képernyőre arról, hogy a kiszolgáló milyen különleges szolgáltatásokat nyújt. Az *xqryinfo* program igen hasznos lehet az X kiszolgáló beállítása közben, amikor a végső simításokat végezzük.



Persze a puding próbája az evés, ezért az Xvideo szolgáltatást általában úgy próbáljuk ki, hogy használjuk. Ha telepítve van a számítógépünkre az *mplayer* lejátszóprogram, akkor indítsuk el a lejátszást a *-ao xv* és a *-zoom no* kapcsolóval, amelyek egyrészt előírják az Xvideo használatát, másrészt tiltják a szoftveres képnagyítást. Ha a lejátszás nem indul el, vagy a lejátszás közben nem lehet teljes képernyős üzemmódra kapcsolni (az billentyűvel), az XVideo szolgáltatás nem érhető el.

Az extmod tehát hasznos lehet, ha videólejátszót használunk. Végyük sorra az ilyen hasznos, egyszerűen kezelhető modulok listáját!

dbe A modul a DBE (*double buffer extension*) szolgáltatást valósítja meg. Ennek lényege, hogy amikor az alkalmazás a képernyőre rajzol, az nem látszik azonnal, csak akkor, amikor az alkalmazás úgy dönt, hogy a teljes rajz elkészült, azt a képernyőre lehet rajzolni. A képernyőtartalmat a DBE használata esetén két helyen kell tárolni: a régi kép a képernyőn látszik, az új, félkész kép pedig nem. A háttérben való rajzolásnak köszönhetően a kép az újrajzolás során villódzásmentes lesz, hiszen a változások egyszerre jelennek meg.

A tapasztalatok szerint a modul problémát nem okoz, érdemes betölteni.

extmod Az extmod sokféle szolgáltatás gyűjtőmodulja. A modulban található szolgáltatásokhoz általában nem kell különleges beállításokat elvégeznünk, egyszerűen csak be kell tölteni a modult. Mivel a modul a tapasztalatok szerint problémát nem okoz, mindenki érdemes betölteni.

fbdewhw A modul elsősorban az fbdev meghajtóprogram számára, a videóáram-körökkel való kapcsolattartásra készült, de más meghajtóprogramok is használják a szolgáltatásait.

A modul betöltése szükséges, ha az fbdev meghajtóprogramot használjuk, és felesleges lehet, ha más meghajtóprogrammal kezeljük a videókártyát. Mivel a modul betöltése automatikus, és általában amúgy sincs rá szükség, a betöltését elhagyhatjuk.

record A modul lehetővé teszi az alkalmazások számára, hogy a felhasználó által végzett műveleteket rögzítsék és visszajátszák. A modul gyakorlati jelentősége csekély.

xtrap A modul segítségével az alkalmazások betekintést nyerhetnek az X Window által kezelt eseményekbe, és eseményeket hozhatnak létre, mintha azok a felhasználótól származtak volna. A modul gyakorlati jelentősége csekély.

A különleges szolgáltatások közt külön csoportot alkotnak a háromdimenziós műveletek, amelyekre általában játékok és tervezőprogramok építenek. Az X Window alapjában véve kétdimenziós ábrázolást valósít meg, de segíti a háromdimenziós világ kétdimenziós megjelenítését is, és lehetővé teszi a videóáramkörökbe épített háromdimenziós eszközök használatát.

A számítógépes grafika háromdimenziós változatát is támogató programok közül GNU/Linux rendszereken az OpenGL (*open graphics library*, nyílt grafikus programkönyvtár) terjedt el leginkább. Az OpenGL a háromdimenziós számításokat támogató függvények szabványa, amelynek egy szabad forrású változata a Mesa. (A Mesa név nem rövidítés, egyszerűen jól hangzik.) A programok a Mesa

programkönyvtár segítségével háromdimenziós számításokat végezhetnek, a háromdimenziós világ elemeit kétdimenzióban ábrázolhatják, és meg is jeleníthetik, akár mozgás közben is.

Az OpenGL programok X Window segítségével a GLX (*OpenGL extensions to the X Window System*, OpenGL kiegészítések az X Window System számára) programkönyvtár segítségével futtathatók. A GLX programkönyvtár biztosítja a kapcsolatot a megjelenítéshez az OpenGL (Mesa) és az X kiszolgáló között.

A modern videókártyák olyan kiegészítő áramköröket tartalmaznak, amelyek alkalmasak háromdimenziós számítások elvégzésére. Ha ilyen videókártyával rendelkezünk, a háromdimenziós számításokat rábízhatjuk a videókártyára is, így akár nagyságrendekkel gyorsíthatjuk a számítások elvégzését. A videókártyában végzett háromdimenziós számításokat a DRI (*direct rendering infrastructure*, közvetlen testmodellező eszköz) felület biztosítja, amely a grafikus programkönyvtár és a videókártya háromdimenziós csatolófelülete.

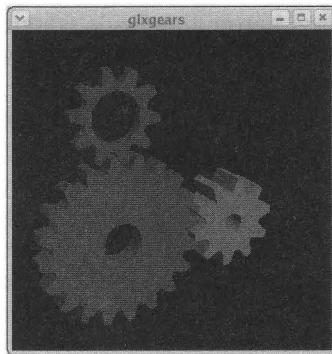
Összefoglalva azt mondhatjuk, hogy az OpenGL egy szabvány, amelyre a Mesa nevű, háromdimenziós számításokat végző programkönyvtár épül. Az OpenGL szabvány alapján megírt programokat lefordíthatjuk és futtathatjuk a Mesa segítségével, amely képes megjeleníteni a program kimenetét X Window alatt a GLX segítségével. Ha a számítógépen a használt videókártyához DRI felületet biztosító meghajtóprogram áll a rendelkezésünkre, a számítások elvégzését nagymértékben gyorsíthatjuk azzal, hogy azokat a videókártyára bízzuk.

Az OpenGL felületet biztosító Mesa könyvtárral és a GLX felülettel általában könnyen boldogulunk. Ezek az eszközök a legtöbb GNU/Linux terjesztés telepítése után elérhetők. Más a helyzet azonban a DRI felülettel, amely erősen épít a használt videókártyára, és így hardverfüggő. Ha a videókártyánk rendelkezik háromdimenziós számítások elvégzésére készült kiegészítéssel és a videókártya meghajtóprogramja képes ezt kihasználni, a DRI bekapcsolásával fel tudjuk gyorsítani a háromdimenziós számítások elvégzését. Mivel sok alkalmazás használatához ez elengedhetetlenül fontos, a következő néhány bekezdésben ehhez adunk némi segítséget.

glx A GLX felületet megvalósító modul, amely lehetővé teszi, hogy OpenGL alkalmazásokat futtassunk. A modul csak a kapcsolatot valósítja meg az alkalmazás és az X kiszolgáló között, lehetővé teszi OpenGL (Mesa) programok futtatását, de nem tartalmazza az OpenGL függvényeket, és nem teszi lehetővé a videókártyába épített gyorsító áramkörök használatát.

dri A modul DRI felületet valósít meg, és lehetővé teszi a videókártyára épített gyorsító áramkörök használatát GLX felületen.

A modul működéséhez szükség van a videókártyára épített gyorsító áramkörökre és arra, hogy a videókártya X kiszolgáló számára beállított meghajtóprogramja támogassa a DRI felület használatát. (Bizonyos meghajtóprogramok nem, vagy csak bizonyos feltételekkel támogatják a gyorsító áramkörök használatát.)



10.2. ábra. A GLX kipróbálása

A `dri` modul használatához a videókártyát vezérlő Linux magmodulra is szükség van. A magmodul általában automatikusan betöltődik az X kiszolgáló indításakor.

A `dri` modul betöltése után az alkalmazás közvetlenül vezérelheti a videókártyát, ami biztonsági kérdéseket vet fel. A jogosultságok felhasználók számára való biztosítására az X kiszolgáló beállítóállományában elhelyezhetjük a DRI szakaszat. A DRI szakasz megadja, hogy melyik felhasználói csoport használhatja a DRI szolgáltatást és milyen jogok tartozzanak a kártyával való kapcsolattartáshoz. A következő részlet magától értetődő:

```

1 Section "DRI"
2     Group          0
3     Mode           0666
4 EndSection

```

A GLX szolgáltatásról és a DRI állapotáról szerezhetünk információkat a `glxinfo` program segítségével. A program kiírja a szabványos kimenetére, hogy a háromdimenziós számításokra a videókártyára épített áramkörök használhatók-e. A háromdimenziós számítások kipróbálására használható a `glxgears` (10.2. ábra), amely az animáció során megméri a számítások sebességét, és a szabványos kimenetre írja.

A GLX és DRI beállításához nyújt segítséget a következő példa.

142. példa Az X kiszolgáló beállítóállományában sem a `glx`, sem pedig a `dri` modul nincs betöltve. Vizsgáljuk meg, hogy a `glxinfo` program mit ír a kimenetére!

```
$ glxinfo
name of display: :1.0
Xlib: extension "GLX" missing on display ":1.0".
```

```
Xlib: extension "GLX" missing on display ":1.0".
Xlib: extension "GLX" missing on display ":1.0".
Error: couldn't find RGB GLX visual
$
```

Amint látjuk, a GLX szolgáltatás nem érhető el, így nem meglepő, hogy a glxgears program el sem indul:

```
$ glxgears
Xlib: extension "GLX" missing on display ":1.0".
Error: couldn't get an RGB, Double-buffered visual
$
```

Helyezzük el a glx modul betöltésére vonatkozó sort az X kiszolgáló beállítóállományában! A modul nem igényel különleges áramköröket és beállításokat, ezért a betöltésére irányuló kísérleteink nem fulladhatnak kudarcba. Az X kiszolgálót természetesen újra kell indítanunk, miután a beállítóállományt módosítottuk.

Ha a glx modult betöltöttük, a glxinfo a GLX szolgáltatásról a következő formában ad jelentést (az egyszerűség kedvéért néhány sort eltávolítottunk):

```
$ glxinfo | head -n 5
name of display: :1.0
display: :1 screen: 0
direct rendering: No
server glx vendor string: SGI
server glx version string: 1.2
```

Amint a program kimenetének 3. sorában olvashatjuk, a DRI nem érhető el, de a GLX szolgáltatás működik. Ez nyilvánvalóan azt jelenti, hogy a háromdimenziós számításokat a számítógép processzora végzi el, nem a videókártyába épített áramkörök.

Ha a GLX szolgáltatás elérhető, a glxgears program működőképes:

```
$ glxgears
470 frames in 5.0 seconds = 94.000 FPS
407 frames in 5.0 seconds = 81.400 FPS
408 frames in 5.0 seconds = 81.600 FPS
X connection to :1.0 broken (explicit kill or server shutdown).
$
```

A program 5 másodpercenként méri a számítási sebességet, és a kapott értéket kiírja a szabványos kimenetre. A használt mértékegység az fps (frame per second), azaz a másodpercenkénti képkockák száma.

Helyezzük most el az X kiszolgáló beállítóállományában a dri modul betöltésére utasító parancsot. A dri modulnak a DRI-t támogató videókártyára és videókártya-meghajtóprogramra van szüksége, így könnyen lehet, hogy a modul nem töltődik be. Ilyen esetben az X kiszolgáló induláskor egy figyelmeztető üzenetet ír a szabványos hibacsatornára, de egyébként a szokott módon elindul.

Ha sikeres a dri betöltése, a glxinfo jelenti, hogy a DRI elérhető:

```
$ glxinfo | head -n 5
name of display: :1.0
display: :1 screen: 0
direct rendering: Yes
server glx vendor string: SGI
server glx version string: 1.2
$
```

Ha most elindítjuk a glxgears programot, és megvizsgáljuk, hogy milyen sebességet mér, jól láthatjuk, hogy a DRI felgyorsította a háromdimenziós számítások elvégzését.

```
$ glxgears
2484 frames in 5.0 seconds = 496.800 FPS
3074 frames in 5.0 seconds = 614.800 FPS
3074 frames in 5.0 seconds = 614.800 FPS
X connection to :1.0 broken (explicit kill or server shutdown).
```

Más számítógépeken az elérhető sebességnövekedés mértéke természetesen más lehet.

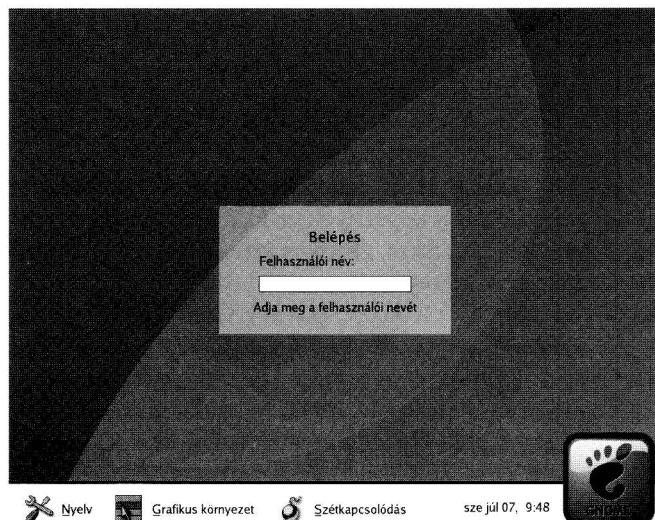
10.4. Az X képernyőkezelő

Manapság a felhasználók többsége kizártan grafikus felületen használja a számítógépet, így szerencsés, ha már a bejelentkezés is grafikus felületen történik. A GNU/Linux terjesztések mindegyike fel van szereelve olyan programmal, amely grafikus bejelentkezási felületet biztosít helyi és távoli bejelentkezéshez. A következő oldalakon ilyen programokról lesz szó.

Az xdm (*X display manager*, X képernyőkezelő) az XFree86 és az Xorg programokhoz tartozó képernyőkezelő, amely helyi és távoli gépek számára biztosít grafikus bejelentkező képernyőt. Az xdm a legtöbb GNU/Linux terjesztés esetében a grafikus felületet megvalósító X Window részeként a rendelkezésünkre áll.

Az xdm mintájára sok hasonló program készült. Ezek a programok általában sokkal látványosabbak, és a testreszabásra is több lehetőséget adnak. Az egyik ilyen továbbfejlesztett program a Gnome projekt[9] gdm (*Gnome display manager*, Gnome képernyőkezelő) programcsomagja.

A két program különbözik ugyan egymástól, de mindenki ugyanazon feladat elvégzésére született. Mind az xdm, mind pedig a gdm helyi és távoli bejelentkezéshez biztosít grafikus felületet. A két program ugyanazon szabványok alapján működik, és mivel ezeket a szabványokat nem kizártan a GNU/Linux rendszerek használják, mindenki program képes kapcsolatot teremteni más rendszerekkel is.



10.3. ábra. Grafikus bejelentkezés a gdm segítségével

10.4.1. A képernyőkezelő indítása

A legtöbb GNU/Linux gyűjteményben szokás a képernyőkezelőt az 5. futási szinten automatikusan elindítani. Ha a számítógépet az 5. futási szintre kapcsoljuk (lásd 6.2.2. példa), a grafikus bejelentkező képernyő automatikusan elindul. Ha a grafikus bejelentkező képernyőre nincs szükségünk, a 3. futási szintre kapcsolhatunk, és így megszabadulhatunk tőle.

Az 5. futási szinten tehát automatikusan elindul a képernyőkezelő (például a `xdm` vagy a `gdm`), hogy helyi és távoli bejelentkező képernyőt biztosítson. A program a következő lépésekben indul:

1. A képernyőkezelő elindítja az X kiszolgálót, amely grafikus felületet, billentyűzet és mutatóeszközöt biztosít.
2. Elindítja X ügyfélprogramként a programot, amely felhasználói nevet és jelszót kér a felhasználótól. Az X ügyfélprogram felveszi a kapcsolatot az X kiszolgálóval, és létrehozza a szükséges képernyőelemeket.
3. A képernyőkezelő ezután a hálózatot figyelve XDMCP (*X display manager control protocol*, X képernyőkezelő; vezérlőüzenetek szabványa) kérésekre vár. Az XDMCP kérések a távoli számítógépek bejelentkezési igényeit jelzik, tehát az XDMCP grafikus hálózati bejelentkezések számára készült.

Ha a felhasználó bejelentkezett a helyi számítógépen, az X kiszolgáló tovább fut, X ügyfélprogramként pedig elindul az ablakkezelő, amely a felhasználó számára munkafelületet biztosít.

Ha a felhasználó kijelentkezik (például lenyomja a **[Ctrl]+[Alt]+[Backspace]** billentyűkombinációt), a grafikus felületet biztosító ügyfélprogram és a vele kapcsolatban álló X kiszolgáló befejezi futását. Ekkor a képernyőkezelő – amely a hátterében továbbra is fut – újra elindítja az X kiszolgálót és a bejelentkezést biztosító grafikus ügyfélprogramot.

Ha a számítógéphez XDMCP kérés érkezik – és annak fogadása a képernyőkezelő számára engedélyezett –, a képernyőkezelő létrehozza a kapcsolatot a távoli X kiszolgáló és a helyben elindított grafikus bejelentkezőprogram közt. A bejelentkezés ezután a már bemutatott módon történhet.

A távoli számítógépen a hálózaton, keresztül történő bejelentkezést kétféleképpen kérhetjük. Az X -query kapcsolója után megadva a távoli számítógép nevét, a számítógéphez kapcsolódhatunk és grafikus bejelentkező ablakot kérhetünk.

Kissé másnéppen zajlik a bejelentkezés, ha az X -indirect kapcsolója után adjuk meg a távoli számítógép nevét. Ilyenkor a következő lépésekben történik a kapcsolat kiépítése:

1. Az X kiszolgáló a -indirect kapcsoló után megadott számítógép számára XDMCP üzenetet küld, amelyben jelzi, hogy közvetett módon szeretne kapcsolatot kiépíteni.
2. A címzett számítógép fogadja a kapcsolatfelvételi kérelmet, és elindítja a választóprogramot (*chooser*), ügyfélprogramként pedig grafikus választóablakot hoz létre.
3. A választóprogram körüzenetet küld a hálózaton és a válaszokból összegyűjti, hogy a környékén ki fogad grafikus kapcsolatfelvételi kérelmeket. A választó ezen gépek listáját egy grafikus ablakban megjeleníti, hogy a felhasználó választhasson közülük.
4. Ha a felhasználó választott a listából, a kapcsolatfelvétel a már bemutatott módon folytatódik.

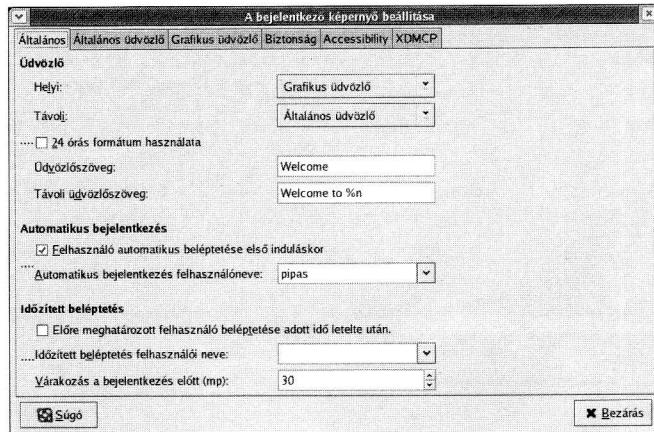
Nyilvánvaló, hogy a közvetett kapcsolatfelvételi mód akkor hasznos, ha egy távoli alhálózaton keresünk grafikus bejelentkezést biztosító számítógépet, hiszen távoli alhálóra nem lehet körüzenetet küldeni.

A következő oldalakon arról olvashatunk, hogy miképpen befolyásolhatjuk az itt bemutatott folyamatot az `xdm` és a `gdm` esetében.

10.4.2. A Gnome képernyőkezelő beállítása

A Gnome grafikus munkafelület részeként használhatjuk a `gdmsetup` programot, amelynek segítségével beállíthatjuk a grafikus bejelentkező képernyőt, a `gdm` programot.

A `gdmsetup` elindulása után a 10.4. ábrán látható ablak jelenik meg. Az ablakban a Gnome grafikus bejelentkező képernyőjét részletesen beállíthatjuk. Igen



10.4. ábra. A gdm általános beállításai

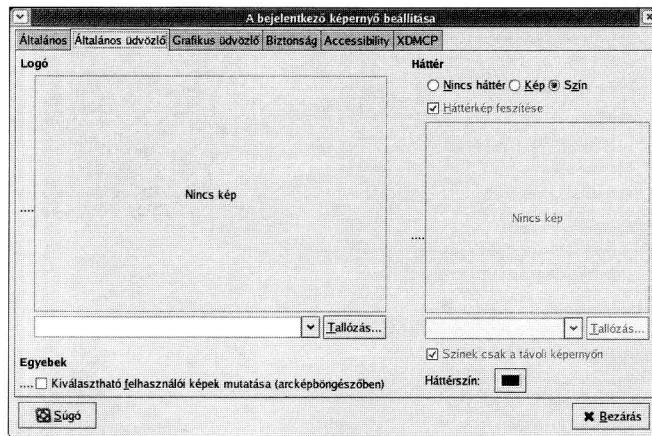
érdekes módon a beállítások átállítása azonnal érvényesül, azaz ha a számítógéphálózaton valaki éppen a bejelentkező képernyőt látja, a változtatásokat néhány másodpercen belül megfigyelheti.

Az ablak alsó részén található *Súgó* nyomógombbal egy részletes és igényes leíráshoz juthatunk, a *Bezárás* nyomógombbal pedig bezárhatjuk az ablakot.

Az ablak felső részén az *Általános* pontot kiválasztva beállíthatjuk a következőket (10.4. ábra):

1. Kiválaszthatjuk, hogy helyi, illetve távoli bejelentkezés esetén a grafikus vagy az általános bejelentkező képernyő jelenjen meg. A grafikus képernyő díszesebb, míg az általános bejelentkező képernyő egyszerűbb és kevesebb erőforrást használ. Általában szerencsés, ha távoli bejelentkezésre az egy-szerűbb, általános üdvözlőt használjuk.
2. Beállíthatjuk a bejelentkezéshez használt üdvözlő szöveget, külön a helyi, és külön a távoli bejelentkezéshez.
3. Beállíthatjuk az automatikus beléptetést, amely a jelszó és a felhasználói név begépelése nélkül belépteti a felhasználót a számítógép bekapcsolásakor. Ez a lehetőség nyilvánvalóan akkor használható, ha a számítógépet általában egy valaki használja, és az egyébként mások számára hozzáférhetetlen.
4. A beállítások között szerepel az időzített beléptetés, amely felhasználói név és jelszó begépelése nélkül belépteti a felhasználót, ha a megadott idő leteltéig más valaki be nem lépett.

Az ablak felső részén az *Általános üdvözlő* részt kiválasztva beállíthatjuk az egyszerűbb bejelentkező ablakot biztosító általános üdvözlőt (10.5. ábra). Az általános üdvözlő esetében a következő tulajdonságokat állíthatjuk be:

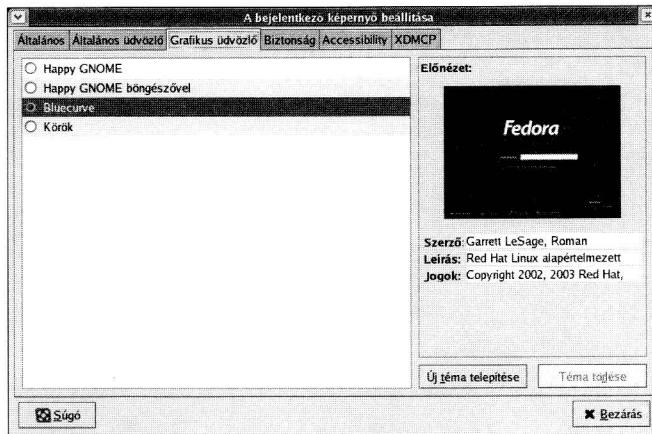


10.5. ábra. A gdm általános üdvözlőjének beállítása

- Kiválaszthatunk egy kisméretű képet, amelyet emblémaként elhelyezhetünk a bejelentkező ablakban. Célszerű akkora képet választani, mint amekkora egy ikon szokott lenni a képernyőn.
- Kérhetjük, hogy a felhasználók listája jelenjen meg az általános üdvözlőben, hogy ne kelljen begépelni a nevüket, egyszerűen kiválaszthassák magukat a listáról. Ezt a lehetőséget természetesen csak olyan esetben ajánlatos használni, amikor csak néhány felhasználó használja a számítógépet.
- A bejelentkezéskor a felhasználók neve mellett megjelenik a fényképük is. A felhasználók a gdmphotosetup nevű program segítségével állíthatják be milyen képet látnának szívesen magukról a bejelentkező képernyőn.
- A képernyő bal oldalán beállíthatjuk, milyen háttérkép vagy háttérszín jelenjen meg az általános üdvözlőben. Beállíthatjuk azt is, hogy távoli bejelentkezés esetén egyszínű háttér legyen a bejelentkezés alatt a képernyőn, így lassú hálózat esetén sem kell sokat várni a bejelentkező ablak megjelenítésére.

Az ablak felső részén található *Grafikus üdvözöl* rész segítségével a grafikus üdvözlőt állíthatjuk be (10.6. ábra). Ebben a részben a következő tulajdonságokat állíthatjuk be:

- A bal oldalon beállíthatjuk a grafikus üdvözlőhöz tartozó témat, amely alapjában véve meghatározza, hogyan nézzen ki a grafikus üdvözlő. A jobb oldalon a képernyőképen azonnal láthatjuk, hogyan néz ki a képernyő az adott téma kiválasztása esetén. Ügyeljünkn arra, hogy ha megtetszett valamelyik téma, a téma neve előtti kiválasztógombot is átkapcsoljuk!

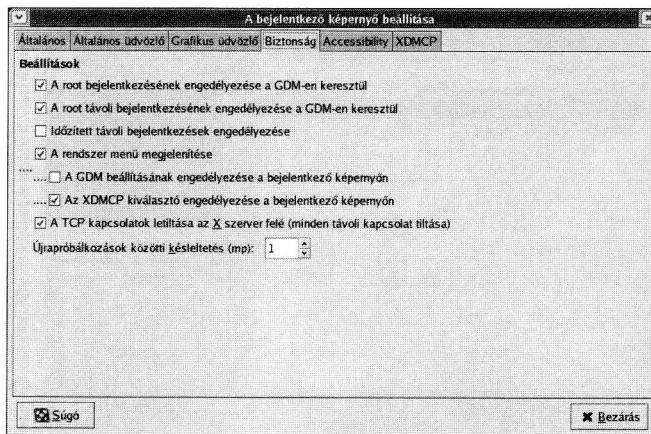


10.6. ábra. A gdm grafikus üdvözlőjének beállítása

- A képernyő jobb alsó részében új téma telepítését kérhetjük. A témát előzőleg fel kell kutatnunk az Interneten, és le kell töltönünk a gépünkre, majd az itt található nyomógombbal telepítenünk kell.

Az ablak felső részén található *Biztonság* rész segítségével a legfontosabb biztonsági kérdéseket állíthatjuk be (10.7. ábra). Ebben a szakaszban a következő kérdésekre adhatunk választ:

- Beállíthatjuk, hogy a rendszergazda beléphessen a grafikus felületet használva. Ha ezt nem engedélyezzük, a rendszergazda csak karakteres felületen léphet be.
- Beállíthatjuk, hogy a rendszergazda távoli számítógépről grafikus felületen bejelentkezhessen. Ha ezt nem engedélyezzük, a rendszergazda távoli gépről a grafikus felület segítségével nem jelentkezhet be, ami megfontolandó elővigyázatosság kiterjedt számítógép-hálózatok és az Internet esetében.
- Engedélyezhetjük az időzített bejelentkezést távoli csatlakozás esetében. Mivel ez azt jelenti, hogy az önként jelentkező távoli idegenektől nem kerünk jelszót, beengedjük őket a számítógépre, csak különleges esetekben használjuk ezt a lehetőséget.
- Beállíthatjuk, hogy megjelenjen-e a rendszermenü, amelynek segítségével újra lehet indítani és le lehet állítani a számítógépet.
- Beállíthatjuk, hogy engedélyezett legyen-e a kiválasztó menü, amelyet a távoli bejelentkezéskor a -indirect kapcsolóval lehet kérni.
- Letilthatjuk a TCP kapcsolatokat, ami azt eredményezi, hogy a távoli számítógépek nem kaphatnak grafikus bejelentkező képernyőt. Ha nem kívánunk



10.7. ábra. A gdm biztonsági beállításai

távolról grafikus bejelentkező ablak segítségével bejelentkezni a számítógépre, mindenkorban javasolt letiltani ezt a lehetőséget, hiszen így egy támadási felülettel kevesebb áll a támadók rendelkezésére.

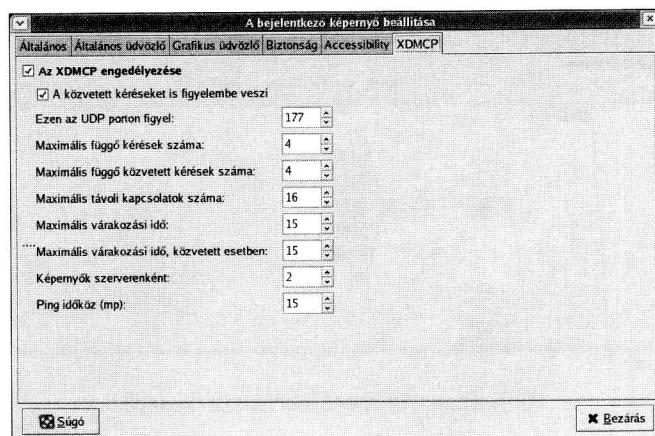
Az ablak felső részén az *XDMCP* rész kiválasztásával beállíthatjuk, hogyan kezelje a program a távoli kapcsolatokat (10.8. ábra). Itt a következő beállításokkal találkozhatunk:

- Engedélyezhetjük az *XDMCP* szabványú kérések kezelését. Ha ezt nem tesszük meg, a számítógép-hálózaton keresztül nem lehet grafikus képernyőt kérni a számítógéptől. Ha nem akarunk távolról grafikus képernyőn keresztül bejelentkezni a számítógépre, mindenkorban érdemes letiltani az *XDMCP* kezelését.
- A hálózati kapcsolatok kezelésének különböző jellemzőit szintén beállíthatjuk az ablakban, de az alapértelmezett értékek a legöbb esetben megfelelőek, azokon változtatni nem kell.

10.4.3. Az X képernyőkezelőjének beállítása

Az *xdm* az X képernyőkezelője (*X Display Manager*). Az *xdm* fogadja a hálózatról érkező bejelentkezási kérelmeket, grafikus bejelentkező ablakot biztosít távoli gépek és a helyi gép számára, valamint kérésre összegyűjti a helyi hálózaton grafikus bejelentkezést biztosító szolgáltatásokat.

Az *xdm* beállítóállománya rögzíti a futásához elengedhetetlenül fontos állományok elhelyezkedését és a működésének alapjait. A beállítóállomány formai szempontból – nyelvtana szerint – az X Window erőforrások (*resources*) kezelésére szolgáló alrendszerére épül.



10.8. ábra. A gdm XDMCP beállításai

Az X erőforrásai

Az X Window az alkalmazások beállításainak tárolására általában erőforrások leíróit használja.

Az erőforrások névvel és értékkel ellátott változók, amelyek általában faszerkezetbe szerveződnek. A faszerkezetet az erőforrások neve biztosítja. A nevekben pont karakterekkel elválasztott mezők vannak, amelyek tagokra osztják az erőforrásokat. Ez a forma a könyvtárak leírására használt abszolút elérési úthoz hasonlóan faszerkezet leírására használható.

Az erőforrások tárolására általában szöveges állományokat használunk, amelyekben az erőforrások neve után kettősponttal elválasztva az értéke szerepel. Az erőforrásleíró állományokban az erőforrások neveiben szerepelhet a * karakter, melynek itt betöltött szerepe hasonló az állománynév-helyettesítő karakterként betöltött szerepéhez. Ilyen módon az állomány egy sorában több erőforrás hoz rendelhetjük ugyanazt az értéket.

Az erőforrások leírására használható állományokban a ! karakter segítségével megjegyzéseket helyezhetünk el. A megjegyzéseket tartalmazó sorok első karakterének a ! karakternek kell lennie.

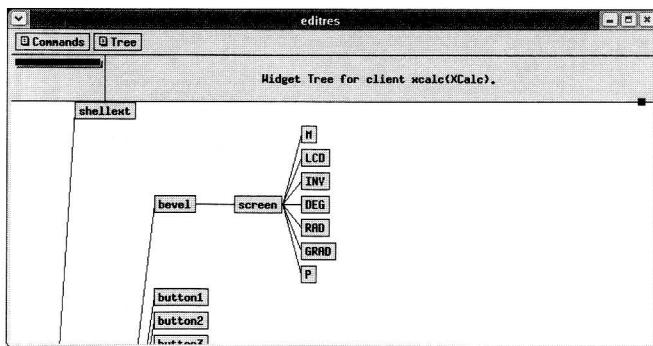
143. példa A következő állományrészlet X erőforrások leírására használható:

```

1 ! Beállítások...
2 XConsole.text.geometry: 480x130
3 XConsole.verbose:      true
4 XConsole*iconic:       true
5 XConsole*font:         fixed

```

*Figyeljük meg a faszerkezetet és a többszörös névmegadásra használt * karaktert!*



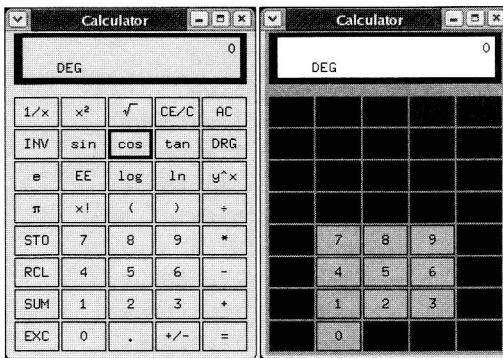
10.9. ábra. Az X erőforrás-szerkesztő

Az erőforrások kezelésére használható elmés program az `editres` (10.9 ábra), amellyel az alkalmazások által használt erőforrások neve lekérdezhető, és az erőforrások értéke megváltoztatható. A program segítségével az erőforrásokat használó alkalmazások beállítása a következő lépésekben végezhető el:

1. Indítsuk el grafikus felületen az `editres` nevű programot.
2. Indítsuk el grafikus felületen az alkalmazást, amelyet be akarunk állítani.
3. Válasszuk az `editres Commands` menüpójéből a *Get tree* menüpontot. Ekkor az egérmutató megváltozik, jelezve, hogy a beállítandó alkalmazást ki kell választanunk.
4. Keressük meg a beállítandó alkalmazás ablakát, és kattintsunk rá az egérrel.
5. Ha az alkalmazás fel van készítve az X erőforrásrendszerének kezelésére, az `editres` ablakában megjelenik az erőforrások szerkezetét ábrázoló fa.

Ha néhány másodpercen belül nem jelenik meg a fa, biztosak lehetünk benne, hogy az alkalmazás nem az X erőforrások rendszerét használja a beállításainak tárolására. Sajnos a modern programok a legtöbb esetben az X erőforrásoknál sokkal bonyolultabb eszközöket használnak a beállítások kezelésére.

6. Az erőforrásokat ábrázoló fa tetszőleges elemét kijelölhetjük, ha rágattink, és beállíthatjuk, ha kiválasztjuk a *Commands* menü *Show resource box* menüpontját. Amint átállítjuk az egyes erőforrások értékét, az `editres` jelzi az új értéket az alkalmazásnak.
7. Ha a beállításokkal elégedettek vagyunk, a megváltoztatott erőforrások értékét szöveges állományba menthetjük az erőforrás lehetséges értékeit tartalmazó ablak megfelelő nyomógombjaival.



10.10. ábra. A számológép erőforrások nélkül és erőforrásokkal

144. példa Készítsünk erőforrás-leíró állományt az xcalc számára! A billentyűket színezzük ki, hogy könnyebb legyen megkülönböztetni őket!

```

1 !
2 ! A~billentyűk háttérének színe
3 !
4 .xcalc.ti.button1.background: blue
5 .xcalc.ti.button2.background: blue
6 .xcalc.ti.button3.background: blue
7 .xcalc.ti.button4.background: red
8 .xcalc.ti.button5.background: red
9 .xcalc.ti.button6.background: blue
10 .xcalc.ti.button7.background: blue
11 .xcalc.ti.button8.background: blue
12 .xcalc.ti.button9.background: blue
13 .xcalc.ti.button10.background: blue
14 .xcalc.ti.button11.background: blue
15 .xcalc.ti.button12.background: blue
16 .xcalc.ti.button13.background: blue
17 .xcalc.ti.button14.background: blue
18 .xcalc.ti.button15.background: blue
19 .xcalc.ti.button16.background: blue
20 .xcalc.ti.button17.background: blue
21 .xcalc.ti.button18.background: blue
22 .xcalc.ti.button19.background: blue
23 .xcalc.ti.button20.background: blue
24 .xcalc.ti.button21.background: blue
25 .xcalc.ti.button22.background: lightblue
26 .xcalc.ti.button23.background: lightblue

```

```

27 .xcalc.ti.button24.background: lightblue
28 .xcalc.ti.button25.background: blue
29 .xcalc.ti.button26.background: blue
30 .xcalc.ti.button27.background: lightblue
31 .xcalc.ti.button28.background: lightblue
32 .xcalc.ti.button29.background: lightblue
33 .xcalc.ti.button30.background: blue
34 .xcalc.ti.button31.background: blue
35 .xcalc.ti.button32.background: lightblue
36 .xcalc.ti.button33.background: lightblue
37 .xcalc.ti.button34.background: lightblue
38 .xcalc.ti.button35.background: blue
39 .xcalc.ti.button36.background: blue
40 .xcalc.ti.button37.background: lightblue
41 .xcalc.ti.button38.background: blue
42 .xcalc.ti.button39.background: blue
43 .xcalc.ti.button40.background: blue
44 !
45 !
46 ! A~program háttérének színe
47 !
48 .xcalc.ti.background: grey

```

A számológép megjelenése a beállított erőforrások nélkül és az itt bemutatott erőforrásokkal megfigyelhető a 10.10. ábrán.

Ha ez erőforrásokat állományba mentettük, azokat a grafikus felületen az xrdb program segítségével bármikor betölthetjük. A betöltött erőforrások az alkalmasztok számára elérhetővé válnak, azok hatása érvényesül. Ha azt akarjuk, hogy az xrdb az állományból az erőforrásokat betöltsse, de a már betöltött erőforrásokat ne törölje, használjuk az xrdb -merge kapcsolóját.

A legtöbb GNU/Linux terjesztés automatikusan betölti a felhasználó saját könyvtárában található .Xresources állományból az erőforrásokat. Ha tehát azt szeretnénk, hogy az erőforrások minden belépéskor érvényesüljenek, az erőforrásokat a .Xresources állományban kell elhelyeznünk.

Az X képernyőkezelő erőforrásai

Az X képernyőkezelőjének (xdm) működését alapjaiban meghatározza a beállítóállomány, amelynek neve xdm-config. Az xdm-config állomány általában a /etc/X11/xdm/ könyvtárban található.

Amint azt már említettük, az xdm beállítóállomány tulajdonképpen egy X erőforrásleíró állomány. Itt a megadható legfontosabb erőforrások a következők:

DisplayManager.errorLogFile Az erőforrás segítségével megadhatjuk, melyik állományba kerüljenek a grafikus felület futása közben keletkező hibaüzenetek.

DisplayManager.servers Az X kiszolgálókat tartalmazó állomány megadása.

Ma már a legtöbb számítógépre egyetlen X kiszolgáló van telepítve, de régebben minden egyes videókártya-típushoz önálló X kiszolgálót használtunk. Régebben ugyanis a videókártyák meghajtóprogramjai nem betölthető modulok voltak, így minden videókártya-családhoz külön X kiszolgáló tartozott. A **DisplayManager.servers** által jelzett állományban adhatjuk meg, hogy az egyes képernyőkhöz melyik X kiszolgáló tartozik.

Mivel ma általában már csak egyetlen X kiszolgálót használunk minden számítógépen, a **DisplayManager.servers** erőforrásban megadott állományt, azaz az X kiszolgálók listáját a legritkább esetben kell módosítanunk.

DisplayManager.requestPort Az erőforrás megadja, hogy a képernyőkezelő melyik UDP kaput figyelve várakozzék a távoli kapcsolatfelvételi kérésekre.

Alapértelmezés szerint ennek értéke 177, amit ritkán kell megváltoztatnunk, viszont sok GNU/Linux terjesztésben a beállítóállományban ez az erőforrás 0-ra van állítva, hogy a képernyőkezelő ne fogadjon külső kéréseket. Ha tehát engedélyezni szeretnénk a grafikus bejelentkezési felületet távoli számítógépek számára is, a **DisplayManager.requestPort** erőforrás értékének beállítását ellenőriznünk kell a képernyőkezelő beállítóállományában.

DisplayManager.*.resources Ez az erőforrás annak az állománynak a nevét adja meg, amelyből az erőforrásokat be kell tölteni a felhasználó azonosítása előtt.

Az erőforrás nevében szereplő * helyére beírhatjuk az egyes képernyők nevét (_0, _1 stb. formában), így minden képernyőre külön erőforrásleírót készíthetünk.

DisplayManager.*.chooser Az erőforrás segítségével megadhatjuk, melyik program induljon el a felhasználó számára választóprogramként.

A választóprogramra akkor van szüksége a képernyőkezelőnek, ha a távoli kapcsolatfelvételi kérés a grafikus bejelentkezésre alkalmas számítógépek összegyűjtésére vonatkozik. (Az X -indirect kapcsolója.)

A * helyére az egyes képernyők nevét írhatjuk be (_0, _1 stb.), így minden képernyő számára más-más választóprogramot állíthatunk be.

DisplayManager.*.systemPath Az erőforrás segítségével megadhatjuk, mi legyen a \$PATH környezeti változó értéke a **Displaymanager.*.startup** és a **DisplayManager.*.reset** erőforrásokban megadott külső programok futtatása ideje alatt.

A * helyére az egyes képernyők nevét írhatjuk be (_0, _1 stb.).

DisplayManager.*.userPath Az erőforrás segítségével megadhatjuk, mi legyen a \$PATH környezeti változó értéke a Displaymanager.*.session erőforrásban megadott külső program – vagyis a felhasználói munkafelület – futtatásának ideje alatt.

A * helyére az egyes képernyők nevét írhatjuk be (_0, _1 stb.).

DisplayManager.*.setup Az erőforrás segítségével megadhatjuk, hogy melyik programot használjuk a képernyő előkészítésére. Ez a program indul el, mielőtt a képernyőn megjelenik a felhasználói nevet és jelszót kérő ablak.

A * helyére beírhatjuk az egyes képernyők nevét(_0, _1 stb.), így minden képernyőre egyedi előkészítő programot állíthatunk be, ami hasznos, mert a távoli képernyőket a hálózati erőforrásokkal való takarékoskodás érdekében sokkal egyszerűbb grafikus elemekkel szokás díszíteni.

Az erőforrás használata során ügyelniük kell arra, hogy a képernyő előkészítése a felhasználó bejelentkezése előtt történik, a képernyőt előkészítő program a rendszergazda nevében fut.

DisplayManager.*.startup Az erőforrás segítségével megadható, hogy melyik program fusson le rendszergazdaként, miután a felhasználó bejelentkezett. Ezt a programot általában a bejelentkezés naplázására használhatjuk.

Az erőforrás nevében a * karakter helyén elhelyezhetjük a képernyők nevét (_0, _1 stb.), így minden képernyőhöz saját értéket adhatunk meg.

DisplayManager.*.session Az erőforrás segítségével megadhatjuk, hogy a felhasználó bejelentkezése után melyik program fusson le a felhasználó nevében.

Ennek a programnak a feladata roppant fontos, nem más, mint a munkafelület biztosítása. A képernyőkezelő a sikeres belépés után az azonosított felhasználó nevében elindítja a DisplayManager.*.session erőforrásban található nevű állományt, és a felhasználót kilépettnek tekinti, ha az állományban tárolt program befejezi a futását.

A * helyére beírhatjuk az egyes képernyők nevét (_0, _1 stb.), így minden képernyőre egyedi munkafelületet állíthatunk be.

DisplayManager.*.reset Az erőforrás segítségével megadhatjuk, hogy melyik program fusson le rendszergazdaként, amikor a felhasználó kijelentkezett.

A * helyére beírhatjuk az egyes képernyők nevét (_0, _1 stb.).

DisplayManager.accessFile Az erőforrás megadja, hogy melyik állományban tároljuk a távoli számítógépek neveihez tartozó jogokat. Az állományban megadhatjuk, hogy mely távoli számítógépeknél van jog a grafikus bejelentkező képernyőt kérni a képernyőkezelőtől.

`DisplayManager.willing` Az erőforrás segítségével megadható, melyik programot használjuk rövid bemutatkozószöveg kiírására.

Ha az alhálózatunk valamelyik számítógépe összegyűjt, hogy mely számítógépek vállalnak különböző grafikus kapcsolatkéréseket, ez a program lefut, és az általa a szabványos kimenetre kiírt szöveg megjelenik a felhasználó számára bemutatott listában.

A bejelentkező ablak megjelenése

Amint láttuk, az X képernyőkezelő a felhasználó azonosítása előtt betölti a `DisplayManager.*.resources` erőforrásban tárolt néven található állományból az erőforrásokat. Ezek között az erőforrások között szokás beállítani a felhasználó azonosítására használt bejelentkező ablak megjelenésére és viselkedésére vonatkozó erőforrásokat is.

A bejelentkező ablak megjelenésének megadásakor szám vagy szöveg jellegű értékeket rendelünk az egyes erőforrásokhoz. Ha betűtípusokat szeretnénk meghatározni, akkor az adott betűtípus vagy betűtípusok X Window számára megszokott neveit kell megadnunk (lásd a 10.2.1. példát a 291. oldalon). Ha színeket szeretnénk meghatározni, az `rgb.txt` állományban található neveket vagy a színkeverés kódját kell megadnunk. A színkeverés kódja a # karakter után írt vörös, zöld és kék színösszetevők tizenhatos számrendszerben megadott súlyával történhet (például #4444aa).

A bejelentkező ablak megjelenését és viselkedését befolyásoló erőforrások közül a legfontosabbak a következők:

`xlogin.Login.width` A bejelentkezőablak szélessége képpontban.

`xlogin.Login.height` A bejelentkező ablak magassága képpontban.

`xlogin*borderWidth` Az ablakkeret vastagsága képpontban.

`xlogin.Login.x` Az ablak bal szélének távolsága a képernyő bal szélétől.

`xlogin.Login.y` Az ablak tetejének távolsága a képernyő tetejétől.

`xlogin*greetColor` Az ablak tetején használt üdvözlő szöveg színe.

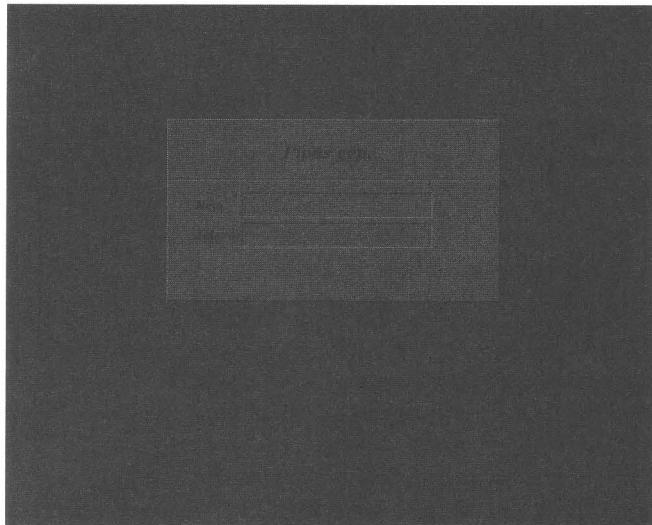
`xlogin.Login.foreground` A felhasználó által begépelezett szöveg színe.

`xlogin.Login.background` Az ablak háttérszíne.

`xlogin.Login.failColor` A hibás jelszó begépelésekor megjelenő szöveg színe.

`xlogin.Login.greeting` Az ablak tetején megjelenő üdvözlő szöveg.

Az üdvözlő szövegen használhatjuk a `CLIENTHOST` vagy `HOST` szavakat a számítógép nevének rövidítéseként.



10.11. ábra. Az X képernyőkezelője

`xlogin.Login.namePrompt` A név kérésére használt szöveg.

`xlogin.Login.passwdPrompt` A jelszó kérésére használt szöveg.

`xlogin.Login.fail` A sikertelen bejelentkezés jelzésére használt hibaüzenet.

`xlogin.Login.Font` Az ablakban használt betű típusa.

`xlogin.Login.greetFont` Az üdvözlő szöveg betűtípusa.

`xlogin.Login.promptFont` A név és jelszó bekérését jelző szöveg betűtípusa.

`xlogin.Login.failFont` A hibás jelszó begépelését jelző szöveg betűtípusa.

`xlogin.Login.allowRootLogin` A rendszergazda belépésének engedélyezése.

Ha ennek az erőforrásnak az értéke `false`, a rendszergazda nem léphet be a grafikus felületen.

`xlogin.Login.allowNullPasswd` Az üres jelszó használatának engedélyezése.

Ha ennek az erőforrásnak az értéke `true`, akkor azok is beléphetnek, akiknek a jelszava egyetlen karaktert sem tartalmaz.

145. példa Készítsünk erőforrásleíró állományt, amely az X képernyőkezelője számára állítja be a bejelentkező ablak megjelenését (a bemutatott állományban a sorokat nyomdai okokból rövidítettük)!

```
1 # A szöveget hordozó erőforrások
2 xlogin*greeting: Pipás gépe
3 xlogin*namePrompt: Név:
4 xlogin*passwdPrompt: Jelszó:
5 xlogin*fail: Hibás jelszó
6
7 # Betűtípusok
8 xlogin*greetFont: *-new century schoolbook-\bold-i-normal-*-240-
9
10 xlogin*font: *-new century schoolbook-\bold-i-normal-*-180-
11
12 xlogin*promptFont: *-new century schoolbook-\bold-i-normal-*-180-
13
14 xlogin*failFont: *-new century schoolbook-\bold-i-normal-*-180-
15
16
17 # Színek
18 xlogin*borderWidth: 1
19 xlogin*frameWidth: 2
20 xlogin*innerFramesWidth: 2
21 xlogin*background: #2222aa
22 xlogin*shdColor: #0000aa
23 xlogin*hiColor: #4444aa
24 xlogin*foreground: Black
25 xlogin*greetColor: Black
26 xlogin*failColor: Red
```

Keressük meg, hogy melyik állományból tölti be a képernyőkezelő az erőforráskat a felhasználó bejelentkezése előtt, és másoljuk be az állományt ide!

```
$ grep resources: /etc/X11/xdm/xdm-config
DisplayManager*resources:          /etc/X11/xdm/Xresources
$ cp res.txt /etc/X11/xdm/Xresources
$
```

Az xdm újraindítása után a 10.11. ábrán látható megjelenéssel indul el a grafikus bejelentkező ablak.

A munkafelület biztosítása

Amint láttuk, az X képernyőkezelő sikeres azonosítás után a felhasználó nevében elindítja a `DisplayManager.*.session` erőforrásban tárolt nevű programot, hogy a felhasználó számára a munkafelületet biztosítsa. Ennek a programnak tehát igen fontos feladata van.

A munkafelületet biztosító program – amelyet az xdm-config állományban adunk meg – a felhasználó által használt felületet alapvetően meghatározza. Mivel sok részletre kell ügyelnie a programnak, és könnyen módosíthatónak kell lennie, a legtöbb GNU/Linux terjesztés valamelyen héjprogramot használ erre a célra.

146. példa A következő egyszerű héjprogram a felhasználó számára grafikus felületet biztosít. A program alkalmas arra, hogy a képernyőkezelővel együttműködve a szokásos módon elindítson egy ablakkezelőt.

```
1 #!/bin/bash -login
2 #
3 # Ez az állomány a felhasználó azonosítása után fut,
4 # a felhasználó névében. Amikor ez a program befejezi
5 # futását, a grafikus felület új bejelentkező ablakot
6 # jelenít meg.
7 #
8
9 # A háttér beállítása feketére
10 xsetroot -solid Black
11
12 #
13 # Adjunk hozzá az elérési úthoz néhány könyvtárat, amelyek
14 # a grafikus felület programjait tartalmazzák.
15 #
16 ADDTOPATH="/usr/X11R6/bin /usr/local/bin"
17 for TOADD in $ADDTOPATH; do
18     if ! printenv PATH | grep -q $TOADD; then
19         PATH="${PATH}":$TOADD
20     fi
21 done
22
23 #
24 # Betöltsük a mindenire érvényes és a felhasználó
25 # által a saját könyvtárban elhelyezett erőforrásokat.
26 #
27 RESOURCES="/etc/X11/Xresources $HOME/.Xresources \
28 $HOME/.xresources"
29 for RES in $RESOURCES; do
30     if [ -f $RES ]; then
31         xrdb -merge $RES
32     fi
33 done
34
35 # Ha vannak egyéb teendők, azokat is elvégezzük.
36 for INIT in /etc/X11/xinit/xinitrc.d/* ; do
37     if [ -x "$INIT" ]; then
```

```

38      .  "$INIT"
39      fi
40 done
41
42 #
43 # Nézzük meg, hogy milyen programot tudunk indítani
44 # a munkafelület biztosítására!
45 #
46 INITSCS="$HOME/.xsession $HOME/.Xclients \
47 /etc/X11/xinit/Xclients"
48 for INITS in $INITSCS; do
49   if [ -x $INITS ]; then
50     exec $INITS
51   fi
52 done
53
54 #
55 # Ha semmit sem találtunk, indítsuk az xterm
56 # programot.
57 #
58 exec xterm

```

Figyeljük meg, hogy a példaprogram a \$HOME/.xsession, a \$HOME/.Xclients és a /etc/X11/xinit/Xclients állományokban keresi a munkafelületet megvalósító programot!

Könnyedén készíthetünk olyan héjprogramot, amely megkeresi a számítógépre telepített ablakkezelőket és az elsőt, amelyet megtalált, elindítja. A következő program ezt teszi, és a /etc/X11/xinit/Xclients állományban gondoskodik a munkafelület elindításáról:

```

1 #!/bin/bash
2 # Ez az állomány elindít egy ablakkezelőt, ha talál.
3 #
4 #
5 # Soroljuk fel itt azokat az ablakkezelőket, amelyeket
6 # keresünk!
7 #
8 AKEZELOK="icewm qvwm fvwm fvwm2 xterm"
9 for AK in $AKEZELOK; do
10   for KVT in $(echo $PATH | tr : " "); do
11     if [ -x "$KVT/$AK" ]; then
12       exec "$KVT/$AK"
13     fi
14   done
15 done

```


Irodalomjegyzék

- [1] Werner Almesberg. *LIO Generic boot loader for Linux, Technical overview*.
- [2] Werner Almesberg. *LIO Generic boot loader for Linux, User's guide*.
- [3] Edward C Bailey. *Maximum RPM*. Red Hat Press, Indianapolis, IN, USA, 1997.
- [4] Remy Card, Thodore Ts'o, and Stephen Tweedie. Design and implementation of the second extended filesystem. In *Proceedings of the First Dutch International Symposium on Linux*, number ISBN 90 367 0385 9, <http://web.mit.edu/tysso/www/linux/ext2intro.html>. Laboratoire MASI — Institut Blaise Pascal and Massachussets Institute of Technology and University of Edinburgh.
- [5] Mike G. Bash programming - introduction how-to. World-Wide Web document. Also available in Hungarian [6] translation.
- [6] Mike G. Bevezetés a bash programozásba hogyan. World-Wide Web document. Hungarian translation by Kovács Ferenc [5].
- [7] Daniel Gilly and Tim O'Reilly. *The X Window System in a Nutshell: Desktop Quick Reference for Version 11 Release 4 of the X Window System*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, 1990.
- [8] Daniel Gilly and Tim O'Reilly. *The X Window System in a Nutshell*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, second edition, 1992. Updated for X11 R5 by Ellie Cutler.
- [9] Gnome project homepage. World-Wide Web document.
- [10] Paul Gortmaker. The linux bootprompt-howto. World-Wide Web document.
- [11] Tony Harris and Kristian Koehntopp. Linux partition howto. World-Wide Web document.

- [12] Russel Ingram. Linux + xfs hogyan. World-Wide Web document. Hungarian translation by Daczi László of [13].
- [13] Russel Ingram. Linux + xfs howto. World-Wide Web document. Also available in Hungarian [12] translation.
- [14] Thorsten Kukuk. Linux nis(yp)/nys/nis+ hogyan. World-Wide Web document. Hungarian translation by Bábos Balázs of [15].
- [15] Thorsten Kukuk. The linux nis(yp)/nys/nis+ howto. World-Wide Web document. Also available in Hungarian [14] translation.
- [16] Gordon Matzigkeit and Okuji Yoshinori. *The GRand Unified bootloader*.
- [17] Linda Mui and Eric Pearce. *Volume 8: X Window System Administrator's Guide*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, October 1992.
- [18] Cameron Newham and Bill Rosenblatt. *Learning the bash Shell*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, October 1995.
- [19] Andrew Oram and Steve Talbott. *Managing Projects with make*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, second edition, October 1991.
- [20] Chet Ramey. What's GNU: bash — the GNU shell, part 1 of 2. *Linux Journal*, 3, July 1994.
- [21] R. W. Scheifler. RFC 1013: X Window System Protocol, version 11: Alpha update April 1987, June 1987. Status: UNKNOWN.
- [22] Miroslav „Misko” Skoric. Lilo mini-hogyan. World-Wide Web document. Hungarian translation by Horváth Albert of [23].
- [23] Miroslav „Misko” Skoric. Lilo mini-howto. World-Wide Web document. Also available in Hungarian [22] translation.
- [24] Winfried Trümper. Cd-writing howto. World-Wide Web document.
- [25] Stephen C. Tweedie. Journaling the linux ext2fs filesystem. Technical report, LinuxExpo 98, 1998.
- [26] Ralf van Dooren. Kvóta mini-hogyan. World-Wide Web document. Hungarian translation by Szijjártó László of [27].

- [27] Ralf van Dooren. Quota mini-howto. World-Wide Web document. Also available in Hungarian [26] translation.
- [28] David A. Wheeler. Program library howto. World-Wide Web document. Also available in Hungarian [29] translation.
- [29] David A. Wheeler. Programkönyvtár hogyan. World-Wide Web document. Hungarian translation by Szalai Ferenc Attila of [28].

Tárgymutató

(), 209, 210	/boot/map, 102, 104
(fd0), 110	/cdrom/, 35, 76
(hd0), 110	/contrib/, 34, 35
(hd0,0), 109, 110	/dev/, 20, 32, 109
(hd0,4), 110	/dev/cua0, 299
(hd1,0), 110	/dev/cua1, 299
*, 274, 291, 329, 333, 334	/dev/fb0, 302
+, 86	/dev/fb1, 302
+1, 115	/dev/fd0, 22
-, 176, 229	/dev/fd1, 22
-bash, 158	/dev/hda, 20, 21
., 42, 43, 46, 160, 229, 230	/dev/hda1, 108, 116
.., 42, 43, 46	/dev/hda2, 22
./configure, 244	/dev/hda3, 60
.SUFFIXES, 210, 211	/dev/hda6, 73
.Xresources, 332	/dev/hdb, 20
.bash_login, 159	/dev/hdc, 20
.bash_logout, 159	/dev/hdd, 20, 21
.bash_profile, 159	/dev/initctl, 130
.bashrc, 159, 160	/dev/loop0, 65, 66
.hushlogin, 156	/dev/loop1, 65
.profile, 159	/dev/psaux, 299
.rej, 217	/dev/scd0, 21
.so, 227	/dev/scd1, 21
/, 41, 42, 181	/dev/scd2, 21
/bin/, 31, 126, 127	/dev/sda, 21, 50
/bin/bash, 120, 159	/dev/sda1, 52, 56, 70
/bin/gzip, 43	/dev/sdb, 21
/bin/sh, 160	/dev/sdb6, 22
/boot/, 32, 37, 100	/dev/sdc, 21
/boot/boot.NNNN, 102	/etc/, 32, 74, 135
/boot/boot.b, 102, 103	/etc/X11/, 294, 295
/boot/chain.b, 102	/etc/X11/fs/config, 292
/boot/grub/, 112	/etc/X11/xdm/, 332

/etc/X11/xinit/Xclients, 339
/etc/X11/állománynév, 294
/etc/X11/, 294
/etc/X11/XF86Config, 294
/etc/X11/XF86Config-4, 294
/etc/X11/\$XF86CONFIG, 294
/etc/X11/\$XORGCONFIG, 294
/etc/X11/xorg.conf, 295
/etc/X11/xorg.conf-4, 295
/etc/X11/állománynév, 294
/etc/apt/sources.list, 269, 277
/etc/init.d/keymaps-lct.sh, 193
/etc/security/console.app/, 170
/etc/security/limits.conf, 175,
 177
/etc/xorg.conf, 295
/etc/bashrc, 160
/etc/console-tools/, 193
/etc/console-tools/config, 193
/etc/default/useradd, 145
/etc/fstab, 14, 74, 77, 81, 83, 122,
 180, 182
/etc/group, 140, 141, 143, 152
/etc/gshadow, 143, 152
/etc/init.d/, 134–136
/etc/init.d/rc.local, 154
/etc/init.d/xfs, 292
/etc/inittab, 126, 130–132
/etc/issue, 153, 154
/etc/issue.net, 153, 154
/etc/ld.so.cache, 226, 228
/etc/ld.so.conf, 228
/etc/lilo.conf, 98, 102
/etc/motd, 154, 174
/etc/mtab, 74, 75
/etc/nologin, 155, 174
/etc/nsswitch, 172, 173
/etc/pam.conf, 167
/etc/pam.d/, 167
/etc/pam.d/xeyes, 170
/etc/passwd, 140–143, 172
/etc/profile, 66, 67, 159–161,
 163, 164
/etc/rcn.d/, 132

/etc/rc.d/rc, 131–133
/etc/rc.d/rc.local, 154
/etc/rc1.d/, 135
/etc/rc2.d/, 135
/etc/securetty, 155, 156, 174
/etc/security/opasswd, 173
/etc/shadow, 143, 152, 172
/etc/shells, 148
/etc/skel/, 144, 145, 178
/etc/sysconfig/i18n, 193
/etc/sysconfig/keyboard, 193
/floppy/, 35
/grub/, 109, 112
/home/, 14, 33, 34, 37, 76, 82, 181
/home/fred/, 49
/lib/, 32, 226
/mnt/, 35
/mnt/cdrom/, 66
/proc/, 35, 69, 74, 118
/proc/cmdline, 118
/proc/mounts, 74
/proc/swaps, 59, 60
/root/, 32
/sbin/, 31, 32, 126, 127, 169, 170
/sbin/fsck, 32
/sbin/init, 118
/sbin/nologin, 148, 149
/sbin/syslogd, 138
/tmp/, 32, 33, 37, 73
/usr/, 34, 37, 47, 74, 82
/usr/X11R6/etc/X11, 295
/usr/X11R6/etc/X11/, 294
/usr/X11R6/etc/X11/
 \$XF86CONFIG, 294
/usr/X11R6/etc/X11/
 \$XORGCONFIG, 295
/usr/X11R6/etc/X11/állománynév,
 294
/usr/X11R6/etc/X11/állománynév,
 294
/usr/X11R6/lib/modules/
 drivers, 301
/usr/X11R6/lib/modules/input/,
 298

/usr/doc/, 245
/usr/share/doc/, 245
/usr/src/redhat/SOURCES/, 239, 241, 246
/usr/src/redhat/SPECS/, 239, 246
/usr/bin/, 127, 170
/usr/lib/, 226
/usr/local/, 221
/usr/local/sbin/, 127
/usr/sbin/, 127
/usr/share/consolefonts/, 193
/var/, 35, 37
/var/cache/apt/archives/, 268, 273
/var/log/lastlog, 173
/var/run/, 132, 138
/var/run/xxxx.pid, 134
/var/run/utmp, 126
/var/spool/mail/, 33, 156, 173, 181
/var/tmp/, 32
;, 44, 214
<>, 280
=, 209
[], 268
#, 127, 167, 209, 235, 267, 298, 335
#include, 208
\$, 209
\$BASH_ENV, 160
\$EDITOR, 184, 186
\$HOME/.Xclients, 339
\$HOME/.bash_login, 159
\$HOME/.bash_profile, 159
\$HOME/.bashrc, 159, 160
\$HOME/.profile, 159
\$HOME/.xsession, 339
\$KEYTABLE, 193
\$LANG, 192
\$LCA_ALL, 192
\$LD_AOUT_LIBRARY_PATH, 226
\$LD_AOUT_PRELOAD, 225, 226
\$LD_LIBRARY_PATH, 226
\$LD_PRELOAD, 225, 226
\$PATH, 161, 169, 333, 334
\$PS1, 161
\$RPM_BUILD_ROOT, 248, 249, 251
\$VISUAL, 184, 186
\$XF86CONFIG, 294
\$XORGCONFIG, 294, 295
%{ARCH}, 233
%{BUILDHOST}, 233
%{BUILDTIME}, 233
%{DESCRIPTION}, 233
%{DISTRIBUTION}, 233
%{GROUP}, 233
%{INSTALLTIME}, 233
%{LICENSE}, 233
%{NAME}, 232
%{OS}, 233
%{PACKAGER}, 233
%{RELEASE}, 232
%{SIZE}, 233
%{SUMMARY}, 233
%{URL}, 233
%{VENDOR}, 233
%{VERSION}, 232
%build, 244
%clean, 244, 245, 249
%config, 245
%configure, 244
%description:, 241
%dir, 245
%doc, 245
%files, 245, 248, 249, 251
%install, 244, 249
%makeinstall, 251
%patch, 243
%post, 252
%postun, 252
%pre, 252
%prep, 242, 244
%preun, 252
%setup, 242, 243, 247
állományfoglaltsági táblázat, 70
állománynév-végződések
 .a, 224, 316
 .c, 201, 211
 .jigdo, 257–259

.o, 205, 211	sysinit, 128
.plan, 151	wait, 127
.rpm, 229, 230, 240, 242	0x00, 100
.rpmsave, 236	0x01, 100
.spec, 239	0x02, 100
.src.rpm, 239	0x03, 100
.txt, 297	0x04, 100
állományrendszer, 39	0x06, 100
_0, 333, 334	0x07, 100
_1, 333, 334	0x08, 100
{}, 44	0x09, 100
DefaultDepth szám, 309	0x0C, 100
Depth szám, 309	0x10, 101
/etc/fstab	0x11, 101
acl, 78	0x20, 101
async, 79	0x40, 101
atime, 79	0x80, 101
auto, 79	0x82, 50, 60
defaults, 79	0xBB, 101
dev, 79	1024x768, 310
exec, 80	4.0.12, 227
noatime, 79	5.2.11, 227
noauto, 79	5.3.0, 227
nodev, 80	640x480, 310
noexec, 80	800x600, 310
nosuid, 80	
nouser, 80	0, 125
owner, 81	1, 120, 125
ro, 80	2, 120
rw, 80	3, 125
suid, 80	5, 125, 237
sync, 79	6, 125
users, 81	8193, 55
user, 81	16384, 55
/etc/inittab	32768, 55
bootwait, 128	
boot, 128	a, 25
ctrlaltdel, 129	a.out, 224–226
initdefault, 129	AC_INIT(), 218
once, 128	AC_OUTPUT(), 218
powerfailnow, 129	AC_PROG_CC, 218
powerokwait, 128	Access, 282
powerwait, 128	access control list, 78, 82
respawn, 127	account, 168, 171–174

- AceCad, 299
ACL, 57, 78, 82–88, 90, 91
acl, 78, 82, 83
acllocal, 219
acllocal.m4, 219
adstyl, 291
advanced package tool, 266
agetty, 130
all, 222
alpha, 230
AM_INIT_AUTOMAKE(), 218
American Standard Code for Information Interchange, 188
apm, 302
append=, 105
APT, 266–269, 271, 276, 277, 282, 283
apt-cache, 277–279, 281, 282
apt-cdrom, 270
apt-get, 257, 259, 262, 270–274, 276, 277, 279, 280, 282, 285
aquota.group, 180–182
aquota.user, 180–182
ar, 224
ASCII, 188–190
ask, 119
async, 79
atime, 79
auth, 168, 171–174
AUTHORS, 219
Auto, 299
auto, 14, 79
autoclean, 272, 273
autoconf, 217, 218, 220, 221
automake, 217–221, 251
automake(), 218
AutoReqProv: no, 254
autoscan, 218
avgWidth, 291
awk, 142

b, 25
BASH, 158, 159

bash, 253
bash_completion, 273
betöltősáv, 94
betölthető, 25
bin, 234
binaries, 31
BIOS, 93–95, 98–102, 104
BIOS setup, 93
bit per pixel, 309
bitmap, 290
bitmap, 316
block special file, 19
blocks, 184
blokkeszköz-meghajtó állomány, 19
BoardName, 306
bold, 291
boot, 32
boot, 114, 115, 128
boot loader, 93
boot sector, 94
boot:, 96
boot=, 103
bootable, 25, 114
bootable CD-ROM, 63
bootwait, 128
build-dep, 272
BuildArch: *típus*, 241
BuildRequires: *eszköz1*, *eszköz2*, 255
BuildRoot:, 248
BuildRoot: *könyvtárnév*, 248
BusMouse, 299

c, 25, 237
címke, 77
cat, 60, 110, 112
cc, 200
CCW, 302
cd, 112
CD-ROM, 21, 35, 61, 63–68, 72, 78, 94, 99, 268
cdrecord, 64, 66, 67
cdrom, 121
cdrom:, 268

- CDRW, 67
chainloader, 115
change shell, 148
ChangeLog, 219
charset, 195
check, 272
chfn, 149
chips, 302
chkconfig, 136
chmod, 163, 164
chooser, 324
chsh, 148
clean, 213, 222, 272
CLIENTHOST, 335
cmp, 111
command, 207
compact, 100, 101, 103
condensed, 291
conf=állománynév, 175
Config, 283
config-files, 261
configfile, 111
configure, 220–222
configure.in, 218, 219
Conflicts:, 280
consolechars, 191
consolehelper, 169, 170
continue, 53
contrib, 268, 269
contribution, 34
copy:, 268
core, 162, 176
CoreKeyboard, 298, 312
CorePointer, 298, 312
cp, 91
cpio, 238
cpu, 176
CRC, 101
csereterület, 58
ctrlaltdel, 129
CW, 302
cylinder, 23, 94
d, 25, 53
data, 176
DBE, 318
dbe, 318
deb, 267
deb-src, 267
debconf, 265, 266
default, 90, 113
default.kmap, 193
default.kmap.gz, 193
default=, 103
DefaultDepth, 309
defaults, 79
delay=, 103, 104
depends, 280
Depends:, 280
dev, 79, 80
development, 255
Device, 296, 301, 305, 306, 309
devices, 32
df, 35, 36
diff, 214, 215, 217
dinamically loaded libraries, 228
dir=könyvtárnév, 173
direct rendering infrastructure, 319
Display, 309, 310
display power management services,
 308
DisplayManager.*.chooser, 333
DisplayManager.*.resources,
 333, 335
DisplayManager.*.reset, 334
DisplayManager.*.setup, 334
DisplayManager.*.startup, 334
DisplayManager.*.systemPath,
 333
DisplayManager.*.userPath, 334
DisplayManager.errorLogFile,
 333
DisplayManager.requestPort, 333
DisplayManager.*.reset, 333
DisplayManager.*.session, 334,
 337
Displaymanager.*.session, 334
Displaymanager.*.startup, 333

DisplayManager.accessFile, 334
DisplayManager.requestPort, 333
DisplayManager.servers, 333
DisplayManager.willing, 335
DisplaySize szélesség magasság, 308
dist, 222
dist-upgrade, 272, 277
distribution, 17
DMA, 100
docbook-utils, 236
dot per inch, 291
double buffer extension, 318
dpkg, 257, 259–266, 273, 276, 287
dpkg-reconfigure, 266
drdos8x16, 191
DRI, 319–322
DRI, 320
dri, 319–321
dselect, 282, 283
dump, 78, 279
dumpavail, 277, 279
dumpkeys, 190, 192, 194
DVD, 64

e2fsck, 52, 54–56
editres, 330
edquota, 184, 185
El Torito, 64
ELF, 224, 225, 228
elsődleges lemezrész, 19
elsődleges- és másodlagos esz-közszám, 20
encdng, 291
EndSection, 295
et cetera, 32
exec, 79, 80
executable and linking format, 224
ExplorerPS/2, 299
export, 161
ext2, 72, 75
ext3, 72, 75
extended 2 filesystem, 40
extended partition, 19
extmod, 317, 318
fájlnév, 43
FAT, 70, 109
fb, 302
fbdev, 302
fdbewhw, 318
fdisk, 23–25
felhasználói adatbázis, 139
FHS, 30
file allocation table, 70
file:, 268
file=állománynév, 174
Files, 296, 297, 316
filesystem, 39
filesystem hierarchy standard, 30
find, 43–45, 47, 49, 111, 151
find-provides, 254
find-requires, 254
finger, 150, 151
fmy, 291
fndry, 291
font server, 290
FontPath, 297
fonts.dir, 292, 297
frame buffer, 302
frame per second, 321
freetype, 316
frozen, 269
fsck, 31, 78
fsize, 176
FTP, 268
ftp, 148
ftp://, 235, 268
fuser, 76
futási szintek
 0, 125
 1, 125
 3, 125
 5, 125
 6, 125
 S, 125
G, 120, 237

Gamma vörös zöld kék, 308
gcc, 200, 202, 203, 205
gdm, 322–324
gdmphotosetup, 326
gdmsetup, 324
getfacl, 83, 84
GID, 41, 139
GlidePoint, 299
GlidePointPS/2, 299
glint, 303
GLX, 319–321
glx, 319–321
glxgears, 320–322
glxinfo, 320, 321
Gnome display manager, 322
gpm, 197
grace period, 184
grand unified boot loader, 108
grand unified bootloader, 93
grep, 234
group, 84, 85, 152
Group: csoport, 240
group:csop:jogok, 85
groupadd, 151
groupdel, 151
groups, 151
grpck, 152
grpconv, 152
grpquota, 180, 182
grpunconv, 152
GRUB, 93, 108–110, 112–114, 116
grub, 110, 112
GRUB parancsok
 boot, 114, 115
 cat, 110
 chainloader, 115
 cmp, 111
 configfile, 111
 default, 113
 find, 111
 halt, 111
 help, 111
 hiddenmenu, 113
 initrd, 114
 install, 111
 kernel, 114
 makeactive, 114
 md5crypt, 113
 password, 113
 quit, 112
 reboot, 112
 rootnoverify, 114
 root, 112, 114
 savedefault, 113
 saved, 113
 setup, 112
 splashimage=, 113
 testvbe, 112
 timeout, 113
 title, 114
 vbeprobe, 112
grub.conf, 112
gshadow, 152
guests, 177
gyökérkönyvtárat tartalmazó lemez-
 rész, 31
gzip, 47, 113
half-configured, 261
half-installed, 261
halt, 111, 112
hard, 162
hard, 176, 184
hard disk, 20
hda1, 124
hda2, 124
head, 94
help, 111
henger, 23
hiddenmenu, 113
home, 33
HorizSync, 307
HorizSync frekvenciák, 307
HOST, 335
HTTP, 268
http://, 235, 268
http://www.debian.org, 257, 259
httpd, 135

- hu, 191, 299
hu101, 191
hu_qwerty, 299
hush, 156
Hz, 307
i, 291
I128, 303
i128, 303
i386, 230
i586, 230
i810, 303
IBM, 18, 19, 72, 93
IDE, 20–22, 109
ide-cd, 121
Identifier, 309, 312
image.xpm.gz, 109
image=, 104, 106
ImPS/2, 299
init, 121, 126–132, 138, 182
init=, 118
init=/bin/bash, 120
initdefault, 129
initial RAM disk, 119
initrd, 114
initrd=, 119
inode, 40
inodes, 184
InputDevice, 296, 298, 300
Install, 283
install, 111, 112, 222, 272
install=, 103
installed, 261
integrated device electronics, 20
IntelliMouse, 299
interactive shell mode, 157
ISO-8859, 189
ISO-8859, 194
ISO-8859-1, 189
ISO-8859-2, 189
iso-8859-2, 194
iso-8859-X, 192, 194
ISO-9660, 61–64, 72
ISO-9958, 189
iso9660, 61–63, 65–67, 72
italic, 291
jfs, 72
jigdo, 257
jigdo-lite, 258, 259
JPEG, 223
K, 120
kbd, 299
kernel, 114
keyboard, 298, 299
keycode, 195
keymaps, 195, 196
kHz, 307
kibővített lemezrész, 19
L, 99
l, 25
látszólagos memória, 59
label, 77
LABEL=, 81
label=, 104
LBA, 104
lba32, 104
ld, 205, 224
ld.so, 225–227
ldconfig, 226–228, 252
ldd, 226
lemezrész-táblázat, 18
lemezrészek, 17
LI, 99
lib, 227, 316
libbitmap.a, 316
libraries, 32
library, 227
License: engedély, 240
LIL, 99
LIL-, 99
LIL?, 99
LILO, 93, 95–103, 106
lilo, 97–103
LILO hibakódok
 0x00, 100

0x01, 100
0x02, 100
0x03, 100
0x04, 100
0x06, 100
0x07, 100
0x08, 100
0x09, 100
0x20, 101
0x40, 101
0x80, 101
0xBB, 101
LILO parancsok
append=, 105
boot=, 103
compact, 103
default=, 103
delay=, 103, 104
image=, 104
install=, 103
label=, 104
lba32, 104
linear, 103
literal=, 105
lock, 104
map=, 104
message=, 104
other=, 104
password=, 105
prompt, 104
read-only, 105
read-write, 105
restricted, 105
root=, 105
vga=, 105
lilo.conf, 106
linear, 100, 101, 103
linking, 205
Linux loader, 93
linuxrc, 122
literal=, 105
ln, 228
loadkeys, 190, 191, 193
lock, 104
locks, 176
logical block addressing, 104
logical partition, 19
logikai lemezrész, 19
login, 154–159, 168
login shell mode, 157
Logitech, 299
losetup, 65
lpd, 135
ls, 14, 41, 43, 45, 49, 83, 84, 86
lsof, 76
M, 120, 237
m, 24, 25, 53, 85, 203
main, 268, 269
main(), 204
main.c, 201, 204, 205, 208, 218,
 219
main.o, 205, 207, 211
major device number, 20
make, 206–214, 217, 219, 221
makeactive, 114
Makefile, 206, 208, 209, 211, 218,
 220, 221, 251
makefile, 206
Makefile.am, 219
Makefile.in, 219, 220
man, 202, 298, 302
map installer, 98
map=, 104
mask, 84–86
mask:jogok, 85
master boot record, 95
math.h, 202
maxlogins, 176
MBR, 95
md5, 172
md5crypt, 113
medium, 291
mem=, 120
memlock, 176
message=, 104
Meta_asciitilde, 196
mga, 303

MHz, 307
Microsoft, 299
microsoft, 299
minor device number, 20
mkdosfs, 70
mke2fs, 50, 51, 55
mkfontdir, 293
mkfs.ext2, 50, 51
mkfs.xfs, 57–59
mkinitrd, 118, 119, 122
mkisofs, 63–65, 67
mknod, 20
mkswap, 60, 61
mkzftree, 64, 67, 68
MMHitTab, 299
MMSeries, 299
Mode, 308, 310
Module, 296, 316
ModulePath, 316
Monitor, 296, 306, 309, 310
MOTD, 154
motd=állománynév, 174
mount, 31, 58, 73–76, 78, 79, 180
mouse, 299, 300
MouseMan, 299
MouseManPlusPS/2, 299
MouseSystems, 299
mplayer, 317
msdos, 70–72
mutatópont, 40

n, 25, 291
név1, 45
név2, 45
Name: név, 240
ncpfs, 72
neomagic, 303
NetMousePS/2, 299
NetScrollPS/2, 299
netselect, 271
NEWS, 219
nfs, 72
NIS, 49, 173, 178
nis, 173

NIS+, 49
nm, 228
noarch, 230, 241, 247
noatime, 79
noauto, 79
nobody, 140
nodelay, 172
nodev, 80, 81
noexec, 80, 81
nofile, 176
non-free, 268
non-us, 269
none, 77, 82
normal, 291
normal, 291
nosuid, 80, 81
nouser, 79, 80
nproc, 176
ntsysv, 136
nullok, 172
nv, 303

o, 25, 85, 90
objdump, 228
object code, 205
once, 128
open graphics library, 318
open source, 200
OpenGL extensions to the X Window System, 319
Option BlankTime szám, 313
Option OffTime szám, 313
Option StandbyTime szám, 313
Option SuspendTime szám, 313
optional, 169
osszeg(), 204
other, 84, 85, 89, 168
other:jogok, 85
other=, 104
owner, 81

p, 25
PAM, 167, 168, 171, 174, 178
pam_cracklib.so, 173, 175

pam_deny.so, 171
pam_lastlog.so, 173
pam_limits.so, 175
pam_mail.so, 173
pam_mkhomedir.so, 178
pam_motd.so, 174
pam_nologin.so, 174
pam_permit.so, 171
pam_securetty.so, 174
pam_unix.so, 172
pam_warn.so, 171
panic, 53
partition, 17
partition table, 18
passwd, 145, 146, 152, 168
password, 113, 168, 171–173
password=, 105
patch, 215–217, 244
Patch: *URL*, 241
PATH, 127
PC, 18, 19
pc101, 299
pc102, 299
PCI, 301, 306
PID, 126
Pluggable Authentication Modules
 for Linux, 166
powerfailnow, 129
powerokwait, 128
powerwait, 128
prepare, 242
PREVLEVEL, 132
primary partition, 19
printf(), 201, 203, 232
proba, 207
proba.c, 204, 205
proba.h, 204, 208, 211
proba.o, 205, 211
proc, 69, 70, 72, 74, 77, 82, 118
processes, 35
programok
 cdrecord, 67
 chfn, 149
 df, 36
dumpkeys, 192
e2fsck, 55
fdisk, 23
find, 44
gcc, 205
getfacl, 83
init, 130
lilo, 98
loadkeys, 190
login, 157
losetup, 65
mke2fs, 51
mkfs.ext2, 51
mkfs.xfs, 59
mkinitrd, 118
mkisofs, 64
mknod, 20
mkswap, 61
mkzftree, 68
mount, 75
passwd, 146
quota, 187
quotacheck, 181
quotaon, 183
repquota, 188
runlevel, 126
setfacl, 87
setquota, 186
stat, 41
swapoff, 60
swapon, 60
sync, 40
telinit, 130
tune2fs, 53
ulimit, 162
umask, 164
umount, 75
useradd, 144
userdel, 147
usermod, 150
prompt, 161
prompt, 103, 104
Provides: név1, név2, 255
PS/2, 299

- PS/2, 299
- ptSz, 291
- pwck, 152
- pwconv, 152
- pwunconv, 152
- pxlsz, 291
- q, 26
- Quit, 283
- quit, 112
- quota, 178
- quota, 186, 187
- quota.group, 179, 181
- quota.user, 179, 181
- quotacheck, 180–182
- quotaoff, 181
- quotaon, 181–183
- r, 84
- rövidítések
 - ACL, 57, 78, 82–88, 90, 91
 - APT, 266–269, 271, 276, 277, 282, 283
 - ASCII, 188–190
 - BASH, 158, 159
 - BIOS, 93–95, 98–102, 104
 - CD-ROM, 21, 35, 61, 63–68, 72, 78, 94, 99, 268
 - CRC, 101
 - DBE, 318
 - DMA, 100
 - DRI, 319–322
 - DVD, 64
 - ELF, 224, 225, 228
 - FAT, 70, 109
 - FHS, 30
 - FTP, 268
 - GID, 41, 139
 - GLX, 319–321
 - GRUB, 93, 108–110, 112–114, 116
 - HTTP, 268
 - IBM, 18, 19, 72, 93
 - IDE, 20–22, 109
 - ISO-8859, 189
 - ISO-8859-1, 189
 - ISO-8859-2, 189
 - ISO-9660, 61–64, 72
 - ISO-9958, 189
 - JPEG, 223
 - LBA, 104
 - LILO, 93, 95–103, 106
 - MBR, 95
 - MOTD, 154
 - NIS, 49, 173, 178
 - NIS+, 49
 - PAM, 167, 168, 171, 174, 178
 - PC, 18, 19
 - PCI, 301, 306
 - PID, 126
 - PS/2, 299
 - RAM, 69, 70, 72, 119, 121–123
 - SCSI, 20, 21, 23, 66, 109, 118
 - SCSI CD-ROM, 21
 - SGID, 80
 - SMB, 72
 - SPARC, 230, 258
 - SUID, 80
 - TCP, 327
 - TrueType, 316
 - Type1, 317
 - UDF, 64
 - UDP, 333
 - UID, 40, 139
 - URL, 241
 - UTF, 189
 - UTF-16, 190
 - UTF-32, 189, 190
 - UTF-8, 190
 - VESA, 112, 119, 304, 305
 - VGA, 105, 119, 191, 305
 - X11R6, 289
 - XDMCP, 323, 324, 328
 - XFS, 57–59, 82, 83, 179
 - radeon, 304
 - RAM, 69, 70, 72, 119, 121–123
 - ramdisk, 122
 - ramfs, 69, 72

- read-only, 105
read-write, 105
readelf, 228
README, 219
real time, 57
reboot, 112
Recommends:, 280
record, 318
Red Hat package manager, 229
redhat-config-services, 136
reiserfs, 72
Release: változatszám, 240
remember=n, 173
remount-ro, 53
Remove, 283
remove, 272, 276
Replaces:, 280
repquota, 187, 188
required, 168, 169
Requires: eszköz1, eszköz2, 254
requisite, 169
resources, 328
respawn, 127, 130
restricted, 105
resx, 291
resy, 291
retry=n, 173
rgb.txt, 297, 335
RgbPath, 297
rgstry, 291
rm, 249
ro, 80, 119
root, 32
root, 112, 114, 139, 140
root partition, 31
root=, 105, 108, 119, 124
rootflags=, 120
rootfstype=, 120
rootnoverify, 114
rpm, 229–240, 242, 244, 245, 247–
249, 251–255, 266–268,
273, 274, 285
rpm csomagleíró állomány
 AutoReqProv: no, 254
BuildArch: típus, 241
BuildRequires: eszköz1,
 eszköz2, 255
BuildRoot: könyvtárnév, 248
Group: csoport, 240
License: engedély, 240
Name: név, 240
Patch: URL, 241
Provides: név1, név2, 255
Release: változatszám, 240
Requires: eszköz1, eszköz2,
 254
Source: URL, 241
Summary: leírás, 240
Version: változatszám, 240
%build, 244
%clean, 244
%configure, 244
%config, 245
%description:, 241
%dir, 245
%doc, 245
%files, 245
%install, 244
%makeinstall, 251
%patch, 243
%postun, 252
%post, 252
%prep, 242
%preun, 252
%pre, 252
%setup, 242
rpm-src, 268
rpm2cpio, 238
rpmbuild, 239, 241–245, 247–249,
 254
rss, 176
RUNLEVEL, 132
runlevel, 125
runlevel, 126
rw, 79, 80, 119
S, 120, 125, 237
s, 26

s3virge, 304
sans serif, 291
savage, 304
saved, 113
savedefault, 113
scalable, 290
scan code, 193
Screen, 296, 309, 310, 312
Screen szám, 301
SCSI, 20, 21, 23, 66, 109, 118
SCSI CD-ROM, 21
SCSI disk, 20
search, 277, 279
Section, 295
sector, 95
Select, 282, 283
serif, 291
ServerLayout, 296, 311
session, 168, 171–175
setfacl, 85–87
setfont, 191
setquota, 185, 186
setup, 111, 112
SGID, 80
SGID, 226
shadow, 152
shadow password, 142
shared object, 227
shared objects, 225
show, 277, 282
showkey, 196
showkeys, 196
showpkg, 277, 281
showsrd, 277
SIGKILL, 130
signal, 126
SIGTERM, 130
siliconmotion, 304
sin(), 202, 203
sis, 304
slant, 291
small computer system interface, 20
SMB, 72
smbfs, 72
soft, 162
soft, 176, 184
software library, 223
source, 199
source, 160, 272
Source: URL, 241
sources.list, 266–268, 271
SPARC, 230, 258
sparc, 230
sparc64, 230
spc, 291
speedo, 317
splashimage, 112
splashimage=, 113
src, 217, 230
src.rpm, 239
src/, 219
src/Makefile, 218
src/Makefile.am, 219
src/Makefile.in, 220
ssh, 156
stable, 256, 257, 268, 269, 274,
 277
stack, 176
start, 134
stat, 41, 42
static libraries, 224
stats, 277, 278
stdio.h, 201
stop, 134, 135, 138
subsystem, 131
sufficient, 169
Suggests:, 280
SUID, 80
SUID, 226
suid, 79, 80
Summary: leírás, 240
swap, 58, 72, 78
swap space, 58
swapoff, 60
swapon, 60
sWdth, 291
synaptic, 283, 285, 287
sync, 40, 79

- sysinit, 128, 129
syslogd, 138
SysMouse, 299
system binaries, 31
system wide user profile, 161
szektor, 23
- T, 237
t, 26
tar, 47
TCP, 327
tcp/gépnév:port, 297
telinit, 130
temporary, 32
testing, 256, 257, 268, 269, 274, 277
testvbe, 112
ThinkingMouse, 299
ThinkingMousePS/2, 299
timeout, 113
title, 114
to mount, 35
TrueType, 316
try_first_pass, 172
ttmkfdir, 293
tune2fs, 51–53, 55, 56
Type1, 317
type1, 317
- U, 237
u, 26, 85
UD, 302
UDF, 64
UDP, 333
UID, 40, 139
ulimit, 161–163
umask, 163, 164
umount, 75, 76
umsdos, 72
unicode transformation format, 189
unicode_start, 192
unicode_stop, 192
uninstall, 222
unix/:port, 297
- unmet, 277, 279
unpacked, 261
unstable, 256, 268, 269, 274
Update, 282
update, 271, 272, 274
upgrade, 271, 272, 274, 277
URL, 241
us, 299
USB, 299
use_authok, 173
use_authtok, 173
use_first_pass, 172
user, 81, 84, 85
user resources, 34
user:felh:jogok, 85
useradd, 144, 145
userdel, 147
usermod, 149, 150
username, 139
users, 81
usrquota, 180, 182
UTF, 189
UTF-16, 190
UTF-32, 189, 190
UTF-8, 190
utmp, 156
- v, 26
ValuMouseScroll, 299
variable, 35
vbeprobe, 112
Vendorname, 306
Version: változatszám, 240
VertRefresh frekvenciák, 307
VESA, 112, 119, 304, 305
vesa, 304, 305
vfat, 71, 72
VGA, 105, 119, 191, 305
vga, 305
vga=, 105, 119
via, 305
video electronics standards association, 119
video graphics array, 119

VideoRam szám, 301
vim, 185, 216, 233
vim-, 260
vim-common, 237
virtual memory, 59
vmware, 305
void, 300

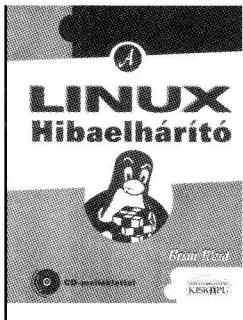
w, 26, 53, 84, 156, 158
wait, 127
webserver, 253
wght, 291
who, 156
WSMouse, 299
wtmp, 156

X, 290, 301, 324, 333
x, 26, 84, 89, 143
X Display Manager, 328
X display manager, 322
X display manager control protocol, 323
X version 11. release 6., 289
X Window beállítóállomány
 Above azonosító, 312
 AceCad, 299
 Auto, 299
 Below azonosító, 312
 BoardName név, 301
 BusID azonosító, 301
 BusMouse, 299
 Chipset áramkör neve, 301
 Device kártyánév, 309
 Device, 296
 DisplaySize szélesség magasság, 308
 Driver meghajtónév, 298, 301
 ExplorerPS/2, 299
 Files, 296
 FontPath állománynév, 297
 Gamma vörös zöld kék, 308
 GlidePointPS/2, 299
 GlidePoint, 299
 HorizSync frekvenciák, 307
 Identifier azonosító, 298, 301, 307, 309, 311
 ImPS/2, 299
 InputDevice, 296
 IntelliMouse, 299
 LeftOf azonosító, 312
 Load modulnév, 316
 Logitech, 299
 MMHitTab, 299
 MMSeries, 299
 Microsoft, 299
 modelName típus, 307
 ModulePath könyvtárnév, 297
 Module, 296
 Monitor monitornév, 309
 Monitor, 296
 MouseManPlusPS/2, 299
 MouseMan, 299
 MouseSystems, 299
 NetMousePS/2, 299
 NetScrollPS/2, 299
 Option AllowMouseOpenFail yes, no313
 Option AutoRepeat szám1 szám2, 299
 Option BlankTime szám, 313
 Option Buttons szám, 300
 Option Device karakteresköz-meghajtó, 299
 Option DontVTSwitch yes, no312
 Option DontZap yes, no313
 Option DontZoom yes, no313
 Option Emulate3Buttons yes, no300
 Option NoPM yes, no313
 Option OffTime szám, 313
 Option Protocol típus, 299
 Option Rotate irány, 302
 Option StandbyTime szám, 313
 Option SuspendTime szám, 313

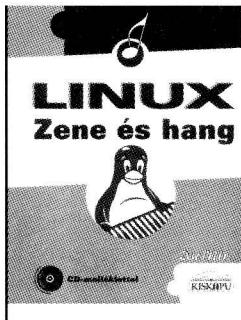
- Option Xinerama yes , no 313
Option XkbLayout billentyűki-
osztás, 299
Option XkbModel típus, 299
Option fbdev karaktereszköz-
meghajtó, 302
PS/2, 299
RgbPath állománynév, 297
RightOf azonosító, 312
Screen szám, 301
Screen, 296
ServerLayout, 296
SysMouse, 299
ThinkingMousePS/2, 299
ThinkingMouse, 299
USB, 299
ValuMouseScroll, 299
VendorName gyártó, 307
VendorName név, 301
VertRefresh frekvenciák, 307
VideoRam szám, 301
WSMouse, 299
XVideo, 317
Xqueue, 299
bitmap, 316
dbe, 318
dri, 319
extmod, 318
fbdehw, 318
freetype, 316
glx, 319
kbd, 299
keyboard, 298
mouse, 299
record, 318
speedo, 317
type1, 317
void, 300
xtrap, 318
DefaultDepth szám, 309
Depth szám, 309
X Window System Video, 317
X11R6, 289
xcalc, 331
Xconfigurator, 293
xdm, 322–324, 328, 332, 337
xdm-config, 332, 338
XDMCP, 323, 324, 328
xdpyinfo, 317
xeyes, 170
xf86cfg, 293
xf86config, 293
xfontsel, 291
XFS, 57–59, 82, 83, 179
xfs, 58, 72, 292
xlogin*borderWidth, 335
xlogin*greetColor, 335
xlogin.Login.allowNullPasswd,
 336
xlogin.Login.allowRootLogin,
 336
xlogin.Login.background, 335
xlogin.Login.fail, 336
xlogin.Login.failColor, 335
xlogin.Login.failFont, 336
xlogin.Login.Font, 336
xlogin.Login.foreground, 335
xlogin.Login.greetFont, 336
xlogin.Login.greeting, 335
xlogin.Login.height, 335
xlogin.Login.namePrompt, 336
xlogin.Login.passwdPrompt, 336
xlogin.Login.promptFont, 336
xlogin.Login.width, 335
xlogin.Login.x, 335
xlogin.Login.y, 335
xlsfonts, 291
Xqueue, 299
xrdb, 332
xtrap, 318
XVideo, 317

LINUX világába

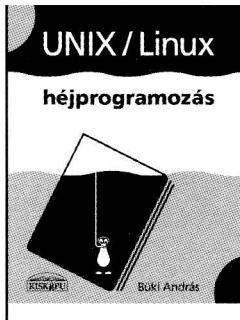
Kapu a



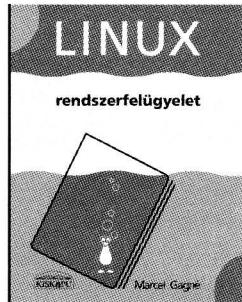
Ár: 3220 Ft
281 oldal
felhasználói szint:
kezdő, haladó
melléklet: CD



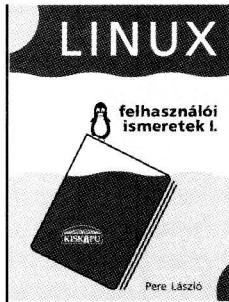
Ár: 4900 Ft
397 oldal
felhasználói szint:
kezdő, haladó
melléklet: CD



Ár: 2660 Ft
256 oldal
felhasználói szint:
kezdő-haladó



Ár: 6440 Ft
672 oldal
felhasználói szint:
kezdő–profi



Ár: 2660 Ft
256 oldal
felhasználói szint:
kezdő

www.kiskapu.hu

