

HALADÓ UHU-LINUX FELHASZNÁLÓK KÓDEXE



1. kiadás, készült 2003. november 27-én

Copyright © 2003, UHU-Linux Kft.

A Kódex tartalmának, illetve részeinek sokszorosítása abban az esetben engedélyezett, ha jelen licencet minden másolt példány tartalmazza.

A Kódexben szereplő információkat a szerzők legjobb tudásuk szerint állították össze, ennek ellenére hiba előfordulása nem kizárható.

A szerzők és az UHU-Linux Kft. semmiféle felelősséget nem vállalnak és semmilyen anyagi kárért nem felelősek, amely bármilyen vélt, vagy valós módon a Kódexben leírtak alkalmazásából eredhet.

A Kódexszel kapcsolatos javaslatokat, megjegyzéseket az alábbi email címre kérjük elküldeni:

doksi@uhulinux.hu

A Kódex készítésekor kizárólag szabad szoftverek kerültek felhasználásra. A szövegszerkesztés az mcedit segítségével történt, a nyomdai előkészítést a \LaTeX tördelőprogram végezte. A teljes folyamat szabad felhasználású Linux operációs rendszer alatt zajlott le.

Szerzők: Ágoston László <endymion@freemail.hu>
 Balázs Tibor <covek@tux.linux.hu>
 Dömsödi Gergely <doome@uhulinux.hu>
 Gibizer Tibor <gibzo@freemail.hu>
 Hazai Géza <janu@kde.hu>
 Koblinger Egmont <egmont@uhulinux.hu>
 Lévai Attila <bronz@bronz.hu>
 Sári Gábor <saga@chello.hu>
 Szilveszter Farkas <info@psoftwares.hu>

Lektorok: Parusef Imre <uzenet@uze.net>

LaTeX tanácsadó: Nagy Bence <bence@freemail.hu>

Szedés: \LaTeX 3.14159 verzió

Tartalomjegyzék

1. Adminisztrációs feladatok	5
2. Az X felület	7
2.1. Bevezetés	9
2.2. A hardver	9
2.3. Csomagok és telepítésük	10
2.4. Mi is valójában az X felület?	11
2.5. Finomhangolás	13
2.6. Tippek	22
2.7. Ablakkezelők és a Desktop Environment	27
3. A rendszer indítási folyamata	29
3.1. Rendszerindítási folyamat	31
3.2. Munka a boot managerekkel	43
3.3. Felhasznált irodalom	51
4. Dokumentumkezelés	53
5. Bejelentkezés	57
5.1. Az Gnome Display Manager (gdm)	59
5.2. A KDE Display Manager (kdm)	61
6. Felhasználókezelés	73
6.1. Több felhasználó, több program	75
6.2. Hozzáférési jogosultságok	77
6.3. Felhasználók és csoportok kezelése	82

7. Forrásnyelvű programok telepítése	87
7.1. A fordításról általában	89
7.2. Egy program lefordulásának lépései	90
7.3. Hol találhatunk forrásban programokat?	90
7.4. Előkészületek	90
7.5. Fordítás másik architektúrára	96
7.6. Felhasznált irodalom	96
8. Hálózati alapismeretek	97
8.1. Alapfogalmak	99
9. Hálózati szolgáltatások	111
9.1. A DNS	113
9.2. A levelezés alapjai	139
10. A Kernel	149
10.1. A kernel szerepe	151
10.2. A Linux kernel története	151
10.3. Kernel fordítás	156
10.4. Munka a kernellel	158
10.5. Felhasznált irodalom	162
11. A Linux környezet mélyebb ismerete	163
12. Biztonság	181
12.1. Biztonsági kérdések	183
13. Segítségkérés, dokumentációk	241
13.1. Segítség, dokumentációk	243
13.2. Segítség helyben	243
13.3. Segítség az interneten	244
13.4. Mit illik és mit nem a linux és más levelező listákon?	245
14. Szövegfeldolgozás	249

15. Szerver programok UHU-Linux alatt	251
15.1. Az APACHE 2 és PHP4 beállítása UHU-Linux alatt	253
15.2. Az Apache finomhangolása	259
15.3. Az Apache telepítése csomagból	261
15.4. A Postfix	262
15.5. A proftpd	262
15.6. A MySql	262
15.7. A PostgreSQL	274
16. Tűzfalak	285
17. Az SSH	287
18. Fejlesztői eszközök, ismertető	289
18.1. Bash shell programozás	291
18.2. Az sed áradatszerkesztő	373
18.3. Perl-ről röviden	373
18.4. Az awk	404
18.5. A grep	404
18.6. Az Xdialog	404
18.7. Alapok	404
18.8. Itt egy c forráskód, fordítsuk le	404
18.9. Szintaktikai elemek	405
18.10. Összetett adatszerkezetek, TÖMB	418
18.11. Fájlfelügyelet	438
18.12. NCURSES	442

Bevezetés

Tisztelt vásárlónk! Szeretettel köszöntjük az UHU-Linux rendszert használók egyre növekvő táborában! A programcsomag készítői folyamatosan azon fáradoznak, hogy az Ön asztalára egy minél inkább kézreállóbb, a kezdőknek egyszerű, a profiknak pedig kellően átgondolt és megtervezett Linux összeállítás kerüljön.

Szeretnénk, ha rendszerünk minél több felhasználó öröme és megelégedésére szolgálna, ezért megkérjük Önt is, hogy észrevételeit, javaslatait küldje el nekünk a <doksi@uhulinux.hu> címre.

Itt is felhívjuk figyelmét a hosszabb idő óta eredményesen működő “UHU-kezdő” levelezőlistára, valamint *Tudásbázisunkra*, melyek a <http://www.uhulinux.hu/> weboldalon megtalálható.

A most megvásárolt programcsomag folyamatos frissítése az Interneten keresztül lehetséges lesz, így a későbbi verziók megvásárlása nem feltétele rendszerének naprakész állapotban tartásának.

E Kódex szerkesztése közben komolyan elgondolkodtunk azon az egyszerűnek tűnő kérdésen, hogy milyen elvet kövessünk a nyelvezet során. Értjük ez alatt azt, hogy megszólításként Önözéssel vagy Tegezéssel éljünk. Végül, középútként azt a módszert használtuk, mintha az UHU-Linux rendszert közösen kezdenénk el megismerni, virtuálisan egymás mellett ülve.

A könyvben használt jelölések

A könyvben alkalmazott tipográfiai jeleket a következő táblázat magyarázza el.

Szöveg kinézete	Jelentés
passwd	a passwd program elindítása a passwd paranccsal
/etc/passwd	egy fájl vagy könyvtár neve
<fájl>	amikor a parancsot megadjuk, a fájl-t helyettesítsük be a kívánt értékkel, de a < > jeleket <i>NE</i> adjuk meg!
PATH	környezeti változó, melynek neve PATH
192.168.1.1	egy változó értéke
ls	az ls parancs
katalin	a katalin nevű felhasználó
bagolyvar:~# df	a root burok a ~ könyvtárban (a ~ az adott felhasználó „saját könyvtára”); a df a parancs, a bagolyvar pedig a számítógép neve
kata@bagolyvar:/tmp\$ > ls	a kati nevű felhasználó parancsértelmezője a /tmp könyvtárban, az ls parancs kiadása előtt
C:\> fdisk	DOS parancssorban az fdisk parancs
(Alt)	az „Alt” billentyű lenyomása
(Ctrl) + (Alt) + (Del)	a ‘+’ jel a megadott billentyűk „együttes” lenyomását jelzi. Az egymás után lenyomandó billentyűket írásban csak egy szóköz választja el.
Engedély megtagadva	rendszerüzenet
’Rendszerfrissítés’	menübejegyzés

Hogyan használjuk a könyvet

Ha probléma merülne fel

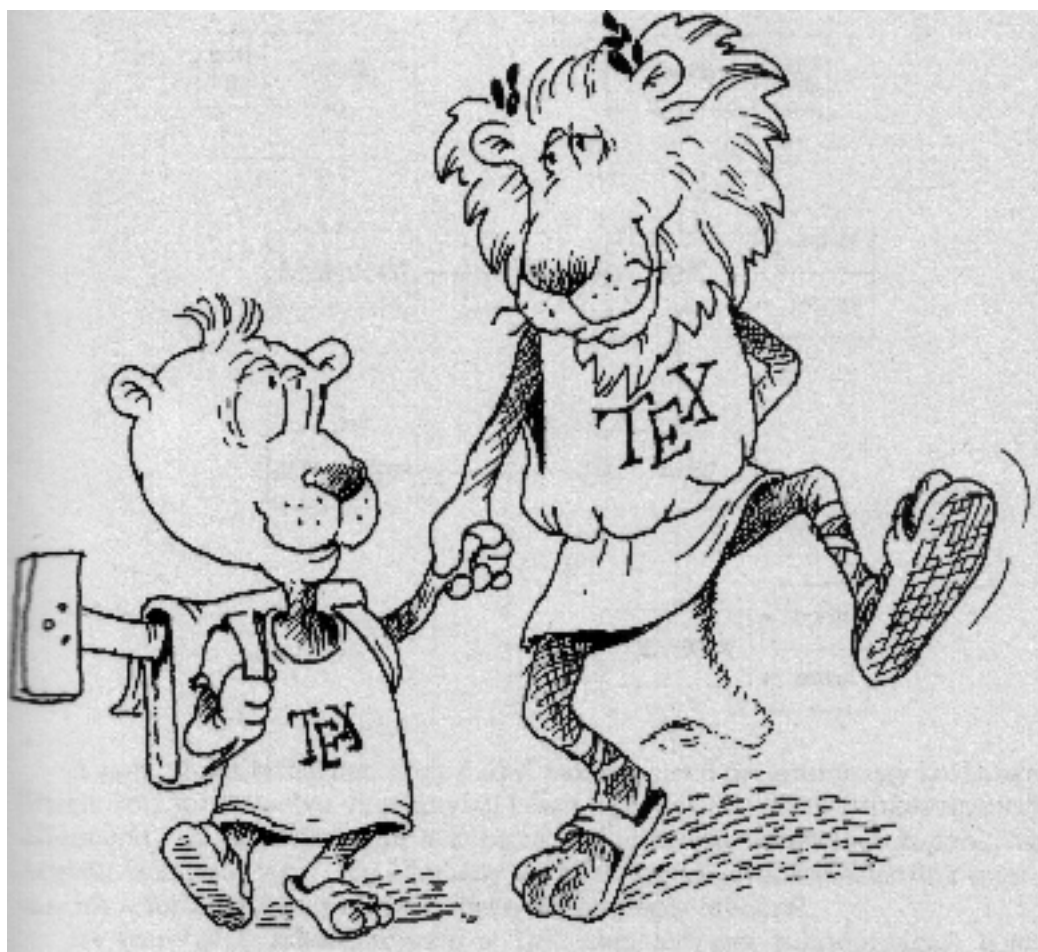
Közreműködők névsora

Köszönetnyilvánítások

az UHU-Linux csapata

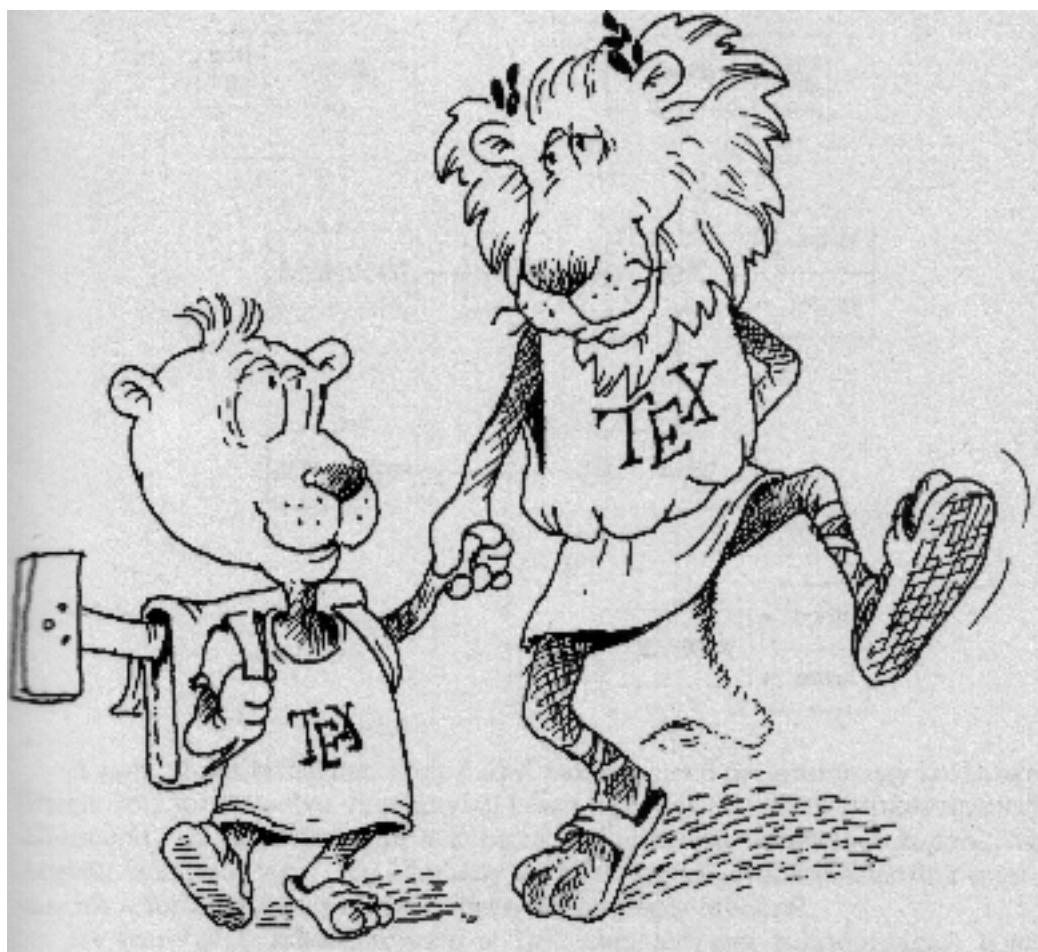
1. fejezet

Adminisztrációs feladatok



2. fejezet

Az X felület



Ez a fejezet az XFree86 részletes installálását, konfigurációját és finom beállításait tartalmazza specifikusan Debian GNU/Linux rendszerre.

2.1. Bevezetés

2.1.1. Előszó

Ha valaki kipróbálta már valamelyik ismertebb Linux disztribúciót, biztosan tapasztalta, hogy már a telepítés során beállításra kerül a szép X, - vagyis a Grafikus felület - mindenféle különleges ablakkezelőkkel egyetemben. Azonban, ha az ember leül telepíteni egy Debian rendszert, az csak a "base system"-et fogja neki feltenni, ami nem tartalmazza a grafikus felületet. Ennek több oka is van, főleg hogy a Debian csak most kezd elterjedni desktop OS-ként, másrészt egy Debian felhasználó sokkal jobban szeret mindent maga beállítani és nem szereti, ha helyette ezt elvégzi automatikusan valami. A Debian igen rugalmas, csomagkezelőjének a dpkg-nak és könnyen kezelhető frontendjének az apt-nek köszönhetően a telepítéshez csupán 1-3 percre van szükségünk, feltéve ha rendelkezünk CD-vel, vagy min. 50kbit/s-es sávszélességgel. A Linux többnyire az X grafikus rendszer szabvány szabad/nyílt forráskódú XFree86 grafikus kiszolgálót használja. Azért a Debian disztribúciót választottam, mert szerintem az egyik legstabilabb GNU/Linux.

Telepítéséhez igaz, hogy egy kicsivel több szakértelem kell, de ha az ember kipróbálja, akkor sohasem tud majd leszokni róla. Az első lépések tényleg nehézkesek, de az eredményül kapott rendszer mindenért kárpótol minket.

2.1.2. Szerzői jog és a felhasználás feltételei

A szerzői jogokat Pápai Ádám birtokolja copyright (c) 2003; A dokumentum másolása, terjesztése és/vagy módosítása a Free Software Foundation által kiadott GNU Free Documentation License 1.1 vagy bármely későbbi verziója szerint lehetséges a dokumentum részeinek, elő- ill. hátoldali szövegeinek módosítása nélkül. A licenc egy másolatát megtalálod ezen a honlapon.

2.2. A hardver

Nagyon fontos, hogy tudjuk és tisztában legyünk azzal, hogy milyen grafikus kártyával rendelkezünk. Ezen nagyon sok múlik. Fontos, hogy lehetőleg olyan kártyát vegyünk, amit támogat a Linux is. Az ehhez kapcsolódó oldal a www.xfree86.org, ahol az aktuális driverek, verziók és a hardvertámogatottság megtekinthetők. Ha még nincs megfe-

lelő driver a kártyánkhoz, akkor egy kicsit türelmesnek kell lennünk. Sajnos a gyártók első dolga nem az, hogy megírják a drivereket Linux alá is. Előbb-utóbb fel fog tűnni a támogatott kártyák között. Rohamos tempóban fejlesztik a grafikus programokat, köszönhetően annak, hogy egyre inkább elfogadják a szabad szoftvereket, ezért próbálnak a fejlesztők minél emberibb külsőt ajándékozni az Operációs rendszernek. Ma már nem lehet elmondani azt, hogy a GNU/Linux nem rendelkezik grafikus felülettel, sőt szinte megszámlálhatatlan fejlesztő dolgozik az ilyen jellegű projekteken. Noha senki sem kötelez, hogy használjuk a grafikus felületet, nem árt, ha van a gépünkön. Például, ha szeretnénk egy filmet jó minőségben megnézni pl. a Xine lejátszó segítségével, vagy ha szeretnénk használni az OpenOffice.org nyílt forráskódú programcsomagot, amely segítségével prezentációkat készíthetünk, táblázatokat és szöveget szerkeszthetünk.

A jelenleg (2003.05.23) támogatott grafikus kártyák dióhéjban:

3Dfx	National Semiconductor
3Dlabs	NCR
Alliance	NeoMagic
ARK Logic	NVIDIA
ATI	Number Nine
Avance Logic	Oak Technologies, Inc.
Chips and Technologies	Paradise/Western Digital
Cirrus Logic	RealTek
Compaq/Digital	Rendition/Micron
Cyrix	S3
Epson	Silicon Graphics, Inc. (SGI)
Genoa	Silicon Integrated Systems (SiS)
IBM	Silicon Motion, Inc.
IIT	Sun Microsystems
Integrated Micro Solutions (IMS)	Trident Microsystems
Intel	Tseng Labs
Matrox	Video 7
Micronix, Inc. (MX)	Weitek

2.3. Csomagok és telepítésük

Az alábbi csomagokra lesz szükségünk. Ezek az XFree86 oldaláról forrásban letölthetők, vagy a Debian rendszer alatt az

```
apt-get install csomagnév
```

paranccsal telepíthetők.

xfree86-common	-	X Window System (XFree86) infrastructure
xfonts-base	-	standard fonts for X
xfonts-100dpi	-	100 dpi fonts for X
xfonts-75dpi	-	75 dpi fonts for X
xbase-clients	-	miscellaneous X clients
xserver-xfree86	-	the XFree86 X server (4.x)
xterm	-	egy grafikus terminál
		mindenképpen ajánlott feltelepíteni

Mindenkinek ajánljuk a 4.x használatát, de még mindig elérhető az XFree86 3.x verziója is (xserver-svga, xserver-* - régebbi kártyák támogatása). Az xserver-xfree86 csomag telepítésekor bejön a csomag debconf (kérdézz-felelek ncurses) alapú menüje, ahol szinte minden beállítást megtehetünk a monitort illetve a grafikus kártyát illetően. Miután készen vagyunk az első konfigurációval, és a csomagok telepítését befejeztük, elég erőt érezhetünk magunkban, hogy kiadjuk a csodálatos parancsot:

```
$ startx
```

Segítség: Ha az X felületről nem tudunk hogyan kilépni, használjuk a ctrl+alt+backspace (escape hatch) billentyűkombinációt.

2.4. Mi is valójában az X felület?

2.4.1. Mit is jelent az X felület valójában és milyen folyamatokat indít el?

A már széles körben ismert és használatos grafikus felületek vezérlőprogramja az xinit. Az xinit konfigurációs fájlok segítségével felépíti az egész X felületet. A startx parancs csak egy héjprogram, amely szintén az xinit-et hívja meg, és azt indítja el.

Tehát milyen folyamatok zajlanak le, ha elindítjuk a startx-et? A következők:

- A héjprogram a /home/user -ben keres egy .xinitrc állományt. Ez a felhasználó X-szel kapcsolatos egyedi beállításait tartalmazza (milyen ablakkezelőt indít és milyen programokat).
- Amennyiben ezt nem találta meg, tovább keresi a /etc/X11/xinit/xinitrc -t.
- Ez az xinitrc állhat akár egy sorból is, vagy akár újabb programot indíthat - pl. Xsession.

- Az Xsession file tartalmaz minden globális beállítást. Azokat is, hogy esetlegesen a helyi beállítófájlokat hol keresse.
- Végül elindul az X.

Amennyiben nem szeretnénk a globális beállításokat használni, abban az esetben létre kell hoznunk a saját /home -unkon belül, egy .xinitrc file-t. Ebben a fileban lévő parancsok, az X indulásakor végrehajtásra kerülnek. Például hogy milyen ablakkezelő induljon el és milyen programokat töltsön be alapértelmezetten. Tehát ha csak ennyit tartalmaz a .xinitrc:

```
exec xterm
```

akkor ez egy alap X felületet fog nekünk futtatni, amelyen belül egy xterm is fut. Ha innen kiadunk bármilyen parancsot pl: xterm vagy xclock, akkor az xterm helyén fog megjelenni az adott program.

Kilépés: a Ctrl+C -vel történik, majd exit/logout

Van még egy igen fontos fájl, a .Xresources. Ez a file tartalmazza az X alatt futó programok beállításait kinézet szempontjából. Ha meg szeretnénk változtatni valamelyik program kinézetét, akkor nincs más tennivalónk, mint elolvasni az adott program kézikönyvét. Abban megtalálhatjuk hogy milyen néven lehet az egyes tulajdonságokra hivatkozni. Változtassuk meg például az Xterm kurzorának színét és a megjelenítendő nevét. Keressük meg, mi is ennek a 2 változónak megfelelő sor:

```
cursorColor  
iconName
```

Tehát akkor a .Xresources file-ba ennyit írunk:

```
xterm*cursorColor: Blue  
xterm*iconName: Ikszterm
```

Látható, hogy az xterm* előtagot használtuk, hogy az xrdb (X resources database), pontosan tudja, hogy melyik X folyamatnak a beállításaira vonatkozik a bejegyzés. Most hogy ez készen van, szeretnénk ezt a beállítást aktivizálni, de úgy, hogy csak az adott programot és ne az egész X-et kelljen újraindítanunk. Erre van az xrdb parancs, melyet a következőképpen kell használnunk:

```
$ xrdb -merge /home/user/.Xresources
```

Az illető programot újraindítva máris láthatjuk a változást. Ha szeretnénk, hogy minden alkalommal automatikusan betöltődjenek ezek a beállításaink, akkor a .xinitrc file-ba írjuk be a következő sort is.

```
$ xrdb -load ~/.Xresources
```

2.4.2. Az xinit használata

Felkészültünk tehát az X futtatására, de még nem rendelkezünk semmilyen ablakkezelővel. Nem kell elkeserednünk, ugyanis nem feltétlenül kell nekünk ablakkezelő is, hogy futtassunk egy grafikus alkalmazást. Például tökéletes ha van egy xterm, amely grafikus felületen biztosít nekünk parancsot. Nézzük meg tehát hogyan is lehet munkára bírni mondjuk egy xterm és egy xclock grafikus programot. Minden alkalmazás nagyon sok "kapcsolóval" rendelkezik, így nagyon sokféleképpen lehet elindítani egy egyszerű xterm-et, vagy xclock-ot is. Alapértelmezésben az

```
$ xinit
```

parancs, elindítja az X alkalmazást a -display 0 értéken, a vt7-es terminalon (Virtual Terminal). Első megjelenés: egy összemaszzkolt háttér és egy X alakú egérkurzor.

Egészítsük ki az xinit parancsot, hogy indítson el egy xterm-et is fekete háttérrel, piros betűkkel, ikszterm névvel és 100x30 pixel nagyságban. A parancs így néz ki:

```
$ xinit /usr/X11R6/bin/xterm -bg black -fg red -title Xterm -geometry 100x30
```

Ugyanígy elindíthatjuk akár az abakkezelőnket vagy a teljes Desktop Environment-et is:

```
$ xinit /usr/bin/startkde
```

Annyit talán még érdemes megemlíteni, ha mindenféle állítgatás nélkül szeretnénk más színmélységben indítani, akkor alkalmazhatjuk az

```
$ xinit -- -depth 16 :0 vt7
```

parancsot is, amely 16 bites színmélységben, a 0 display-en a 7-es virtuális terminálon indítja el a grafikus felületet.

Ha valakit mégjobban érdekel az xinit, és kapcsolói, akkor ajánljuk az xinit kézikönyvének olvasgatását.

2.5. Finomhangolás

2.5.1. Konfiguráció a debconf-al

FONTOS! Mindig készítsünk biztonsági mentéseket a beállításokat tartalmazó fájlokról!!

Az X konfigurációs fájl valószínűleg a /etc/X11/XF86Config-4 lesz 4.x esetén. 3.x esetében a /etc/X11/XF86Config.

Mielőtt nekiesnénk kedvenc szövegszerkesztőnkkel a Config fájlnek, érdemes szót ejteni egy sokkal látványosabb és könnyebben konfigurálható rendszerről. Váltunk át rendszergazdnának és essünk neki:

```
# dpkg-reconfigure xserver-xfree86
```

Ekkor bejön ugyanaz az felület (a debconf), mint amivel az Xserver telepítésénél találkoztunk. Ezzel igen sokat lehet játszani, de lássuk, hogy milyen beállítási lehetőségeink vannak:

Az első kérdés ami felmerül:

```
Manage Xfree86 4.x server configuration file with debconf
```

A kérdés arra vonatkozik, hogy szeretnénk-e az X konfigurációs fájl-t ezzel a módszerrel módosítani. A válasz egyértelműen: YES

```
Select the desired X server driver.
```

Itt van lehetőségünk kiválasztani a kártyánk típusát.

```
Enter an identifier for your video card.
```

Ide egy nevet kell írunk, amivel mi magunk akarjuk majd később esetleg azonosítani a kártyánkat.

```
You may wish to use the "lspci" command to determine the bus location
```

Ha több videokártya van a gépben, akkor az lspci parancs kiadása után eldönthetjük, hogy melyiket akarjuk használni.

```
01:00.0 VGA compatible controller: nVidia Corporation NV11 (GeForce2 MX) (rev b2)
00:0c.0 VGA compatible controller: S3 Inc. ViRGE/DX or /GX (rev 01)
```

Vagyis esetünkben:

```
Section "Device"
    Identifier      "Geforce2MX400"
    Driver          "nvidia"
    BusID           "PCI:01:00:0"
EndSection
```

Please enter the video card's bus identifier.

Ha tudjuk a kártyánk busz azonosítóját (lspci), akkor beírhatjuk, egyébként hagyjuk üresen

Enter the amount of memory (in kB) to be used by your video card.

A videokártyánk mennyi memóriát fogyasszon KB-ben mérve (ha nem akarunk megadni értéket, hagyjuk üresen)

Please select the XKB rule set to use.

Itt nagy valószínűséggel semmit nem kell változtatni, hanem csak egy OK-t kell nyomni

The "pc102" and "pc105" models are versions of the pc101 and pc104

Meg kell adni, hogy hány gombos a billentyűnk

Please select your keyboard model.

Itt elég ha egy OK-t nyomunk

Please select your keyboard layout.

Itt meg kell adnunk a billentyűzet kiosztást, hogy milyen legyen, gondolom mindenki vagy az us vagy a hu betpár beírásával fog próbálkozni :)

Users of U.S. English keyboards should generally leave this entry blank.

Mehet az OK

Please select your keyboard variant.

Ezt célszerűen hagyhatjuk üresen is

Please select your keyboard options.

Ezt is nyugodt szívvel üresen hagyhatjuk

For the X Window System graphical user interface to operate correctly, certain characteristics of your mouse (or other pointing device, such as a trackball) must be known.

Itt figyelmeztet minket, hogy ahhoz, hogy a grafikus felület tökéletesen menjen, ahhoz az egeret is be kell állítanunk!

Please choose your mouse port.

Itt ki kell választanunk, hogy milyen egerünk is van. Itt most egy kicsit részletezem hogy melyik mit jelent

```
/dev/input/mice --> USB-s egér, késbb részletezem.  
/dev/psaux      --> Szabványos PS/2 -es egér.  
/dev/ttyS0-4    --> COM porton kommunikáló egér.  
/dev/sunmouse   --> sun fejlesztés spec egér.  
/dev/gpmdata    --> Ajánlott a /etc/gpm.conf -ba a "repeat_type=raw"  
                  beállítás használata ebben az esetben.
```

Emulate 3 button mouse?

Felmerül a kérdés, hogy ha 2 gombos az egerünk, akkor szeretnénk-e a 2 gomb egyszeri megnyomásával emulálni, egy harmadik nem létező gombot. Ajánlott a YES választ.

Enable scroll events from mouse wheel?

Ha az egerünk görgvel rendelkezik, akkor válasszuk a YES opciót

Enter an identifier for your monitor.

Itt a monitorunknak adhatunk egy egyedi nevet, amire késbb hivatkozhatunk

Is your monitor an LCD device?

A kérdés: LCD monitorunk van vagy sem

The "advanced" option will let you specify your monitor's horizontal

Felhívja a figyelmet arra, hogy ha a következ conf-nál az advanced beállítást választjuk, akkor magunknak kell beírni a monitor hsync - vsync -jét, vagyis a horizontális és vertikális teljesítményét.

Please select your monitor's best video mode.

Ha a simple-t választjuk, akkor csak ki kell választanunk, hogy hány color a monitorunk, ha a mediumot, akkor a monitorunk legjobb felbontását kell megadnunk. Ha az advancedet választottuk, akkor ott meg kell adni mind a vertikális, mind a horizontális értékeket.

```
Select the video modes you would like the X server to use.
```

Itt kiválaszthatjuk, hogy milyen felbontásokat szeretnénk használni a későbbiek során

```
Please select your desired default color depth in bits.
```

Az alapértelmezett színmélységet kell megadni bit-ben. Ez egy fontos lépés, nagyon sokszor emiatt nem megy az X, mert itt nem jó értéket állítottunk be, vagyis olyan érték kell, amit a grafikus kártya is támogat. Ezt idvel úgyis kitapasztalják.

```
Select the XFree86 server modules that should be loaded by default.
```

Az alapértelmezésként betöltendő modulok kiválasztása. Itt külön felhívom a figyelmet arra, hogy bizonyos kártyák/driverok megkövetelik egy-egy modul meglétét, avagy éppen a hiányát. Például: nvidia driver-nél a GLcore és a dri modul `_NEM_` kell!

```
Write default Files section to configuration file?
```

Rákérdez, hogy kimentse-e a módosított file-t az alapértelmezett konfigurációba. A válasz YES

```
Write default DRI section to configuration file?
```

Nem részletezem, a válasz YES

és végeztünk is a konfiggal. Most megint jöhet a:

```
$ startx
```

Amennyiben még mindig nem menne, folytassuk addig a konfigurálást ezzel a módszerrel, ameddig nem sikerül, vagy térjünk át a profibb megoldáshoz. Írjuk át sajátkezűleg. A következő fejezetek a kézi beállításról szólnak. Még nem találkoztam olyan géppel, amire ne tudtam volna felrakni egy X felületet, tehát nem kell feladni, rendületlenül kell folytatni és próbálkozni.

2.5.2. Mit kell tenni, ha meg akarjuk változtatni a felbontást?

Amint láttuk, a `dpkg-reconfigure xserver-xfree86` paranccsal maximálisan csak az állítható be, hogy milyen felbontásokat akarunk használni és hogy mi a maximum felbontása a monitornak, viszont a beállított listából mindig a legnagyobb felbontást fogja nekünk automatikusan betölteni. Tehát hogyan is változtassuk meg? A válasz nagyon egyszerű. Ragadjuk meg kedvenc szövegszerkesztőnket (`vi`, `joe`, `emacs`, `mcedit`, stb.), és kezdjük el a `/etc/X11/XF86Config-4` fájl módosítását.

Találunk benne egy ilyen részt:

```
DefaultDepth      24
```

A mögötte álló érték, lehet akár 15,16,24,32 stb, attól függen, hogy mit állítottunk be alapértelmezett színmélységnek. Ha netán szeretnénk a színmélységet átállítani, csak ezt kell átírni egy használható mélységre. Hogy melyik a használható színmélység?

A `Conf`ban kicsit lejjebb találhatunk:

```
SubSection "Display"
```

hivatkozást, ami alatt fel van tüntetve egy `Depth`, vagyis "mélység" opció. Tehát bármelyik színmélységet beállíthatjuk a `DefaultDepth` értékének, ami szerepel a `Depth` opciókban is.

Na de mégis hogyan változtassuk meg a felbontást?

A `DefaultDepth` az egyik meghatározó eleme ennek a lépésnek. Figyeljük meg, a `DefaultDepth` alatt felsorakozó részeket

```
SubSection "Display"
EndSubSection
```

Az ezek közötti részek tartalmazzák a használható felbontást, színmélységet. Hogy melyik az éppen használatban lévő? Nagyon egyszerű! Mindig az, amelyiknek a `Depth` értéke megegyezik a `DefaultDepth` értékével. Tehát az erre vonatkozó sorokban meg kell változtatnunk a felbontást, a `Modes` opcióban. Lássunk egy példát:

Adott egy konfiguráció, kiragadok belőle egy részletet. Jelölöm a felbontást meghatározó részt:

```
Section "Screen"
    Identifier      "Default Screen"
    Device          "nemtom"
    Monitor         "Generic Monitor"
```



```

DefaultDepth      24                                <--
SubSection "Display"
    Depth          16
    Modes           "1280x1024" "1024x768"
EndSubSection
SubSection "Display"
    Depth          24                                <--
    Modes           "1280x1024" "1024x768" <--
EndSubSection
EndSection

```

Módosítsuk úgy, hogy ne 24-bites színmélységet használjon, hanem, 16 biteset. Nézzük ez hogyan változtat a dolgokon:

```

Section "Screen"
    Identifier      "Default Screen"
    Device          "nemtom"
    Monitor         "Generic Monitor"
    DefaultDepth    16                                <--

    SubSection "Display"
        Depth       16                                <--
        Modes        "1280x1024" "1024x768" <--
    EndSubSection
    SubSection "Display"
        Depth       24
        Modes        "1280x1024" "1024x768"
    EndSubSection
EndSection

```

Most ne csak a színmélységet, hanem az 1280x1024-et cseréljük le 1024x768-ra és mindezt 24 bites színmélységgel. Ez így fog kinézni:

```

Section "Screen"
    Identifier      "Default Screen"
    Device          "nemtom"
    Monitor         "Generic Monitor"
    DefaultDepth    24                                <--
    SubSection "Display"
        Depth       16
        Modes        "1280x1024" "1024x768"
    EndSubSection
EndSection

```

```

EndSubSection
SubSection "Display"
    Depth            24                <--
    Modes            "1024x768"        <--
EndSubSection
EndSection

```

Amint láthatjuk, semmi egyebet nem csináltunk, mint a DefaultDepth értékét megváltoztattuk 24-re és átírtuk az ehhez tartozó felbontást, és az első helyre írtuk az 1024x768-at. Ha most beírjuk hogy startx, akkor az 1024x768-as felbontás tárul elénk 24 bites színmélységgel. Ez volt a fapados módszer, amivel jó sokat lehet dolgozni. Lássuk az egyszerűbb módszert.

Tehát adott a konfigurációs fájl (XF86Config-4), melyben megtalálhatóak a következőhöz hasonló sorok:

```

SubSection "Display"
    Depth    24
    Modes    "1280x1024" "1280x960" "1152x864" "1024x768" "800x600"
EndSubSection

```

Vagyis, beállítottuk, hogy ezeket a felbontásokat szeretnénk használni. Hogyan változtassuk meg a felbontást úgy, hogy közben nem ljuk ki az X-et? A következő billentyűkombinációk szolgálnak ennek megvalósítására:

Ctrl+Alt+Keypad-Plus

Megváltoztatja a felbontást a listában szereplő következő értékre.

Ctrl+Alt+Keypad-Minus

Megváltoztatja a felbontást a listában szereplő előző értékre.

2.5.3. A billentyűzet beállításairól egy kicsit bővebben

Billentyűzet kiosztásunkat szintén az XF86Config-4 fájlban találhatjuk. Ezt is meg lehet változtatni úgy, hogy ne globálisan hasson, hanem csak 1-1 felhasználó egyéni beállításaként működjön. Nézzük először is a conf file-t. A billentyűzetre vonatkozó beállítások a

```

Section "InputDevice"
    Identifier "Configured Keyboard"
    Driver "Keyboard"
    ...
EndSection

```

közötti részekben találhatóak. Az Option rész tartalmazta a leglényegesebb beállításokat. Lássuk milyen értékeket vehet fel:

XkbRules - X 'architektúra' (xfree86 alapértelmezetten) XkbModel - A Hardver típusa. (pl: hány gombos billentyűzet) XkbLayout - A bill. kiosztása (melyik bill. hova legyen bindelve) XkbVariant - A billentyűzet a billentyűk elhelyezkedése a nyelven belül XkbOptions - extra beállítások (pl: átváltás útközben 2 nyelv között stb.)

Nézzünk egy konkrét példát. Magyar és 105 gombos billentyűzettel így néz ki:

```
Section "InputDevice"
    Identifier "Configured Keyboard"
    Driver      "keyboard"
    Option      "CoreKeyboard"
    Option      "XkbRules"      "xfree86"
    Option      "XkbModel"      "pc105"
    Option      "XkbLayout"     "hu"
EndSection
```

Ha egyszerre több nyelvet is szeretnénk betölteni, akkor definiálnunk kell először a nyelveket, aztán definiálni kell egy billentyűt ami átvált a nyelvek között. Erre szolgál az "XkbLayout" és az "XkbOptions". Legyen a két nyelv közötti átváltó billentyű kombináció az alt+shift. Ez így fest:

```
Section "InputDevice"
    Identifier "Configured Keyboard"
    Driver      "Keyboard"
    Option      "XkbModel"      "pc105"
    Option      "XkbLayout"     "hu,us"
    Option      "XkbOptions"     "grp:alt_shift_toggle"
EndSection
```

Vagyis az "XkbLayout" után ha több érték is következik, akkor azt vesszővel elválasztva kell jelölni.

Ezeket a beállításokat azonban megtehetjük "interaktívan" is. Nem kell leállítanunk az X folyamatot, hanem munka közben is nyílik lehetőség ezek módosítására. Erre a feladatra van a setxkbmap parancs. Rengeteg kapcsolóval rendelkezik, de ezek a kapcsolók megegyeznek az Option-ban használt változókkal. pl

```
$ setxkbmap -rules xfree86 -model pc105 -layout "hu,us" -option "grp:alt_shift_toggle"
```

Amennyiben még jobban szeretnénk belemélyedni a témába, ajánljuk Branden oldalát, illetve a kézikönyvek olvasgatását

Még egy trükk. Elég gyakran előfordul, hogy amint elindítjuk az X felületet és futtatunk egy Xterm folyamatot vagy akár egy Mozilla-t, és nem akar működni a backspace. Ilyenkor nincsen más teendők, mint beírni a következő sort az xterm-be:

```
$ xmodmap -e 'keycode 22=backspace'
```

Vagy rendelkezünk egy 3 gombos jobbkezes egérrel, de mi meg akarjuk fordítani a jobb-bal kattintást, akkor szintén csak 1 sort kell begépelnünk:

```
$ xmodmap -e 'pointer = 3 2 1'
```

Amennyiben vissza szeretnénk állítani, abban az esetben írjuk a 3 2 1 helyébe az 1 2 3-t ;-)

2.6. Tippek

2.6.1. Hogyan vehetjük rá USB-s egerünket a működésre?

Mindenekelőtt meg kell bizonyosodnunk róla, hogy a kernel tartalmazza-e az USB-hez illetve az egérhez a megfelelő modulokat. Váltunk root prompt-ra, lépünk be a kernel forrásunk könyvtárába és adjuk ki a következő parancsot:

```
# make menuconfig
```

(Abban az esetben, ha még sohasem forgattunk kernelt, ajánljuk a KERNEL-HOWTO minél előbbi átolvasását!)

Visszatérve tehát ha a kernelünk már tartalmazza a következő beállításokat:

```
2.4.x kernel
```

```
USB support      -->  (*)   Support for USB
                   (M)   UHCI Alternate Driver (JE) support
                   (M)   OHCI (Compaq, iMacs, OPTi, SiS, ALi, ...)
support
                   (M)   USB Human Interface Device (full HID) support

Input core support -->  (M)   Input core support
                   (M)   Mouse support
```

jelmagyarázat: (*) = Kernelbe beleforgatva
(M) = Modul

akkor mehetünk tovább, ellenkező esetben, le kell forgatni a kernelt a modulokkal együtt és ha már minden kész van, akkor a modulokat be kell tölteni, majd jöhet a konfigurálás. A kernelünk már felkészült az USB-s egér beizzítására. Amit tennünk kell: csupán egy szövegszerkesztővel megnyitjuk az /etc/X11/XF86Config-4 állományt, megkeressük azt a részt, ahol ez található, (vastaggal jelölöm a számunkra fontos részt):

```
Section "InputDevice"
    Identifier      "Configured Mouse"
    Driver          "mouse"
    Option          "CorePointer"
    Option          "Device"          "/dev/psaux"  <--
    Option          "Protocol"        "PS/2"
EndSection
```

A célunk az, hogy ezt a PS/2-es egérbeállítást USB-re cseréljük. Mindössze annyi a dolgunk, hogy a "/dev/psaux" -t lecseréljük "/dev/input/mice" -ra, vagyis helyesen így néz ki:

```
Section "InputDevice"
    Identifier      "Configured Mouse"
    Driver          "mouse"
    Option          "CorePointer"
    Option          "Device"          "/dev/input/mice"  <--
    Option          "Protocol"        "PS/2"
EndSection
```

2.6.2. Az egér görgőjének beállítása

Ehhez mindössze rendelkezünk kell egy görgős egérrel és a meglévő config-hoz hozzáírni 3 sort. Amit meg kell változtatnunk az a "Protocol" ("ImPS/2"-re, egyes új egér-típusoknál "ExplorerPS/2", "ThinkingMousePS/2", "NetScrollPS/2", "NetMousePS/2", "GlidePointPS/2", "MouseManPlusPS/2"), illetve fel kell vennünk egy új Option-t a "ZAxisMapping"-ot. Nézzük hogy is fog kinézni az XF86Config-4 file:

```
Section "InputDevice"
    Identifier      "Configured Mouse"
    Driver          "mouse"
    Option          "CorePointer"
    Option          "Device"          "/dev/input/mice"
    Option          "Protocol"        "ImPS/2"
```

```
Option      "Emulate3Buttons" "true"  
Option      "ZAxisMapping"      "4 5"  
EndSection
```

FONTOS! Mindig készítsünk biztonsági másolatot a konfigurációs fájljainkról!

2.6.3. Hogyan állítom be, hogy melyik ablakkezelő fusson?

Egyszerűen létrehozunk egy `.xinitrc` file-t a home-unkon belül, és beleírunk 1-2 sort pl:

```
### .xinitrc ###  
exec blackbox  
### end of file ###
```

vagy

```
### .xinitrc ###  
exec wmaker  
### end of file ###
```

esetleg

```
### .xinitrc ###  
exec icewm  
### end of file ###
```

2.6.4. Xvidtune

Egy kevés szót szeretnénk ejteni az `xvidtune` csomagról (az `xbase-clients` csomag része). Ha már sikerült beizzítani az X-et de még utólag is szeretnénk itt-ott kicsikét módosítani grafikus felület alatt, akkor adjuk ki az `xvidtune` parancsot egy `xterm`-ből. Ez egy nagyon finom hangoló műszer. Mindenki csak óvatosan használja, ínyenceknek ajánlott, akik szeretnek állítgatni a dolgokon. Régebbi, analóg monitorok esetén kifejezetten ajánlott az `xvidtune` használata.

Az `Xvidtune` felületet biztosít a különböző képernyőmódok beállításához a kép élesítését, szélesítését, nyújtását és egyéb beállítását lehetővé téve. X alól indítsuk el egy `xterm`-ből

```
$ xvidtune &
```

A felbukkanó ablak 2 részből áll. Az egyik oldalon a vízszintes, a másik oldalon a függőleges tulajdonságokra vonatkozó beállítások találhatók. A változtatáshoz egyszerűen mozgassuk a csúszkákat. A Test gomb megnyomásával tesztelhetjük le, hogy megfelel-e az elvárásainknak. Amennyiben elnyerte a tetszésünket, abban az esetben nyomjuk meg a show gombot, amely kiírja a megfelelő Modeline-t. A kapott sort csak be kell illesztenünk az XF86Config/XF86Config-4 fájlba és újra kell indítani az X-et.

2.6.5. Virtuális terminálok

Az alcím lehetne az is, hogy miként térjünk vissza a parancssorba anélkül, hogy megszakítanánk az éppen futó X folyamatot. Sőt, miként térhetünk vissza az X folyamatunkhoz, mindenféle galiba nélkül. Szerencsére a Debian GNU/Linux és az XFree86 igen rugalmas rendszer, így minden eshetőségre felkészültek. Mint azt már fentebb említettük, az alapértelmezett X felület a 7-es virtuális terminálon indul (xinit – :0 vt7). A virtuális terminál száma (alapértelmezésben: vt7) akkor érdekes számunkra, ha több X felületünk fut egyidőben. Az átváltás X-ről parancssorba a következőként történik:

```
Alt+Ctrl+F1(F2,F3,F4,F5,F6) --> Az adott F1-F6 érték virtuális
                                terminálba (parancssorba) vált
"vissza" Alt+Ctrl+F7           --> Mivel az alapértelmezett X a 7-es
                                virtuális terminálon fut, ezért e
                                kombinációval térhetünk vissza az X
                                folyamatunkhoz.
```

2.6.6. Több X felület egyszerre

Szeretnék egy kis háttérinformációval szolgálni. Miért pont a 7-es virtuális terminálon indul el az alapértelmezett X folyamatunk? Nézzük meg a /etc/inittab fájl tartalmát. Rábukkanhatunk, a következő sorokra:

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

Ezek az alapértelmezésben beállított virtuális terminálok, vagyis azok a bejelentkezési pontok, amelyek a tty1-tty6 között lehetővé teszik a bejelentkezést és munkálatokat a rendszeren.

Egy kicsit kitérnék arra, hogy mit is jelentenek ezek a számok az inittab állományban, sőt egyáltalán mit tartalmaz az inittab állomány. Amikor gépünket elindítjuk, akkor bizonyos szolgáltatások, démonok elindulnak. Hogy mikor melyik démon indul el, azt a futási szintek határozzák meg (runlevel). Debian Woody esetén az alapértelmezett futási szintet ez a sor határozza meg :

```
# The default runlevel.  
id:2:initdefault:
```

Ez azt jelenti, hogy gépünk automatikusan a 2-es futási szintre kapcsol. A futási szinteknél 6 különböző szintet ismerünk. Ezek a következők:

```
#Runlevel 0 is halt.  
#Runlevel 1 is single-user.  
#Runlevel 2-5 are multi-user.  
#Runlevel 6 is reboot.
```

Az egyes runlevelek különböző démonokat/programokat indítanak el a háttérben. Mind-egyik runlevelnek megvan a saját listája, hogy mit is indít el. Ezek a listák a /etc/rc0.d - /etc/rc6.d könyvtárakban találhatóak meg. Minden egyes file, egy-egy shell script, ami lefutása során elindít egy programot. Minden runlevelhez tartozó értékhez létezik egy könyvtár, amelyen belül a /etc/init.d/ könyvtárban lévő "programokra" találunk hivatkozásokat. Röviden egy adott runlevelhez tartozik egy adott /etc/rcx.d/ könyvtár. A könyvtárban található összes "shell script" le fog futni és ezzel elindítja a kijelölt démonokat, melyek a /etc/init.d/ -ben találhatóak meg.

Tehát az a sor hogy

```
1:2345:respawn:/sbin/getty 38400 tty1
```

annyt jelent, hogy a kettes-hármas-négyes-ötös futási szinten a tty1-es terminálon mindig kérje a logint a gép, még akkor is ha logoutoltunk. Visszatérve az elejéhez, jól látható, hogy a tty1-tty6 a parancssornak fentartott hely.

Arra figyeljünk, ha esetleg hozzá szeretnénk adni még több VT-t, akkor a tty7-t mindig hagyjuk szabadon az X számára, feltéve ha használunk X-et.

Tehát mit kell tennünk, ha egyszerre több X munkafolyamatot is szeretnénk futtatni? Tekintsük alapértelmezettnek a rendszert, tehát 7-től kezdve szabadok a Virtuális Terminálok.

Indítsunk el egy XTerm-et az alapértelmezett VT-n

```
$ xinit /usr/X11R6/bin/xterm -bg black -fg red -title terminalablak
```


Majd váltsunk vissza a második virtuális terminálba (CTRL+ALT+F2) és indítsunk el egy másik X folyamatot a 10-es Virtuális terminálon

```
$ xinit -- :1 vt10 /usr/X11R6/bin/xterm -bg black
```

Ha az egyikről a másikra akarunk váltani, akkor a

```
CTRL+ALT+F7  
CTRL+ALT+F10
```

billentyűkombináció használatával érjük azt el. Tehát a vt7 és a vt12 között akármennyi X felületet futtathatunk egyszerre. Ezek a parancsok persze érvényesek startx-esetén is. Vagyis

```
$ startx -- :1 vt8
```

2.7. Ablakkezelők és a Desktop Environment

2.7.1. Mi a különbség egy ablakkezelő és egy Desktop környezet között?

Érdemes szót ejteni az ablakkezelőkről és a Desktop Environmentről (DE) is, melyeknek egyre nagyobb választéka táru a Linux/BSD/Hurd + a többi Posix rendszert használók elé. Szinte már választani sem tudunk közülük, olyan sokféle van és mindegyiknek megvan az előnye, hátránya.

Hogy mi a különbség egy ablakkezelő és egy Desktop környezet között? Nos az ablakkezelő magára az X-re építkezik, míg egy Desktop környezet egy ablakkezelőre épül. Sokszor keverik a két fogalmat, igaz elfordul hogy vannak átfedések, de a kettő mégsem ugyanaz. Egy DE mégiscsak több tulajdonsággal rendelkezik, több a lehetőség. Tetszés szerint fazonra szabható.

X → Ablakkezelő → Desktop Environment

Mégis mi a különbség ezen kívül egy ablakkezelő és egy Desktop Environment között?

- A méret. Míg pl. az IceWM max. 500K méretű, addig egy KDE alapsomag 50MB + mellé a kiegészítő csomagok (összesen kb. 200MB)
- A rendszer erőforrásainak terhelése. Egy Blackbox/Fluxbox átlagos rendszerterhelése 0.3, míg a KDE-nél ez az érték 1.0 körül jár.

- Bootolási idő. Az IceWM Boot-ideje 0.2 sec, a KDE teljes betöltődéséhez 20-40 mp szükséges.

Nos eddig csak negatívumok hangzottak el szinte a Desktop Environment-ről az ablakkezelőkkel szemben. Most lássuk 1-2 előnyös oldalát is:

- Talán az első és legfontosabb: User friendly, vagyis nagyon felhasználóbarát. Könnyű kezelni, egyértelmű az elrendezése, a használata
- Rengeteg saját beépített és saját fejlesztésű szoftverrel/szoftvercsomaggal rendelkezik (pl. a KDE-ben a KOffice)
- Egy csomó ember dolgozik rajta, fejleszti és szinte naponta jön ki frissített csomag.

Nézzük meg, mi egy ablakkezelő elsődleges tulajdonsága:

- Egy lényeges tulajdonság: méretezhetővé teszi az ablakokat
- A gyökér ablakra ad általában 1-2 supportot (pl: jobb klikkre lejön egy menü, amelyet szerkeszthetünk).

Hogy melyiket válasszuk? Mindenki a saját képességei és elvárásai szerint döntsön. Tanácsnak annyit, hogy egy 100-200MHz körüli gépre ne akarjuk a KDE vagy a GNOME összes legújabb csomagját feltenni, érdemesebb a szerényebb icewm, vagy Fluxbox-al próbálkozni.

A leggyakrabban használt Desktop Environment-ek listája:

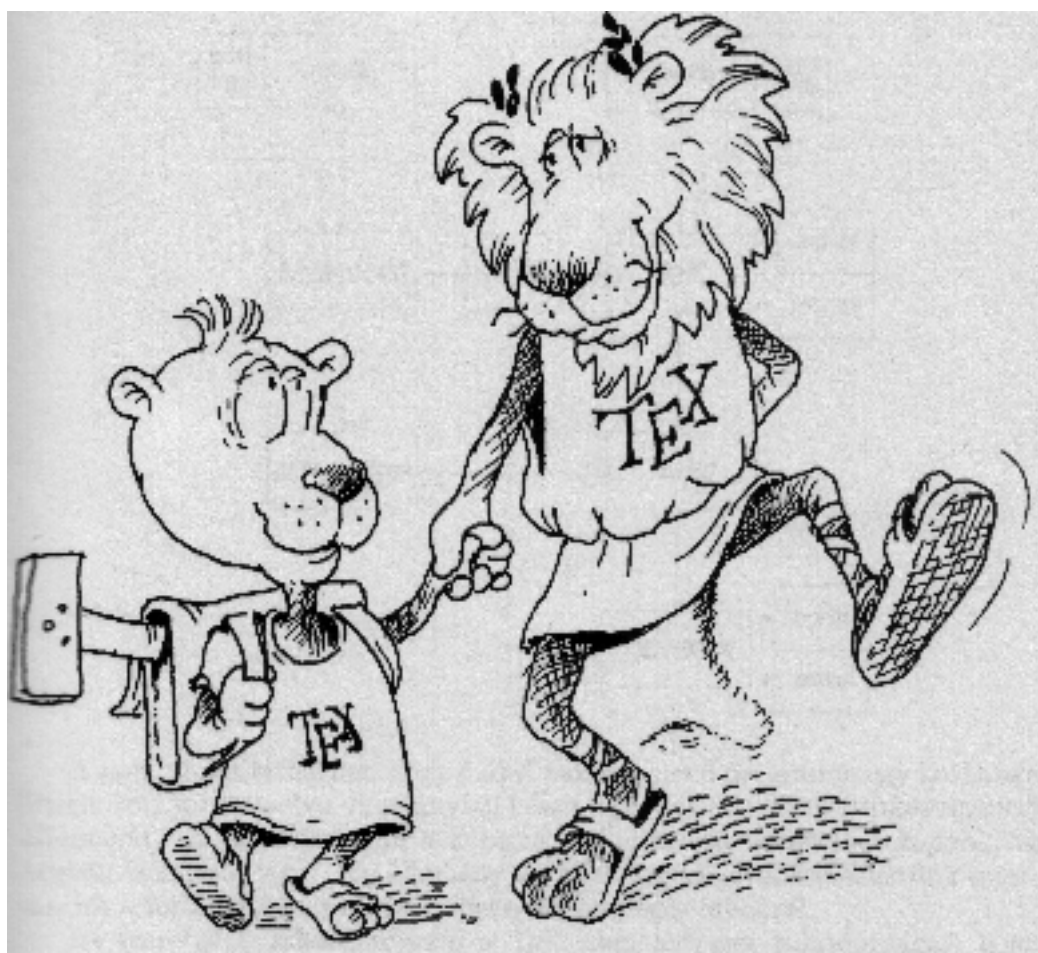
- KDE
- GNOME

A leggyakrabban használt ablakkezelők listája:

- WindowMaker
- BlackBox
- FluxBox
- enlightenment
- iceWM
- TWM
- AfterStep
- fvwm

3. fejezet

A rendszer indítási folyamata



(c) 2003. Tóth Csaba <tothcsaba@cekend.hu>

A dokumentumra az FDL Licenz érvényes.

Ez a fejezet a Linux, de általánosabban a rendszerindítás folyamatát írja le. Nyelvezetét tekintve nem általános célú felhasználásra van tervezve, hanem inkább hozzáértőeknek.

3.1. Rendszerindítási folyamat

A Rendszerindítási folyamatot angolul boot-olásnak hívják.

Ebben a fejezetben arról lesz szó, hogy mi történik a számítógépben az indító gomb megnyomásától kezdve egészen odáig, hogy megjelenik a prompt, és a felhasználó megkapja a vezérlést.

3.1.1. Hardveres rész

Amikor bekapcsoljuk a gépünket, az leellenőrzi saját magát, hogy minden jól működik-e, ezt POST (Power On Self Test) -nak hívjuk. Ezt a folyamatot maga a hardver, illetve az alaplapon egy nem felejtő memóriában lévő kód, a BIOS (Basic Input / Output System) vezérli.

A hardver beindulása

A sorrenddel mi magunk is tisztában lehetünk, ha megfigyeljük, hogy mi történik, mikor bekapcsoljuk a gépünket. Először felzúgnak a ventilátorok, és egy kis ideig nem történik semmi látható sem. Ekkor történik a POST folyamat hardveres része, a chippek kommunikálnak egymással, a közpi chippek (alaplapi chipkészlet) felépít egy listát az elérhető hardverekből, kiosztja az IRQ-kat. Ha bármi hibát talál ezen a ponton, akkor az alaplapon leáll, és hibakódokkal elkezd pittyegni. A POST kódokat mi is le tudjuk ellenőrizni, erre lehet kapni kártyát szaküzletben, ami a PCI/ISA buszra dugva megmondja, hogy a POST éppen melyik szakaszban tart. Értelemszerűen arra jó ez, ha a gép lefagy, akkor pontosan látjuk a POST melyik szakaszában történt a baki. Ezek után megfigyelhetjük, hogy a monitor felvillan, és kiíródik a VGA kártyánk indulási üzenete. Ekkor történik a gépben lévő kiegészítőkártyákon lévő BIOS-ok betöltődése a memóriába. Ezek után a vezérlést megint az alaplapi BIOS veszi át, és ekkorra már a gép hardverei lényegében inicializálva vannak. A BIOS először kiírja a saját infóit, itt van lehetőség valamilyen gomb megnyomásával bejutni egy beállító felületre. Következik a billentyűzet betöltése, ekkor a billentyűzeten található LEDek kigyulladnak. Ezután a memória leellenőrzése következik, majd az alaplapon található IDE vezérlő BIOS-a kapja meg a vezérlést, majd a floppy meghajtó BIOS-a, és végül az egyéb IO eszközök BIOS-ai (ilyennek kell érteni pl. a SCSI, az S-ATA vagy az IDE-RAID csatolókat).

A rendszerindítási sorrend

A rendszerindítási sorrendet (boot szekvenciát) a BIOS beállító felületén tudjuk állítani.

3.1.2. Merevlemez MBR-je

A MBR (Master Boot Record) -nak két funkciója van: ebben tárolódik a merevlemez partíciós táblája, és az a kis 512 byte hosszú (egy szektornyi) programkód, ami a későbbiekben el fogja indítani azt a folyamatot, aminek hatására beindul az operációs rendszerünk, jelen esetben a Linux. Az MBR felépítésével az általános hardverismeretek fejezet, illetve a telepítési fejezet partícionálás része foglalkozik.

3.1.3. A kernel betöltődése

A kernel az operációs rendszer központi része. Ezzel egy külön fejezet foglalkozik.

Az operációs rendszereinknek is magához a géphez hasonlóan van egy elindulási folyamatuk, ezt szintén boot folyamatnak nevezzük. A kernel alaptól nem képes elindulni, mindenképpen szükség van egy előtevékenységre, egyfajta "tereprendezésre", hogy a kernel beindulhasson.

A boot loader szerepe

A kernel betöltésre vannak célprogramok, ezek a boot loader -ek. A két legáltalánosabban használt univerzális program a LILO és a GRUB, ezek képesek más (nem linux) operációs rendszereket is elindítani, illetve vannak csak szimpla linux kernel betöltők, pl. a *Syslinux*, vagy a *Linuxloader*.

A kernel betöltése a következő lépésekben zajlik:

- memória elejének kiürítése
- a kernel beolvasása a memória legelejére
- a kernel memória része után közvetlenül odaírni az esetleges initrd állományokat
- átadni a vezérlést a kernelnek

Ez roppant egyszerű feladatnak tűnik, de ne felejtjük el, hogy ahhoz, hogy a kernelt be tudjuk tölteni a memóriába, először azt be kell olvasni a merevlemezről, ami a sok különböző fájlrendszerek miatt bonyolult feladat. Ez a program semmiképp sem valósítható meg 512 bájt hosszúságban (ez boot sector mérete), ezért e programkód feladata

nem betölteni a kernelt, hanem csak a boot loader többi részét, ami azután majd betölti a kernelt.

A különböző boot loaderok működését a 3.2. fejezetben találhatjuk.

A kernel elindulása

A linux kernel egy nagyon bonyolult "program", éppen ezért itt csak a főbb részek ismertetésére szorítkozunk. Ha valakit érdekel mélyebben is a téma, további információkra lelhet a *Linux-Init HOWTO*-ban, vagy magában a linux forráskódjában.

Amikor a kernel megkapta a vezérlést, a következő műveleteket végzi el:

- a processzort átkapcsolja védett valós módról (real/DOS mode) módra (protected mode)
- kicsomagolja saját magát, ha szükséges
- beállítja a kellő regisztereket a megfelelő értékekre
- feldolgozza a megkapott paramétereket
- felülírja a megszakítás táblát
- betölti a konzol és a PCI busz vezérlőit
- a PCI bus inicializálja magát
- hálózati blokk inicializálja magát
- a kernel elindítja a mellékszálait
 - idle szál (PID 0)
 - init szál (PID 1)
 - esemény lekezelés [keventd] (PID 2)
 - ksoftirqd_CPU0
 - swap lekezelés [kswapd] (PID 4)
 - bdflood (merevlemez cachelés)
 - kupdated
 - khubd (asszem ez USB)
 - kjournald (ext3 vezérlő)
- kernelbe fordított eszközvezérlők betöltése (pl. hangkártya, hálókártya, stb.)

- soros/párhuzamos portok inicializálása
- character típusú eszközök inicializálása
- block típusú eszközök inicializálása
- SCSI busz inicializálása
- ha van `initrd`, akkor előkészíti
- felilleszti a root fájlrendszert a gyökérbe (/)
- ha szükséges felilleszti a `devfs` fájlrendszert
- ha van `initrd`, elindítja rajta a `/linuxrc` programot az `init` szál
- átadja a vezérlést a `/sbin/init` programnak az `init` szál

Ezen folyamatok során a kernel mindenféle információval lát el minket, ezeket a képernyőn jeleníti meg. A pontos inicializálási mechanizmust végső soron nem szükséges megismernünk, de legalább egyszer mindenképp tanulmányozzuk az üzeneteket, ezekből egy pár tipikusát meg is fogunk nézni a későbbiekben. Ha bármiféle kritikus hiba lép fel, akkor a kernel leáll, ezt *kernel panic*-nak hívjuk. Ilyenkor kapunk egy nyomkövetési listát is, és a kernel azonnal leállítja saját magát, tehát a programjaink is leállnak, ilyenkor minden nem mentett adat elvesz.

Szerencsére ez működő rendszerben nagyon ritkán fordul elő. Kernel pánikra ad okot, pl. ha nem találja meg a rendszer a root fájlrendszert, ilyenkor ugyanis nyilván nem fogja tudni elindítani az `init` programot, ami nélkül nem lesz rendszerünk, futó programjaink, tehát értelmetlen lenne minden további működés.

Az `initrd` arra használható, hogy külső eszközvezérlőket tölthessünk be mielőtt még a gyökér fájlrendszert felcsatolnánk. Erre szükségünk lehet olyankor, ha

- mondjuk a gyökér fájlrendszerünk egy bonyolult fájlrendszert használ, és megnövelné a kernelnek a méretét, ami mondjuk úgy már nem férne el egy floppy-n;
- vagy mondjuk ha titkosítva lenne a gyökér fájlrendszer, és ahhoz, hogy elérjük mindenféle egyéb segédprogramnak kell futnia;
- vagy ha RAID tömbbe van rendezve a gyökér fájlrendszer is, és nem tudjuk vagy nem akarjuk bekapcsolni a kernelben az automata RAID felismerés funkciót;
- vagy ha egyszerűen csak le akarjuk ellenőrizni a fájlrendszerek épségét mielőtt még felillesztenénk, mert mondjuk már eleve írható üzemmódban van rá szükségünk;

- vagy ha mondjuk egy hálózati kiszolgálón található a gyökér fájlrendszerünk, és bizonyos hálózati funkcióknak már működniük kell ahhoz, hogy elérjük.

Mint láthatjuk sokszor hasznos lehet az `initrd` használata, mindazonáltal nem javasolt. A kernel készítői szinte mindent megtettek, hogy soha többé ne kelljen `initrd`-t használnunk, például a fentebbiek közül a legelsőhöz és a negyedikhez kellhet csak még.

A kernel paraméterezése

Mivel amikor a kernel elindul még nincsennek fájlrendszerek, ezért Legalább egy paramétert mindenképp tudatnunk kell előre a kernellel: hogy hol található a gyökér fájlrendszer a merevlemezen.

Pl. ha a root fájlrendszer a `/dev/sda1` partíción van, akkor ezt a `root=/dev/sda1` paraméterrel tudjuk megmondani.

De ezen kívül lehetőségünk van paraméterként megadni, hogy melyik programunk az `init`. Pl. ha valami gubanc lenne az `init` fájljal, akkor beírjuk kernel paraméternek, hogy `init=/bin/bash`, és akkor a kernel betöltődés után azonnal root jogokkal fog elindulni egy `bash`.

A kernel paramétereiről bővebben a kernelről szóló fejezet foglalkozik majd.

A kernelüzenetek megtekintése

Most lássuk a kernel beindulását a kiírt üzenetei segítségével. Ha túl gyorsan jelenne meg a képernyőn, akkor a szöveget a későbbiek folyamán megtaláljuk a `/var/log/dmesg` nevű fájlban.

A kernel első feladata a videokártya inicializálása, ha úgy állítottuk be, akkor kérheti a képernyőfelbontás megadását, ellenkező esetben a kernelben meghatározott videomódot állítja be a videokártyán. Ezt a következő üzenettel adja tudtunkra a kernel:

```
Console: colour VGA+ 80x25
```

Ezután kiszámol egy összeget, amit a teljesítmény jelzésére használnak.

```
Calibrating delay loop... 149.50 BogoMIPS
```

A kernel következő feladata a fizikai memória méretének azonosítása:

```
Memory: 63532k/65536k available (656k kernel code, 412k reserved, 904k da  
32k init)
```

Ezután egy humoros mondással nyugtázza a kernel, hogy

```
Checking if this processor honours the WP bit even in supervisor mode...
```

ez a processzor tiszteletben tartja a WP bitet supervisor módban.

Inicializálja a lemezkvótát:

```
VFS: Diskquotas version dquot_6.4.0 initialized
```

Beazonosítja a processzort (és betölt hozzá egy hibajavítást):

```
CPU: Cyrix 6x86L 2x Core/Bus Clock stepping 02
Checking 386/387 coupling... OK, FPU using exception 16 error reporting.
Checking 'hlt' instruction... OK.
Cyrix processor with "coma bug" found, workaround enabled
```

Ezután a PCI eszközök és az ISA buszok inicializálását kísérli meg a kernel:

```
PCI: PCI BIOS revision 2.10 entry at 0xfb470
PCI: Using configuration type 1
PCI: Probing PCI hardware
PCI: 00:38 [1106/0586]: Work around ISA DMA hangs (00)
Activating ISA DMA hang workarounds.
```

A hálózati protokollok betöltése:

```
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
NET4: Unix domain sockets 1.0 for Linux NET4.0.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
```

A soros portok beállítása:

```
Serial driver version 4.27 with no serial options enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
ttyS03 at 0x02e8 (irq = 3) is a 16550A
```

A pszeudo-terminálok konfigurálása:

```
pty: 256 Unix98 ptys configured
```

Az IDE-vezérlők azonosítása:

```
VP_IDE: IDE controller on PCI bus 00 dev 39
VP_IDE: not 100% native mode: will probe irqs later
ide0: BM-DMA at 0x6000-0x6007, BIOS settings: hda:pio, hdb:pio
ide1: BM-DMA at 0x6008-0x600f, BIOS settings: hdc:pio, hdd:pio
```

Az IDE eszközök azonosítása:

```
hda: QUANTUM FIREBALL EX3.2A, ATA DISK drive
hdc: WDC AC31600H, ATA DISK drive
hdd: WPI CDD-820, ATAPI CDROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
hda: QUANTUM FIREBALL EX3.2A, 3079MB w/418kB Cache, CHS=782/128/63, (U)DMA
hdc: Disabling (U)DMA for WDC AC31600H
hdc: DMA disabled
hdc: WDC AC31600H, 1549MB w/128kB Cache, CHS=3148/16/63
hdd: ATAPI 8X CD-ROM drive, 128kB Cache
```

A CD-ROM meghajtóprogramjának betöltése:

```
Uniform CDROM driver Revision: 2.55
```

A merevlemezeken lévő partíciók azonosítása:

```
Partition check:
hda: hda1 hda2 hda3 hda4 < hda5 hda6 >
hdc: [PTBL] [787/64/63] hdc1 hdc2
```

Kíírja, hogy a root partíciót csak-olvashatóra mountolja:

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

A swap terület hozzáadása a rendszerhez:

```
Adding Swap: 32252k swap-space (priority -1)
```

Kernelmodulok betöltése a */etc/modules.conf* fájl alapján:

```
wd.c: Presently autoprobing (not recommended) for a single card.
wd.c:v1.10 9/23/94 Donald Becker (becker@cesdis.gsfc.nasa.gov)
eth0: WD80x3 at 0x280, 00 00 C0 56 CB 68 WD8013, IRQ 10, shared memory at
Soundblaster audio driver Copyright (C) by Hannu Savolainen 1993-1996
SB 4.13 detected OK (220)
```

3.1.4. POSIX szolgáltatásszintek

A POSIX meghatározza, hogy a gyökér fájlrendszer beillesztése után az alábbi programok valamelyikét kell elindítani ilyen sorrendben:

- */sbin/init*
- */etc/init*
- */bin/init*

A kernel *init*= paraméterével meg lehet határozni egyéni *init* programot is. Illetve a szolgáltatásszint számát megadva adott futásszinten fog beindulni a rendszer.

A rendszer különböző üzemmódokban futhat. Ezeket futási szinteknek nevezzük. Úgy képzelhetjük ezt el, mint a Windowsoknál a csökkentett vagy normál, NT-nél a VGA üzemmódot, csak itt menet közben tudjuk ezeket változtatni.

Az LSB a következő futási szinteket definiálja:

- 0 leállítás
- 1 egyfelhasználós üzemmód
- 2 többfelhasználós hálózat nélkül
- 3 normális/teljes többfelhasználós
- 4 nem használt, ugyanaz, mint a 3-as
- 5 többfelhasználós grafikus felülettel
- 6 újraindítás

Látható, hogy nem létezik olyan idő a rendszerbetöltődés kicsi idejét kivéve, amikor nem lennének valamilyen futási szintben.

Másik futási szintre átlépni az *init x* paranccsal tudunk, ahol *x* az új futási szint neve.

3.1.5. LSB/SysV típusú rendszerindítás

A *init* beolvassa a */etc/inittab*-ot, és végrehajtja az ott meghatározott eseményeket.

Áttekintés

A */etc/inittab*-ban meg kell adni az *init*-nek a rendszerindító szkript nevét. Ez a szkript fogja aztán beindítani a rendszert, elvégzi a szolgáltatásokhoz szükséges előkészületeket, pl. leellenőrzi és beilleszti a fájlrendszereket, leellenőrzi a lemezkvótákat, elindítja a szükséges segédprogramokat, stb.

Ehhez meg kell adni, hogy az egyes futási szintekhez milyen programokat kell elindítani. Ez a SysV típusú rendszerindításnál a */sbin/rc x* -et jelenti, ahol *x* a kívánt futási szint neve.

Az LSB innentől nem definiálja, hogy az adott szkriptek hogyan határozzák meg, hogy melyik szolgáltatást kell elindítani az ő futási szintjükön, így ezt itt mi sem tesszük.

Azért nem definiálja az LSB, mivel ez a legtöbb terjesztésben egyedi szokott lenni.

Egy példát mégis hadd hozzunk fel, ami azért többé kevésbé alkalmazható lesz a SysV alapú indítási folyamatot használó disztribúcióknál:

Létezik a */etc/init.d/rcx.d/* nevű könyvtár, ahol *x* a futási szint neve. Ebben a könyvtárban helyezkednek el szimbolikus link formájában a szolgáltatások indító szkriptjei.

A következő módon:

SxxNév megnevezéssel vannak ellátva azok, melyeket ebben a futási szintben el kell indítani

KxxNév névvel pedig azok, amiket le kell állítani

Az *xx*-ek sorszámok lehetnek, és azt határozzák meg, hogy milyen sorrendben induljanak el a szkriptek, pl. ha a DNS szerveret előbb szeretnénk elindítani, mint levelező szerverünket, akkor a levelező szervernek a száma mindenképp nagyobb legyen, mint a DNS szerveré (pl. ha a DNS: 20, akkor mail: 40).

A leállítás is e számozás alapján történik.

Ha átlépünk egy másik futási szintre, akkor sok minden történhet.

Az egyik módszer az, hogy az *init* megnézi, hogy milyen különbség van a régi és az új futási szint *S** fájljai között, és ami különbség van, azokat leállítja a régi futási szintben meghatározott sorrendben, illetve elindítja az újakat az új futási szintnél meghatározott sorrendben.

Az inittab felépítése

Az *inittab* fájl leírja mely processzek indulnak el az induláskor és a normál műveletek közben (például: */etc/rc*, a *getty*-k ...). Az *init(8)* megkülönböztet többszörös futásszinteket, ezek mindegyikének saját beállítása lehet a futtatandó programokról.

Érvényes futási szintek a 0-6, ezenkívül A, B, és C a helybeni bejegyzésekhez. Egy inittab fájlbejegyzésnek a következő formátumúnak kell lennie:

```
azon.:futásszint:tevékenység:processz
```

A # karakterrel kezdődő sorok megjegyzésként viselkednek.

id (azon) Egyedi, 1-4 elemből álló karakterlánc, ami azonosítja a bejegyzést az inittabban (azon *sysvinit* verziók esetében amelyek fordításához tartozó library < 5.2.18, vagy még az *a.out*-os rendszerhez készült, a korlát 2 karakter).

Megjegyzés: *getty* vagy másmilyen login processzeknél az „azon” mező a *tty* megfelelő *tty* rangjának kell lennie, például 1 a *tty1* helyett. Másképpen a bejelentkezési nyilvántartás helytelenül működhet.

futásszint leírja, hogy melyik futási szinteken jön létre az adott tevékenység.

tevékenység leírja, hogy milyen tevékenység jöjjön létre.

processz Meghatározza, hogy melyik processz induljon el. Ha a processz mező „+” karakterrel kezdődik, akkor az *init* nem csinál *utmp* és *wtmp* nyilvántartást a processznek. Ez akkor szükséges, ha a *getty* ragaszkodik a saját *utmp/wtmp* háztartásához. Ez egy történelmi hiba.

A futásszintek mező több értéket is tartalmazhat, a különböző futásszintekhez.

Például az 123 meghatározza, hogy a programnak az 1-es, 2-es és 3-as futásszinten kell futnia. A helyi futásszint bejegyzések A, B, vagy C lehetnek. A *sysinit*, *boot*, és *boot-wait* futásszint bejegyzések figyelmen kívül lesznek hagyva. Ha a rendszer futásszintje megváltozik, az összes olyan program, ami az új futásszinthez nincs bejegyezve, leáll, először *SIGTERM*, majd *SIGKILL* jelzéssel.

Érvényes tevékenységek a tevékenység mezőhöz:

respawn A processz újraindul valahányszor megszakítódik (pl: *getty*).

wait A processz akkor indul el, amikor a megadott futási szintre lép, és az *init* addig vár, amíg a processz fut.

once A processzt egyszer, a futásszintre lépéskor indítja el.

boot A processz a rendszer újraindítása közben indul el. Ilyenkor a futásszint mező tartalma nem érdekes.

bootwait A processzt a rendszer indulása közben indítja el, az *init* megvárja míg lefut (például: */etc/rc*). A futásszint mező mellőzésre kerül.

off Semmit sem csinál.

ondemand Az *ondemand* (helyi) futásszinttel megjelölt processz elindul, valahányszor a megadott helyi futásszint meghívódik. Viszont nem következik be futásszint csere (helyi futási szintek az „a”, a „b” és a „c”).

initdefault Az *initdefault* bejegyzés megadja, hogy melyik futási szintre lépünk be a rendszer újraindítása után. Ha ilyet nem adtunk meg, akkor az *init* a konzolról kér be egy futásszintet. A processz mező ilyenkor figyelmen kívül hagyódik.

sysinit A processz a rendszer újraindítása alatt hajtódik végre, mégpedig minden *boot* és *bootwait* bejegyzés előtt. A futásszint mező tartalma lényegtelen.

powerwait A processz akkor lesz végrehajtva, ha megszakad az áramszolgáltatás. Az *init* erről általában egy olyan processztől értesül, ami egy UPS-sel (szünetmentes áramforrás) kommunikál. Az *init* ilyenkor megvárja, hogy a processz befejeződjön, mielőtt továbbmenne.

powerfail Ugyanaz, mint a *powerwait*, kivéve, hogy az *init* ilyenkor nem várja meg a processz befejeződését.

powerokwait Ez a processz azonnal végre lesz hajtva, amint az *init* arról értesül, hogy az áram visszatért.

powerfailnow Ez a processz akkor lesz végrehajtva, ha azt közlik az *init*-tel, hogy a külső UPS elemei majdnem teljesen üresek, és az áramszolgáltatás megszűnt (feltételezi, hogy a külső UPS és az ellenőrző program képes ezt az állapotot érzékelni).

ctrlaltdel A processz akkor lesz végrehajtva, ha az *init* egy *SIGINT* szignált kap. Ez azt jelenti, hogy valaki a rendszer konzolján lenyomta a CTRL + ALT + DEL billentyűkombinációt. Általában ez azt jelenti, hogy valaki valamiféle shutdown-t akar végrehajtani: vagy egyfelhasználós szintre akar eljutni, vagy pedig újra akarja indítani a gépet.

kbrequest A processz akkor lesz végrehajtva, ha az *init* egy szignált kap a billentyűzetkezelőtől, ami azt jelzi, hogy egy speciális billentyűkombináció lett lenyomva a konzol billentyűzetén.

E funkció leírása még nem teljes; további dokumentációt a *kbd-x.xx* csomagokban lehet találni (a legújabb a *kbd-0.94* csomag volt a dokumentáció írása idején).

Valószínűleg néhány billentyűzetkombinációt akar feltérképezni a „KeyboardSignal” akcióhoz. Például ahhoz, hogy az (Alt + felfelé nyíl) kombinációt e célból feltérképezzük, használjuk a következő bejegyzést a *keymaps* fájljában:

```
alt keycode 103 = KeyboardSignal
```

Példa egy jól működő inittab-ra

```
#Szint amin fussunk
id:2:initdefault:

#Rendszer inicializáció minden más előtt.
si::sysinit:/etc/rc.d/bcheckrc

#0, 6-os futásszint a leállítás és az újraindulás,
#az 1-es pedig a karbantartáshoz van.
10:0:wait:/etc/rc.d/rc.halt
11:1:wait:/etc/rc.d/rc.single
12:2345:wait:/etc/rc.d/rc.multi
16:6:wait:/etc/rc.d/rc.reboot

#Mit csináljunk a „3 ujjas tisztelegés”-nél.
ca::ctrlaltdel:/sbin/shutdown -t5 -rf now

# Mi történjen, amikor az áram elmegy/visszajön.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

#Futásszint 2&3: getty konzolon, 3-as szinten a modemhez is.
1:23:respawn:/sbin/getty tty1 VC linux
2:23:respawn:/sbin/getty tty2 VC linux
3:23:respawn:/sbin/getty tty3 VC linux
4:23:respawn:/sbin/getty tty4 VC linux
S2:3:respawn:/sbin/uugetty ttyS2 M19200
```


Munka a szolgáltatásokkal

A szolgáltatásokat elindító programcskák (szkriptek) a */etc/init.d/* könyvtárban helyezkednek el.

Szolgáltatások menedzselése. A szolgáltatások init szkriptjeinek az alábbi utasításokat lehet megadni:

start elindítja a szolgáltatást

stop leállítja a szolgáltatást

restart leállítja és újraindítja a szolgáltatást (ha fut), egyébként elindítja

reload újraolvastatja a szolgáltatás beállítási fájlját a szolgáltatás újraindítása nélkül

force-reload elindítja a *reload*-ot, ha a szolgáltatás támogatja, egyébként újraindítja

status kiírja a szolgáltatás jelenlegi státuszát

Az alábbi státuszok vannak definiálva:

0 a program fut, és minden OK

1 a program halott, a */var/run* pid fájl létezik

2 a program halott, a */var/lock* lock fájl létezik

3 a program nem fut

4 ismeretlen

A szolgáltatáslista menedzselése. A szolgáltatások menedzselésére nincs bevett gyakorlat, minden terjesztésben más és más módon zajlik. Általában van egy beállító program, melynek segítségével lehet engedélyezni vagy tiltani egy szolgáltatást futási szintként.

3.2. Munka a boot managerekkel

A boot manager tölti be a rendszerünket, ha nem állítjuk be jól, akkor nagyon könnyen kerülhetünk bajba, ezért használatukra érdemes nagyon odafigyelni.

3.2.1. A LILO

A LILO a Linux LOader rövidítése, egy többféle operációs rendszert betölteni képes program (boot manager). Fő célja persze az, hogy a Linuxot betöltse.

Áttekintés

A boot-olás szempontjából a legfontosabb információ az, hogy a Linux nem BIOS hívásokkal kezeli a hardware-t, így a merevlemezt sem. Boot-oláskor viszont a lilo-nak nincs más választása, hisz olyankor még nincs bent a memóriában a kernel. Továbbá a kernelt magát (általában) már a Linux file-rendszeréről kell betöltenie, aminek felépítését, így a benne lévő file-ok helyét is maga a kernel tudja.

Ezt a dilemmát a lilo úgy oldja meg, hogy valójában két részre bomlik:

Van egy 16 bites, real módú része (neve /boot/boot.b), ezt indítja el az MBR-be költöztetett kis programocska. (Az MBR-en kívül máshová is lehet a lilo-t installálni, de ezt hagyjuk most.)

Van aztán egy natív Linux futtatható része, ennek a file-nak a neve valójában a lilo (/sbin/lilo).

Az /sbin/lilo-t Linux alatt futtatva az lekérdezi a kerneltől, hogy a betöltendő file-ok (pl. maga a kernel (tipikusan /boot/vmlinuz)) blokkjai hol (vagyis milyen C/F/S címen) található a lemezen, és ezeket az adatokat beírja a /boot/map file-ba. Persze ennek a file-nak a helyét is lekérdezi, ezt az adatot magába az MBR-ben lévő programocskába írja bele. Így aztán boot-oláskor BIOS hívásokkal tud hozzáférni mindenhez, ami számára fontos.

Ezek után nyilvánvaló, hogy minden kernelfordítás után újra kell futtatni az /sbin/lilo-t, hogy az új kernel elhelyezkedését a map file-ba írhasa. Ha ezt elmulasztjuk, könnyen lehet, hogy a régi kernel indul el, még ha le is töröltük a lemeztől :-)

Minden file, amire a boot-olás során szükség van, és amiket BIOS hívásokkal kell elérni, a /boot/ könyvtárban található. Ezek tehát azok a file-ok, amiknek a lemez első 1024 cilinderén belül kell lenniük, hisz a BIOS (az előzőleg említett BIOS bővítés nélkül) csak ezeket képes kezelni. Nagy kapacitású lemez esetén elegendő egy kis méretű partíciót kreálni a lemez elején és a /boot/ könyvtárat ebbe helyezni, ezzel elérhetjük, hogy a kritikus file-ok BIOS-ból olvashatóak legyenek.

Gondot okoz viszont az, hogyha a kernel a BIOS-tól eltérő geometriát feltételezve adja meg a file-ok helyét, hiszen így boot-oláskor teljesen fals helyről beolvasott adatokat próbál a processzor utasításként végrehajtani. Ennek eredménye tipikusan az, hogy a kezdeti LILO feliratnak csak a fele (LI) jelenik meg a monitoron, aztán a gép lefagy.

Ilyen esetben általában elegendő megoldás az, hogyha bekapcsoljuk a lilo linear opcióját az /etc/lilo.conf-ba írt linear kulcsszóval, vagy az /sbin/lilo -l opcióval. Ennek hatására a

map file-ba nem a C/F/S cím, hanem a lineáris szektorcím kerül bele, és ezek alapján a betöltő program a boot-olás során közvetlenül a BIOS-tól lekérdezett geometria szerint alakítja ki a C/F/S címeket.

A lilo.conf

A /etc/lilo.conf a LILO beállítási fájlja.

Felépítése nagyon egyszerű, az elején vannak az általános beállítások, a végén pedig a boot események.

Általános beállítások:

boot=eszköz ez az eszköz fogja tartalmazni a boot sectort. Lehetőségünk van megadni egy merevlemezt (/dev/hda), vagy csak egy sima partíciót is

default=címke ez lesz az alapértelmezett címke

delay=tmp ennyi időt fog tíz másodpercben várni a LILO hogy kiválasszuk a címkét

install=boot-szektor-fájl ezt a boot sector fájlt fogja telepíteni (vigyázzunk ezzel a beállítással)

lba32 engedélyezi a 32 bites LBA lekezelést, segítségével nem kell a kernelnek az első 1024 cylinderben elhelyezkednie

lock automatikusan megjegyzi az egyes kernelnek megadott paramétereit

map=map-fájl ez a fájl lesz a fentebb vázolt map fájl

message=üzenet-fájl ez a fájl ki lesz írva a képernyőre a LILO: prompt előtt.

Lehetőség van interaktív programokat is megadni, pl. hogy ne LILO: prompt jelenjen meg, hanem egy grafikus vagy menüs felület, és ott lehessen a nyilakkal címkét választani.

nowarn kikapcsolja az esetleges hibákra a figyelmeztetést

password=jelszó ez lesz a jelszó, egyedi paramétereknél, vagy nem alapértelmezett címke kiválasztásánál használatos

prompt felkínálja, hogy a LILO: promptnál várakozzon megadott másodpercig, és ezáltal lehessen másik boot címkét választani

restricted nem engedélyezi az egyedi kernel paraméterek használatát. Ha van jelszó megadva, akkor annak ismeretében ad csak engedélyt

timeout=tmp ennyi időt várakozik az egyes billentyűleütések között, ha az idő lejár, visszaugrik a LILO: prompthoz, ha ott jár le, akkor betölti az alapértelmezett címkét

Az egyes boot eseményeket (címkéket) az image vagy az other mezővel tudjuk megnyitni. Az ezek után következő sorok, egészen egy új image vagy other definiálásáig csak az adott image-re lesznek érvényesek (pl. egyéni jelszót is beállíthatunk, stb.)

image=fájl ezen a fájlban lesz a boot esemény értelmezve, pl. megadhatunk egy linuxos bzImage fájlt.

other=eszköz vezérlés átadása más rendszerindító kódnak, pl. ha egy merevlemezre adtunk meg, akkor az ottani MBR-nek fogja átadni a vezérlést, ha pedig egy partíciót, akkor a partíció boot sectora fogja megkapni azt

label=címke ez lesz a boot esemény címkéjének a neve

lock fentebb tárgyaltuk

optional ha nem található a fájl vagy az eszköz, a LILO nem fog reklamálni, és hibaüzenettel kilépni

password=jelszó egyéni jelszó

restricted lásd fentebb

Csak a linux kernel számára elérhető paraméterek:

append=szöveg ez a szöveg lesz a kernel paramétere, pontosabban ez is odakerül a kernel parancssorának végére

literal=szöveg csak ez a szöveg lesz a kernel paramétere

read-only engedélyezi, hogy a kernel a root meghajtót csak olvashatónak illessze fel, erre azért van szükség, mert ha a fájlrendszer hibákat tartalmaz, akkor azt csak ilyen állapotban tudjuk javítani

read-write engedélyezi, hogy a kernel írhatóként illessze fel a root eszközt

root=gyökér-eszköz ez a partíció tartalmazza a gyökér eszközt

vga=mód ebben az üzemmódban fogja a kernel elidíteni a VGA kártyánkat.

Indítólemez készítése

Az első teendő, hogy létre kell hoznunk egy kis konfigurációs állományt a LILO számára. Így kell kinéznie:

```
boot      = /dev/fd0
install   = /boot/boot.b
map        = /boot/map
read-write
backup     = /dev/null
compact
image      = KERNEL
label      = Bootdisk
root       = /dev/fd0
```

Valószínűleg szeretnénk a merevlemez */etc/lilo.conf* állományából hozzáadni az `append=...`-t tartalmazó sorokat ehhez az állományhoz.

Mentsük el ezt az állományt *bdlilo.conf* néven.

Létre kell hoznunk egy kis állományrendszert, amit kernel állományrendszernek hívunk, hogy megkülönböztethessük a gyökér állományrendszertől.

Először meg kell nézni, hogy milyen nagynak kell lennie ennek az állományrendszernek. Vegyük a kernel blokkokban vett méretét (ezt a méretet mutatja a `ls -l KERNEL`, amit 1024-el kell elosztani, majd felfele kerekíteni), és adjunk hozzá 50-et. Ötven blokk körülbelül elég lesz az indode-ok és néhány egyéb állomány számára. Pontosán is kiszámíthatjuk ezt a számot, ha szeretnénk. De egyszerűen használhatunk 50-et is. Ha két lemezes készletet készítünk, felül is becsülhetjük a szükséges helyet, mivel az első lemezen csak a kernelt fogjuk úgyis tárolni. Ezt a számot `KERNEL_BLOKK`-nak hívjuk. Helyezzünk egy lemezt a meghajtóba (az egyszerűség kedvéért feltételezzük, hogy ez az eszköz a */dev/fd0*), és készítsünk rajta ext2 kernel állományrendszert:

```
mke2fs -i 8192 -m 0 /dev/fd0 KERNEL\_BLOKK
```

Az `-i 8192` mondja meg, hogy egy inode legyen minden 8192 byte-on. Ezután beilleszthetjük az állományrendszert, távolítsuk el a *lost+found* alkönyvtárat és hozzuk létre a *dev* és a *boot* alkönyvtárakat a LILO számára:

```
mount /dev/fd0 /mnt
rm -rf /mnt/lost+found
mkdir /mnt/{boot,dev}
```

Ezután hozzuk létre a `/dev/null` és a `/dev/fd0` eszközöket. Ahelyett, hogy nekiállnánk eszközszámokat keresni, egyszerűen másoljuk át őket a merevlemezről:

```
cp -R /dev/{null,fd0} /mnt/dev
```

A LILO-nak szüksége van egy másolatra a betöltőjéhez (ez a `boot.b`), ezt lemásolhatjuk a merevlemezről, általában ezt a `/boot` könyvtárban találjuk.

```
cp /boot/boot.b /mnt/boot
```

Végül másoljuk át az előzőleg elkészített LILO konfigurációs állományt a kernellel együtt. Mindkettőt a gyökér alkönyvtárba tehetjük:

```
cp bdlilo.conf KERNEL /mnt
```

A LILO-hoz szükséges összes állomány most már a kernel állományrendszeren van, tehát készen áll arra, hogy futtassuk. A LILO-nak az `-r` paraméterrel mondhatjuk meg, hogy más gyökérrel telepítse az indító betöltőt:

```
lilo -v -C bdlilo.conf -r /mnt
```

A LILO-nak hiba nélkül le kell futnia. A kernel állományrendszernek valahogy így kell kinéznie:

```
total 361
  1 -rw-r--r--    1 root    root           176 Jan 10 07:22 bdlilo.conf
  1 drwxr-xr-x    2 root    root          1024 Jan 10 07:23 boot/
  1 drwxr-xr-x    2 root    root          1024 Jan 10 07:22 dev/
358 -rw-r--r--    1 root    root        362707 Jan 10 07:23 vmlinuz
boot:
total 8
  4 -rw-r--r--    1 root    root          3708 Jan 10 07:22 boot.b
  4 -rw-----    1 root    root          3584 Jan 10 07:23 map
dev:
total 0
  0 brw-r-----    1 root    root         2,   0 Jan 10 07:22 fd0
  0 crw-r--r--    1 root    root         1,   3 Jan 10 07:22 null
```

Ne aggódjunk, ha az állományok méretei egy kicsit eltérnek.

A kernel image belsejében van az az információ, ami megadja, hogy hol található a gyökér állományrendszer, több más paraméterrel együtt. Ez az információ az `rdev` utasítással érhető el, és a következőképpen kell értelmezni:

bits 0-10 Az eltolás mértéke, hogy hol kezdődik a ramdisk, 1024 bájtos blokkméretben megadva

bits 11-13 Nem használt

bit 14 Igaz hamis érték, jelzi, hogy van-e ramdisk, amit be kell tölteni

bit 15 Igaz hamis érték, jelzi, hogy kell-e prompt mielőtt betölti a gyökér fájlrendszert

Ha a 15. bit be van állítva, induláskor új lemez meghajtóba tételére szólít fel a rendszer. Erre kétlemezes indító készlet esetén van szükség. Két eset lehetséges, attól függően, hogy egy indító/gyökér lemezt készítünk, vagy kétlemezes *indító+root* lemezkészletet.

Ha egyetlen lemezt készítünk, a tömörített gyökér állományrendszer közvetlenül a kernel után kerül, tehát az eltolás az első szabad blokkra (aminek meg kell egyeznie a KERNEL_BLOKK-al) fog mutatni. A 14. bit 1, a 15. nulla.

Például, tegyük fel, hogy egy lemezt készítünk, és a gyökér állományrendszer a 253. (decimális) blokknál kezdődik. A memórialemez szó értékének 253-nak (decimális) kell lennie, a 14. bitet 1-re, a 15. bitet 0-ra kell állítani.

Ahhoz, hogy kiszámítsuk az értéket, egyszerűen csak össze kell adni a decimális számokat. $253 + 2^{14} = 253 + 16384 = 16637$. Ha nem érthető, hogy honnan jönnek ezek a számok, üssük be őket egy tudományos számológépbe, és alakítsuk át őket binárisra. Ha két lemezes készletet készítünk, a gyökér állományrendszer a második lemez 0. blokkjánál kezdődik, tehát az eltolást nullára kell állítani. A 14. bitet 1-re kell állítani, és a 15. bitet is 1-re kell állítani. Ebben az esetben a decimális érték $2^{14} + 2^{15} = 49152$.

Miután gondosan kiszámítottuk a memórialemez szó értékét, állítsuk be az *rdev -r* parancssal. Figyeljünk rá, hogy a decimális értéket használjuk. Ha LILO-t használtunk, az *rdev* paraméterének a beillesztett kernel útvonalat kell használni, pl. */mnt/vmlinuz*, ha *dd*-vel másoltuk a kernelt, használjuk inkább a lemezeszköz nevét (pl., */dev/fd0*).

```
rdev -r KERNEL\_VAGY\_FLOPPY\_MEGHAJTÓ ÉRTÉK
```

Most lecsatolhatjuk a lemezt.

Példa egy sokoldalú lilo.conf-ra

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=650
restricted
```

```
password=tcstcs

default=32

other=/dev/hda2
    label=win
    table=/dev/hda
image=/boot/zImage32
    label=32
    root=/dev/hda1
    read-only
    optional
image=/boot/vmlinuz-2.0.32
    label=old
    root=/dev/hda1
    read-only
image=/boot.deb/zImage
    label=debiandef
    root=/dev/hda4
    read-only
other=/dev/hdb1
    label=WindowsNT
    table=/dev/hda
    loader=/boot/any_d.b
```

3.2.2. Indítólemez készítése kézzel

A kernel boot loader (vagy másnéven boot manager) nélkül is képes beindulni, de csak floppy lemezről. Ugyanis a kernel legelső 512 bájtja egy speciális boot loader, ami képes betölteni a kernelt, ha annak adatai folyamatosan egymás után vannak felírva az adott médiumra (tehát nem valamilyen fájlrendszeren található!).

A feladat tehát nem más, mint hogy kézzel rámásoljuk a kernel image fájlt egy floppyra, majd ráírjuk a kellő paramétereket is. A másolásra a *dd* program használható:

```
dd if=KERNEL of=/dev/fd0 bs=1k
353+1 records in
353+1 records out
```

Ebben a példában a *dd* 353 teljes rekordot + 1 részleges rekordot írt, így a kernel a lemez első 354 blokkját foglalja el. Ezt a számot *KERNEL_BLOKK*-nak hívjuk.

Végül állítsuk be gyökér eszköznek magát a lemezt, majd tegyük a gyökeret írható/olvasható, és betölthetővé:

```
rdev /dev/fd0 /dev/fd0
rdev -R /dev/fd0 0
```

Legyünk óvatosak, a második *rdev* utasításnál nagybetűt használjunk -R kapcsoló esetén.

Ezután állítsuk be a gyökér állományrendszer elérési helyét a LILO-nál megismert módon!

3.2.3. Linux indítása DOS alól

A linux elindítása DOS alól a LINUXLOADER nevű programmal valósítható meg. A programot csak és kizárólag DOS alól tudjuk használni, ha valamilyen Windows fut, akkor a LINUXLOADER nem fog elindulni!

Ha DOS-os indítómenüt kívánunk alkalmazni, úgy a LINUXLOADER meghívására a legalkalmasabb hely maga a CONFIG.SYS.

A program használata nagyon egyszerű, ki kell másolni a kernelt és ha szükséges, akkor az initrd -t tartalmazó fájlt a LINUXLOADER könyvtárba, majd a megfelelő paramétereket megadva indítsuk el a LINUXLOADER-t.

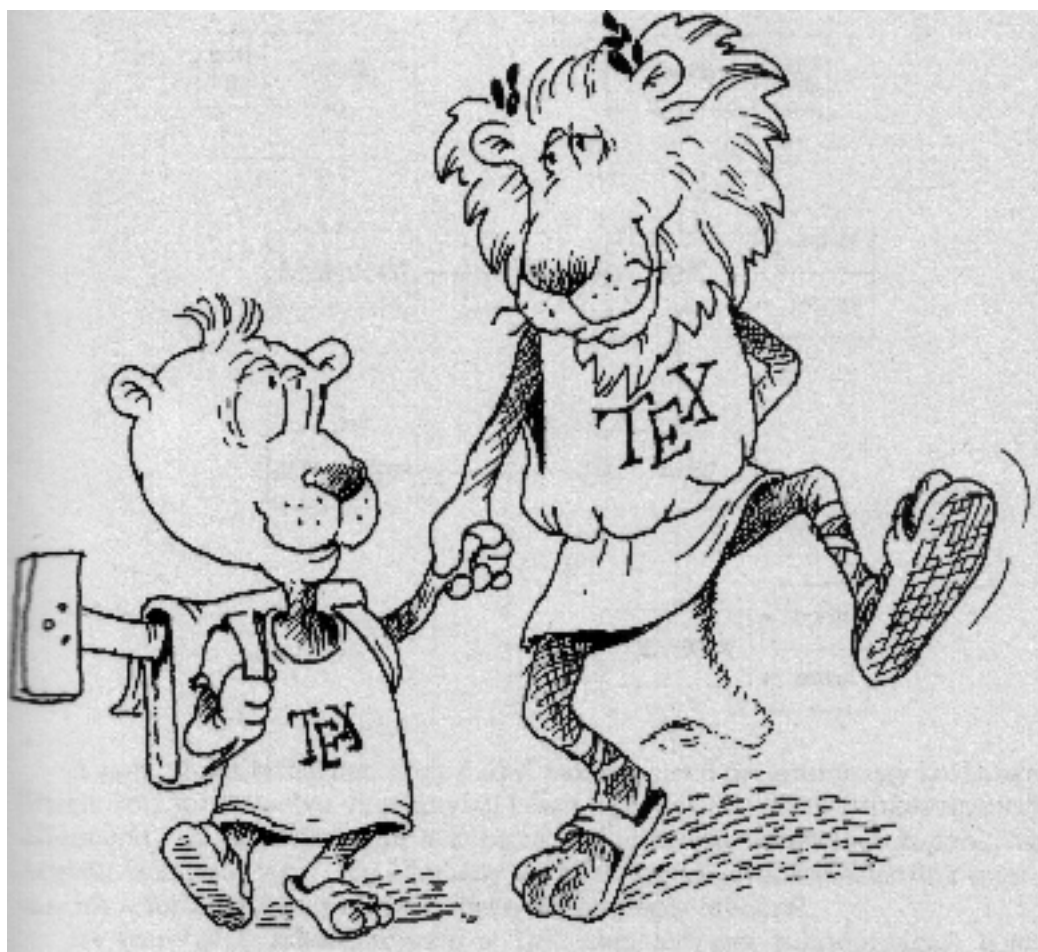
3.3. Felhasznált irodalom

- From Power Up To Bash Prompt HOWTO by Greg O'Keefe <gcokeefe@postoffice.utas.edu.au>
- The Unix and Internet Fundamentals HOWTO by Eric S. Raymond
- Kernel HOWTO by Al Dev (Alavoor Vasudevan) <alavoor@yahoo.com>
- Linux 2.4.x Initialization for IA-32 HOWTO by Randy Diunlap <rddunlap@ieee.org>
- Linux Kernel forráskód by Linus Torvalds <torvalds@transmeta.com>
- Linux Oktatási anyag by Kósa Attila
- Indítólemez Hogyan by Tom Fawcett <fawcett@croftj.net>

- inittab kézikönyv by Sebastian Lederer <lederer@francium.informatik.uni-bonn.de> and Michael Haardt <u31b3hs@pool.informatik.rwth-aachen.de>
- Linux Standard Base spec 1.3 – <http://www.linuxbase.org/>

4. fejezet

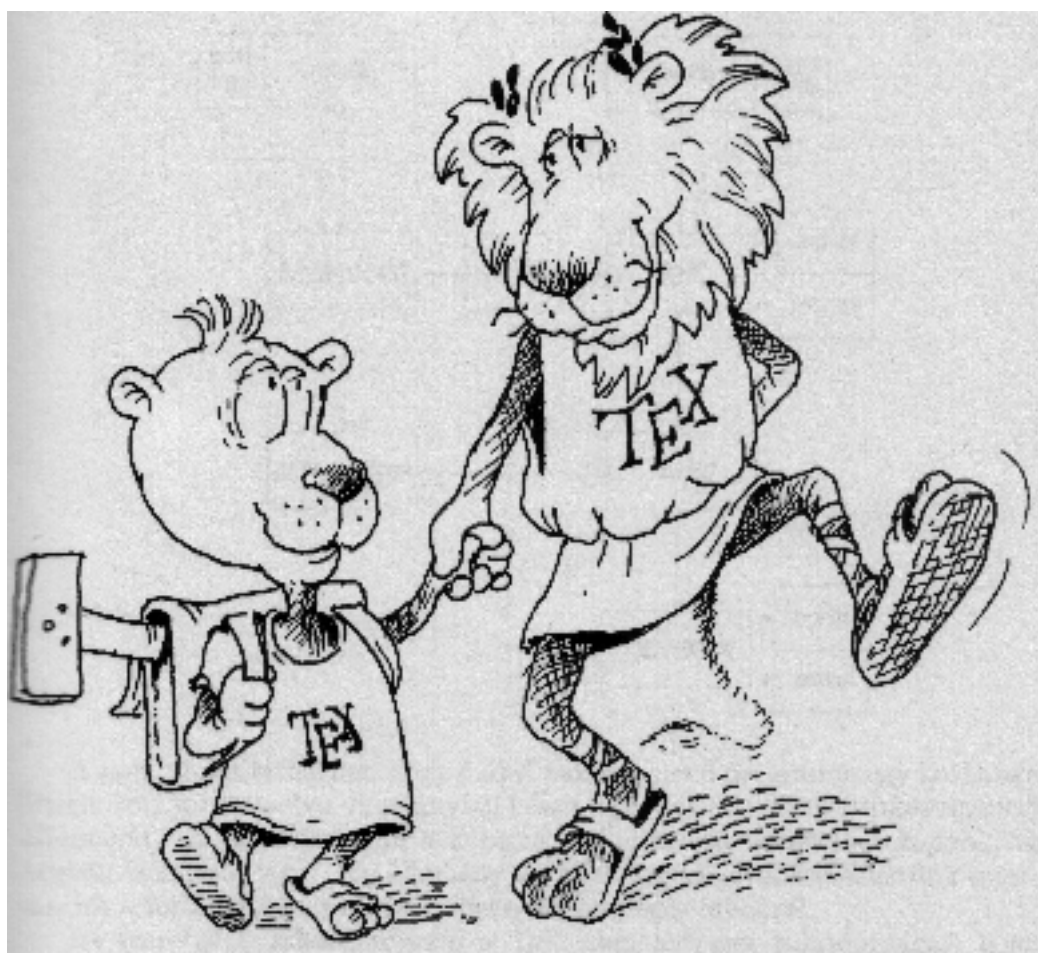
Dokumentumkezelés



Acrobat Reader, xpdf, gpdf, ggv, eog, gqview

5. fejezet

Bejelentkezés



UHU-Linux alatt a ma használatos grafikus kártyák szinte mindegyikén működik a grafikus felület (amit a Unix világban *X servernek* nevezünk).

Amennyiben nem akarunk ilyet használni, vagy régebbi típusú grafikus kártyánk miatt nincs rá lehetőségünk, akkor az ebben a fejezetben leírtak tanulmányozását nyugodtan mellőzhetjük.

A továbbiakban részletesen bemutatjuk a rendszerben használható két *Grafikus bejelentkezés kezelőt*.

5.1. Az Gnome Display Manager (gdm)

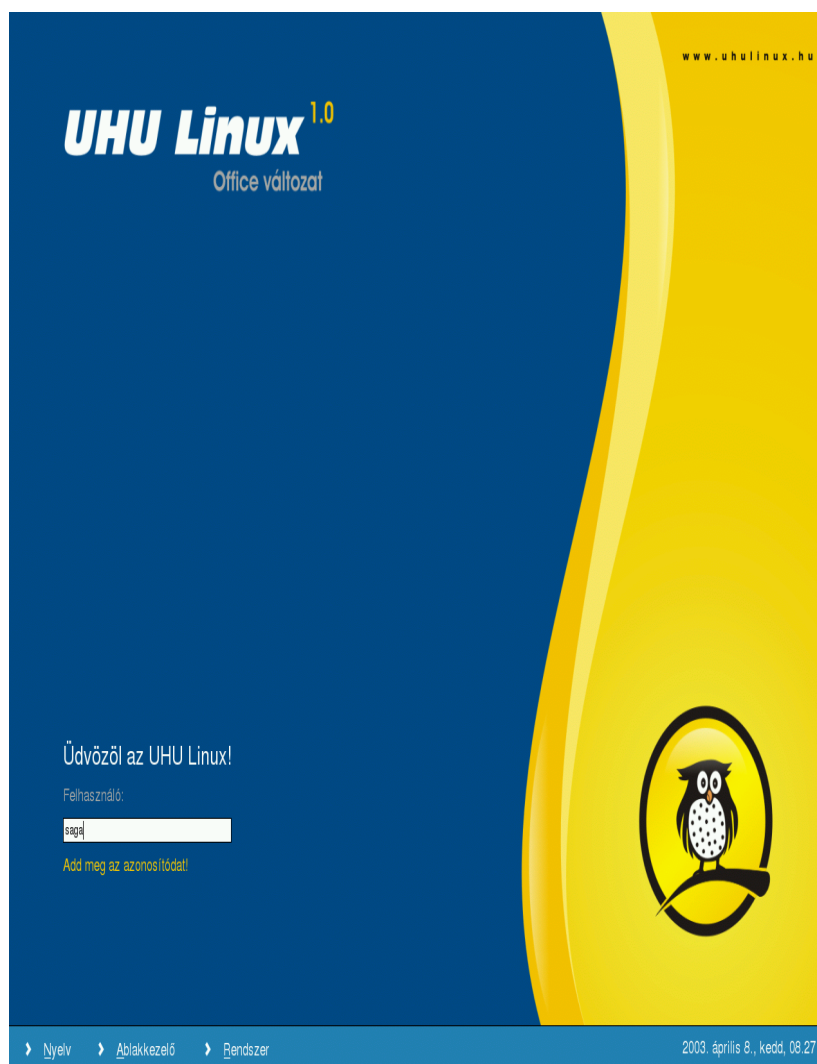
Az UHU-Linux alapértelmezésként a Gnome grafikus rendszert használja, ennek eredményeként ilyenkor a *gdm* (5.1 ábra) kezeli a grafikus felületre történő bejelentkezéseket.

A *gdm* menüsorában találunk egy *Munkamenet* nevű menüpontot, amelyben a kívánt ablakkezelőt választhatjuk ki.

Linux alatt, a grafikus felület teljes mértékben átszabható, a legkisebb mértékben sincs megkötve kezünk e téren. A legismertebbek (*KDE* és *Gnome*) mellett rengeteg egyszerűbb, és emiatt jóval kisebb erőforrást igénylő ablakkezelők is léteznek (pl. *IceWm*, *BlackBox*, *Window Maker*). Ezek némelyike alapesetben olyan puritánnak tűnik, hogy látszólag semmi sem jelenik meg a grafikus munkaterületen.

Az egér gombjai segítségével ilyenkor természetesen előtűnnek az első pillanatban hiányolt menük.

A menüsor második eleme a *Nyelv* kiválasztása. Ezzel a kiválasztott grafikus felület nyelvezetét állíthatjuk be. Mint látható, elég széles körű az alkalmazható nyelvek választéka.



5.1. ábra. A gdm ablaka

A harmadik menüpont a *Rendszer*, amely az UHU-Linux leállítására és újraindítására szolgál. Használatuk értelemszerű.

A *gdm* teljeskörű beállítása az *Alkalmazások/Rendszereszközök/GDM Beállító* pontból indítható *gdmsetup* nevű programmal végezhető el.



5.2. ábra. Munkamenet kiválasztása

5.2. A KDE Display Manager (kdm)

A *kdm* bejelentkezéskezelő használatának beállítása az *UHU Vezérlőpult/Szolgáltatások* pontjában történhet, a következő módon:

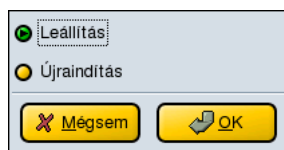
- A *gdm*-re állítsuk be, hogy *Ne induljon el*, és a megjelenő kérdésre válaszolva ne “állítsuk le a szolgáltatást”!
- A *kdm*-re állítsuk be, hogy *Induljon el*, de a megjelenő kérdésre válaszolva ne “indítsuk el a szolgáltatást”!
- Lépjünk ki a grafikus felületről, és indítsuk újra a UHU-Linux-ot.
- Újraindulás után már a *kdm* fog bejelentkezni.

Fentiek természetesen újraindítás nélkül is megoldhatók, de mivel ez a Kódex kezdőknek szól, a legegyszerűbb megoldást ismertettük.

A felhasználókat jelképező ikonokat (sok egyéb mellett) a *KDE Vezérlőközpontban*, a *Rendszeradminisztráció/Bejelentkezéskezelő (KDM)* pontban (5.13 ábra) tudjuk igényünk szerint beállítani. Természetesen előtte meg kell majd adnunk a rendszergazda jelszavát.

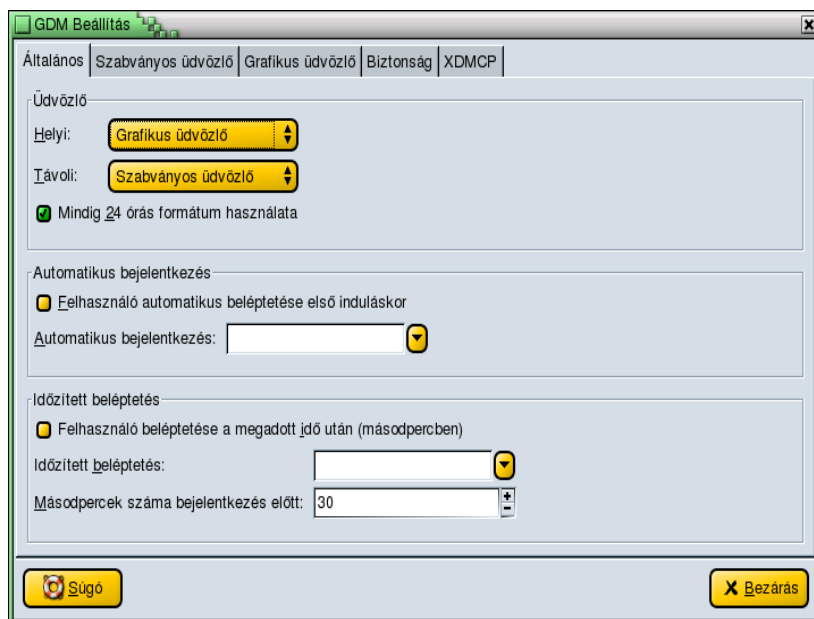


5.3. ábra. Nyelv meghatározása

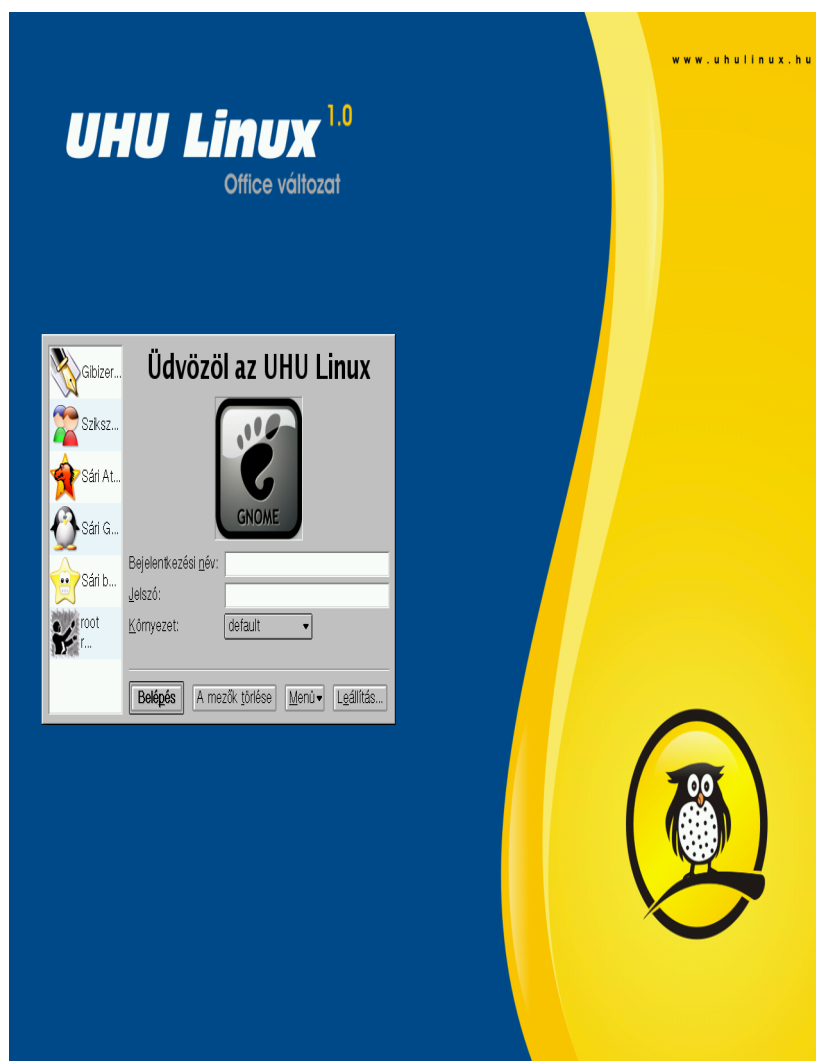


5.4. ábra. A Rendszer menü

A Megjelenés fülön a *kdm* ablakában látható Üdvözlő szöveget, az embléma területen megjelenő ábra kinézetét, a *kdm* által használt nyelvet, az ablak pozícióját, stílusát, és a jelszó mezőben megjelenő karaktereket módosíthatjuk.

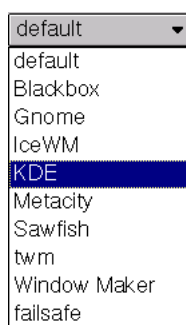
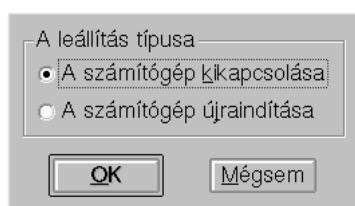
5.5. ábra. A *gdmsetup* általános beállításai

A Betűtípus fülön a *kdm* által használt három betűtípust módosíthatjuk, valamint az Élsimított betűtípusok használatát kapcsolhatjuk ki és be.

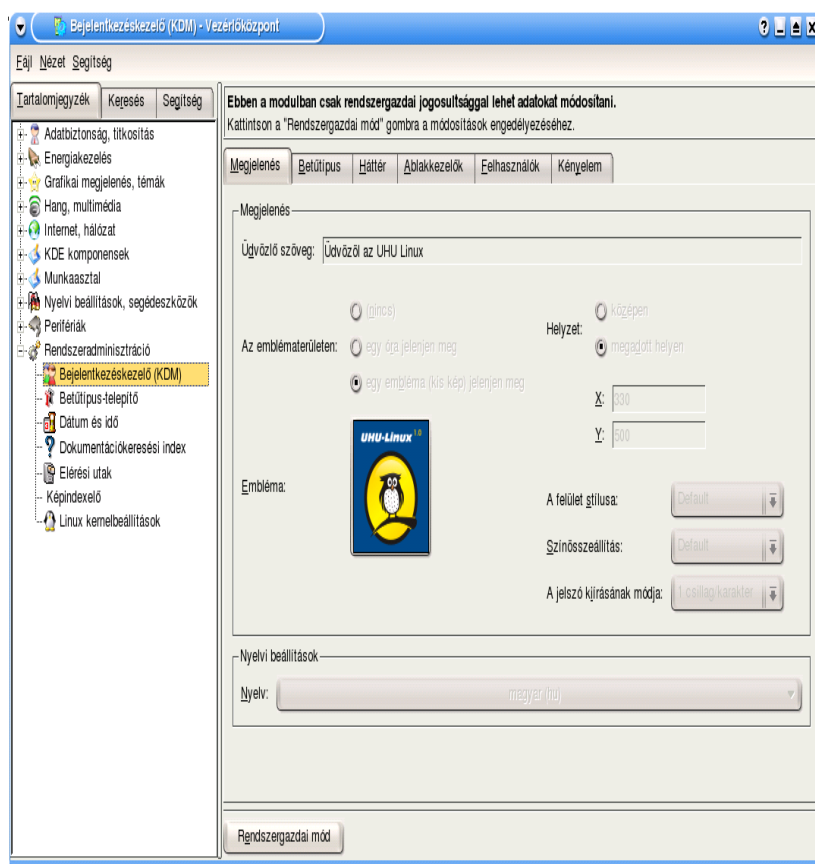


5.6. ábra. A *kdm* grafikus bejelentkezéskezelő

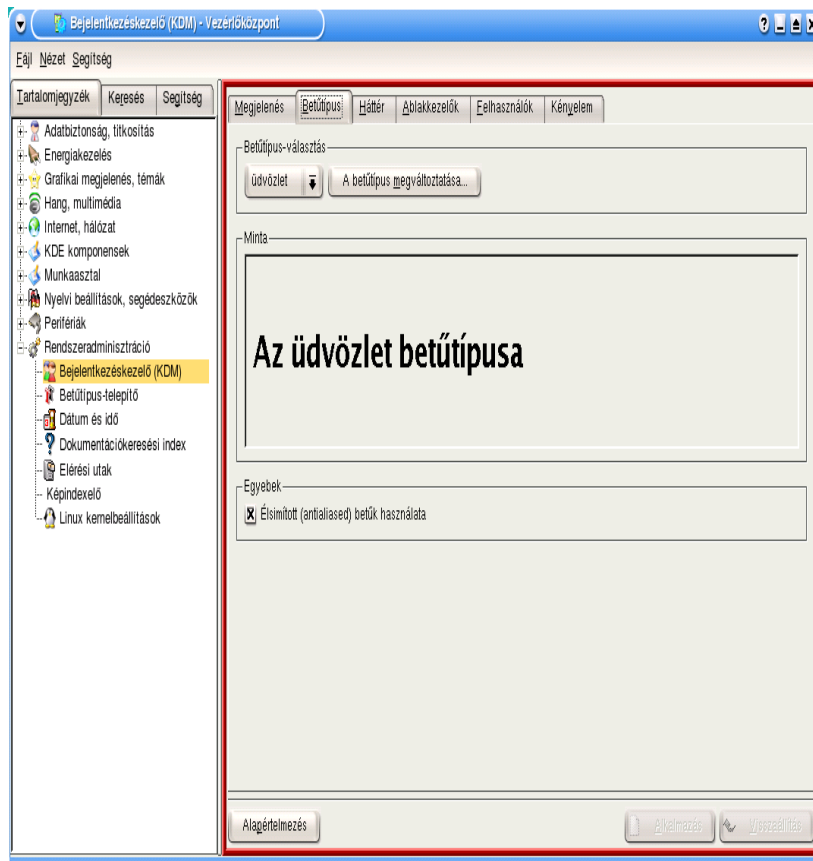
A Háttér fülön a *kdm* háttérét alakíthatjuk át kedvünk szerint.

5.7. ábra. A *kdm* Környezet beállításai5.8. ábra. A *kdm* Leállítás menüje

Az Ablakkezelők fülön a *kdm* Környezet menüjében megjelenő és kiválasztható ablakkezelők indító parancsait szerkeszthetjük.

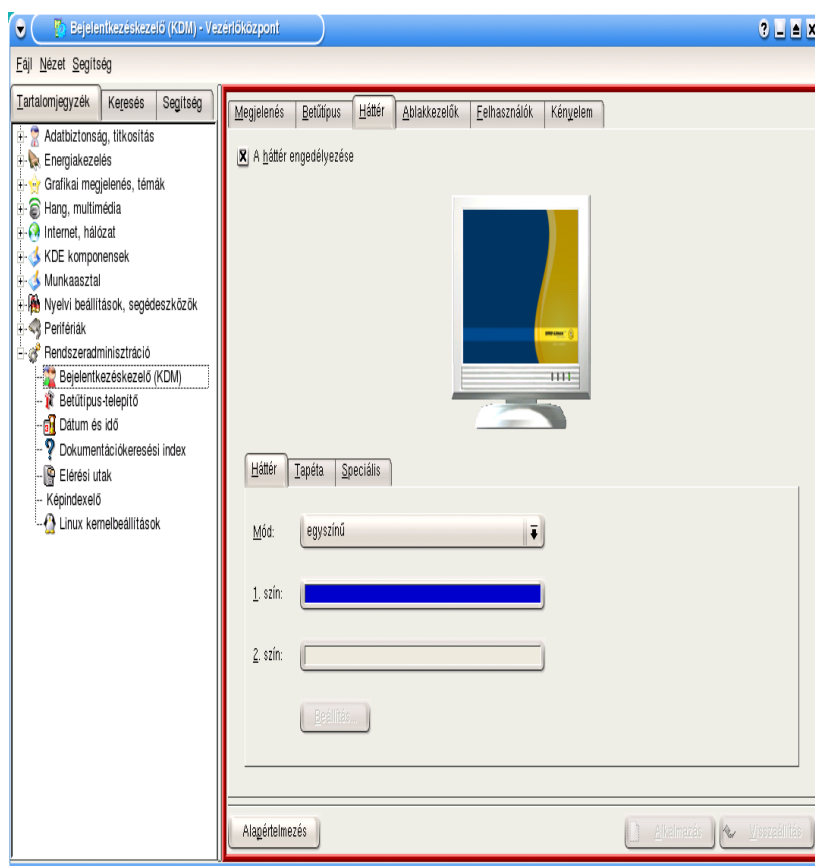
5.9. ábra. A *kdm* testreszabása (Megjelenés)

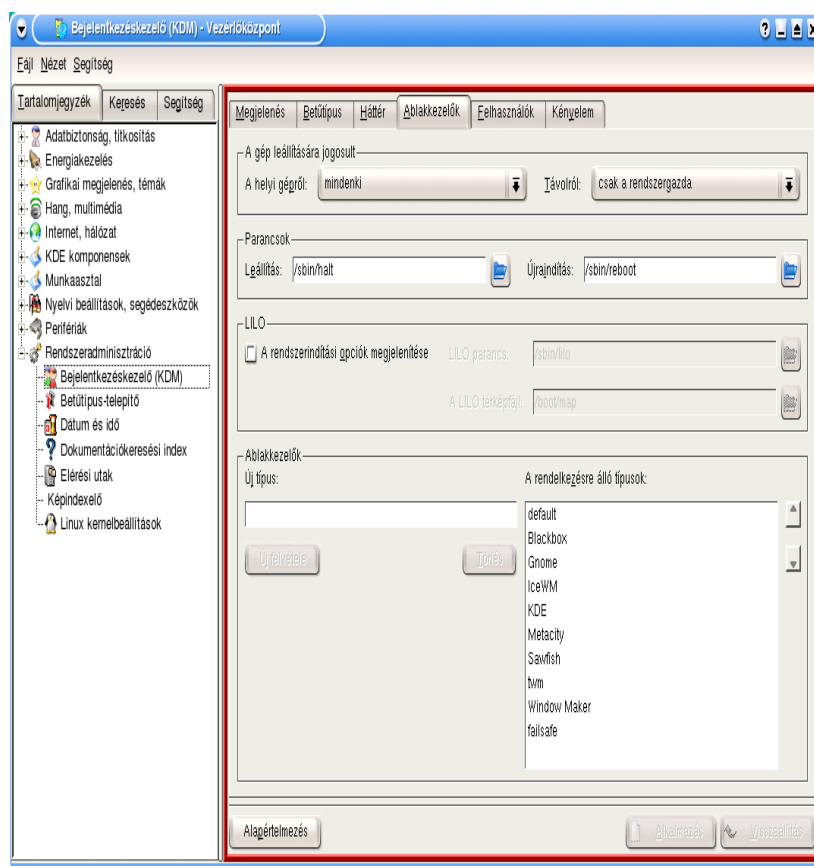
A Felhasználók fülön a megjelenő felhasználókat tudjuk beállítani és itt tudjuk a már említett ikon hozzárendeléseket elvégezni.

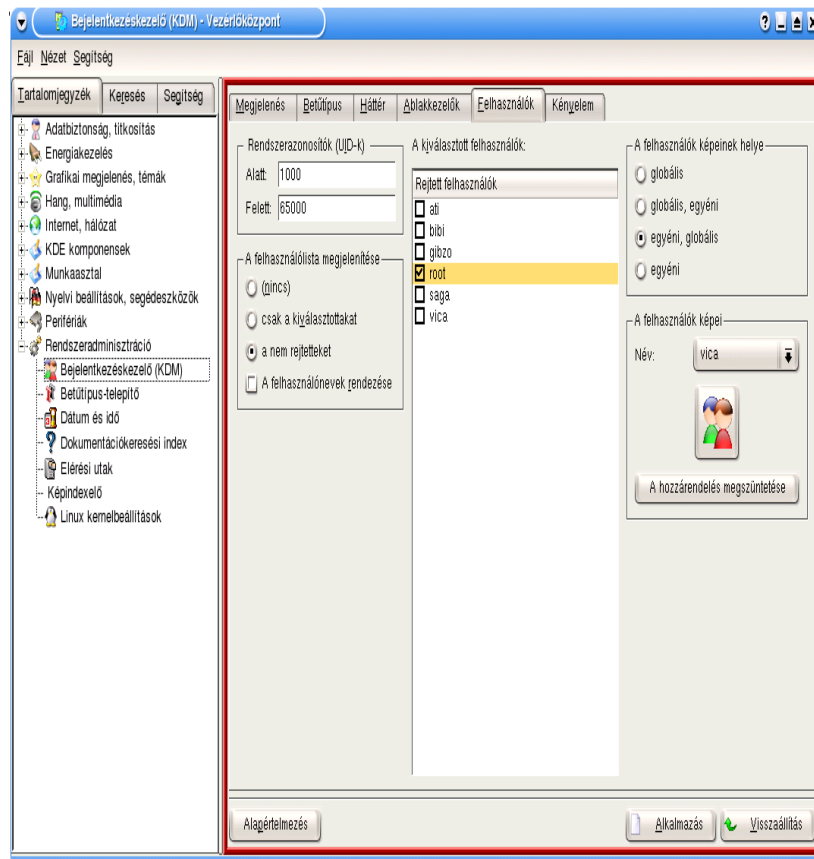
5.10. ábra. A *kdm* testreszabása (Betűtípus)

A Kényelem fülön olyan, látszólag hasznos dolgokat tudunk beállítani, amik valójában egy megfelelően beállított rendszeren akaratlanul is biztonsági lyukakat nyitnak. Ha törekszünk a biztonságra, akkor itt lehetőleg semmiféle kényelmes tulajdonságot ne állítsunk be!

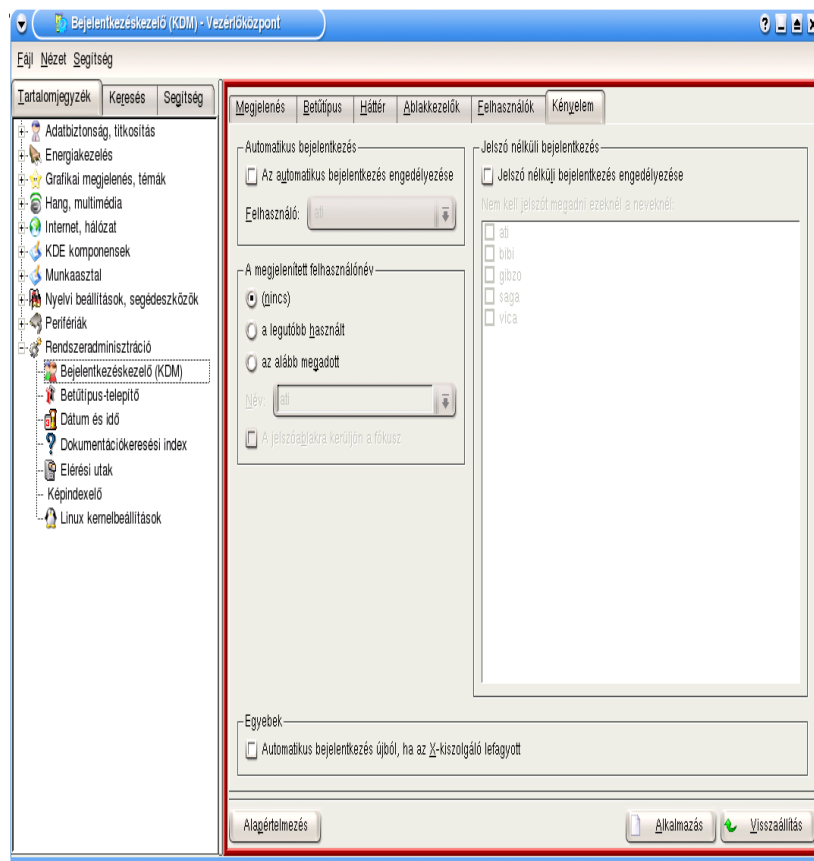
Ne feledkezzünk meg arról sem, hogy minden grafikus kiegészítés, mint például a háttérképek vagy az animációs mozgások, csökkentik gépünk szabad erőforrásait. A processzoridőből viszonylag keveset vesznek el, de a memória kihasználtságát erősen befolyásolhatják.

5.11. ábra. A *kdm* testreszabása (Háttér)

5.12. ábra. A *kdm* testreszabása (Ablakkezelők)

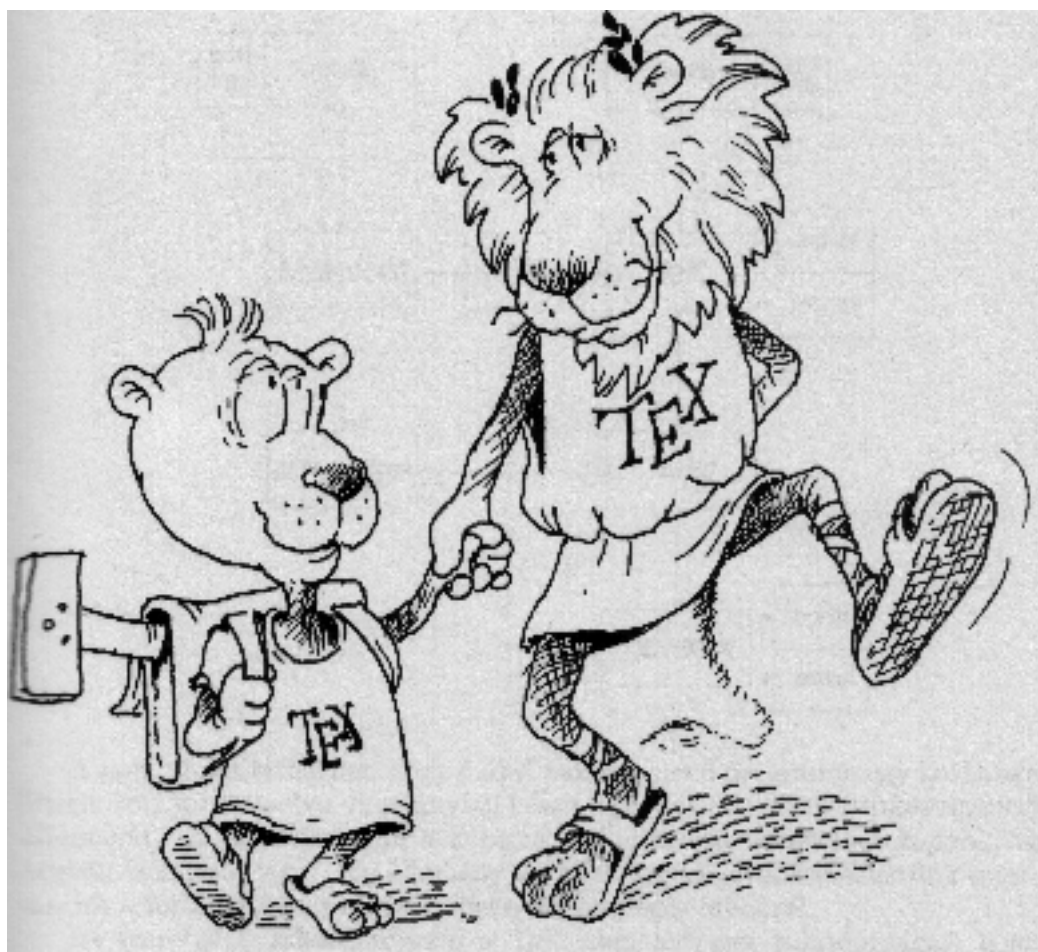


5.13. ábra. A kdm testreszabása (Felhasználók)

5.14. ábra. A *kdm* testreszabása (Kényelem)

6. fejezet

Felhasználókezelés



6.1. Több felhasználó, több program

A Linux egy valódi több felhasználós rendszer. Ez azt jelenti, hogy egyszerre több felhasználó kezelésére is alkalmas.

Multiuszer

Több felhasználó egyidejű kiszolgálását jelenti. Ez nem csak fájlok megosztását jelenti, hanem inkább programok futtatását. Tehát egy gépre több ember jelentkezhet be egyszerre, és egyszerre tudnak dolgozni anélkül, hogy zavarnák egymás munkáját. Ez maga után vonja azt, hogy a rendszernek meg kell tudnia különböztetni egymástól a felhasználókat.

Minden felhasználóhoz előre definiált jogok és engedélyek tartoznak, amelyeket a rendszer mindig ellenőriz, amikor a felhasználó szeretne hozzáférni valamihez. Ez szükséges az erőforrások megfelelő szétosztása, és a biztonság érdekében.

Multitask

Több feladat egyidejű végrehajtását jelenti. Az egy processzorral rendelkező számítógépeken az egyidejű végrehajtás csak látszólagos, hiszen a processzor csak egy feladattal tud foglalkozni egyszerre; tehát a feladatok felváltva kapják meg a processzort.

A legkisebb egység, amely párhuzamos feldolgozásra kerülhet - a processz. A feladatok váltogatását az ütemező végzi, amely különböző stratégiák szerint dolgozhat. Csak felsorolásszerűen a lehetséges stratégiák:

- Round robin ütemezés,
- prioritásos ütemezés,
- a legrövidebb feladatot először elv,
- garantált ütemezés,
- sorsjáték-ütemezés,
- valós idejű ütemezés,
- kétszintű ütemezés.

A Linux prioritási szinteket használ ezáltal lehetővé téve, hogy a felhasználó megválogtassa saját processzeinek prioritását.

A Linux preemptív multitaszkos operációs rendszer, ami azt jelenti, hogy amikor egy adott folyamat számára kijelölt időszak letelt, akkor a kernel megszakítja a folyamat futását, és másik folyamatnak adja át a vezérlést. Az operációs rendszer nem teszi lehetővé, hogy egy folyamat a végtelenségig magánál tartsa a vezérlést, és így megakadályozza a többi folyamat futását.

Azonban a prioritási soron belül lehetőség van a szokásos (Round robin) ütemezés helyett FIFO ütemezés kérésére is - ezáltal szoft-real-time ütemezést is meg lehet valósítani.

Ilyenkor a rendszer nem veszi el a futás jogát a processztől, csak ha az lemond róla. Ezért a fejlesztőknek kell arra figyelniük, hogy ne blokkolják a teljes rendszert. Linuxban az ütemezés alapegysége 1/100 másodperc. Azonban a Linux nem real-time operációs rendszer (de van ilyen irányú fejlesztése is), és ez azt jelenti, hogy több futó folyamat esetén bizonyos időközönként mindegyikre rákerül a vezérlés, azonban a két aktív állapot között eltelt időre nincs szigorú időkorlát.

Egy frissen telepített rendszerben már a telepítés közben találkozhatunk a felhasználók létrehozásával. Természetesen később is lehetőségünk van új felhasználókat felvenni a rendszerbe, vagy onnan törölni, tulajdonságaikat-, jogosultságaikat megváltoztatni.

A rendszerben a felhasználókat felhasználói név és jelszó alapján különböztetjük meg. A felhasználók között található egy kiemelt felhasználó, akit root-nak, azaz rendszergazdának hívunk. A rendszergazda az a személy, aki a rendszerben bármit csinálhat.

Alapértelmezés szerint Ő az egyetlen, aki felhasználókat vehet fel, rendszerszintű alkalmazásokat indíthat, stb. Ügyetlenségével adott esetben komoly károkat is okozhat. Az egyszerű felhasználók jogai, lehetőségei lényegesen korlátozottabbak, bár ezeket is a root határozza meg.

A felhasználókat csoportokba szervezzük. Ennek előnye, hogy a csoport jogait már előre definiálhatjuk, így nem kell egyenként minden felhasználónak jogot adni pl. arra, hogy nyomtathasson, internetezzen, vagy zenét hallgathasson. A csoport azonosításhoz számokat használunk. Egy felhasználó lekérdezéséhez, tehát ahhoz, hogy megállapítsuk milyen csoportba tartozik, használjuk az *id* parancsot.

useradd felhasználó hozzáadása

userdel felhasználó törlése

users az aktuális hoszton bennlevő felhasználók neveit írja ki

userpasswd felhasználó jelszavának módosítása

userinfo felhasználó adatainak módosítása

usrmod felhasználó adatainak módosítása

userhelper felhasználó adatainak lekérdezése

UHU-Linuxban a felhasználókezelést az *uhu-control-center* végzi. Ez egy grafikus program, aminek bal oldalán a fa struktúrában ki lehet választani a "Felhasználók és csoportok" alatt található "Csoportok" és "Felhasználók" elemet.

6.2. Hozzáférési jogosultságok

A fájlokhoz tartozó hozzáférési jogosultságok meghatározzák, hogy melyik felhasználó melyik fájlra hajthat végre műveletet, miközben magát a műveletet is meghatározza. A Linux a felhasználókat három csoportra osztja, amikor a fájlokhoz és könyvtárakhoz való viszonyukat vizsgálja:

- a fájl tulajdonosa (user),
- csoport (group),
- egyéb (others).

A "csoport" segítségével csoportosíthatjuk a felhasználókat – lehetőleg – valamilyen logika szerint, hogy meg tudják osztani egymással a megfelelő fájlokat vagy szabályozni tudjuk a hozzáféréseket.

A fájlokkal és könyvtárakkal három dolgot tehetnek a felhasználók:

- olvasás (read)
- írás (write)
- végrehajtás (execute)

Mindhárom felhasználói csoportra külön-külön be lehet állítani ezeket az engedélyeket. Fájloknál elég egyértelmű, hogy mit takar ez a három dolog: olvasás, írás és végrehajtás, de a könyvtárak esetében már nem az.

Egy könyvtár olvasása azt jelenti, hogy egy erre a célra szolgáló paranccsal ki tudjuk listázni a könyvtár tartalmát.

Egy könyvtárat akkor nevezünk írhatónak, ha a megfelelő parancsokkal bejegyzéseket tudunk benne létrehozni, módosítani vagy törölni az adott könyvtárban, illetve törölhetjük magát a könyvtárat.

Talán a végrehajtás a legnehezebben megérthető, mert hasonlít egy kicsit az olvasás engedélyhez.

Abban az esetben, ha egy könyvtárba csak olvasási jogunk van, akkor nem tudjuk megnevezni a fájloknak a bejegyzéseit; míg ha végrehajtási jogunk van csak, akkor - ha ismerjük a fájl teljes nevét - az "ls" parancs megmutatja a fájlt (de csak azt az egy fájlt, mert a könyvtárban esetleg lévő többi fájlt már nem fogjuk látni), és el is tudjuk indítani (amennyiben az egy végrehajtható script vagy program, és van engedélyünk a fájlra).

Egy példán keresztül magyarázzuk el a fájlokra vonatkozó engedélyeket, és azok hatását a felhasználókra:

```
-rwxr-xr-- 1 gibzo users 3450 2003-06-27 19:16 program.sh
```

A legelső karakter a "-" jel helyén a következő jelölések állhatnak:

- közönséges fájl
- b** blokkos eszköz (például: adattároló eszközök)
- c** karakteres eszköz (például: nyomtató, terminál, stb.)
- d** könyvtár
- l** szimbolikus link
- p** pipe (csatorna)
- s** socket

A következő kilenc karakter azt mutatja, hogy ki mit tehet a fájllal:

- r** olvasási engedély (read)
- w** írási engedély (write)
- x** végrehajtási engedély (execute)
- az engedélyek hiánya
- s** suid bit
- t** sticky bit

Tehát a fenti fájl jelölései sorrendben a következő dolgokat jelentik:

- közönséges fájl

r olvasási engedély a tulajdonos számára

w írási engedély a tulajdonos számára

x végrehajtási engedély a tulajdonos számára

r olvasási engedély a csoport számára

- az írási engedély hiánya a csoport számára

x végrehajtási engedély a csoport számára

r olvasási engedély bárki más számára

- írási engedély hiánya bárki más számára

- végrehajtási engedély hiánya bárki más számára

Egy példán keresztül elmagyarázzuk a könyvtárakra vonatkozó engedélyeket, és azok hatását a felhasználókra:

```
drwxr---x    2 gibzo    users          4096 2003-06-27 19:15 konyvtar
```

A "konyvtar" nevű könyvtár jelölései sorrendben a következő dolgokat jelentik:

d könyvtár

r olvasási engedély a tulajdonos számára

w írási engedély a tulajdonos számára

x végrehajtási engedély a tulajdonos számára

r olvasási engedély a csoport számára

- az írási engedély hiánya a csoport számára

- végrehajtási engedély hiánya a csoport számára

- olvasási engedély hiánya bárki más számára

- írási engedély hiánya bárki más számára

x végrehajtási engedély bárki más számára

Egy kicsit bővebben leírva: a tulajdonos bármit tehet a könyvtárral (könyvtárban) - ki-listázhathatja (olvashatja) a tartalmát, fájlokat és könyvtárakat hozhat létre benne és elindíthatja az itt található programokat (amennyiben azokra a fájlokra van engedélye).

A csoport számára csak olvasási engedély van adva, tehát láthatja a könyvtárban lévő fájlneveket, de magukat a fájlokat nem. Tehát amennyiben a könyvtárban található olyan fájl, ami olvasható a csoport számára, akkor sem fogják tudni azt elolvasni a csoport tagjai, mert magát a fájlt sem látják! Csak a bejegyzését látják a könyvtár fájlban.

Bárki más számára pedig csak végrehajtási engedély van megadva. Tehát sem írni, sem olvasni nem tudják a könyvtárat. De ha tudják egy fájl (vagy könyvtár) nevét, akkor azt ki tudja listázni, tehát látni fogják a fájl (vagy könyvtár) összes bejegyzését. Sőt, ha a fájlra vonatkozó jogosultságok megengedik, akkor akár írhatják is az adott fájlt (vagy könyvtárt).

6.2.1. A SUID és a SGID bitek

A `suid` egy rövidítés, a "Set User Identification" kifejezés rövidítése. Azt jelenti, hogy a "felhasználói azonosító megváltoztatása".

Megértéséhez tudnunk kell azt, hogy időnként szükség van arra, hogy egy egyszerű felhasználó egy privilegizált felhasználó jogaival rendelkezzen. Talán a legegyszerűbb eset a jelszó megváltoztatása.

Egy egyszerű felhasználó nem írhatja közvetlenül a rendszer jelszófájlját (*/etc/passwd*), hiszen akkor bármikor korlátlan jogokhoz juthatna, de a saját jelszavát meg kell tudnia változtatni.

Ehhez viszont írnia kell a jelszófájlba. Ezt az ellentmondást oldják fel úgy, hogy a programot ruházzák fel privilegizált jogokkal, a `suid` bit beállításával.

A `passwd` parancs engedélyei a következők:

```
-rwsr-xr-x    1 root      root          24152  2003-04-07 05:18 /usr/bin/passwd
```

Látható, hogy a `suid` bit be van kapcsolva, így futásának idejére az őt futtató felhasználó rendszergazdai jogkört kap, tehát a `root` jogosultságaival olvassa és írja a */etc/passwd* fájlt (mivel a `root` tulajdonában van a fájl).

A Unix típusú rendszerekben a programok azokkal a felhasználói jogokkal futnak, amivel az őket elindító felhasználó rendelkezik. Ezt lehet megváltoztatni a `suid` és az `sgid` bitekkel. Egy `suid` bittel rendelkező program elindításakor a program a program tulajdonosának jogaival fog futni, fájlokat olvasni, írni és más programokat futtatni.

Az `sgid` bit is egy rövidítés a "Set Group Identification" kifejezés rövidítése. Azt jelenti, hogy a "csoportazonosító megváltoztatása".

Beállítása esetén a program annak a csoportnak a jogaival fog futni, akinek az adott programfájl a birtokában van.

Az sgid bitet könyvtárak esetén is be lehet kapcsolni.

Ennek eredménye a következő lesz: ha ebben a könyvtárban bárki létrehoz egy fájlt (ehhez a többi jognak természetesen rendben kell lennie), akkor a fájl csoporttulajdonosa nem az a csoport lesz, amelyikbe a felhasználó tartozik, hanem az, akinek a könyvtár a birtokában van.

6.2.2. A Sticky bit

A sticky bit bekapcsolása fájlok esetén azt jelzi az operációs rendszernek, hogy a fájlt tartsa a memóriában a végrehajtás után is. Ennek a tulajdonságnak akkor van értelme, ha azt szeretnénk, hogy egy program minél gyorsabban induljon el, ne kelljen várni arra, hogy betöltődjön a memóriába.

A sticky bitet be lehet kapcsolni könyvtárak esetén is.

Egy ilyen bittel ellátott könyvtárban bárki írhat fájlokat (a többi jognak is rendben kell lennie), de mindenki csak a sajátját törölheti. Ezt a lehetőséget pontosan azért tervezték, hogy az olyan, mindenki által írható könyvtárakban, mint például a */tmp*, a felhasználók ne tudják a másik felhasználó által írt fájlokat módosítani, letörölni.

A fájlokhoz tartoznak egyéb információk is a jogosultságokon kívül:

- a fájlra mutató linkek száma
- a fájl tulajdonosának azonosítója
- a fájl csoporttulajdonosának azonosítója
- a fájl mérete
- a fájl utolsó módosításának dátuma
- a fájl utolsó hozzáféréseinek dátuma
- a fájl létrehozásának vagy utolsó státuszmódosításának dátuma
- a fájl típusa (könyvtár, sima fájl, stb.)

6.3. Felhasználók és csoportok kezelése

Mivel több felhasználó is használhatja egyszerre a rendszert, annak el kell tudni döntenie, hogy ki milyen jogosultságokat kaphat. Ezért a felhasználóknak azonosítaniuk kell magukat a rendszer számára. Tehát minden felhasználó egyedi azonosítóval rendelkezik (UID - felhasználói azonosító), és egy vagy több csoportba tartozik (GID - csoportazonosító).

Ezeket az információkat a */etc/passwd* és a */etc/group* fájlok tartalmazzák. Létezik még ezeken kívül a */etc/shadow* fájl is, ami a titkosított jelszavakat tartalmazza és csak a root számára olvasható.

A név azonban nem minden. Egy számla (account) az összes fájlt, erőforrást és információt jelenti, mely egy felhasználóhoz tartozik. Az elnevezés a bankokra utal, és az üzleti rendszerekben valóban pénz kapcsolódik a számlához, amely fogy a rendszer használatával. Pl. a lemezterületnek lehet napi ára megabájtonként, a feldolgozási időnek pedig másodpercenkénti ára lehet.

A Linux kernel maga egyszerű számokként kezeli a felhasználókat. Mindegyik felhasználót egy egyedi egész szám, az ún. felhasználói azonosító (user id vagy uid) azonosít, mivel a számítógép gyorsabban és könnyebben kezeli a számokat, mint a szöveges meghatározásokat. A kernelen kívül létezik egy adatbázis, mely egy szöveges nevet, a felhasználói névet hozzákapcsolja egy felhasználói azonosítóhoz. Az adatbázis ezen kívül más információkat is tartalmaz.

Egy felhasználó létrehozásához a felhasználó személyről információkat kell bevinni az adatbázisba és egy home könyvtárat kell létrehozni számára, ahol saját fájljait tárolhatja. Szükséges lehet még némi alapképzés az új felhasználó számára, és egy megfelelően beállított induló környezet létrehozása is.

A legtöbb Linux terjesztésben található program a számlák létrehozására.

Több ilyen létezik. UHU-Linuxban a szabványos *useradd*; de léteznek grafikus felületű (GUI) programok is, ilyen például az *uhu-control-center*. Bármelyik programot is használjuk, kevés, jóformán semennyi munkát kell végeznünk "puszta kézzel". Még ha a részletek tekervényesek is, ezek a programok csupa magától értetődő dolgot művelnek.

A UNIX rendszerek alapvető felhasználói adatbázisa a */etc/passwd* szöveges fájl, amit jelszófájlnak (password file) nevezünk. Ez felsorolja az összes érvényes felhasználói nevet és a hozzájuk kapcsolt információkat. A fájlban minden felhasználói névhez egy sor tartozik, egy-egy sor 7, kettősponttal elválasztott mezőre oszlik:

1. Felhasználói név (username).
2. Titkosított jelszó.
3. Felhasználói azonosító szám (uid).

4. Csoportazonosító szám (gid).
5. Teljes név, vagy egyéb leírás.
6. Home könyvtár.
7. Bejelentkezési burok (login shell), azaz a bejelentkezéskor futtatandó program.

A formátumot a (man passwd) kézikönyv lap részletesebben elmagyarázza.

A rendszer minden felhasználója olvashatja a jelszófájlt, így pl. megtanulhatják a többi felhasználó nevét. Ez azt is jelenti, hogy még a jelszó (a második mezőben) is mindenki számára elérhető. Igaz, itt csak egy titkosított változata van, azaz elméletileg ez megfelelő megoldás.

Azonban a titkosítás feltörhető, különösen gyenge jelszavak esetén (pl. ha az túl rövid, vagy benne van egy szótárban). Ezért nem jó ötlet, hogy itt vannak a titkosított jelszavak. Sok Linux, így az UHU-Linux is már alapértelmezéskor rendelkezik az árnyék jelszó (shadow password) lehetőségével: a titkosított jelszó ekkor egy külön fájlban, a */etc/shadow*-ban van, melyet csak a root olvashat. Ekkor a */etc/passwd* fájl csak egy speciális jelzést tartalmaz a második mezőben. Minden program, melynek egy felhasználó azonosításával kell törődnie, el kell érje az árnyék jelszófájlt. A szokásos programok elérhetnek a jelszón kívül minden információt az eredeti jelszófájlból, de magát a jelszót nem.

6.3.1. Felhasználói- és csoportazonosító számok választása

A legtöbb rendszeren teljesen mindegy, milyen számok a felhasználói- és csoportazonosítók (uid és gid), de ha a hálózati fájlrendszert (NFS) használjuk, a felhasználóknak azonos uid-jük és gid-jük kell legyen a gépeken.

Ez azért van, mert az NFS is a számszerű azonosítókat használja. Azonban ha nem használunk NFS-t, a konkrét számértékek közömbösek, rábízhatók valamilyen automatikus rendszerre.

Az NFS használata esetén szükséges a számlainformációk automatikus szinkronizálása. Az egyik mód, amivel ez megtehető, a NIS rendszer.

6.3.2. Kezdeti környezet. A */etc/skel* könyvtár

Amikor egy új felhasználó home könyvtára létrejön, a */etc/skel* könyvtár tartalma alapján töltődik az új tartalommal. Ezért ebben a könyvtárba a rendszeradminisztrátor létrehozhat egy szép környezetet az új felhasználók számára. Pl. létrehozhat egy

/etc/skel/.profile fájlt, amely az EDITOR környezeti változót egy népszerű, új felhasználóknak szánt szövegszerkesztőre állítja be.

Általában az a legjobb, ha a */etc/skel* a lehető legkisebb, mivel csaknem lehetetlen a már létrehozott felhasználói fájlok frissítése.

Pl. ha a közkezdvelt editor megváltozik, minden már létező felhasználónak meg kell szerkeszteni a */.profile* fájlját. A rendszeradminisztrátor megkísérelheti ezt automatikusan, shell szkriptből megtenni, de ilyenkor nagyon nagy valószínűséggel sérülhet valakinek a módosított *.profile* fájlja.

Hacsak lehetséges, a globális konfigurációt globális fájlokba érdemes tenni, mint pl. a */etc/profile*. Ezáltal a felhasználói fájlok megsértésének veszélye nélkül változtathatjuk meg a beállításokat.

Felhasználók létrehozása kézzel

Egy új felhasználói számla kézzel is létrehozható, ha követjük az alábbi lépéseket:

1. Szerkesszük meg a */etc/passwd* fájlt a *vipw* programmal, és adjunk egy új sort hozzá, mely az új számlát írja le.
Nagyon vigyázzunk a szintaxisra! Ne szerkesszük egy általános szövegszerkesztővel a jelszófájlt! A *vipw* lefoglalja a fájlt, így más programok nem fogják frissíteni a szerkesztés ideje alatt. A jelszó mezőjébe írjunk "*" -ot, hogy még ne lehessen bejelentkezni a számlára.
2. Hasonlóan szerkesszük meg a */etc/group* fájlt, ha új csoportot is létre kell hoznunk.
3. Hozzuk létre a megfelelő *home* könyvtárat az *mkdir* paranccsal.
4. Másoljuk a */etc/skel* tartalmát a *home* könyvtárba.
5. Korrigáljuk a tulajdonosokat és az engedélyeket a *chown* és *chmod* parancsokkal. A -R opció hasznos lehet.
6. Állítsuk be a jelszót a *passwd* paranccsal.

Az utolsó lépés után a számla működni fog. Ezt valóban csak legutoljára szabad megtenni, különben a felhasználó bejelentkezhet, miközben készítjük a számláját, ami bonyodalmakhoz vezethet.

Néha szükséges olyan buta számlák létrehozása, melyet nem használ egy ember sem. Pl. egy anonymous FTP-szerver létrehozásához egy *ftp* nevű felhasználót létre kell hozni. Ilyen esetekben nem szükséges a jelszót beállítani, sőt, jobb, ha nem tesszük meg, mert így senki sem tudja azt használni, csak a root, mivel ő minden felhasználó nevében dolgozhat.

Felhasználók törlése

Egy felhasználó törléséhez először le kell törölni az összes fájlját, levelesládáját, nyomtatási feladatait, cron és at feladatait, és minden egyéb hivatkozást a felhasználóra.

Ezek után törölhető csak a megfelelő sor a */etc/passwd* fájlból, és a rá vonatkozó bejegyzések a */etc/group*-ból. Jó ötlet először kikapcsolni a számlát, ahogy fentebb láttuk, majd utána elvégezni ezeket, hogy ne használhassa a rendszert a felhasználó, miközben töröljük számláját.

Figyeljünk arra, hogy a felhasználónak lehetnek fájljai a home könyvtárán kívül is. A *find* felhasználható ezek megkeresésére.

Ez a parancs hosszú ideig futhat, ha nagyok a lemezeink. Ha NFS-en keresztül csatolt lemezeink is vannak, akkor vigyázni kell, hogy ne terheljük túl a hálózatot vagy a szerveret.

Néhány Linux terjesztés speciális parancsokat tartalmaz a felhasználók törlése céljából, melyek szokásos neve *deluser* vagy *userdel*. Igaz, ez a munka kézzel is könnyen elvégezhető, és ekkor biztosak lehetünk abban, hogy minden rendben és hiánytalanul zajlott.

Egy felhasználó időleges kikapcsolása

Néha szükséges egy számla ideiglenes kikapcsolása anélkül, hogy törölnénk. Pl. ha a felhasználó nem fizette a használati díjat, vagy a rendszeradminisztrátor észrevette, hogy egy cracker megszerezte a jelszót arra a számlára.

Egy számla ideiglenesen kikapcsolható az alábbi parancssal (LOCK):

```
passwd -l gibzo
```

Egy kikapcsolt számla vissza kapcsolása az alábbi parancssal történhet meg (UNLOCK):

```
passwd -u gibzo
```

Ez a módszer ugyan egyszerű, de a felhasználó nem kap semmiféle visszajelzést.

Emiatt a legjobb mód az időleges kikapcsolásra a bejelentkezési burok egy speciális programra való kicserélése, mely egy üzenetet ír ki, majd azonnal kilép. Ily módon akárki próbál bejelentkezni a számlára, nem fog tudni semmilyen parancsot kiadni, de megtudja ennek okát. Célszerű, ha az üzenet felhívást tartalmaz a rendszeradminisztrátorral való kapcsolatfelvételre a hiba elhárítása érdekében.

Azt is megtehetjük, hogy a felhasználói nevet vagy a jelszót cseréljük másra, de akkor a felhasználó nem tudja, mi történt. A megzavart felhasználó pedig több munkát jelent.

A legegyszerűbb mód ilyen üzenőprogram létrehozására egy "tail szkript" írása:

```
#!/usr/bin/tail +2
```

Ezen számla a biztonság megsértése miatt lett lezárva.

Hívja fel szolgáltatója telefonszámát.

Az első két karakter ('#!') azt mondja a kernelnek, hogy a sor további része egy parancs, mely értelmezni fogja a fájl hátralevő részét. A példabeli tail parancs az első sor kivételével mindent kiír a standard kimenetre.

A tail szkripteket külön könyvtárban tarthatjuk, hogy nevük ne zavarja a normális parancsokat.

A NIS

A *Sun Microsystems* fejlesztette ki a NIS-t a *SunOS* operációs rendszer részeként. Először *Yellow Pages*-nek nevezték el, de mivel Angliában ez védett név volt, ezért meg kellett változtatniuk NIS-re.

Az alkalmazások nevében azonban máig is fellelhető az eredeti név az *yp* előtag révén. A NIS protokoll nyilvánosan elérhető és felhasználható, ezért a UNIX-ok számos változatán létezik.

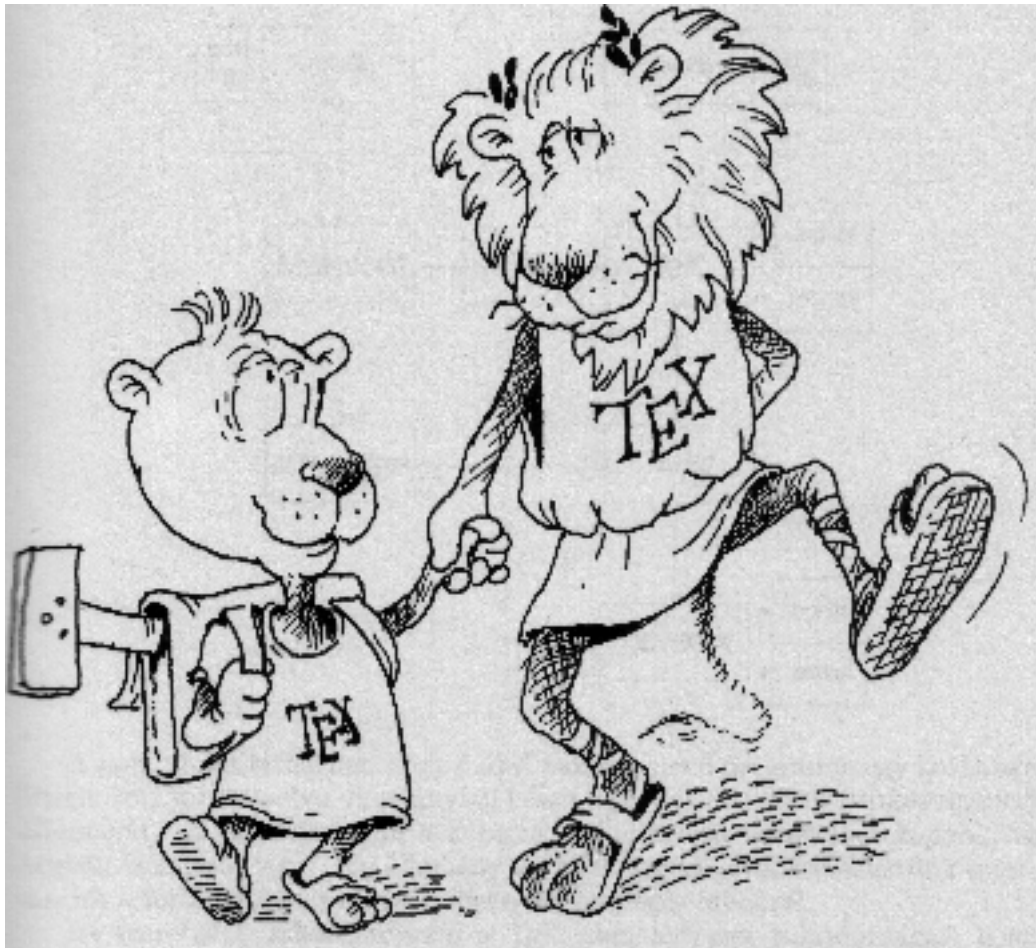
A NIS egy egyszerű, általános kliens-szerver alapú adatbázis rendszer. Legtöbbször jelszó- és csoportfájlok megosztására használják a hálózaton. Segítséget nyújt a hálózat átlátszóvá tételében azzal, hogy egy bejelentkezéssel használhatjuk az egész hálózat számunkra engedélyezett erőforrásait.

A NIS szerver az információkat egyszerű adatbázis-formátumú fájlokban (DBM) tárolja, amelyek lehetővé teszik a gyors keresést. A kliensek RPC hívásokkal tudnak információkat lekérni a szerverről.

A fenti manuális folyamatokat a kép teljessé tétele érdekében adtuk közre. Használtuk UHU-Linux alatt felesleges, hiszen az *uhu-control-center* éppen azért készült, hogy ezeket a feladatokat levegye a rendszergazda válláról.

7. fejezet

Forrásnyelvű programok telepítése



Ez a fejezet a Linux, de általánosabban a UNIX alatti "általános" programok fordítását ismerteti.

Nyelvezetét tekintve nem általános célú felhasználásra van tervezve, hanem legfőképpen hozzáértőknek, hogy lássák, mikor mit kell hogy elmondjanak.

7.1. A fordításról általában

7.1.1. Miért kell(het) fordítani?

A válasz nagyon röviden: sebesség illetve gazdaságosabb működés miatt, vagy mert az adott programhoz nem tudunk csak forráskód formában hozzájutni. Ahogy fejlődnek a processzorok, egyre újabb és újabb utasításokat ismernek, az ezeket kihasználó programokat pedig csak újrafordítással lehet elkészíteni. Példának említsük az egyik legnagyobb vihart kavart utasításkészlet kiegészítést, az MMX-et, aminek segítségével a multimédiás alkalmazások gyorsabbak lettek. Persze fejlődnek a fordítóprogramok is. Vagy lehet, hogy valami hibát fedeznek fel a fordítóprogramban, amitől hibásak lesznek a vele lefordított programok, amit csak újrafordítással lehet kijavítani. Például az AMD K6-os processzorokban volt egy szekvencia hiba, ami Turbo Pascal 7-es fordítóval lefordított programokban jelentkezett.

7.1.2. Mikor kell(het) fordítani?

Egy disztribúcióban kapott előre lefordított (bináris) programok általában i486-os processzorhoz lettek lefordítva, hogy minden x86 (IA32) kompatibilis processzor alatt futtatni lehessen őket. Ez azt jelenti, hogy a processzorunk ismeri az MMX utasításkészletet, akkor azt a programok nem fogják használni, mert az a multimédiás utasításkészlet nem része az eredeti x86-os architektúrának.

7.1.3. Mit kell(het) fordítani?

Természetesen bármit nem érdemes újrafordítani, hiszen maga a fordítási folyamat sok macerával járhat. Amit mindenképp érdemes újrafordítani azok a programok, amiket gyakrabban használunk, vagy amiknél biztosan sebességnövekedést érünk el, ha specifikus, feladatorientált utasításkészletekre optimalizáljuk.

7.2. Egy program lefordulásának lépései

Igazából nem létezik tipikus fordítási menet. Minden program mögött ember(ek) áll(nak), és mivel minden ember más, mindenki másképp dolgozik, minden program máshogy van felépítve, más parancsokkal kell lefordítani. Léteznek IDE felületek (egységesített fejlesztői felületek), ahol csak egy fájlt kell megnyitnunk, és kiválasztani egy menüből a fordítás parancsot, és kész is, de ezek nagyon nem jellemzőek az általánosan használt linuxos, vagy éppenséggel GNU-s programokra.

Azért általánosságban elmondható, hogy parancssoros felületen egy C-ben írt programot az alábbi lépésekkel lehet lefordítani:

* előkészületek:

```
autoconf  
./configure
```

* fordítás

```
make all
```

* telepítés

```
make install
```

Ezek az utasítások egy csomó GNU-s fordítást segítő program meglétét feltételezik. És persze nemcsak nekünk kell ezeket feltelepíteni, hanem a forráskódban meglévő programot is úgy kellett kifejleszteni, hogy a fordítás ezek segítségével történjen.

7.3. Hol találhatunk forrásban programokat?

Vannak nyílt forráskódú programok közzétételét elősegítő kiszolgálók, ezek közül a két legismertebb:

* <http://www.sourceforge.net/>
* <http://savanna.gnu.org/>

7.4. Előkészületek

7.4.1. Letöltés

Egy programot általában *.tar.gz* (*.tgz*) vagy *tar.bz2* formában találunk meg. Ezt a fájlt töltsük le, és mentjük el valamelyik könyvtárunkba, javasolt a */usr/local/src/-n* belül létrehozni nekik egy tárolókönyvtárat.

Kicsomagolás

Kicsomagolni a *.tar.** összecsomagolt állományokat a *tar* programmal tudjuk: *tar.gz* (.tgz) esetén a

```
tar -xvzf ./forráskód.tar.gz
```

paranccsal,
.tar.bz2 esetén a

```
tar -xvjf ./forráskód.tar.bz2
```

paranccsal. Régebbi Linux terjesztéseken, a *-xvjf* helyett még a *-xvlf* paramétereket kellett hogy megadjuk.

A *tar* parancsot a */usr/local/src/* könyvtárból indítsuk el, és akkor általában a */usr/local/src/* könyvtárban létrejön egy forráskód-verzió nevű könyvtár, ami a forráskódot és mellette pár információs fájlt tartalmaz.

Függőségek ellenőrzése

Ezek után olvassuk el a *README* majd pedig az *INSTALL* fájlokat nagyon alaposan! Ezek tartalmazzák a szükséges információkat a fordításhoz.

Az alábbiakra figyeljünk oda:

1. milyen osztott függvénykönyvtárakra (shared library) van szükségünk a sikeres fordításhoz
2. milyen lépésekkel kell lefordítanunk a programot?
3. ha kell a *./configure* programot indítani, milyen paraméterei vannak?
4. lesz-e elég szabad helyünk a fordításhoz?

Ha mindezekkel tisztába kerültünk, akkor a megadott lépéseknek megfelelően akár el is kezdhethük a fordítást.

Javítások (foltok) keresése

A fordítás előtt azonban érdemes hibajavításokat, vagy funkcióbővívéseket keresni. Ezeket úgynevezett foltok (patch) formájában tudjuk letölteni a netről. Vigyázzunk arra, hogy olyan foltot használjunk csak, ami a mi általunk birtokolt forráskódú program verziószámunknak megfelelő, és akkor is csak egyet tegyünk fel egyszerre. Ez alól persze kivételt lehet tenni, ha olyan foltokat telepítünk, ami direkt egy másik folt meglétét igénylik.

A foltokat a *patch* nevű programmal tudjuk felrakni. Lépünk be abba a könyvtárba, ahol az eredeti forráskód van, majd a

```
patch -p0 -i foltfájl
```

paranccsal rakhatjuk fel.

7.4.2. Fordítás

Van egy általános modell, ami GNU programok meglétét feltételezi. Ezek a programok:

autoconf ezt a forráskód testesztelésére használhatjuk. Segítségével lehet beállítani pl., hogy melyik igényelt osztott könyvtárat merre találunk a rendszerben, vagy a programban szolgáltatásokat lehet engedélyezni/tiltani, vagy processzor utasításkészleteket tudunk engedélyezni, illetve a telepítéskor a célkönyvtárat tudjuk előre meghatározni. Az *autoconf* létrehoz egy *configure* nevű programot a forráskód gyökérkönyvtárában, aminek segítségével tudjuk a fordítás előtt a szükséges paramétereket beállítani, és ez a program már nem igényli az *autoconf* meglétét a fordításra használt számítógépen.

make ez a program már a fordításban és a telepítésben nyújt segítséget. Segítségével egy központi fájlban (Makefile) lehet beállítani, hogy a fordítás milyen lépésekből álljon, pl. előbb kell a *.lib* könyvtárban lévő *main.c*-t lefordítani, és utána a *.util* könyvtárban lévő *runme.c*-t, mert ez utóbbi igényli az előbbi meglétét. A *Makefile* fájlt általában a *configure* program hozza létre, teljesen a rendszerünkre és a megadott paraméterekre szabva.

Az autoconf és a ./configure

Ha nem létezik *./configure* fájl, akkor az *autoconf* parancs kiadásával létre tudjuk hozni. Akkor is célszerű kiadni ezt a parancsot, ha úgy gondoljuk, hogy a csomagban található *configure* program régi verziójú, mert ekkor az *autoconf* a legújabb verziójú *configure*-ra fogja azt frissíteni. Az *autoconf* program a *configure.in* fájl meglétét igényli.

A *configure* parancs paramétereit a “./configure –help” paranccsal listázhatjuk ki.

A *configure* parancs mindig elérhető paramétereit:

```
--version : kiírja a verziószámot

--prefix=dir : a megadott könyvtárba fogja telepíteni az
               architektúra független fájlokat
               alapértelmezett: dir = /usr/local/

--exec-prefix=dir : a megadott könyvtárba fogja telepíteni az
                   architektúra függő fájlokat
                   alapértelmezett: dir = PREFIX

--bindir : felhasználó által futtatható fájlok elérési helye
           alapértelmezett: dir = EPREFIX/bin

--sbindir : rendszergazda által futtatható fájlok elérési helye
           alapértelmezett: dir = EPREFIX/sbin

--libexecdir : program által futtatható fájlok elérési helye
              alapértelmezett: dir = EPREFIX/libexec

--datadir : csak olvasható architektúra specifikus adatok elérési helye
           alapértelmezett: dir = PREFIX/share

--sysconfigdir : csak olvasható gépfüggő adatok elérési helye
               alapértelmezett: dir = PREFIX/etc

--libdir : objektum kódok elérési helye
          alapértelmezett: dir = EPREFIX/lib

--includedir : C fejléc fájlok elérési helye
              alapértelmezett: dir = PREFIX/include

--mandir : kézikönyv fájlok gyökér elérési helye
          alapértelmezett: dir = PREFIX/man

--infodir : infó fájlok elérési helye
           alapértelmezett: dir = PREFIX/info
```

Illetve a program specifikus paramétereket a

```
--enable-FEATURE vagy a --disable-FEATURE paranccsal  
tudjuk a szolgáltatásokat engedélyezni/tiltani  
  
--with-PACKAGE vagy a --without-PACKAGE paranccsal tudjuk  
a programrészeket engedélyezni/tiltani
```

A make és Makefile

A *make* parancs nagyon egyszerűen működik a mi oldalunkról, paraméternek csak egy szócskát kell megadni, ami a végrehajtandó művelet parancsszava. A műveletnek megfelelő utasításokat a *make* végrehajtja.

Pl. fordításhoz írjuk csak szimplán be a *make* vagy a *make all* parancsot, telepítéshez pedig a *make install* parancsot.

Ha nem adunk meg paramétert, mindig van egy alapértelmezett utasítás, ez fog lefutni. Ez általában a fordítás szokott lenni.

7.4.3. Egy tipikus fordítás forrásból

Példát majd legközelebb

7.4.4. Automatikus fordítást segítő rendszerek

Látható, hogy minden egyes program fordítása egyedi, ezért egy olyan komplex fordítási folyamat, mint mondjuk egy komplett linux terjesztés, vagy hogy egy kicsit kisebbet is mondjunk, pl. a *KDE* ablakkezelő lefordítása is egy egész fejlesztő csapatot megmozgathat.

A Linux terjesztést fejlesztő cégek ezért kidolgoztak egy eljárást, aminek segítségével a fordítást egy soros parancsra lehet leegyszerűsíteni.

Az elmélet lényege az, hogy egy információs fájlban eltároljuk, hogy milyen utasításokat kell kiadni milyen sorrendben egy sikeres fordításhoz, illetve milyen programok, osztott könyvtárak szükségesek a fordítás folyamán, milyen foltokat kell feltenni, és így tovább.

A program forráskódját, a szükséges foltokat, és ezt az információs fájlt aztán beteszik egy összecsomagolt fájlba, és nekünk csak annyi a dolgunk, hogy elindítjuk a megfelelő programot, ami aztán azt információs fájl alapján el tudja végezni a teljes fordítás menetét.

7.4.5. RPM

Az RPM csomagkezelőt a RedHat fejlesztette ki. Általában a forráskódot tartalmazó fájlt *.srpm* kiterjesztéssel szokták ellátni, de mivel ez is csak egy sima *RPM* fájl, ezért egyes terjesztésekben a bináris tartalmazó RPM csomaghoz hasonlóan *.rpm* a kiterjesztése.

Az *rpm* program használata:

```
rpm {-bÁLLAPOT|-tÁLLAPOT} [egyéb-opciók] FÁJL ...
```

A *-b* paramétert kell használnunk, ha az infófájlt adjuk meg paraméterül, és a *-t* paramétert, ha egy összecsomagolt állományt, és az *RPM*-nek saját magának kell belőle kinyernie az infófájlt. Az ÁLLAPOT egy egy karakter hosszúságú parancs lehet, és a fordítási folyamatot tudjuk vele szabályozni:

```
-ba: elkészíti a bináris .rpm és a forrás .srpm fájlt is.  
-bb: csak a bináris csomagot készíti el.  
-bp: kicsomagolja a fájlokat, és felteszi a foltokat  
-bc: lefordítja a már kicsomagolt csomagot  
-bi: feltelepíti a rendszerünkbe a lefordított csomagot  
-bl: leellenőrzi, hogy minden fájl lefordult-e  
-bs: csak a forrás csomagfájlt készíti el
```

Egyéb paraméterek:

```
--clean: a fordítás befejeztével letörli az ideiglenes tárolókönyvtárakat  
--nobuild: nem fog tényleges fordítást elvégezni, csak teszteléshez haszn
```

Ha csak fel szeretnénk telepíteni egy forrás RPM fájlban található programot, akkor adjuk ki az

```
rpm --recompile FORRÁSRPM` parancsot is.
```

7.4.6. A dpkg

Ez most nem készült el idő hiányában.

Mivel az anyag úgyis Debian specifikus, és az LSB csak az RPM-et követeli meg, nem is baj.

7.5. Fordítás másik architektúrára

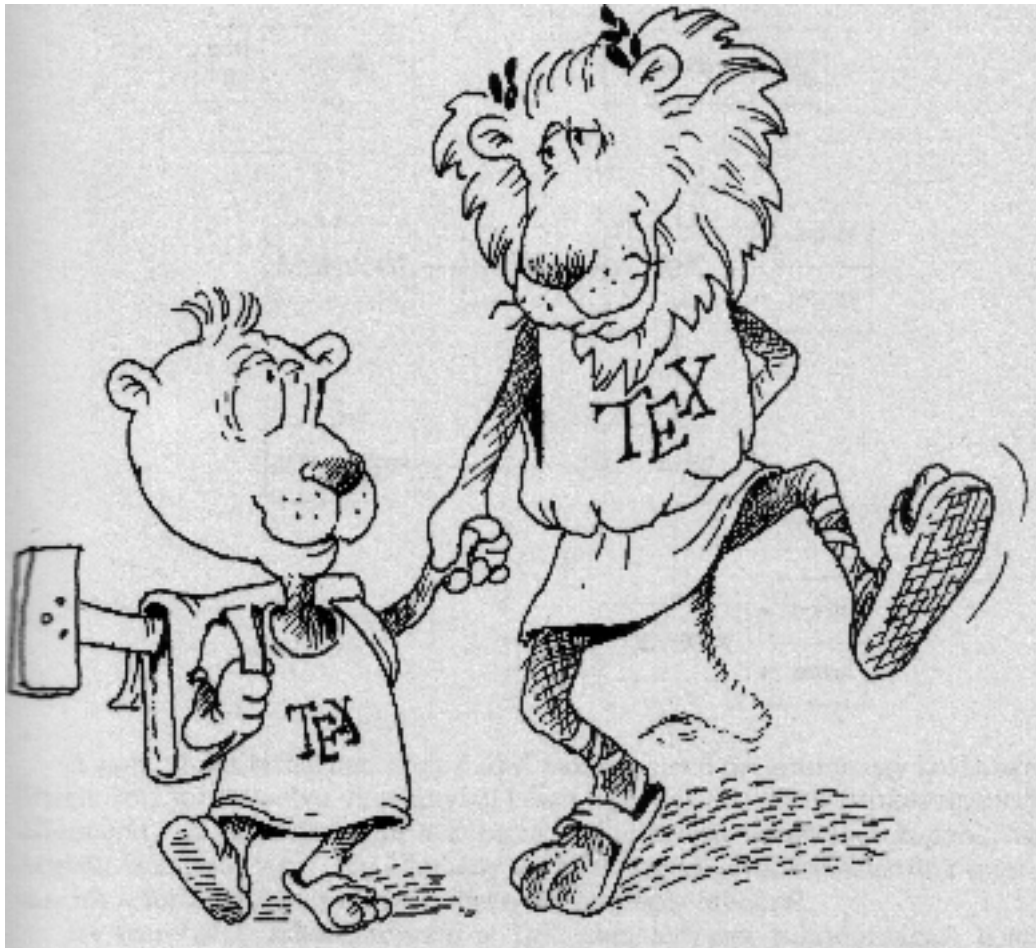
Ez most nem készült el idő hiányában.

7.6. Felhasznált irodalom

- Building and Installing Software Packages for Linux by Mendel Cooper – <http://person.riverusers.com/thearendel/>
- RPM kézikönyv

8. fejezet

Hálózati alapismeretek



8.1. Alapfogalmak

A Linux különböző szerepkörben a hálózaton belül, mint munkaállomás, mint kiszolgáló.

8.1.1. Mit értünk "Hálózat" alatt?

Hálózatnak nevezzük azt a kapcsolódási formát két vagy több gép között, amely lehetővé teszi a közvetlen kommunikációt a gépek között. Tehát ha csak két gépet kötünk össze egymással és azok kommunikálnak egymással, akkor már hálózatról beszélhetünk.

Méretét tekintve megkülönböztetünk 3(4) fajtát:

- LAN (Local Area Network - Helyi hálózat),
- MAN (Metropolitan Area Network - Városi Hálózat),
- WAN (Wide Area Network - Nagy kiterjedésű Hálózat),
- GAN (Global Area Network)

8.1.2. Hogyan kommunikálnak egymással a gépek a hálózatban?

A számítógépek egymás között különféle protokollokkal kommunikálnak. Az internetes adattovábbítás a TCP/IP -n alapszik. A TCP/IP jelentése: Transmission Control Protocol/Internet Protocol vagyis Átvitelt Vezérlő Protokoll/Internet Protokoll.

- TCP/IP

Tekintsük a nevet egy összefoglaló névnek, vagyis a TCP/IP protokollkészletnek. Protokollok halmaza, melyek "egy nyelvet" beszélnek. A TCP/IP protokollal küldött vagy fogadott adatok "adatcsomagok", amelyek az IP alapú protokoll alapvető adat egységei.

A gépek közötti kommunikáció során ezek a csomagok alkalmazási protokollokhoz kerülnek, amely többnyire TCP vagy UDP (User Datagram Protocol).

- TCP (Kapcsolat alapú protokoll)

Lényege, hogy egy alkalmazáshoz kapcsolódik és addig él a kapcsolat, ameddig meg nem szakítjuk. A TCP protokollra jellemző, hogy a nagyméretű csomagokat számozott IP csomagokká botja és úgy továbbítja különféle útvonalon. A csomagok azért számozottak, mert a csomagok továbbítása törtenhet más úton is, így az eredeti adatot később a célállomás összeállítja.

- UDP (Kapcsolatmentes protokoll) /User Datagram Protocol/

Az UDP az adatokat adatcsomag (datagram) formájában továbbítja. Ezek a csomagok nincsenek megszámozva és nem garantált a csomagok sorrendbeli érkezése. Az adatok épségének a vizsgálata az alkalmazás dolga.

A legtöbb alkalmazási protokoll mind a TCP, mind az UDP protokoll-t is használja. Például: SMTP, POP, FTP.

- ICMP (Internet Control Message Protocol)
- IGMP (Internet Group Multicast Protocol)
- Protokollok (szolgáltatások), portok

A számítástechnika és a hálózatok fejlődésével egyre több protokoll jelent meg. Ma már több száz protokollal rendelkezünk, amely hol bonyolultabb az előzőnél, hogy egyszerűbb.

Linux rendszerünk legfontosabb folyamata az init, amely minden más alkalmazás előtt indul el és indít el minden más folyamatot. A hálózat számára fontos szolgáltatás, amit az init elindít az inetd. Ennek a szolgáltatásnak a feladata, hogy a hálózati kéréseket figyelje. Ezek a kérések történhetnek socket és port alapján. Az inetd tehát azt figyeli, hogy mikor melyik daemon-t kell elindítani egy kapura érkező kérés teljesítéséhez.

A */etc/services* file-ban találhatóak bizonyos szolgáltatások nevei, kapui és az általuk használt protokoll(ok) nevei.Pl:

ftp	21/tcp		
fsp	21/udp	fspd	
ssh	22/tcp		# SSH Remote Login Protocol
ssh	22/udp		# SSH Remote Login Protocol
telnet	23/tcp		
smtp	25/tcp	mail	
time	37/tcp	timserver	
time	37/udp	timserver	
rlp	39/udp	resource	# resource location
nameserver	42/tcp	name	# IEN 116
whois	43/tcp	nickname	

Természetesen csak az alapértelmezett kapukat tartalmazza a */etc/services* file. Először a szolgáltatás neve, majd a kapu száma következik és a szolgáltatás által használt protokoll. Ami ezek az adatok után következik, az általában megjegyzés.

8.1.3. Mik azok az IP címek és hogyan épülnek fel?

Ipv4 (Internet Protocol Version 4)

Felépítésük: négy ponttal elválasztott 8 bites részből, amely 32 bites címet jelent, a 0.0.0.0 - 255.255.255.255-ig terjedő számskálán. Ez közel 4.3 millió IP címet jelent. A jövőre nézve, azonban ez a 4,3 millió IP-cím kevés lesz, ezért 1993 óta elkezdtek fejleszteni egy új protokollt, amely majd felváltja a jelenlegi ipv4-et. Ez az ipv6, amely az eddigi 32 bites cím helyett 128 bites címezést tesz lehetővé sok egyéb biztonsági javítással együtt.

Az IP címek kiosztása. Az IP címek kiosztását szigorú szabályok határozzák meg. Ezeket a szabályokat az úgynevezett RFC-kben találhatjuk meg. A belső hálózat IP címeinek kiosztása a mi feladatunk. Erre léteznek külön fenntartott IP tartományok. 3 ilyen osztályt különböztetünk meg.

A: 10.0.0.0 - 10.255.255.255
B: 172.16.0.0 - 172.32.255.255
C: 192.168.0.0 - 192.168.255.255

Illetve léteznek még speciálisan fenntartott IP tartományok. Ezek a következők:

D: 224.0.0.0 - 239.255.255.255 (multicast)
E: 240.0.0.0 - 255.255.255.255 (broadcast)

Mint láthatjuk a fenti címfelosztásból kimaradt a 127.0.0.0/255.0.0.0 címtartomány. Ez az ún loopback-nek fenntartott rész. A loopback egy olyan álesztköz, ami a saját számítógépünket jelenti. Bármelyik cím a 127.0.0.0 tartományon belül a saját számítógépünkkel kommunikál.

RFC. Az RFC-ről érdemes egy kicsit beszélni, legalább annyit, hogy tudjuk, hogy mi is az.

Az RCF rövidítése Request For Comments vagyis Felhívás Hosszászólásra. Az első RFC 1969-ben látott napvilágot és a mai napig is akár hetente - naponta jelenhetnek meg új RFC-k, vagy átdolgozottak. A céljuk, hogy meghatározzák az eredetileg ARPA később internet és a számítógépes adattovábbítás módozatait, általában a hálózatokat, protokollokat stb. Később, mikor megalakult az IETF (Internet Engineering Task Force) az RFC-eket az Internet szabványává alakították, így az RFC-k jelenleg az Internet tervrajzai. Ha valaki érdekelnek, akkor a www.rfc.org-on mazsolázhat belőlük.

8.1.4. Hálózati maszkok

Az A, B, C osztályú hálózatok mindegyikének van egy úgynevezett alapértelmezett hálózati maszkja.

A: 255.0.0.0

B: 255.255.0.0

C: 255.255.255.0

Hogy miért is? A magyarázat nagyon egyszerű: A hálózati maszkoknak le kell fedniük azt a részt, amely nem tartozik a hálózathoz, vagyis a *192.168.0.1* címhez tartozó hálózati maszk így néz ki fedésben egymással:

11000000.10101000.00000000.00000001 -> 192.168.0.1

11111111.11111111.11111111.00000000 -> 255.255.255.0

A 255 a kettes számrendszerben elfedi a 192-t, így ez más hálózatból láthatatlanná válik és így tovább. Röviden a hálózati maszk meghatározza, hogy az IP-cím melyik része jelenti a hálózatot.

A .0 az IP-cím végén a hálózatot jelöli, míg a .255 a Broadcast-ot.

8.1.5. Üzenetszórásos adatküldési cím, vagyis Broadcast

Ha adatot küldünk az üzenetszórásos címre, az olyan, mintha a hálózaton lévő összes gépre elküldenénk egyszerre. Például:

```
ping -b 172.16.0.255
```

Egy adott hálózatot, az IP-cím, a hálózati maszk és a üzenetszórásos adattobábitási cím határozza meg. Egy hálózaton akkor vannak a gépek, amikor az egyik hálózaton lévő gép elérheti útválasztó nélkül a másik gépet. Az útválasztó pedig egy olyan eszköz, amely hálózatok között mozgat adatokat, így lehetőséget biztosít a nem egy hálózaton lévő gépeknek a kommunikációjához.

8.1.6. Tartományok, tartománynevek (domain)

A tartománynevek a következő elgondolás szerint épülnek fel.

számítógépnev.egytartomány.elso_szintu_tartomany

Azért öltenek ilyen formát, hogy ne kelljen megjegyezni a 32 bites vagy 128 bites címekből álló hosszú számsorozatokot, csak értelmesnek tűnő neveket. Ezeket a "neveket" a névkiszolgálók fordítják le újra IP címekre.

Elsőszintű amerikai tartományok:

```
.org  
.com  
.edu  
.mil  
.gov  
.net
```

Új elsőszintű tartományok

```
.aero  
.biz  
.coop  
.info  
.museum  
.name  
.pro
```

A kétbetűs, országokat jelölő felsőszintű tartományokat az IANA (Internet Assigned Numbers Authority) felügyeli és szabályozza. <http://www.iana.org/>

8.1.7. A hálózat beállítása

Ahhoz, hogy legyen hálózatunk (LAN), szükségünk van minimum 1 hálókártyára a saját gépünkben. Fontos, hogy olyan hálókártyát vegyünk, amit a kernelünk támogat (Habár itt jegyezném meg, hogy lassan nincs olyan hálókártya, amit ne támogatna a Linux!).

A kernel

Megoszlanak a vélemények, hogy egy hálókártyát hogyan is helyezzünk el a kernelben. Forgassuk bele, vagy csupán modulként használjuk. Mindkettő helyes döntés lehet, azonban én azt ajánlom, hogy mint modul használjuk! A döntés okát a kernel alaposabb tanulmányozása meg is fogjuk érteni.

Interfacek, parancsok

Linux rendszerünkön az interfacek különböző neveken ismertek:

```
eth0, eth1, eth2..  
lo  
sit0  
ppp0  
tunl0  
stb...
```

Számunkra `_most_` jelenleg egyedül az `eth0` a fontos. Ez az elsődleges hálózati interface.
(Nomeg a `lo` = loopback :)

Tehát egy interface adatainak lekérdezésére szolgál az `/sbin/ifconfig` parancs. Van neki egy kapcsolója az `ifconfig -a`, amely a gépen található összes csatolóról megjelenít minden megjeleníthető információt. Adjuk ki hát a parancsot:

```
eth0      Link encap:Ethernet  HWaddr 00:04:E2:33:91:D0  
          inet addr:192.168.0.2  Bcast:192.168.0.255  Mask:255.255.255.0  
          inet6 addr: fe80::204:e2ff:fe33:91d0/10 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:1919036 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1862386 errors:4 dropped:0 overruns:0 carrier:8  
          collisions:0 txqueuelen:100  
          RX bytes:2480908330 (2.3 GiB)  TX bytes:147416873 (140.5 MiB)  
          Interrupt:5 Base address:0xd400  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:16436  Metric:1  
          RX packets:1881 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:1881 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:458517 (447.7 KiB)  TX bytes:458517 (447.7 KiB)
```

Valami ilyet fogunk látni, feltéve, hogy előzőleg beállítottunk valamit, ha nem, akkor sem kell elkeseredni. Mindjárt megtanuljuk, hogyan kell beállítani a címeket, és hogy melyik sor mit is jelent számunkra.

```
# Az alábbi sor jelenti az adott hálókártya MAC
# (Media Access Control ) címét. Ez teljesen egyedi!

HWaddr 00:04:E2:33:91:D0

# A hálózati IP-cím, Broadcast és a Netmaszk.

inet addr:192.168.0.2 Bcast:192.168.0.255 Mask:255.255.255.0
```

A harmadik sor arról árulkodik, hogy éppen fel van húzva vagy sem (UP)

Egy interface leállítása. *ifconfig eth0 down*, vagy ha az összes felhúzott kártyát le akarjuk állítani, akkor *ifdown -a*

Feladat: állítsuk be eth0-t a következőképpen:

IP: 192.168.0.1, netmask: 255.255.255.0

Megoldás:

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

Ezeket automatikusan is be lehet állítani, hogy bootnál egyből felhúzza az interface-t. Debian rendszerünkön, ezek a beállítófájlok a */etc/network/interfaces* állományban található meg. Ez a file alapértelmezésben valahogy így néz ki:

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback
```

Írjuk hozzá, hogy az eth0 indításkor felhúzza a 192.168.0.1 címet a 255.255.255.0 netmaskot és a 192.168.0.0 networkot 192.168.0.255 -os Broadcast-al.

```
auto eth0
iface eth0 inet static
address 192.168.0.1
network 192.168.0.0
broadcast 192.168.0.255
netmask 255.255.255.0
```

Ha út közben leállítottuk a kártyát és most az átírt fájl szerint akarjuk felhúzni a hálókártyát, akkor csak ennyit kell beírunk:

```
ifup eth0
```

ez csak az eth0-t húzza fel, vagy

```
ifup -a
```

ami az összes */etc/network/interfaces* fájlban lévő interface-t felhúzza.

A resolv.conf fájl. A */etc/resolv.conf* az az állomány, ahova a nameserverek IP-címeit helyezzük el. Beállíthatjuk, hogy csatlakozáskor automatiusan beálítsa nekünk azokat (pl. dhcp, bootp), vagy ezt megtehetjük kézzel mi is, feltéve, ha ismerünk fejből nameservereket. Az állomány maga így néz ki:

```
nameserver 195.38.96.20
nameserver 195.38.97.1
```

A hosts fájl. A hosts állomány a */etc/hosts*-t jelenti. Az ebben szereplő hostok azonnal névfeloldásra kerülnek. Minden domaint először innen kérdez le a gépünk, utána kérdezi csak a nameservereket. Ha egy név itt egyezik, akkor az fordítja vissza IP-címre. Például a hosts állományban ez szerepel:

```
127.0.0.1      base      localhost
192.168.0.2    base
195.38.104.98  duffy
80.98.22.136   gyuri
80.202.52.30   norveg1
62.201.118.58 ivi
213.222.168.65 waq
195.228.170.137 lnx
192.168.0.1    rei.org
192.168.0.3    petiserk
```

Ha tehát a webböngészőnkbe beírjuk, hogy rei.org, akkor a 192.168.0.1-es IP-címen fogja keresni az adott címet, attól függetlenül, hogy ez a cím valóban él az interneten.

8.1.8. Route, útválasztás

Hogyan csomagok milyen úton távoznak és érkeznek az ethernetkártyánkhoz, azt a route parancs begépelésével meg is tudhatjuk:

```
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use
Iface
192.168.0.0      *                255.255.255.0    U        0      0      0 eth0
default          192.168.0.1     0.0.0.0          UG       0      0      0 eth0
```

Ez megmutatja, hogy a 192.168.0.0 hálózaton belülről, vagy belülré érkező csomagok az eth0-án keresztül kommunikálnak.

Azonban, amikor az adatcsomagok címe ezen a hálózaton kívül található, akkor létre kell hoznunk egy default route alapértelmezett útvonalat. Például beállítom, hogy a gépem a hálózat egy másik gépét keresse meg, az összes, helyi hálózaton kívülre irányuló forgalommal:

```
route add default gw 192.168.0.1
```

8.1.9. Programok, amelyek segíthetik tájékozódásunkat a hálózat útvesztőjében.

Ping

A *ping* tulajdonképpen egy adatcsomagot küld egy célgépnek. A célgép válaszol egy másik csomaggal (icmp reply), így a küldés és fogadás közötti időpontot mérve megállapítja, hogy az adott hálózaton milyen gyors az adatáramlás.

man ping

Netstat

A *netstat* használata során betekintést nyerhetünk a gépünk éppen aktuális kapcsolataiba és nyitott portjairól kapunk általános információkat. A *netstat -p -a* kapcsolóval arról is értesítést kapunk, hogy milyen programok használják a nyitott portokat.

man netstat

Traceroute

A traceroute program monitorozza nekünk, hogy egy adott csomag egy adott másik géphez milyen úton, milyen útválasztók között jut el.

Példa:

```
traceroute to web.caesar.elte.hu (157.181.151.150), 30 hops max, 38 byte packets
 1 col-gw-bix.integrity.hu (195.56.44.65)          0.234 ms  0.159 ms  0.132 ms
 2 bix.vha.iif.hu (193.188.137.13)                 0.289 ms  0.228 ms  0.293 ms
 3 orion.hbone-2.elte.hu (157.181.141.10)          0.581 ms  0.733 ms  0.693 ms
 4 vega.gigabit-backbone.elte.hu (157.181.0.19)    0.952 ms  0.899 ms  1.420 ms
 5 draco.atm-backbone.elte.hu (157.181.120.140)    4.382 ms  2.625 ms  3.927 ms
 6 web05.caesar.elte.hu (157.181.151.150)         3.350 ms  1.733 ms  1.814 ms
```

whois

A whois parancs lekérdezi egy adott domainnek az információit.

Példa:

```
wooh@base:~$ whois lme.hu
% Whois server 1.99A
```

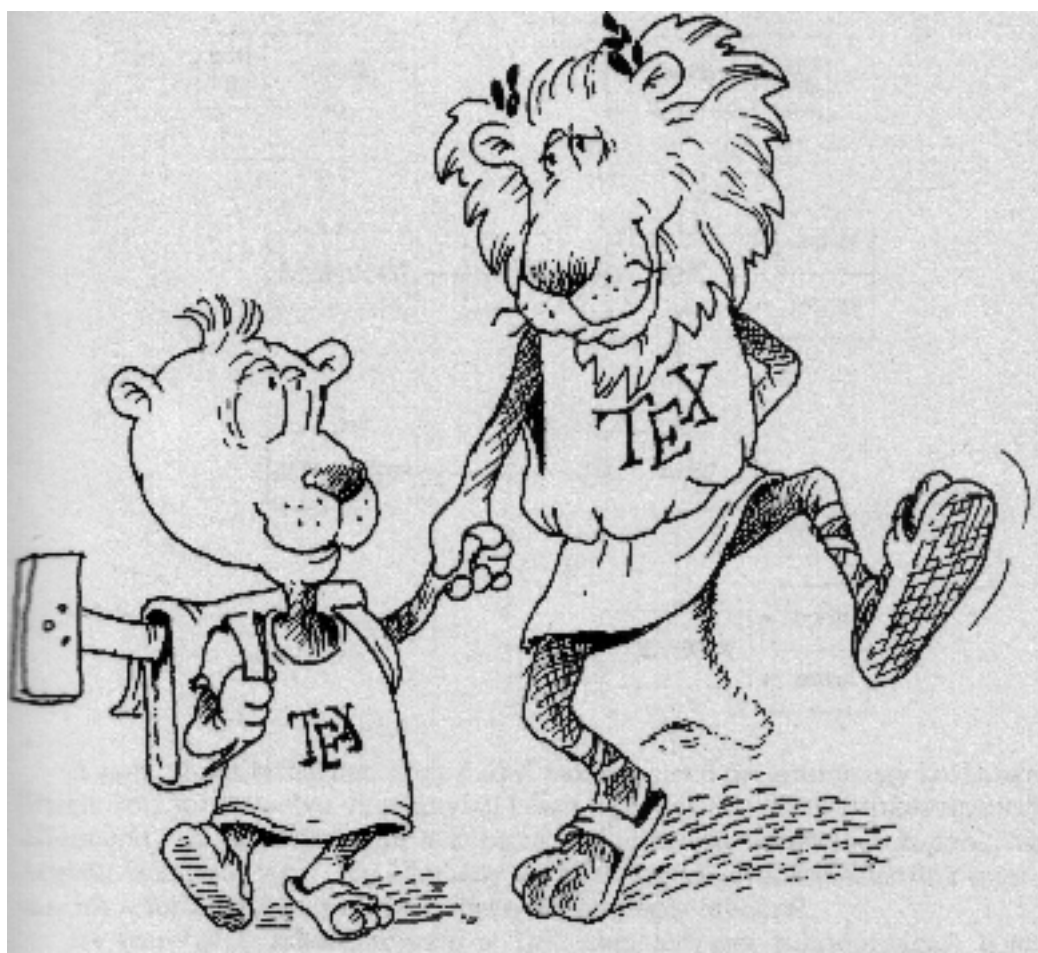
```
Rights restricted by copyright. Szerzői jog fenntartva.
-Legal usage of this service requires that you agree to
abide by the rules and conditions set forth at
http://www.domain.hu/domain/English/domainsearch/feltetelek.html
-A szolgáltatás csak a
http://www.domain.hu/domain/domainsearch/feltetelek.html címen
elérhető feltételek elfogadása és betartása mellett
használható legálisan.
```

```
domain:      lme.hu
org:         org_name_eng: Association of Hungarian Linux Users
org:         org_name_hun: Linux-Felhasználók Magyarországi Egyesülete
address:     Párkány u. 34. X/62.
address:     H-1138 Budapest
address:     HU
phone:       +36 20 9503055
fax-no:
hun-id:      0000256961
admin-c:     2000215479
tech-c:      2000235518
```

```
zone-c:          2000215479
domain_pri_ns:   ns.linux.hu[212.40.96.72]
registered:      2000.06.24 03:01:26
changed:         2003.02.26 15:52:34
registrar:       1990520006
```


9. fejezet

Hálózati szolgáltatások



9.1. Az internet DNS

Elv és konfiguráció

Pásztor Miklós
MTA SZTAKI/ASZI
pasztor@sztaki.hu

Lektorálta: Kiss Gábor, Martos Balázs

Az interneten használt osztott név adatbázis, a DNS (Domain Name Service) folyton használatos: minden web lap letöltésnél, levél közvetítésnél szerepe van, nélküle megbénulna a hálózat, mégis sokan még a létezéséről sem vesznek tudomást, a szolgáltatás csendesen dolgozik a háttérben.

A DNS egy *osztott, hierarchikus adatbázis*: az adatbázist jelenleg név szerverek száz-ezrei szolgáltatják nevek millióiról. A tervezéskor gondoltak redundanciára és a hibatűrésre: a névszerverek sokszor nem elérhetők, konfigurációjuk tele van hibával, hiánysággal, elavult adatokkal, az egész mégis bámulatos módon működik. A DNS rendszer legfontosabb feladata a név – IP cím feloldás, de – ahogy azt látni fogjuk – egy sor más információt is szolgáltat a domain nevekről. A rendszergazdák fontos feladata a DNS konfigurálás. Ebben a írásban a DNS rendszer – nem is bonyolult – elvét ismertetjük, és leírjuk a konfigurálás legfontosabb elemeit.

9.1.1. IP címek, nevek

Az interneten levő hálózati eszközök, számítógépek mindegyikének egyedi azonosítója, (4 byte-on tárolt) IP címe van¹. A felhasználók azonban olyan neveket szeretnek használni, amelyek könnyebben megjegyezhetők, mint egy ilyen hosszú szám, és a névből következtetni tudnak a gép, a szolgáltatás helyére, a szolgáltatás típusára is. Ezért kezdettől fogva neveket rendeltek az IP címekhez. Amikor az internet még csak pár ezer számítógépből állt, ezt a név-cím hozzárendelést egy folyamatosan növekvő fájl, *host táblázat* tartalmazta. Ezt a táblázatot minden számítógépen lokálisan tárolták és egy központi helyről rendszeresen frissítették. Ennek nyoma mind a mai napig megvan: pl. a unix rendszerekben az `/etc/hosts` fájl éppen ilyen.

Az internet növekedtével azonban ez a megoldás tarthatatlanná vált: a fájl hatalmasra dagadt, egyre sűrűbben kellett módosítani, egyre többen töltötték le, egyre gyakrabban. Ezért jött létre a DNS (Domain Name Service), az internetes kommunikáció egyik fundamentuma. Kidolgozásában fő szerepet játszott P. Mockapetris, az ISI (Information

¹IPv4, de terjed az IPv6 is ami egy külön történetnek tekinthető a működését tekintve

Science Institute) munkatársa. A DNS elve egyszerű és ötletes, frappáns bizonyítéka annak, hogy a szubszidiaritás elve milyen jól működik a gyakorlatban.

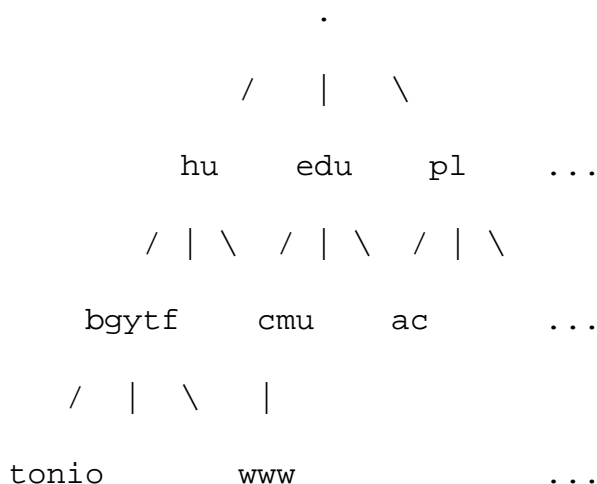
A nevek feloldása hálózati kommunikáció által történik. A névszerverek feladata kettős:

látni azaz az elosztott DNS adatbázist kérdezni, a hálózati szolgáltatások számára az érvényben levő név-cím hozzárendelésről információt adni és

láttatni, mutatni az elosztott adatbázis ide kiosztott részére információforrásként viselkedni, azaz a nevek egy bizonyos halmazáról a többi név szerver számára – mint illetékes – adatokat szolgáltatni.

Ha egy név dolgában egy szerver az internet számára elsődleges információforrás, azaz illetékes, azt úgy szokás kifejezni, hogy az ő adata *autoritatív*.

Mindenki ismer internet neveket: `mail.whitehouse.com`, `reklam.radio.hu`. Az internet nevek fordított fa szerint szerveződő hierarchiát alkotnak:



A fa *fordított*, mert a gyökér a hierarchia legmagasabb foka. A nevek feloldása a gyökértől kezdődik, és fokról fokra halad előre. A név-fa különböző elágazási pontjaiért és ágaiért különböző szerverek felelősek. Egy-egy szerver több ágért is felelős lehet. A név-fa egy egy pontját *domain*-nak, *domain névnek* vagy egyszerűen *név*-nek nevezzük.

9.1.2. A név hierachia

A hierarchia csúcsát *root*-nak, gyökérnek nevezzük. Az ez alatti neveket *top level domain*-oknak, *TLD*-knek mondjuk. Amikor az internet még csak USA hálózat volt, a következő TLD-k voltak használatosak:

edu – amerikai egyetemek, oktatási intézmények

com – vállalatok

mil – katonai szervezetek

gov – kormányhivatalok

net – hálózati szervezetek

org – mindenféle más szervezet

arpa – az internet ősében, az Arpanetben levő gépek neveire szolgált kezdetben. Az inverz nevek feloldásánál (ld. később) mind a mai napig fontos szerepe van.

Az USA-n kívüli domain-ok számára az *ISO 3166*² szabványban meghatározott kétkarakteres országkódot kezdték használni. Példák:

be – Belgium

pl – Lengyelország

hu – Magyarország

A hierarchia nagyon hasonlít az operációs rendszerek hierarchikus fájlstruktúrájához (pl. `/etc/bind/named.conf`), csak az alá-főlérendeltség itt éppen fordítva, jobbról balra olvasható le. Pl. `gep.csoport.osztaly.intezet.hu`. A TLD elnevezés mellett használatos még az SLD (second level domain) kifejezés is, a hierarchia második szintjén levő domain-okra.

9.1.3. Zónák

A név-fa zónákra oszlik: egy-egy zóna a *fa egyben kezelt része*. Sokszor – de nem feltétlenül –, egybeesik egy aldomainnel. Például egy zóna lehet az `osztaly.intezet.hu` és minden név, ami a hierarchiában ez alatt van. Egy zóna például az összes TLD-t tartalmazó root zóna is. Egy zóna a „láttató”, az „autoritatív” szerver szempontjából egy egység, rendszerint egy fájl. Egy-egy zónát több szerver is láttat(hat). Ezek közül az egyik az elsődleges, a többi (ha van) másodlagos.

Az elsődleges szerveren az adatok a zóna adminisztrátor munkájának eredményeképpen ténylegesen változnak.

A másodlagos szerver(ek) a zóna adatait meghatározott rend szerint az elsődleges szervertől tükrözi(k). A tükrözés rendjét az elsődleges szerveren a rendszeradminisztrátor a zóna konfigurációjával határozza meg.

²<ftp://ftp.ripe.net/iso3166-countrycodes>

9.1.4. Delegálás

A hierarchia egyes darabjait a zóna adminisztrátora tovább delegálhatja más szervekre. Például az `intezet.hu` domain gazdája az `osztaly.intezet.hu` aldomain láttatását, autoritását az illető osztály egy meghatározott gépére bízhatja a konfigurációban: mindenki felelős és úr lehet a saját illetékességi körében (szubszidiaritás elve). A root zóna sőt még a TLD-k (`edu`, `gov`, `hu` stb.) is jóformán mást sem tartalmaznak mint ilyen delegálást. Így jön létre a hierarchikus, osztott adatbázis. A delegálás azonban nem feltétele a több szintű név megadásának. Például lehetséges, hogy az `osztaly.intezet.hu` nincs delegálva, nem különálló zóna, mégis létezik a `gep.osztaly.intezet.hu` domain, mert az `intezet.hu` zóna gazdája bevezette a pontot (.) tartalmazó `gep.osztaly` nevet. Ezt éppen úgy megteheti, mint a `gep-osztaly` vagy az `osztalygepe` nevek bevezetését, melyeknek hatása a `gep-osztaly.intezet.hu`, illetve az `osztalygepe.intezet.hu` nevek létrejötte.

9.1.5. Domain nevek

A hierarchia következtében *minden név egyedi*. Lehet, hogy az internet több pontján is elneveznek egy gépet pl. `jupiter`-nek, de nevük egyértelmű, ha a teljes domain nevüket mondjuk: `jupiter.osztaly.intezet.hu`.

`jupiter.arizona.edu`.

A domain neveknek ezt a teljes alakját, ami a nevet a gyökér domain-ig tartalmazza *FQDN*-nek (Fully Qualified Domain Name), a domain név pontokkal elválasztott darabjait pedig *szegmenseknek* nevezzük. Annak jelzésére, hogy a domain név teljes, a név végére pontot teszünk. Valójában a TLD-re (`hu`, `edu`) való végződés nem garantálja, hogy a név FQDN: elképzelhető és tökéletesen szabályos a `jupiter.arizona.edu.osztaly.intezet.hu` domain név is.

Domain nevekben megengedett karakterek a *latin ABC* betűi [`a-z`], a *számjegyek* [`0-9`] és a *kötőjel* (`-`). Kis- és nagybetű egyformán használható, és nem jelent különbséget. Sajnos nem állhat domain névben ékezetes karakter. Gyakori hiba, hogy aláhúzás (`_`) karaktert adnak meg domain nevekben. Az eredeti definíció (RFC1035)³ az egyes szegmensek elején csak betűt engedett meg, a későbbi (RFC1123)⁴ megengedi a számmal kezdődő szegmenst is. Például szabályos a `3com.com` domain. Kötőjel viszont nem állhat továbbra sem se szegmens név elején, sem végén.

³<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1035.txt.Z>

⁴<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1123.txt.Z>

9.1.6. Cím -> név hozzárendelés

Az interneten nem csak arra van szükség, hogy nevekből IP címeket nyerjünk, hanem arra is, hogy *IP címekből domain neveket*. Ez a szolgáltatás – amit inverz, vagy reverz feloldásnak neveznek –, a hálózati biztonság szempontjainak erősödése miatt egyre nagyobb jelentőségű. Például sok FTP vagy levelező szerver nem fogad el kéréseket csak olyan gépekről, amiknek címéből a hozzájuk tartozó domain nevet ki lehet deríteni. Vannak szolgáltatások, amik csak bizonyos domain-okból érhetők el.

A cím–név feloldás érdekében bevezették az `in-addr.arpa` domain-t. IP címeket általában úgynevezett pontozott decimális (dotted decimal) alakban szokás megadni, ilyesformán: `150.151.152.153`. Az ehhez a címhez tartozó nevet úgy kapjuk meg, hogy a domain rendszertől megkérdezzük a `153.152.151.150.in-addr.arpa` névhez tartozó rekordot.

Az `in-addr.arpa` domainban éppen úgy delegálják az egyes aldomain-eket mint minden más zónában.

9.1.7. Rezolverek és DNS szerverek

Hogyan is zajlik a névfeloldás? Tételezzük fel, hogy a `jupiter.arizona.edu` nevet kell feloldani, mert pl. oda akarunk egy levelet továbbítani, vagy ftp-vel belépni. Ezért az általunk használt programnak – pl. a web böngészőnek –, megadjuk a `jupiter.arizona.edu` domain nevet. Programunknak ekkor meg kell állapítania, hogy milyen IP cím is tartozik ehhez a domain névhez. Ezt a funkciót ellátó egységet nevezzük *rezolvernek, feloldónak*. Gépünkön a TCP/IP szoftver telepítésekor, konfigurálásakor meg kellett adni egy vagy több DNS szerver-t. Ezekhez fordul a rezolver. A DNS szerver lehet a gép saját maga, vagy – elvben – tetszőleges gép az interneten. Tehát elvben lehetséges, hogy egy Indonéziában levő számítógép egy Dániában levő name szerver-t állít be a rezolver konfigurációjában. Persze az ésszerűtlen. Célszerű egy hálózati értelemben közeli szerver-t beállítani. A rendszergazdák kedves kötelessége erre vonatkozó információval ellátni felhasználóikat. A rezolver rendszerint néhány konfigurációs fájlból és könyvtári szubrutinból áll. Gyakorlatilag minden TCP/IP-t használó, internetbe kapcsolt számítógépen szükség van rá. A rezolver tehát nem végez közvetlenül névfeloldást, hanem bizonyos általa ismert névszervereket kér meg arra, hogy a feloldást elvégezzék.

A rezolver konfigurációban a DNS szerverek megadásánál értelemszerűen IP címeket kell használnunk. Sok konfiguráló program a szerverek megadásánál használja az „elsődleges” (primary), „másodlagos” (secondary) kifejezéseket. Ez gyakran zavart okoz, mert összekeverik a zónáknál használatos hasonló kifejezésekkel. A *rezolver* konfigurációnál megadott elsődleges/másodlagos névszerver a *látásra* vonatkozik, vagyis arra,

hogyan kéri a kliensünk milyen név szervereket kérdez. A zóna definíciójánál pedig az elsődleges névszerver az, amiről a másodlagos szerverek *tükrözik* a láttatott, mutatott zónát.

Amikor a rezolver a konfigurációjában megadott névszerverhez fordul, hogy például a `jupiter.arizona.edu` névhez tartozó IP címet megtudja, akkor a szerver általában nem válaszol azonnal. Példánkban legyen a kérdezett névszerver a `ns.intezet.hu`. Az `ns` konfigurációjának archimédeszi pontja – hasonlóan a rezolver konfiguráció DNS szerver IP címeihez –, a *gyökér névszerverek IP címe*. Ezek valamelyikét kérdezi az `ns` névszerver. Egy root névszervert kérdezve például a `jupiter.arizona.edu` névről, az nem ad mást, mint a `.edu` zónáért felelős névszerverek listáját. Az `ns` névszerver ekkor egy újabb kérdést intéz a `.edu` névszerveréhez, aki újra csak arra vonatkozóan ad információt, hogy hova lehet fordulni az `arizona.edu` nevek feloldásáért. Ilyen módon a `ns` rekurzív módon oldja fel a nevet, melynek végén a kérdező kliens gép rezolverének megadja a választ. A DNS szerverek általában nem végeznek bármely kliens számára ilyen rekurzív feloldást, hanem csak a konfigurációjukban meghatározottakra.

9.1.8. Cache, TTL

A névszerverek az általuk megtudott neveket tárolják azzal a céllal, hogy ha újra megkérdezik tőlük, akkor ebből a cache-ből *azonnal tudjanak válaszolni*. Ennek többszörös haszna van: *csökkenti* a hálózati forgalmat, és *gyorsítja* a névfeloldást. A cache-ben minden megtudott nevet, csak egy bizonyos ideig tárolnak. Ha ez az idő lejárt, akkor egy újabb kéréskor – hiába lenne a cache-ben az információ –, a névszerver újra kérdezi azt. Ilyen módon, ha a névhez tartozó információ esetleg változik, arról tudomást szerezhet. Azt az időt, ameddig a cache-ben van egy-egy információ, nem a tárolóban, hanem a láttató, az autoritativ szerverben döntik el: minden rekordhoz tartozik egy – sokszor implicit módon megadott – *TTL (Time To Live)* érték. Ennyi másodpercig tárolják a szerverek a cache-ükben az információt.

9.1.9. Névszerverek funkció szerint

Caching only szerverek

A névszerverek egy része nem autoritás semmilyen névre, hanem csak arra szolgál, hogy feloldja a neveket a kliensek számára. Ezeket nevezzük „caching only” – csak cache-elő – névszervereknek. Általában ajánlatos minden lokális hálózaton legalább egy névszervert működtetni. Ha nincs „láttató” feladat, akkor caching-only szerverre van szükség.

Láttató, autoritatív szerverek

Ahogy már erről szó volt, ezek azok a név szerverek, melyeknek az (is) feladata, hogy bizonyos neveket ők mutassanak meg mások számára. A domain név-fa egy *egyben delegált ágát, melyért egy szerver felelős, zónának nevezzük*. Egy zónáért felelős név-szerverek közt van egy kitüntetett, amelyen az *adminisztrátor* a konfigurációt változtatja. Az (esetleges) többi ezt a zónát tükrözi. A kitüntetett szerverre elterjedt kifejezés az „elsődleges”, „primary” a tükröző szerverekre pedig a „másodlagos”, „secondary”. Újabban (elsősorban a 8. változatú BIND megjelenésének hatására) inkább a *master* és a *slave* neveket használják. A master és slave név azért szerencsésebb, mert nem keveredik a rezolver konfigurációknál megadható „primary”/„secondary” szerverekkel. Sajnos a „slave” szerver kifejezés is használatos már régebben és más értelemben: az olyan szerverekre mondjuk hogy „slave”, amelyik csak forwarderek közvetítésével érintkezik az internet nagyobb részével. Egy szerver lehet egy zónára „master” egy másikra „slave”. Valójában gyakori is, hogy két intézmény kölcsönösen „slave” autoritatív szerver a egymás zónáira. A névfeloldás szempontjából a „master” és a „slave” szerverek között semmi különbség nincsen: egyformán autoritatív mindegyik. A névszerverek a név feloldás során bármelyikhez fordulhatnak. A valóságban a kód úgy működik, hogy a szerverek egy-egy zóna autoritatív szerverei közül igyekeznek azt kérdezni, amelyik gyorsabban válaszol, aminek érdekében egy ravasz algoritmust használnak: kezdetben mindegyik névszervert megkérdezik, méri a válaszidőt, aztán azt preferálják, ami gyorsabban válaszolt, de a lassabb szerverek idővel újra szót kaphatnak, mert minden kérdésnél „csökken a büntetésük”.

Forwarder szerverek

Egy névszerver gyakorlatilag kiegészítheti a cache-ét más szerverek cache-ével, ha a forwarder opciót használják a konfigurálásánál. Ha pl. `kicsi.valahol.hu` gépen a DNS konfigurációban megadják, hogy a `nagy.valahol.hu` forwarder legyen számára, akkor a `kicsi`-n történő névfeloldás úgy zajlik, hogy ha a `kicsi` cache-ében nincs benne a kért név, akkor a `kicsi` DNS szerver mielőtt a világban a név-fa hierarchiának megfelelő módon elkezdene érdeklőni, megkérdezi a `nagy`-ot. Ha annak a cache-ében megtalálható a keresett rekord, akkor válaszol, és így a `kicsi` gyorsan megtalálja a választ. Elképzelhető, hogy egy-egy intézménynél több kisebb szerver használ egy közös nagyobb forgalmú forwardert. Például nem csak a `kicsi.valahol.hu`, hanem a `pici.valahol.hu` is a `nagy.valahol.hu`-t. A több irányból érkező, több kérés hatására a `nagy.valahol.hu`-nak nagy cache-e keletkezik.

Slave szerverek

Az olyan szervert, ami csak forwardert (esetleg többet) használ a nevek feloldására, slave szervernek nevezzük. Slave szerverre van szükség tűzfal mögött, ahol a szervernek módja sincs, hogy közvetlenül kilásson az internetre. Ahogy már említettük ez a fajta „slave” fogalom nem keverendő össze a „slave” fogalmával egy-egy zóna szempontjából: a forwarder(ek)re támaszkodó slave szerver korlátozott a *látás* szempontjából, egy-egy zóna slave szervere pedig az illető zóna *mutatása, láttatása* szempontjából.

9.1.10. Zónafájlok

A névszerverek az egyes zónák adatait általában egy-egy fájlban tárolják. A „master” szerveren az adminisztrátor személy közvetlenül, vagy valamilyen program közvetítésével maga módosítja ezt a fájlt. A „slave” szervereken a fájl a tükrözés eredménye.

A zónafájl rekordokból, RR-ekből (resource record) áll. Nagyon sok fajta rekordot tesznek lehetővé az RFC-kben megadott definíciók. A következőkben ezek közül ismertetjük a legfontosabbakat.

A rekordok formáját az RFC1035⁵ határozza meg, és az a következő:

cimke ttl osztály típus adatok

A „*cimke*” a domain rekord neve. Lehet üres, ilyenkor az előtte levő rekord címkéje érvényes. A „*ttl*” a rekordhoz tartozó time to live időt adja meg másodpercben. Nem kötelező paraméter. Ha elhagyjuk, akkor a zónára vonatkozó alapértelmezés lesz a rekordhoz tartozó érték. A következő paraméter értéke gyakorlatilag mindig *IN*, azaz internet osztály. Ez is elhagyható. A „*típus*” mondja meg, hogy milyen fajta információról is van szó. Pl. IP cím (A rekord), name szerver információ (NS rekord) stb. Az „*adatok*” mező a rekord típusától függő információt tartalmaz.

9.1.11. Rekordok

SOA - Start of Authority rekord, zóna kezdő rekord

A SOA rekord adja meg egy zónára vonatkozó közös információkat. A rekord formáját egy példán mutatjuk be:

valami.hu.	SOA	gep.valami.hu.	mester.valami.hu. (
		1999093001	;Serial nr.

⁵<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1035.txt>.Z

86400	;Refresh
1800	;Retry
604800	;Expire
43200)	;TTL

A *cimke* (valami.hu.) a zóna neve. A SOA kulcsszó utáni első paraméter a zónához tartozó elsődleges szerver domain neve. A második paraméter egy e-mail cím, melyet úgy kapunk, ha az első olyan . karaktert, amit nem előz meg backslash (\), at jelre, @-ra cseréljük. A *serial nr.* a zóna sorszáma. Arra szolgál, hogy a slave (másodlagos) szerverek ellenőrizhessék, hogy a náluk levő zóna tartalom nem avult-e el. Akkor töltik le az master (elsődleges) szerverről a zóna tartalmát, ha a náluk levő zóna sorszám kisebb. Arra kell tehát vigyázni az elsődleges szerver adminisztrátorának, hogy ez a szám mindig növekedjen, ha valamit változtat, ha új változat keletkezik a zónából. Szokás ezt a sorszámot ÉÉÉÉHHNNVV alakban megadni, ahol ÉÉÉÉ az év négy jegyen, HH a hónap két jegyen, NN a nap két jegyen, VV a napon belüli változat két jegyen ábrázolva. Az ez után következő négy paraméter mind *másodpercben* megadott érték. Az első a *refresh*, a *frissítés idő* azt mondja meg, hogy mennyi időnként kell a slave szervereknek a master-től megkérdezni, hogy a zóna sorszáma mennyi, vagyis, hogy szükséges-e a zónát frissíteni náluk. A *retry* idő azt mutatja, hogy ha a frissítés nem sikerült, akkor mennyi időt várjanak, mielőtt újra próbálkoznának. Az *expire* azt mondja meg, hogy ha nem sikerül a master-rel kommunikálniuk, ennyi ideig szolgáltatják a zónát a világ számára. A TTL érték lesz a zóna rekordjaira érvényes alapértelmezés.

Figyelni kell rá, hogy észszerűen állítsuk be a zóna SOA rekordjában az idő értékeket. A legtöbb esetben az 1 napos (86400) refresh, 1 órás (3600) retry, 1 hetes (604800) expire és 1 napos (86400) TTL megfelelő. Ha gyors változás várható, akkor érdemes a TTL értéket kicsire venni. A dolog természetéből adódóan súlyos zavarokat okoz, ha az expire idő nem nagyobb mint a refresh: a másodlagos zóna nem fogja szolgáltatni az adatokat az idő egy részében.

A 8-as változatú Bind-nál a másodpercben értendő dimenzió nélkül megadott számok helyett használhatunk emberek számára könnyebben kezelhető mértékegységekben megadott számokat, ilyenformán: 1W2D3H

A W (week) heteket, D (day) napokat, H (hour) órákat jelent.

A – Address, cím rekord

Ez a leggyakrabban használt rekord, amely arra szolgál, hogy egy domain névhez IP címet rendeljünk. Például:

```
masina A 190.111.222.3
```

Sokszor használt tulajdonságát látjuk itt a zónafájlnak: nem írjuk ki egy domain (jelen esetben a masina) teljes domain nevét, csak annak első részét. A végére oda kell érteni azt a *vonatkoztatási rendszert*, ahol éppen vagyunk. Ezt először is maga az a zóna adja meg, amire ez a fájl vonatkozik. Például ha a valami.hu zónáról van szó, akkor a „masina” a végére biggyesztett pont nélkül úgy értendő, mint masina.valami.hu. Ez a tulajdonság legtöbbször igen kellemes, mert például egy 200 A rekordot tartalmazó zóna esetében nem kell 200-szor megismételniünk a zónában, hogy „egyik.valami.hu., masik.valami.hu.” hanem elég annyit írunk „egyik, masik”. Vigyáznunk kell azonban, mert könnyen elfelejtezhetünk arról, hogy pontot kell tennünk a domain név végére, ha azt teljes egészében kiírjuk valahol. Figyeljük meg ebből a szempontból a SOA rekordra felhozott példát fentebb.

NS - Name Server, névszerver rekord

Ez a rekord szolgál arra, hogy egy domain névszervereit megadjuk. Ilyen módon a domain egy delegálási pont. Példa:

```
osztaly NS gep.osztaly.valami.hu.
```

Ezzel a rekorddal deklaráljuk, hogy az „osztaly” aldomain névszervere a gep.osztaly.valami.hu. Ajánlatos – bár technikai értelemben nem kötelező – legalább két névszervert megadni. Ilyen módon a zóna adatai akkor is elérhetők a világból, ha az egyik gép, vagy a hozzá vezető vonal valami miatt kiesne. A felsőbb szinten – példánkban a valami.hu zóna alatt –, nem látszik, hogy a szerverek közül melyik a master és melyik a slave. Szigorúan véve az NS rekordoknak csak a felsőbb szinten, az „apuka” zónában van szerepe, indokolt azonban a zónában is felsorolni. Az NS rekord paramétere egy gép domain neve. Szükséges, hogy ehhez a névhez közvetlenül A rekord tartozzon. Elő-előfordul, de hibás CNAME rekorddal definiált domain nevet megadni.

Glue rekord. Gyakori, hogy a delegált zóna egyik name szervere éppen a zónában van, mint a fenti példában. A gep.osztaly.valami.hu rekordnak az osztaly zónában van a helye, de mégis szükség van arra, hogy egy szinttel feljebb, a valami.hu zónában is felsoroljuk, különben csapdába kerülünk. Ezért fel kell vennünk egy nem oda való A rekordot:

```
gep.osztaly A 190.1.2.3
```

Az ilyen, idegen A rekordot nevezik glue (ragadvány) rekordnak. Előfordul, hogy adminisztrátorok akkor is felsorolnak nem a zónába való A rekordot, amikor az nem egy onnan delegált aldomainban van, ez hiba. Semmi haszna és zavart okoz. Tehát például ha

az osztaly.valami.hu zónának egy másik névszervere a mas.nevszerver.intezet.hu, akkor ehhez nem kell glue rekordot csatolni a valami.hu zónában, hiszen ennek a névszervernek az A rekordját ettől a delegálástól teljesen függetlenül lehet megtudni.

Lame delegálás. Ha valahova delegálunk egy zónát, akkor az ottani adminisztrátorral meg kell beszélnünk, hogy azt folyamatosan szolgáltatssa is. Ha ez nem történik meg, akkor beszélünk „lame” delegálásról. Sokszor előfordul például amiatt, mert a delegált zóna slave szervere nevet változtat, vagy meg is szűnik, és erről elfelejtik értesíteni a felettes zóna gazdáit.

CNAME - Canonical Name, kanonikus név rekord

Ez a rekord arra való, hogy egy hostnak becenevet adjunk. Például:

```
www CNAME gep
```

Ha ez a rekord van mondjuk a valahol.hu zónában, az azt mutatja, hogy a www.valahol.hu egy másik neve a gep.valahol.hu-nak. Nagyon hasznos az ilyen név például a következő esetben: tegyük fel, hogy egy idő után a gep.valahol.hu meg is szűnik, és a szolgáltatást az ujdivat.valahol.hu veszi át. Ilyenkor elég csak a CNAME rekordot módosítani, így:

```
www CNAME ujdivat
```

A világ számára a valahol.hu web lapjai továbbra is a www.valahol.hu gépen lesznek elérhetők. Az is gyakori, hogy egy gép több funkciót is ellát, és a funkciók mindegyikéhez tartozik egy-egy CNAME rekord, ami ugyanarra a gépre mutat. Például news.valahol.hu, ftp.valahol.hu mind mutathatnak ugyanoda.

Mint látjuk, a CNAME rekord paramétere egy domain név. Általában ez a név már A rekorddá oldható fel. Megengedett, de nem ajánlatos a CNAME-ra mutató CNAME rekord.

MX – Mail eXchanger, levelező szerver rekord

Ez a rekord szolgál arra, hogy egy domainba érkező levelek levelező szerverét kijelölje. A rekord formátuma egy példán:

```
valahol.hu.      MX      10      masina.valahol.hu.
                  MX      20      mas.mashol.hu.
```

Ezek a sorok azt jelentik, hogy a `valaki@valahol.hu` alakú címre érkező leveleket a `masina.valahol.hu`, vagy a `mas.mashol.hu` gépekre kell küldeni. Az MX rekordok első paramétere egy szám, ami a rekord preferenciát jelenti. Kötelező paraméter, de csak akkor van jelentősége, ha több MX rekord tartozik ugyanahhoz a névhez: kisebb szám nagyobb preferenciát jelent. Példánkban tehát csak akkor fogják a levelező szerverek a `mas.mashol.hu`-ra küldeni a `valahol.hu` domainba szóló leveleket, ha a preferáltabb `masina.valahol.hu` nem elérhető. Lehetséges több MX rekordot egyenlő preferenciával megadni. Ilyenkor véletlenszerű, hogy melyikre érkezik be egy-egy levél. Az MX rekord második paramétere egy domain név. Fontos, hogy ehhez a névhez már A rekord tartozzon. Nem megengedett olyan domain nevet magadni, ami csak egy CNAME-ra, vagy másik MX-re mutat.

Az MX rekord gyakori alkalmazása, amikor egy intézményben egységes, egyszerűsített, és könnyen megjegyezhető levélcímeket vezetnek be a segítségével. Például a Firma cégnél `kovacs.janos@firma.hu` alakú levelezési címe lehet mindenkinek, ha a `firma.hu` MX rekord egy – akár időben változó – levelező szerverre mutat, ahol aztán feloldják a levél cím első részében a név alias, esetleg tovább küldik a levelet egy másik szerverre.

TXT – Text, szöveges rekord

Ez a rekord tetszőleges szöveges információt tartalmazhat. Példa:

```
modern      TXT      "Ez a gep mar megszunt "
```

A TXT rekord paramétere egyetlen, idézőjelek közé zárt ASCII karaktersorozat.

HINFO - Hardware information, hardver információ rekord

Akárcsak a TXT rekord ez a rekord is emberi olvasásra szánt, egy számítógépről nyújt felvilágosítást. Példa:

```
masina      HINFO      VAX      VMS-4.7
```

Mint látható, két paramétere van. Az első a hardver típust, a második az operációs rendszert szokta jelölni.

PTR – Pointer rekord

Ahogy arról már szó volt, nem csak név–cím, hanem cím–név hozzárendelésre is szükség van. Ezt a szolgáltatást elsősorban nem emberek, nem is kliens programok, hanem

szerver programok használják, annak kiderítésére, hogy egy hozzájuk érkezett IP csomag milyen domainhoz is tartozik. DNS rendszerben az `in-addr.arpa` domain alá tartozó ág szolgálja a cím-név felosztást. Itt a zónák delegálása az IP címtartomány egyes darabjainak megfelelően történik. Példa:

```
140.in-addr.arpa.      NS      ...
```

Ez a zóna a `140.x.y.z` alakú IP címek inverz domain név szolgáltatásánál játszik szerepet. Ha egy intézmény egy C osztályú címet kap, vagyis gazdálkodhat pl. a `192.84.124.x` alakú címekkel, akkor célszerű, ha nála van a `124.84.192.in-addr.arpa` zóna elsődleges névszervere is. Ebben a zónában vannak azután a PTR rekordok. Például:

```
22      PTR      gep.valahol.hu.
```

A PTR rekord egyetlen paramétere az a domain név, ami az illető IP címhez tartozik. A paraméterként megadott domain név A rekorddá kell forduljon az „egyenes” feloldáskor. Amikor egy domain-adminisztrátor – elterjedt kifejezéssel „hostmaster” – egy gépnek, vagy valamilyen hálózati interfésznek nevet, és IP címet oszt, fontos, hogy gondoskodjon az *inverz feloldásról is*: általában párhuzamosan van szükség egy-egy A rekord és PTR rekord bejegyzésére. A dolog természete miatt az „egyenes” és az inverz zónák *nem járnak feltétlenül együtt*. Ha az `osztaly.intezet.hu` zónát kezeljük, és gazdálkodunk egy IP címtartománnyal, akkor nem nyilvánvaló, hogy mi is a kiosztott IP címekhez tartozó inverz zóna, vagy hogy azt egyáltalán mi kezeljük. Kezdő domain név adminisztrátoroknál gyakori hiba, hogy elfelejtkeznek az inverz domainről. A DNS hierarchikus szerkezetéből következik azonban, hogy bárki számára egyértelműen kideríthető, hogy ki is a felelős az általunk osztott IP címekhez tartozó `in-addr.arpa` zónáért. Neki kell azután szólni, hogy a megfelelő bejegyzést végezze el, vagy delegálja tovább a zóna egy darabját nekünk. Például ha a „host” parancsot használjuk a DNS nézegetésre, és arra vagyunk kíváncsiak, hogy kinek is kell bevezetni a `202.103.132.169` IP címhez tartozó inverz rekordot, akkor a következő láncon haladhatunk:

```
$host -t any 202.in-addr.arpa
202.in-addr.arpa      NS      NS.RIPE.NET
202.in-addr.arpa      NS      NS.TELSTRA.NET
202.in-addr.arpa      NS      NS.APNIC.NET
202.in-addr.arpa      NS      SVC00.APNIC.NET
202.in-addr.arpa      SOA      NS.APNIC.NET inaddr.APNIC.NET (
1999091501           ;serial (version)
86400                ;refresh period (1 day)
7200                  ;retry interval (2 hours)
```

```

                2592000           ;expire time (4 weeks, 2 days)
                345600           ;defaultttl (4 days)
            )
$

```

Tehát a 202.x.y.z alakú IP címekhez tartozó inverz zónákat az ns.apnic.net gépen kezelik, és szolgáltatja még három másik name szerver.

Haladjunk tovább:

```

$host -t any 103.202.in-addr.arpa
103.202.in-addr.arpa      NS      ns.telstra.net
103.202.in-addr.arpa      NS      svc00.apnic.net
103.202.in-addr.arpa      SOA      ns.apnic.net      inaddr.apnic.net (
                1999081001          ;serial(version)
                86400               ;refresh period (1 day)
                7200                ;retry interval (2 hours)
                2592000             ;expire time (4 weeks, 2 days)
                345600             ;default ttl (4 days)
            )
$

```

A 202.103.x.y alakú IP címek zónájának hazája ezek szerint szintén az ns.apnic.net.

És itt:

```

$host -t any 132.103.202.in-addr.arpa
132.103.202.in-addr.arpa does not exist, try again
$

```

és:

```

$host -t any 169.132.103.202.in-addr.arpa
169.132.103.202.in-addr.arpa does not exist, try again
$

```

Vagyis a helyzet kulcsa annak a személynek a kezében van, akit az inaddr@apnic.net címen érhetünk el: vagy tovább kell delegálnia megfelelő helyre a 132.103.202.in-addr.arpa zónát, vagy neki kell bevezetnie a 169-es IP címhez tartozó PTR rekordot.

De Groot féle inverz feloldás. A klasszikus interneten az IP címeket A, B, és C osztályú hálózati darabokban osztották, és amikor egy intézmény egy címtartományt kapott, pontosan meg lehetett mondani, hogy melyik `a.in-addr.arpa`, `b1.b2.in-addr.arpa` vagy `c1.c2.c3.in-addr.arpa` zóna tartozik a kapott címtartományhoz. Ennek a delegálását kellett az intézmény adminisztrátorának kérnie, és ettől kezdve könnyen kezelhette az egyenes és `in-addr.arpa` zónáit. A CIDR (Classless Inter-Domain Routing) elterjedésével gyakori, hogy egy-egy intézmény például csak egy negyed részét kapja meg egy C osztályú címnek. Az ilyen címtartományt úgy szokás jelölni, hogy a legkisebb használható cím után / jellel elválasztva megadjuk a tartományt jellemző bitmaszk egyeseinek számát. Például: `193.225.86.128/26` jelenti a `193.225.86.128`-tól `193.225.86.191`-ig terjedő címtartományt. Előfordulhat ilyen módon, hogy egy C osztályú cím 4 vagy még több egymástól távol eső intézmény között oszlik meg. Ha az ilyen tartományhoz tartozó inverz domaint a hagyományos módon szeretnénk kezelni, akkor minden intézménynek egy központi helyen kellene a PTR rekordjait beírni. Ez bonyolult és kellemetlen. Sokkal jobb, ha – mint a klasszikus esetben – minden intézmény saját maga jegyezheti be a saját PTR rekordjait. A problémára Geert Jan de Groot adott megoldást, és azt az RFC2317⁶ írja le. A megoldás a DNS technika szellemes alkalmazását mutatja.

A darabokra szabdalt C osztályú címhez tartozó `in-addr.arpa` zónában nem vezetünk be PTR rekordokat, viszont minden egyes rekordhoz bevezetünk egy CNAME rekordot. Ez a CNAME rekord olyan domain névre mutat, ami a címet birtokló intézmény adminisztrátora definiál. Például ha a `193.225.86.0` hálózatról van szó, akkor a `86.225.193.in-addr.arpa` zónába bevezetünk 256 CNAME rekordot. Ezek jobb oldalán elvben tetszőleges domain név lehet, de szokás olyat megadni, ami az illető címtartomány kezdetét és nagyságát is jelzi, ilyenformán:

```
131.86.225.193.in-addr.arpa. CNAME
131.128/26.86.225.193.in-addr.arpa.
```

Az egyes kiosztott IP címtartomány darabokhoz megfelelően delegált `in-addr.arpa`-beli zónák tartoznak. Például a fenti esetben:

```
128/26.86.225.193.in-addr.arpa.          NS          ns.intezmeny.hu.
```

Ilyen módon a C osztályú címhez tartozó zónában a címek kiosztása után egyszer s mindenkorra rögzíteni lehet a bejegyzéseket, a kis címtartományt birtokló helyen pedig csak arra van szükség, hogy az inverz zóna neve `c1.c2.c3.in-addr.arpa` alak helyett `cim/maszk.c1.c2.c3.in-addr.arpa` alakú legyen. Ebbe a zónába aztán éppen úgy kell PTR rekordokat felvenni mintha teljes C osztályú címhez tartozna a zóna. Például:

⁶<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2317.txt.Z>

131 PTR bagoly.intezmeny.hu.

Ha ez a rekord a 128/26.86.225.193.in-addr.arpa. zónában van, akkor a fenti CNAME rekorddal együtt két lépcsőben feloldást ad a 131.86.225.193.in-addr.arpa domain névre, melynek eredménye bagoly.inetzmény.hu.

9.1.12. BIND - Berkeley Internet Name Domain

BIND a neve az interneten *leggyakrabban használt DNS implementációnak*. A program elsősorban unix típusú gépeken fut, de van pl. NT-s változata is. Fejlesztését az Internet Software Consortium⁷ támogatja, és a BIND „apukája”, Paul Vixie koordinálja. A BIND program forráskódban is szabadon letölthető az ftp.isc.org szerverről.

BIND változatok

E sorok írásakor a BIND kurrens változata 8.2.2. Használatosak azonban ennél sokkal régebbi változatok is. Jelentős ugrást jelentett 1998-ban a 4.9.x változatok után a 8.x változatok megjelenése. Ekkor a konfigurációs fájl szintaxisa is, a fő konfigurációs fájl neve is megváltozott.

4.9.x konfiguráció

Ezen változatok konfigurálásának archimedeszi pontja az /etc/named.boot fájl. Ebben kell leírni azt, hogy milyen zónákra elsődleges vagy másodlagos a szerver, és hogy a zónák milyen fájlokban tárolódjanak. Például:

```
;
tipus          domain          source

primary        valahol.hu       valahol.zona
primary        1.2.199.in-addr.arpa  inverz.zona
secondary      amicus.hu        192.84.3.4      amicus.zona
```

Láthatjuk, hogy a primary kulcsszó után két paramétert kell megadnunk: a zóna nevét, és a fájlt, ami a zóna adatait tartalmazza. A secondary kulcsszóhoz három paraméter tartozik: a zóna neve, a név szerver(ek) ami(k)ről a zónát tükrözni kell, és a fájlnev, ahova a tükrözött adatokat mentjük.

A 4.9.x konfigurációk fontos sorai még:

⁷ <http://www.isc.org>

```
directory      /var/named
cache           .                               root.cache
```

A *directory* kulcsszó után megadott könyvtárhoz relatívan helyezkednek el a konfigurációban megadott fájlok.

A *cache* direktíva arra szolgál, hogy a gyöker névszerverek neveit és címeit tartalmazó fájlt megadjuk. A névfeloldás – a szerver látó funkciója –, úgy fog zajlani, hogy ebből a fájlból veszi a szerver a root szerverek adatait. A root szerverek száma egyre bővül. E sorok írásakor 13 névszerver autoritás az interneten a . zónára. Fontos, hogy a root névszerverek listáját naprakészen tartsuk. A mindenkori lista megszerezhető a DNS segítségével. Ha például a *host* parancsot használjuk, akkor a következő parancs a *friss.cache* fájlba teszi az aktuális listát:

```
host -v -t ns -l . >friss.cache
```

A DNS gyökeréért felelős névszerverek listája letölthető ftp–vel is például az *ftp.internic.net* szerverről: `ftp://ftp.internic.net/domain/named.root`

8.x konfiguráció

A nyolcas változatú BIND–ok konfigurálásának archimedeszi pontja az `/etc/named.conf` fájl. Ennek sokkal bonyolultabb lehet a szerkezete, *árnyaltabb* feltételek megfogalmazására ad módot, mint egy 4.x konfiguráció. Ha valaki 4.x változatról 8.x–re akar áttérni, a *named.boot* fájl helyett az új szintaxisú *named.conf*–ra van szüksége. A 8–as disztribúció tartalmaz egy perl scriptet, ami *automatikusan konvertál* egy *named.boot*–ot *named.conf*–ra. Persze egy ilyen konfiguráció messze nem meríti ki az összes lehetőséget a lehetséges, és hasznos finomságokat illetően. Amint láttuk, a *named.boot/named.conf* fájl csak a kiindulási pontja a konfigurációnak: a tényleges DNS rekordok ettől különböző zóna fájlokban vannak. Ezek formátuma nem változott, nem is nagyon változhat: ami abban van, azt RFC⁸ írja le, nem implementációs kérdés.

A következőkben ismertetjük a 8.x konfiguráció néhány elemét.

Options utasítás. Ezzel az utasítással a konfiguráció egészére állíthatunk be tulajdonságokat, alapértelmezéseket. Példa:

```
Options
{
```

⁸ [ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1035.txt](http://ftp.sztaki.hu/pub/nic/rfc/rfc1035.txt).Z

```
directory      "/usr/local/named";
named-xfer     "/usr/local/sbin/named-xfer";
notify         yes;
check-names    master fail;
check-names    slave warn;
check-names    response ignore;
recursion      yes;
transfers-in   20;
allow-transfer {"amici";};
allow-query    {"anybody";};
listen-on      192.84.225.2;
}
```

A *directory* opció egy könyvtárra mutathat. A konfigurációban szereplő fájlnevek ehhez a könyvtárhoz relatívan értendők.

A *named-xfer* opciónak akkor van jelentősége, ha szerverünk slave (másodlagos) bizonyos zónákra. Ilyenkor a *named-xfer* opcióval annak programnak a nevét adhatjuk meg, amelyik a zónatranszfert végzi.

A *notify* egy új, és igen hasznos elem a 8. BIND-okban. Mód van arra, hogy a slave szerverek azonnal értesítést kapjanak, ha a zóna változott. Így nem kell kivárni amíg a SOA rekordban meghatározott „refresh” értéknek megfelelő időzítés lejár, a slave szerverek azonnal kezdeményezhetik a frissítést. Persze ez feltételezi, hogy a slave olyan BIND-ot futtat, ami ezt támogatja. Önmagában ez a tulajdonság is indokolja, hogy ne futtassunk 4.x változatú BIND-ot, hanem térjünk át frissebb változatra.

A *check-names* opció azt szabályozza, hogy ha a konfigurációs hibával szembesül a program, hogyan viselkedjen. Külön lehet szabályozni, mit tegyen, ha a saját master (elsődleges) zónáiban, a slave (másodlagos) zónákban, vagy egy kérdésre adott válaszban talál hibát. Mindegyik esetben három értéket lehet megadni:

ignore Ügyet sem vet a hibára

warn A logfájlba hibaüzenetet ír

fail A logfájlba hibaüzenetet ír, és az adatot nem veszi figyelembe.

Ajánlatos legalább „master” zónákra beállítani a „fail” értéket. Így ha elrontjuk a zónafájlt, akkor szerverünk egyáltalán nem szolgáltatja, ezáltal hamar felfedezzük a hibát, és módunk lesz kijavítani.

A *recursion* opció azt határozza meg, hogy szerverünk csak láttat (az autoritatív zónákról nyújt információt), vagy hajlandó klienseknek kérdéseket megválaszolni a többi

DNS szerverrel való kommunikáció révén. A TLD–ket szolgáltató szerverek általában nem rekurzívak.

A *transfers-in*, *transfers-out* opciókkal azt szabályozhatjuk, hogy egy időben hány zónatranszfer működhessen be, illetve kifelé.

Az *allow-transfer* opcióval meghatározhatjuk, hogy kinek engedjük meg egész zónák elvitelét. Azoknál a zónáknál van ennek jelentősége, amelyekre autoritatívak vagyunk. Természetesen meg kell engedni a slave zónáknak, hogy a master–től elvigyék a zónát. Ezen kívül azonban megtilthatunk minden zónatranszfert. Régen, amikor nem volt olyan sok a visszaélés az interneten, minden name szerverről le lehetett tölteni bárhova a zónákat. Ezen alapulnak például az internet host count-ok és statisztikák: az internet bármely pontjáról meg lehet(ett) csinálni, hogy a root szerverektől kezdve bejárja egy program az egész név–fát, minden zónáról zónatranszfert kér, és például összeszámolja az A rekordokat (ez a host count statisztika). Amikor ezek a sorok íródnak nagyon sok name szerver adminisztrátor korlátozza a zónatranszfert, és így kétes eredménnyel járna, akár csak a .hu alatti A rekordok összeszámolása is. Az *allow-transfer* opció paramétere egy *acl* (access control list), egy címlista. A címlistában hálózatokat, IP címeket sorolhatunk fel ilyenformán:

```
acl
amici {
    199.2.2.4;
    192.3.4.5;
    195.2.3/24;
};
```

Az első két sor két IP címről – a slave–ekről –, engedi meg a zónatranszfert. A harmadik sorban a /24 egy bitmaszkot jelent: a 193.2.3. hálózat bármely gépének megengedjük a zónatranszfert. Az így definiált lista nevére aztán hivatkozhatunk más utasításokban (pl. *options/allow-transfer*, *zone*). Az *acl* utasításnak azonban meg kell előznie a névre való hivatkozást.

Az *allow-query* opcióval korlátozhatjuk, hogy honnan jövő kéréseket válaszoljon meg szerverünk. Paramétere ennek is egy címlista.

Az *options* utasításban megadott „*allow-transfer*” és „*allow-query*” opciót az egyes zónák definiálásánál hasonló szintaxisú utasítással felülbírálhatjuk.

A *listen-on* opciónak akkor van jelentősége, ha több IP címe van szerverünknek: meghatározhatjuk, hogy melyik IP címekre érkező kérdésekre válaszoljon. Alapértelmezésben mindegyiken elérhető a DNS szerver.

A 4.x konfigurációnál használt „*primary*” és „*secondary*” és „*cache*” utasításokat a *zone* utasítás váltotta fel. Példák:

```
zone valami.hu {
type master;
file "valami.hu";
allow-transfer {"amici";}
};
```

```
zone mas.hu {
type slave;
masters { 192.2.3.4;
          193.4.3.2;
        }
file "sec/mas.hu";
}
```

```
zone "." {
type hint;
file "root.hints";
};
```

A `valami.hu`-ra elsődleges (master) a `mas.hu`-ra másodlagos (slave) szerverek vagyunk. A `valami.hu` zónát a `valami.hu` fájlban kell prezentálnunk a szervernek. A `mas.hu` zónát a szerver a `sec` alkönyvtár `mas.hu` nevű fájljába kérjük. A `valami.hu` zónánál felülbíráltuk az „options”-ban megadott alapértelmezését a zónatranszfer engedélyezésnek: itt csak az „amici” nevű címlista tagjainak engedjük meg az átvitelt. A `mas.hu` zónánál két „master” szervert is felsoroltunk. Ezek közül értelemszerűen (legalább) az egyik szintén „slave”, vagy ugyanannak a „master” szervernek két különböző IP címéről van szó. A „masters” utasításban megadott IP címek sorrendje prioritást jelent: elsősorban az elsőről szinkronizálja a szerver a zónát, ha az nem sikerül, akkor megy tovább a listán. Általában nem érdemes több IP címet megadni, de hasznos lehet, ha komoly esély van arra, hogy az elsőről nem sikerül szinkronizálni. A gyökér szerverekre vonatkozik a „hint” típusú zóna. Az itt megadott fájl-ba (`root.hints`) tegyük a root name szerverekre vonatkozó NS és A rekordokat.

9.1.13. Hasznos segédprogramok

A DNS adatok lekérdezésének klasszikus eszköze az `nslookup`. Ez a program szinte minden operációs rendszernek szabványos része. Vannak azonban szabadon terjeszthető alternatívái, amiknek talán kényelmesebb a felhasználói felülete, és a DNS kérdezésen kívül sok mást - például hibafelderítést - is közvetlenül támogatnak.

Host. Ennek a programnak több változata is elterjedt. Igen jó, sokat tudó és folyamatosan fejlődő az Erik Wassenar féle. Kurrens változata letölthető az ftp.nikhef.nl-ről. Érdeemes letölteni és installálni az friss változatot akkor is, ha operációs rendszerünk már tartalmazná a program valamely változatát. A BIND disztribúcióval is jön egy változat, de általában az sem elég friss.

Dig. A Dig hasonló célokat szolgál, mint a host. Ízlés és szokás kérdése, hogy ki melyiket használja.

Grafikus DNS adminisztrációs segédletek. Felmerül az igény, hogy ne kézzel editáljuk a DNS konfigurációs fájlokat, hanem pl. egy grafikus felületen át. Erre több megoldás is készült. Egyik ezek közül a Webmin. Ez több rendszeradminisztrációs feladatra – többek közt DNS adminisztrációra – alkalmas web-es felületet nyújtó eszköz. Otthona: <http://www.webmin.com/webmin/>

Megjegyzendő, hogy az ilyen eszköz nem pótolja a hozzáértést.

9.1.14. Regisztrálás a .hu TLD és a .hu alatti közcélú SLD-k alatt

A .hu alatti regisztrálás feltételeit e sorok írásakor (1999. november) az Internet Szolgáltatók Tanácsa szabályozza. Létrejött néhány közcélú második szintű (SLD) domain: org.hu, co.hu, info.hu stb. Az ezek alatti és a .hu alatti domain név igénylésről részletes tájékoztató olvasható a www.nic.hu web lapokon. A lényeg az, hogy van egy sor szolgáltató, akiknél egyformán be kell tartani néhány formai és technikai szabályt, de a szolgáltatás díját a szolgáltatók önállóan, egymással versenyben állapítják meg.

Az egyik formai szabály, hogy .hu alatt csak olyan nevet lehet regisztrálni, ami az igénylő neve, nevének rövidítése vagy valamilyen védjegye.

Két fő technikai szabály:

1. A bejegyzendő domaint legalább két, független hálózati elérésű domain név szerverrel kell szolgáltatni.
2. A postmaster@domain.hu levelezési címnek – ahogy azt az RFC822⁹ is előírja – működnie kell.

9.1.15. Példák

Néhány példa konfigurációs fájl következik, a sorok között magyarázatokkal. Named.conf fájl csak látó, nem láttató (caching-only) szerver számára:

⁹ <ftp://ftp.sztaki.hu/pub/nic/rfc/rfc822.txt>.Z

```
options {
    directory "/usr/local/named";
    recursion yes;
};

zone "." {
    type hint;
    file "root.hints";
};

zone "0.0.127.in-addr.arpa" {
    type master;
    file "127.0.0";
};
```

Named.conf láttató szerver számára:

```
/*
    Elsődleges a valahol.hu és a 193.111.222
    C osztályú címtartomány inverze számára, másodlagos
    a mas.hu zóna számára. Szükséges fájlok még:
    root.hints - a root szerverek adataival
    valahol.hu - a szükséges NS, A, stn. rekordokkal
    193.111.222 - a szükséges PTR rekordokkal
    127.0.0      - a loopback hálózat számára
*/

options {
    directory "/usr/local/named";
    recursion yes;
    notify yes;
    check-names slave warn;
    check-names master fail;
    allow-transfer {"slaves";};
};

/* csak két címről engedünk meg zónatranszfert: */

acl "slaves" {
    193.222.111.1;
    193.111.222.5;
```

```
};

zone "." {
    type hint;
    file "root.hints";
};

zone "0.0.127.in-addr.arpa" {
    type master;
    file "127.0.0";
};

zone "valahol.hu" {
    type master;
    file "valahol.hu";
};

zone "222.111.193.in-addr.arpa" {
    type master;
    file "193.111.222";
};

zone "mas.hu" {
    type slave;
    masters { 193.111.222.5; }
    file "sec/mas.hu";
};
```

A 0.0.127.in-addr.arpa zónára a következő fájl elegendő:

```
@          SOA      ns.valahol.hu. hostmaster.valahol.hu. (
                                1999091001      ; Serial
                                86400      ; Refresh
                                7200      ; Retry
                                604800      ; Expire
                                86400) ; Minimum TTL
          NS       ns.valahol.hu.

1          PTR     localhost.
```

A valami.hu fájl lehet ilyen:

```

@           SOA      ns.valahol.hu. hostmaster.valahol.hu. (
                        1999091001          ; Serial
                        86400          ; Refresh
                        7200          ; Retry
                        604800         ; Expire
                        86400)         ; Minimum TTL

                        NS      ns.valahol.hu.
                        NS      ns.mas.hu.
ns          MX      10 masina.valahol.hu.
masina      A       193.111.222.1
ns          A       193.111.222.3
www         CNAME   masina

```

A 193.111.222 fájl lehet ilyen:

```

@           SOA      ns.valahol.hu. hostmaster.valahol.hu. (
                        1999091001          ; Serial
                        86400          ; Refresh
                        7200          ; Retry
                        604800         ; Expire
                        86400)         ; Minimum TTL

                        NS      ns.valahol.hu.
                        NS      ns.mas.hu.
1           PTR      ns.valahol.hu.
3           PTR      masina.valahol.hu.

```

9.1.16. Források az interneten

iDNS-sel kapcsolatos RFC-k. Elérhetők például itt: <ftp://ftp.sztaki.hu/pub/nic/rfc>

1. rfc974¹⁰, Mail routing and the domain system
2. rfc1032¹¹, Domain administrators guide
3. rfc1033¹², Domain administrators operations guide
4. rfc1034¹³, Domain names – concepts and facilities

¹⁰<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc974.txt.Z>

¹¹<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1032.txt.Z>

¹²<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1033.txt.Z>

¹³<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1034.txt.Z>

5. rfc1035¹⁴, Domain names – implementation and specification
6. rfc1101¹⁵, DNS encoding of network names and other types
7. rfc1122¹⁶, Requirements for Internet hosts – comm. layers
8. rfc1123¹⁷, Requirements for Internet hosts – application
9. rfc1536¹⁸, Common DNS implementation errors
10. rfc1537¹⁹, Common DNS data file configuration errors
11. rfc1591²⁰, Domain Name System structure and delegation
12. rfc1597²¹, Address allocation for private internets
13. rfc1627²², Network 10 considered harmful
14. rfc1700²³, Assigned numbers
15. rfc1706²⁴, DNS NSAP resource records
16. rfc1712²⁵, DNS encoding of geographical location (GPOS)
17. rfc1713²⁶, Tools for DNS debugging
18. rfc1794²⁷, DNS support for load balancing
19. rfc1876²⁸, Expressing location information in the DNS (LOC)
20. rfc1884²⁹, IP v6 addressing architecture

¹⁴<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1035.txt.Z>

¹⁵<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1101.txt.Z>

¹⁶<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1122.txt.Z>

¹⁷<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1123.txt.Z>

¹⁸<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1536.txt.Z>

¹⁹<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1537.txt.Z>

²⁰<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1591.txt.Z>

²¹<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1597.txt.Z>

²²<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1627.txt.Z>

²³<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1700.txt.Z>

²⁴<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1706.txt.Z>

²⁵<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1712.txt.Z>

²⁶<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1713.txt.Z>

²⁷<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1794.txt.Z>

²⁸<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1876.txt.Z>

²⁹<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1884.txt.Z>

21. rfc1886³⁰, DNS extensions to support IP v6 (AAAA)
22. rfc1912³¹, Common DNS operational and configuration errors
23. rfc1982³², Serial number arithmetic
24. rfc1995³³, Incremental zone transfer in DNS (IXFR)
25. rfc1996³⁴, Prompt notification of zone changes
26. rfc2010³⁵, Operational criteria for root nameservers
27. rfc2052³⁶, Specification of location of services (SRV)
28. rfc2065³⁷, DNS security extensions (KEY/SIG/NXT)
29. rfc2317³⁸ Classless IN-ADDR.ARPA delegation

News csoportok:

1. comp.protocols.tcp-ip.bind³⁹ news csoport Ennek FAQ-ja:
<http://www.intac.com/cdp/cptd-faq>
2. comp.protocols.dns.bind⁴⁰

Ez a news csoport a bind-users@isc.org lista tükörképe. A lista archívuma:
<http://www.isc.org/ml-archives/bind-users/index.html>

A bind lapjai: <http://www.isc.org>

A Bind Operations Guide, a BOG. Része a Bind disztribúciónak, de letölthető külön is:
<http://www.dns.net/dnsrd/docs/bog>

DNS-sel kapcsolatos információk: <http://www.dns.net>

Kitűnő és olvasmányos könyv a DNS-ről: Cricket Liu & Paul Albitz: DNS and BIND
<<http://www.oreilly.com/catalog/dns3/>> O'Reilly kiadó <<http://www.ora.com>>

A .hu alatti regisztrálásról: <http://www.nic.hu>

A linux disztribúciók DNS HOWTO-ja minden disztribúciónak része. Letölthető pl.
innen: <ftp://ftp.kfki.hu/pub/linux/sunsite.unc.edu/docs/HOWTO/DNS-HOWTO>

³⁰<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1886.txt.Z>

³¹<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1912.txt.Z>

³²<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1982.txt.Z>

³³<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1995.txt.Z>

³⁴<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc1996.txt.Z>

³⁵<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2010.txt.Z>

³⁶<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2052.txt.Z>

³⁷<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2065.txt.Z>

³⁸<ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2317.txt.Z>

³⁹<news:///comp.protocols.tcp-ip.bind>

⁴⁰<news:///comp.protocols.dns.bind>

9.2. A levelezés alapjai

9.2.1. A szereplők

Kliensek

A levelezésben kulcsfontosságú szerep a klienseké, hiszen nélkülük nem lenne értelme a levelezésnek. A mostani anyagban azonban kevesebbet foglalkozunk velük, a kiszolgálókra koncentrálunk. Azt azonban érdemes tudni, hogy a kliens programok nem közvetlenül, hanem legalább egy közbelső kiszolgáló által kapják meg a leveleket. A kliensek csak a kiszolgálók által átvett levelet érik el különféle módokon - helyi olvasás, letöltés saját gépre - és ugyanígy, nem közvetlenül egymásnak küldik a levelet, hanem annak a kiszolgálónak - programnak -, mely a címzett leveleiért felelős. Elképzelhető, hogy nem közvetlenül a címzett levelező kiszolgálójának, hanem a saját kiszolgálójának küldi a levelet és arra bízza rá, hogy elküldje azt a címzettnek. Azt, hogy hogyan találják meg a címzett kiszolgálóját, nemsokára megtudjuk. Addig azonban nézzük át a kiszolgálók szerepét és feladatait.

Kiszolgálók

A kiszolgálók felelősek az általuk kezelet e-mail címeknek szánt levelek fogadásáért és az engedélyezett számítógépekről rajtuk keresztül küldött levelek elküldéséért, esetleg a nem általa kezelt domének számára tartalék kiszolgálóként történő működés lehet még feladatuk. A kiszolgálók feladatai kibővíthetők a víruskereséssel, spam levelek - kéretlen reklám e-mailek - kiszűrésével, azonban ezek a feladatok nem tekinthetők a klasszikusan levelező kiszolgálónak - angolul MTA - szánt alkalmazások céljainak. Erre csak később, az új kihívások legyőzésének megkönnyítése céljából került sor.

Fogalmak

Felmerül a kérdés, hogyan és milyen módon találják meg a kiszolgálók illetve a kliensek a többi kiszolgálót. Ha ismerjük a DNS felépítését, akkor már valószínűleg tudjuk a választ, mely röviden: MX rekord. Nézzük át bővebben, mit is takar ez.

MX rekord. Egy adott domén leveleit annak elsődleges levelező kiszolgálója fogadja, melyet a DNS-ben *MX* rekorddal jelölnek. Egy doménhez több *MX* rekord is tartozhat. Ezek között a sorrendet az dönti el, hogy melyik *MX* rekordhoz tartozik kisebb szám. Példaként nézzük meg a fiktív nevű kukuriku.hu domén esetét. Ehhez a doménhez tartozzon két *MX* bejegyzés:

```
MX 10 mail.kukuriku.hu  
MX 20 mail.szolgáltato.hu
```

Ekkor a levelek minden esetben először a mail.kukuriku.hu kiszolgálóhoz fognak menni, mert az rendelkezik kisebb számmal, azaz nagyobb preferenciával.

Tartalék MX kiszolgáló. Ha azonban a levél nem jut el a megfelelő kiszolgálóra, felmerül a kérdés, hogy mi történik? Ekkor van lehetősége a küldő kiszolgálónak a kisebb preferenciával rendelkező gép felé küldeni a leveleket. Ez a gép a tartalék MX gép, melyből több is lehet, ha igazán biztosra akarunk menni. Ez a kiszolgáló természetesen nem tudja, hogy melyik levél kinek szól és nem kézbesíti azt a felhasználó postafiókjába, hanem egyszerűen csak gyűjti a leveleket, arra várva, hogy az elsődleges levelezési kiszolgáló újra működőképes illetve hálózati kommunikációra alkalmas legyen és elkérje tőle a begyűjtött leveleket. Ennek módszerét később ismertetjük.

Használt protokollok

9.2.1. SMTP

Ismertetés

A levelezés alapját jelentő protokollt 1982-ben egységesítették és kapott RFC számot (821-es RFC). Az SMTP egy szöveg alapú, utasításokkal és hibakódokkal működő protokoll. Működése a következőképpen írható le. Egy küldő SMTP kapcsolatot létesít egy fogadóval. A parancsokat csak a küldő adhatja, melyre csak a fogadó adhat SMTP-nek megfelelő válaszokat. A folyamat elején nem kötelező jelleggel a küldő kiadhatja a *HELO* parancsot, melyben közli a nevét. Majd a *MAIL* paranccsal továbbítja a küldő nevét. Ha a fogadó elfogadja a levelet erről a címről, akkor egy *OK* választ küld vissza. Ekkor a küldő elküldi a címzett nevét az *RCPT* parancs segítségével. Ha a címzett megfelelő, azaz a kiszolgáló felel az adott e-mail cím leveleinek fogadásáért vagy tartalék kiszolgálóként fogadja a domén leveleit, akkor szintén *OK* választ ad vissza, egyébként visszautasítja a levél fogadását a megadott címre. A küldő megadhat másik címet is, mert a fogadó nem az egész folyamatot, csak ezt az elemét utasította el. A leveleknél több címzett is megadható. Ha az összes címzettet közölte a küldő és a fogadó *OK* válasszal tért vissza, elkezdődik az adatcsomag, melyet a *DATA* paranccsal közöl a küldő kiszolgáló a fogadóval. Figyelem! A levél olvasásakor látható Címzett, Feladó, Tárgy, Dátum, Másolat mezők is itt kerülnek "kitöltésre", azaz itt adja meg a küldő kiszolgáló. (Természetesen olyan formában, ahogy azt a kliens számára megadta.)

Az adatok átküldésének végét és a teljes tranzakció befejezését egy, a sorban csak egymagában szereplő "." - pont - jel zárja le. Ha minden rendben történt, akkor a fogadó

még küld egy OK-t és a fogadott levél helyi azonosítóját, amilyen néven a helyi levél-továbbítási sorba bekerült a levél. Ha a küldő nem akar több levelet küldeni, akkor a *QUIT* paranccsal lezárja a kapcsolatot a fogadóval.

Parancsok

A legfontosabb illetve legismertebb parancsok listája a következő:

HELO A küldő kiszolgáló azonosítására szolgáló parancs. Csak a küldő kiszolgáló nevét tartalmazza.

VRFY A küldő megerősítést kér a fogadótól, létezik-e a címzett. Ma már nem használják, mert sok spam küldő így szerezte meg az e-mail címeket.

EXPN A parancs segítségével kéri meg a küldő a fogadót, hogy igazolja, levelezőlistára küld-e levelet. A válasz visszaadja a címek listáját. A parancs az alias fájlokban is keres. Például megtudható a root felhasználó levelei kinek érkeznek meg, azaz ki a root felhasználó. Ezért ezt az opciót sem használják már éles szervereken.

MAIL A levél küldés elején szereplő parancs. Használt formája a MAIL FROM <email@domain.hu>. Ez a cím eltérhet a levélben szereplő From: (Feladó:) mezőtől, ha teljes levelet megnézzük, akkor ez a levél elején szereplő From sorban található meg.

RCPT A levél címzetjét adja meg. Használt formája az RCPT TO <email@domain.hu>. Ez a cím eltérhet a levélben szereplő To: (Címzett:) mezőtől.

RSET Ha a küldést bármilyen okból megszeretnénk szakítani és újra szeretnénk indítani a kapcsolatot egészen a HELO parancsig, akkor az RSET parancsot kell közölni a fogadó kiszolgálóval.

DATA Ennek a parancsnak a kiadása után a fogadó úgy kezeli az ezután érkező sorokat, mint a levelet magát. Az adatok küldésének befejezését egy olyan sor jelzi, melyben csak egy darab "." - pont - jel szerepel.

HELP A HELP PARANCSNÉV egy rövid, hasznos összefoglalót ad a kért parancsról.

NOOP a no operation kifejezés rövidítése. Semmilyen hatása nincs a küldési folyamatra, csak a fogadó küld egy OK választ a küldőnek. Kapcsolatok tesztelésére használatos.

TURN Ma már szintén nem használatos, ma már az ETRN parancsot használják helyette.

QUIT A kapcsolatot zárja le a küldő és a fogadó kiszolgáló között.

Válaszok

500 Szintaxis hiba, ismeretlen parancs

501 Szintaxis hiba a paraméterekben

502 A parancs nincs implementálva

503 Rossz sorrendben lettek kiadva a parancsok

504 A parancs paramétere nincs implementálva

211 A rendszer állapota vagy a rendszer súgójának válasza

214 Help üzenet

220 <domain> A szolgáltatás készen áll

221 <domain> A szolgáltatás lezárja az átviteli kapcsolatot

421 <domain> A szolgáltatás nem elérhető, átviteli csatorna lezárása

250 A kérés rendben van, teljesítve

251 Nem helyi felhasználó, a levél nem lesz továbbítva

450 A kérés nem lett végrehajtva

550 Kérés nem lett végrehajtva

451 A kért tevékenység meg lett szakítva, hiba a végrehajtásban

551 Nem helyi felhasználó, próbáld a <címet>

452 Kért tevékenység nem lett végrehajtva, nincs elegendő tárolóhely (megtelt a me-revlemez)

552 Kért tevékenység nem lett végrehajtva: a rendelkezésre álló terület elfogyott

553 Kért tevékenység nem lett végrehajtva, mailbox név nem engedélyezett

354 Kezdődhet a levél Start mail input; end with <CRLF>.<CRLF>

554 Sikertelen küldés

9.2.2. ESMTP

Ismertetés

Az időközben megjelenő kihívások az *SMTP* protokollt is érintették, ezért létrejött az *ESMTP*, ami egy olyan keretrendszer, melyet az *SMTP* kiegészítésére hoztak létre, már több kiegészítés is létezik hozzá. Most a parancsokon keresztül nézzük át a legfontosabb változásokat.

Parancsok

EHLO Minden ESMTP képes küldőnek és fogadónak ezt kell használnia a **HELO** parancs helyett. A köszöntés után a fogadónak ki kell listáznia milyen szolgáltatásokat és kiegészítéseket nyújt a küldőnek.

ETRN A **TURN** parancs helyett vezették be. Ha tartalék levelező kiszolgáló gyűjtötte össze a leveleinket vagy olyan szolgáltatást bérlünk, mely a leveleinket gyűjti egy szerveren, ahonnan időnként letöltjük, akkor azt az **ETRN** parancs segítségével tudjuk megtenni. A parancsot "ETRN doménnev" vagy "ETRN hostnév" paraméterekkel adjuk ki. Biztonsági megfontolásokból nem ugyanazt a már kiépített adatkapcsolatot használja, mint a **TURN** parancs - ezért is tekintenek el ennek használatától -, hanem egy új kapcsolatot épít fel a fogadó, az új kapcsolattal küldőként megjelenve. Küldőként megnézi, hogy az adott domént kezelő kiszolgálónak mi az IP címe és neki próbálja elküldeni azokat a leveleket, melyek a doménnek szólnak.

AUTH Miután egyre több a mobil felhasználó, akik bárholnan szeretnék elküldeni leveleiket, de csak a céges kiszolgálót szeretnék - vagy szabad - leveleik elküldésére használni. Ezt természetesen nem szabad engedélyezni, hiszen ha bárki bárholnan küldhetne levelet, akkor hamar rajtunk keresztül küldenének több ezer, esetleg több tízezer spam levelet. Ezért alkották meg az **AUTH** bővítményt, melynek segítségével a felhasználók felhasználónév és jelszó segítségével azonosítják magukat a kiszolgálónak, majd ezek után az adott IP címről küldhet levelet az adott tranzakció keretében.

Gyakorlati példa: Postfix

9.2.1. Fordítás forrásból

Az egyik legjobb levelező kiszolgáló a postfix, melyet gyorsasága, biztonságossága és nagy fokú konfigurálhatósága emel ki a többi közül. Amennyiben `dpkg/apt` vagy `rpm`

alapú rendszert használunk, biztosan találunk csomagot belőle. Amennyiben mégis forrásból szeretnénk telepíteni, a 1.1+<http://www.postfix.org>+ oldalról letölthetjük a legutolsó stabil változatot.

Egyszerű kiszolgáló

Egyszerű kiszolgáló alatt azt értjük, hogy a klasszikus passwd és/vagy shadow fájlokban tároljuk a felhasználók neveit és jelszavait, nem pedig LDAP vagy MySQL adatbázisokban. A kitömörített forráskódot tartalmazó könyvtárban adjuk ki a következő parancsokat:

```
make tidy
make
```

Az első sor az esetlegesen ott lévő másik fordításból származó fájlokat törli, míg a második sor lefordítja a postfixet az alapértelmezett kapcsolókkal. Ha készen van, akkor a make install parancs kiadása előtt érdemes megkeresni és strip-pelni a futtatható bináris fájlokat, ugyanis ezzel nagy mértékben csökken méretük. A strip parancs a debug üzeneteket veszi ki a bináris állományokból, melyek nem szükségesek a program működéséhez. Ezt a strip fájlnev parancs segítségével tehetjük meg, mely könnyen szkriptelhető. Ezután adjuk ki a make install parancsot és értelemszerűen válaszoljunk a feltett kérdésekre. A továbbiakban feltételezni fogjuk, hogy az alapértelmezésként felajánlott opciókat fogadjuk el, melyek megegyeznek egy csomagból telepített rendszer paramétereivel.

LDAP alapú kiszolgáló

LDAP támogatással rendelkező postfix fordításához szükségünk van az OpenLDAP fejlesztői fájljaira és függvénykönyvtáira. Fordítás előtt a következőképpen kell előkészíteni a Makefile-t:

```
make -f Makefile.init makefiles \
'CCARGS=-DHAS_LDAP .I/usr/include' \
'AUXLIBS=-L/usr/lib -ldap -L/usr/lib/ -llber'
```

(Feltételeztük, hogy a fejlesztői fájlok és a függvénykönyvtárak a fent megadott helyre települtek.)

Majd a szokásos make és make install páros, előtte természetesen a szükséges binárisok strip-pelésével.

A konfigurációs paraméterek beállításához nézzük át a postfix csomagjában a README fájlok közé tartozó LDAP_README-t.

MySQL alapú kiszolgáló

Amennyiben MySQL-ben szeretnénk tárolni a felhasználók adatait, akkor szükségünk lesz a MySQL és a libz fejlesztői fájljaira valamint a függvénykönyvtárakra.

A make tidy után a következőképpen kell kiadni a make parancsot:

```
make -f Makefile.init makefiles \  
'CCARGS=-DHAS_MYSQL -I/usr/include/mysql' \  
'AUXLIBS=-L/usr/lib/ -lmysqlclient -lz -lm'
```

(Feltételeztük, hogy a fejlesztői fájlok és a függvénykönyvtárak a fent megadott helyre települtek.)

Majd adjuk ki a make és make install parancsokat, előtte természetesen strip-pelhetjük a szükséges fájlokat.

A konfigurációs fájlban ezután, mysql: előtaggal hivatkozhatunk a MySQL alapú tárolókra, például:

```
alias_maps = mysql:/etc/postfix/mysql-aliases.cf
```

Az adatbázis elérést szolgáló fájl beállításait a postfix forrásával szállított README fájlok között megtaláljuk, MYSQL_README néven. Ez foglalkozik a fordítással is, érdemes megnézni, változott-e legutolsó fordításunk óta a tartalma.

9.2.2. Alapvető beállítások

Ahhoz, hogy leveleket küldjünk és fogadjunk, csak pár alapvető beállítás szükséges.

A konfigurációs fájlok

A két legfontosabb konfigurációs fájl amit postfix alatt használunk, a következő:

- master.cf
- main.cf

Általános telepítés esetén mindkettő a /etc/postfix könyvtár alatt található.

A master.cf feladata, hogy a postfix egészét átfogó és irányító master démon számára a modulok futtatási paramétereit megadja. Itt kerül beállításra, hogy egy modul mekkora erőforrást használhat fel, milyen módon fusson, milyen néven hivatkozhatunk rá a main.cf fájlban.

A `main.cf` a modulok viselkedését befolyásolja, ennek segítségével állíthatóak be a futás idejű viselkedés paraméterei minden modul számára. Ez alatt azt értjük, hogy a leveleket fogadó modul innen tudja meg, hogy kitől tilos levelet fogadnia, mely alhálózatokról érkezhetsz hozzá levél, mely domén(ek) számára fogadunk levelet stb. Egy paraméter későbbi értékeire a `$paraméter_neve` formában hivatkozhatunk, például:

```
myorigin = $mydomain
```

A modulok részletes beállításait a `sample-modulneve.cf` fájlok tartalmazzák, igaz angolul.

Levelek fogadása

Az első lépés, hogy megmondjuk, mely domének számára fogadjuk a leveleket és milyen névre hallgat kiszolgáló gépünk. Ezt a következő opciók beállításával érjük el:

myhostname = gep.domain.hu a számítógép neve, ezt adja vissza az *EHLO/HELO* parancs fogadása után és ezt adja meg, ha kapcsolatot kezdeményez. Később többször hivatkozunk majd még erre a beállítási lehetőségre.

mydomain = domain.hu a domén melyet kezelünk, alapértelmezésként a `myhostname` első tagja nélküli szöveg

myorigin = \$mydomain A lokálisan elküldött levelek milyen doménű feladóval menjenek el gépünkről.

inet_interfaces = all mely hálózati kapcsolaton fogadunk leveleket, alapértelmezett az `all`, de beállíthatjuk akár csak az egyik hálózati kártyát, a `localhost`ot vagy bármelyik hálózati kapcsolatot, mely rendelkezésünkre áll.

mydestination = \$myhostname, localhost.\$mydomain Látható, hogy itt is hivatkozunk már korábban beállított értékekre.

`$mydomain` - mely domének és e-mail címek számára fogadunk leveleket.

Ebben a beállításban elfogadjuk, hogy levél érkezzon a teljes internetes nevünkre (*FQDN* - full qualified domain name), domén nevünkre és a `localhost.domén` nevünkre. Jelen esetben: `gep.domain.hu`, `domain.hu`, `localhost.domain.hu`. A `localhost` beállítása fontos, ha kifelejtjük problémánk lehetnek! Ha a kiszolgáló több géppel is rendelkezik, például ez a web szerver is, és fogad leveleket a `www.domain.hu` címen, akkor be kell írni a `www.$mydomain`-t is.

local_recipient_maps = \$alias_maps unix:passwd.byname Beállítható, hogy a rendszer hol találja a meg az érvényes felhasználó neveit. Jelen esetben a UNIX típusú `/etc/passwd` fájlban és az úgynevezett alias táblákban.

```
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
```

A fentiek adják meg, hogy hol tartjuk az alias fájlokat, mely segítségével egy felhasználónak több címe is lehet, például a maris nevű felhasználó szeretne olyan e-mail címet, hogy `Kis.Maris@domain.hu` (elnézést minden Kis Maritól), akkor csak fel kell őt venni az alias táblába a következő módon:

```
Kis.Maris : maris
```

Majd ki kell adni a `newaliases` parancsot, ami legyártja a postfix által is használható adatbázist.

Levelek küldése

mynetworks = 10.0.0.0/24, 127.0.0.0/8 Megadjuk, hogy mely alhálózatról fogadunk el leveleket továbbításra, azaz relay-ezünk. Itt fontos, hogy a localhost címe (`127.0.0.0/8`) is benne legyen.

relayhost = mail.szolgaltato.hu Ha nem közvetlenül mi küldjük a leveleket, hanem csak szolgáltatónk szerverén keresztül tudjuk azt kiküldeni, akkor ezen paraméteren keresztül kell beállítani.

Vírusirtás

Vírusirtásnál az amavis nevű eszközt fogjuk használni, mely sok féle vírusirtóval használható. Telepítéséhez nézzük át az amavis vagy disztribúciónk dokumentációját. Az amavishoz kétféleképpen is beállítható, démonként és csak SMTP kapcsolatokhoz rendelve. Ezek a beállítások csak a postfix-szel kapcsolatos beállításokat tükrözik, az Amavis vírusirtóval történő együttműködéséhez annak dokumentációját nézzük át!

Minden levél átvizsgálása a megadott fájlokban a következő változtatások szükségesek: A `main.cf`-ben állítsuk be szűrőprogramnak az Amavis-t:

```
content_filter = smtp:localhost:10025
```

A `master.cf` egy újabb sorral gazdagodik:

```
localhost:10026 inet n - - - smtpd -o content_filter
```

Az *Amavis* konfigurációjánál - `amavis.conf` - pedig az alábbi részeket kell beállítani:

```
[global]
mail-transfer-agent = SMTP
```

```
[SMTP]
```

```
input address = localhost
input port = 10025
output address = localhost
output port = 10026
```

Csak SMTP-n érkező levelek szűréséhez az alábbi beállítások szükségesek.

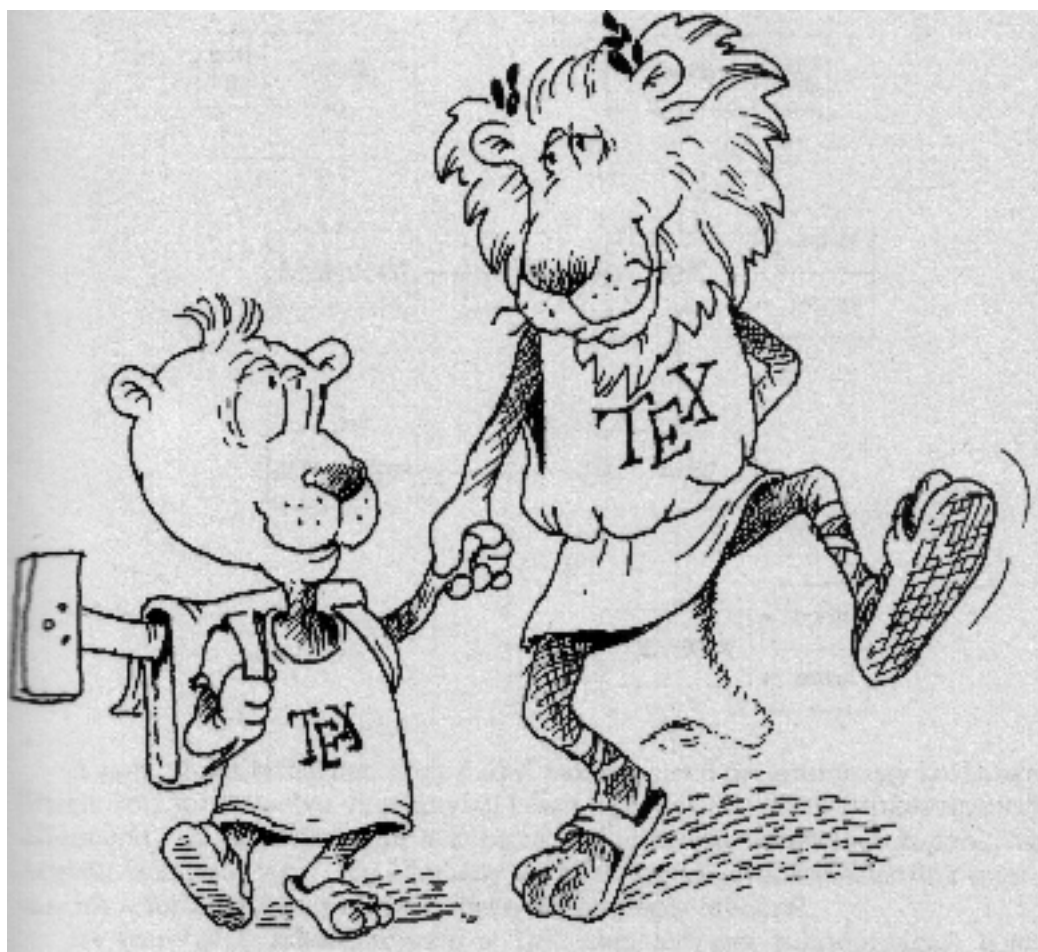
Vegyünk fel egy új szolgáltatást a `master.cf`-be és állítsuk át az `smtpd` modul beállítását:

```
filter unix - n n - - pipe flags=Rq user=mail
argv=/usr/bin/amavis ${sender} -- ${recipient}
```

```
smtp inet n - - - smtpd -o content_filter=filter:
```

10. fejezet

A Kernel



E fejezet bemutatja a munkát a Linux kernelével. Nyelvezetét tekintve nem egy általános célú felhasználásra van tervezve, hanem legfőképpen hozzáértőknek, hogy lássák, mikor mit kell hogy elmondjanak.

10.1. A kernel szerepe

A modern operációs rendszerek több szintből épülnek fel. Minden szintnek megvan a maga szerepe. Az egyik a grafikus felhasználói felületről (GUI) gondoskodik, a másik a lemezhozzáférést biztosítja. De az alapvető szint, ami nélkül az operációs rendszered el sem indul, na őt hívjuk kernelnek. A kernelben találhatók többek közt az alacsony szintű szolgáltatások, amilyen a memória-menedzsment, hardver összefüggések kezelése stb.

A Unix kernel a közvetítő a programok és a hardver között. Menedzseli a memóriát az összes futó program (processz) számára és biztosítja, hogy mind egyenlően (vagy egyenlőtlenül, ha úgy tetszik) részesüljenek a processzor ciklusaiból. Továbbá egy kényelmes, hordozható felületet nyújt a programoknak, amelyen át kommunikálhatnak a hardverrel.

Ennél persze jóval több szót érdemelne a kernel működése, de ezeket az alapvető funkciókat a legfontosabb ismerni.

10.2. A Linux kernel története

A Linux születése, csecsemőkora (0.01-0.10)

A Linux fejlesztésének kezdetén Linus Torvalds a 80386 processzor védett módú (protected mode), feladat-váltó (task-switching) lehetőségeivel szeretett volna megismerkedni. Ez kb. 1991 nyarának elején lehetett. A pontos dátumra maga a szerző sem emlékszik, de amikor egyszer megkérdezték, mikor van a Linux születésnapja, azt mondta, hogy nem tudja megmondani, de egy e-mail tanúsága szerint július 3-án már a POSIX szabvány után érdeklődött, így akkor már biztos futott az alaprendszer.

A program fejlesztése egy korábbi PC-s UNIX, a Minix alatt történt, eleinte assembly-ben. Az első fázisban kialakuló 0.01-es változat még semmire sem volt használható, csak egy lépcső volt a továbbfejlesztéshez.

Amikor Linus áttért a C nyelvre, a fejlesztés lényegesen gyorsabbá vált, és olyan nagyratörő tervek fogalmazódtak meg, hogy valaha le lehessen fordítani a GNU C fordítóját Linux alatt. (Ma már csak csodálkozni lehet azon, hogy 1991-ben ez volt a nagy álom, és azóta hol tart a rendszer.)

Ez a legelső változat még nem volt használható: csak Minix alatt lehetett lefordítani, és semmi hasznos funkciója nem volt azon kívül, hogy írója megismerkedett a processzorral.

1991. október 5-én hirdette meg Linus az első "hivatalos", 0.02-es Linux-ot az Interneten. Ekkor már néhány alapprogram futott a rendszeren (pl. a GNU "gcc" nevű C fordítója, valamint a "bash" burokprogram), így már el lehetett kezdeni használni a rendszert. Ekkor nem is a rendszer használhatóságának növelése volt a fő cél, hanem a rendszer-mag fejlesztése. Ezért ekkor nem készültek dokumentációk, installációs csomagok, stb. A Linux ekkor még csak a megszállott hackereknek készült.

Linus ekkor elhatározta, hogy az Interneten keresztül bevonja a fejlesztésbe a szabad kapacitással rendelkező programozókat, és egy mára elhíresült hirdetményt tette közzé a comp.os.minix hírcsoportban:

"Sóvárogsz a Minix-1.1 szép napjai után, amikor a férfiak igazi férfiak voltak, és mindenki maga írta a saját eszközmeghajtóját? Egy szép projekt nélkül vagy, és épp fened a fogad egy operációs rendszerre, amit igényeidnek megfelelően alakíthatsz? Frusztrálónak találsz, ha minden működik Minix alatt? ...

Akkor ez a levél lehet hogy pont neked szól.

Ahogy egy hónapja említettem, egy szabad Minix-szerűségen dolgozom AT-386 számítógépre. Végül is elérkeztem egy olyan állapotra, amikor ez egyáltalán használható (bár ez függhet attól, mit akarsz), és a program forráskódját szélesebb körben tervezem szétosztani. Ez még csak a 0.02-es változat, de sikeresen futtattam a bash, gcc, gnu-make, gnu-sed, compress, stb. programokat alatta."

Megjegyzendő, hogy ekkor, és még egy darabig a Linux erősen kapcsolódott a Minix-hez: önállóan nem is létezett, csak alatta lehetett lefordítani, futtatni, továbbá az Interneten is a Minix hírcsoportjában folyt a Linux-ról szóló tárgyalás.

A 0.03-as verzió két-három hét alatt megszületett, majd 1991. decemberében Linus kibocsátotta a 0.10-eset is. Ez az ugrás a számozásban azt tükrözte, hogy jelentősen megnőtt a Linux alatt futtatható alkalmazások száma, de a Linux még mindig nem volt önálló, szerzője szerint is "egy hacker által hackereknek írt" rendszerről van szó, így a rendszernek csak fejlesztői vannak, felhasználói nem.

A Linux gyermekkora (0.11-0.99)

1991. december 19-től, a 0.11-es változat kibocsátásától számíthatjuk a Linux gyermekkorát. Ez volt az első önálló rendszer, tehát nem kellett Minix a használatához. Sok

olyan tulajdonsággal rendelkezett, amely jelezte, hogy itt valami komoly készül. Ezeket Linus felsorolásában adjuk közre:

A 0.11-nek a következő újdonságai vannak:

- demand loading
- kód és adatmegosztás nem kapcsolódó processzek közt
- sokkal jobb floppy-vezérlők (most már többnyire működnek)
- hibajavítások
- Hercules/MDA/CGA/EGA/VGA támogatás
- a konzol hangot is ad (Óh! Fantasztikus rendszermag!)
- mkfs/fsck/fdisk (fájlrendszer karbantartó programok)
- amerikai/német/francia/finn billentyűzet
- a com1/2 sebessége beállítható

A 0.12-es változat 1992. január 15-én látott napvilágot, néhány bővítéssel: Már volt init/login szolgáltatás (nem root-ként kellett először bejelentkezni, és inicializálni a rendszert), közeledett a POSIX szabványhoz, virtuális memóriát is használt és kisebb korrekciókat tartalmazott.

Ez egy elég stabil változat lett, ekkortól kezdődött el a Linux igazi hódítása. A 0.12-es Linuxot ugyanis elkezdtek használni “egyszerű” felhasználók is, nemcsak megszállottak.

Szintén ehhez a változathoz kapcsolódik a Linux fejlesztésének kiszélesedése: a 0.12-es már lényeges részeket tartalmazott, melyeket nem Linus Torvalds írt. Pl. a “job control”-t Theodore Ts’o, a virtuális konzolokat Peter MacDonald programozta.

Az így előálló rendszer már több vonatkozásban jobb volt a Minix-nél, de még mindig nem volt látható, hogy ebből akkora mozgalom lesz, mint ami mára kialakult.

A sikeren felbuzdulva a verziószám hirtelen ugrott: a 0.95-ös 1992. márciusában, a 0.96 áprilisban lett kibocsátva. Ekkortól kezdve hihetetlen ütemben gyarapodott a Linux-felhasználók és -programozók száma.

Ekkor a verziószám hirtelen “befékezett”. 1993. decemberében a verziószám 0.99pl14 volt. (A pl14 a “patch level 14” rövidítése, azaz ez a 14. javított változat.) Bár a 0.95-ös verziótól kezdve a szolgáltatások száma, a megbízhatóság, és sok egyéb szempontból jelentős javulás következett be, és hihetetlenül sokan használták ezeket a rendszermagokat, az 1.0 verziószámot csak akkor merték kiadni (1994. elején), amikor a POSIX szabvánnyal való kompatibilitás kielégítővé vált.

A 0.95-0.99 rendszermagra épülő rendszereknek óriási népszerűségük volt. Egyes egyetemeken, pontosabban azok bizonyos intézeteiben gyakorlatilag likvidálták az összes nem Linux-os programot, és a PC-ken nem lehetett DOS-t vagy Windows-t találni. (Legfeljebb a titkárságokon.) Ez főleg olyan helyeken következett be, ahol a kutatók UNIX alatt dolgoztak, mert egy Linux-os PC-n otthon is fejleszthették a programjukat, és ezeket egyszerű volt az intézet nagykapacitású gépeire áttenni.

Hazánkban ekkortájt (1993) kezdett igazán elterjedni a Linux, mert ekkorra kötötték be a felsőoktatási intézmények nagy részét az Internetbe, így sokaknak megnyílt a lehetősége a Linux beszerzésére.

A Linux és a Minix szétválását meggyorsította, hogy a Minix szerzője, Andrew Tannenbaum nem nézte jó szemmel a Linux-ot. Alapvetően elhibázottnak tartotta a Linux rendszermag típusát; Linus Torvalds ugyanis ún. "monolitikus kernelt" írt, míg Andrew Tannenbaum (elméleti megfontolások alapján) a "mikrokernelt" jobb választásnak tartotta. Sajnos, a vitába személyes elemek is keveredtek, és a vita jórészt az Internet hírcsoportjaiban zajlott, meglehetősen nagy nyilvánosság előtt. Így kissé ideges hangulatban zajlott le a Minix és a Linux szétválása.

A Linux fiatalkora (1.0.0-1.2.13)

A POSIX szabványosítás megfelelő szintű elérésével 1994. márciusában megjelent az 1.0.0 sorszámú kernel. Ekkortól kezdve egy speciális sorszámozási eljárást vezettek be a fejlesztők: A verziószámot három, ponttal elválasztott nemnegatív egész jelzi. Az első a fő verziószám, ami csak a rendszermag lényegét érintő változásoknál vált eggyel nagyobbra. A második szám elég speciális jelentésű: ha páros, akkor stabil, tesztelt kernelről van szó, amit bárkinek ajánlanak használatra, míg a páratlan szám tesztváltozatot jelöl, amit inkább azoknak javasolnak, akik tesztelni, fejleszteni szeretnék a kernelt, és akiknek nem számít, ha a rendszer néha "elszáll". A harmadik szám pedig kisebb módosításokkor ugrik egyet.

Ennek megfelelően egyszerre két szálon fut a legújabb verziószám: pl. a legfrissebb két kernel sorszáma 2.4.21 illetve 2.5.76. A stabil verziókba csak olyan modulok kerülhetnek bele, amelyek a fejlesztői változatokban már üzembiztosnak bizonyultak.

Ez a fura sorszámozás lehetővé teszi, hogy az egyszerű felhasználók csak a valóban használható változatokat kapják meg, de közben az esetleg még hibákat tartalmazó fejlesztői változatok is hozzáférhetőek legyenek. A későbbiekben mi csak a stabil verziószámokkal foglalkozunk, mert ezek a "hivatalos" változatok.

Ezen változatok nem hoztak újabb hatalmas áttörést, mert az már korábban bekövetkezett. A fejlesztés során a rendszermag egyre hatékonyabb lett, beépítették a legújabb hardverek meghajtóit (CD-olvasók, PCI-buszok, újabb SCSI-eszközök, stb.). Talán leginkább az 1.2.x-es kernelekkel bevezetett új végrehajtható programformátum, az ELF megjelenését kell itt megemlíteni.

Ebben az időben a Linux alatti felhasználói programok száma nőtt meg hihetetlenül. Míg korábban főleg már meglevő szoftvereket vittek át a Linux alá, addig ekkor már megjelennek azok a programok, melyeket Linux alatt fejlesztenek, és innét viszik át a többi rendszerre. Ekkor már nagy szoftvercégek is elkészítik programjaik Linux-os változatát (pl. Maple V, Motif 2.0). Ezek természetesen nem szabadterjesztésűek, és az, hogy megéri Linux-ra is adaptálni őket, egyértelműen jelzi, hogy a Linux-felhasználók tábora világméretekben is piacot jelent.

Ekkortájt egyre több Linux-terjesztés (disztribúció) kezd megjelenni, azaz több cég olyan programcsomagot állít elő, amelyekkel a Linux telepítése, karbantartása sokkal könnyebb, mintha mindenki egyenként gyűjtené be a rendszer részeit.

Napjaink Linux-a (2.0.0-2.4.21)

1996. augusztusában jelent meg a 2.0.0 sorszámú rendszermag. Ennek fő újítása a modulok megjelenése volt: a kernel bizonyos részei modulként is elkészíthetők, és ezek a modulok akár automatikusan, akár kézzel betölthetők a memóriába, ahonnan a rendszer kipakolja őket, ha régóta nem használjuk.

Például a nyomtató, floppy-vezérlő, nem Linux-os fájlrendszereket kezelő részeket célszerű modulba tenni, mert ekkor ezek csak addig foglalják a memóriát, amíg éppen használjuk őket, és ez többnyire a munkaidő jelentéktelen része. Ezzel az az érdekes helyzet állt elő, hogy a rendszermag memóriaigénye kisebb lett, míg hatékonysága és megbízhatósága megnőtt.

A 2.2.0 -ás kernelt 1999. január 26.-án adta ki Linus, jelentősen átdolgozták a memória lekezelést, a vezérlők felépítését, és még sorolhatnánk, ugyanis ekkortájt a fejlesztés olyannyira kiszélesedett, és felgyorsult, hogy képtelenség összefoglalni még a főbb változtatásokat is. Ma a 2.2.x-es fát Alan Cox tarja karban, és nagyon ritkán jelenik meg új verziója.

A 2.4.0 -ás kernelt 2001. január 4.-én bocsátották útjára, az egyik legfontosabb újítás a hálózati algoritmusok teljes újraírása volt. A 2.4-es fa fejlesztését Marcello Tosatti, egy alig 20 éves fiatalember vezeti.

A következő stabil verzió a 2.6.0-ás lesz, és most januárra várható. Nagyon sok fontos dolgot újraírtak benne, legfőképpen a kernel legfontosabb részeit (pl. memóriakezelés, ütemező algoritmusok), emiatt egy kisebb vita is volt, hogy ne 2.6.0-nak, hanem 3.0.0-ának nevezzék majd, de végül is Linus a 2.6 mellett döntött.

Akit érdekel a Linux kernel fejlődése, az nyomon követheti a heti rendszerességgel megjelenő Linux Weekly News (<http://lwn.net/>) Weekly Edition, illetve a szintén heti rendszerességgel megjelenő Kernel Traffic (<http://kt.zork.net/kernel-traffic/latest.html>) oldalakon. A régebbi verziókról a LinuxHQ -n (<http://www.linuxhq.com/>) lehet tájékozódni.

10.3. Kernel fordítás

Jogosan tehetjük fel a kérdést, hogy ha a kernel olyan fontos szerepet tölt be, mért kell hozzányúlnunk. Nos, az újabb kernelek általában többféle hardverrel tudnak együttműködni, mint a régebbiek (azaz több eszközmeghajtójuk van). Lehet, hogy az új kernelnek jobb a processz-kezelése, gyorsabb vagy stabilabb, mint a régi verzió, és már ki van benne javítva az előző ismert hibája. A legtöbben az eszközmeghajtók és a kijavított hibák miatt frissítenek kernelt.

Előkészületek

A kernel lefordításához szükségünk lesz a legfrissebb stabil kernel (jelen pillanatban ez a 2.4.21) forráskódjára. Ahonnan érdemes kezdeni:

```
http://www.kernel.org/pub/linux/kernel/  
ftp://ftp.hu.kernel.org/pub/linux/kernel/
```

Esetleg ha nem lenne elegendő az alap kernel tudása, vagy olyan eszközünk van a gép-
hez csatlakoztatva, amihez az alap kernelben nincs vezérlő, beszerezhetünk egy fejlesztői foltot. Javasolt fejlesztők foltjai:

```
Alan Cox: http://www.kernel.org/pub/linux/kernel/people/alan/  
Andrea Arcangeli: http://www.kernel.org/pub/linux/kernel/people/andrea/
```

Itt különösen igaz az a mondás, hogy csak a megfelelő verziószámú kernelre alkalmazuk a foltot, illetve csak egy foltot használjunk. A megfelelő verziószámokra gondosan ügyeljünk oda!

Ha letöltöttük, csomagoljuk ki a kernel forráskódját a */usr/src/linux-verzió* könyvtárba, majd állítsuk be a */usr/src/linux-2.4* szimlinkeket arra a könyvtárra. (Ügyeljünk arra, hogy létezzen a */usr/src/linux* szimlink, ami pedig az előbb említett */usr/src/linux-2.4* szimlinkre mutasson.) A foltot ugyanúgy tudjuk feltenni, mint máshol: lépünk be a könyvtárba, majd adjuk ki a *patch -p0 -i foltfájl* parancsot.

Ezek után adjuk ki a *'make mrproper'* parancsot, aminek hatására letörlődik *_minden_* felesleges fájl (megjegyzések, ideiglenes fájlok, előző fordítás maradványai, már lefordított kódok, beállítási fájlok), és csak és kizárólag a szintiszta forráskód fog megmaradni. Értelemszerűen, ha egy új fordítást szeretnénk készíteni, szintén ezt a parancsot alkalmazzuk. Létezik még a *'make clean'* parancs is, de ez sokszor szemetet hagy a foltozott kernelek esetében, illetve a beállítási fájlunkat sem törli le.

Testreszabás

A testreszabás előtt érdemes saját verziószámmal ellátni a kernelt, hogy ha újra lefordítjuk ezt a verziót, akkor is egyedi számozást kapjon, és ne keveredjenek össze a két egyforma verziószámú kernel cuccai. Ehhez egy szerkesztőben (lehet akár az *mcedit* vagy a *vi* is) nyissuk meg a *Makefile* fájlt, ahol a verziószámot a legelején láthatjuk:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 21
EXTRAVERSION =
```

Írjuk oda a mai dátumot, vagy amit szeretnénk az EXTRAVERSION után, pl:

```
EXTRAVERSION = -200307150725
```

Ezek után a lefordított kernelünk verziószáma 2.4.21-200307150725 lesz.

A kernel testreszabásához beépített segédeszközök állnak rendelkezésünkre. Léteznek más segédeszközök is, pl. a *KDE* rendszerben is van egy, de a beépített programok minden igényeket kielégítenek. Három program áll rendelkezésünkre, a lehetőségeinkhez mérten:

make config ez egy kérdezz-felelek elven működő program, végigmegy a legfontosabb beállításokon, ám nem az összesen, és visszatérni egy korábbi kérdésre nem lehet, ezért használata nem javallott, ámde minden helyzetben működik.

make menuconfig ez egy konzolos felületű menü vezérelt program, nagyon hasznos, ámde egy hibája mégis van, nem láthatjuk a nem elérhető beállításokat.

make xconfig a leghasznosabb segédprogram, ámde csak grafikus felület alatt működik, ráadásul ezt is forráskódban kapjuk, ezért a lefordításához grafikus felülethez szükséges fejlesztőeszközök megléte szükséges.

Az elkészült beállításunk a *.config* nevű fájlban fog tárolódni.

menuconfig

A menüpontok között a kurzormozgató billentyűkkel vagy a forrógombokkal mozogatsz. A kiválasztás **(Enterrel)** vagy **(Szóközzel)** történhet, feltéve ha a **(Select)** van kijelölve alul. Ha az **(Exit)** van, akkor egyel feljebb lépsz vagy kilépsz a programból. A **(Help)** gombot választva kapsz helpet (logikus) persze angolul. Szóval **(Select)** és bejutsz a választható opciókhoz. Az opciók előtt kétféle dolgot láthatsz:

[] csak ki- és bekapcsolható
< > Modulba is rakható

Ha valami be van kapcsolva, akkor * van a dobozban, ha *M* van ott akkor modulban van, ha üres akkor nyilván nem kerül bele a kernelbe.

Kilépéskor a *menuconf* megkérdi, hogy el szeretné-e menteni az új beállításokat, ekkor a régi *.config* átneveződik *.config.old* -ra, és az új beállítások elmentődnek *.config* néven.

A program induláskor a *.config* fájlt tölti be, ha létezik, ha nem létezik, akkor a processzor architektúráknak megfelelő alapértelmezett beállításokat hozza föl. A menüben lehetőség van egyéni beállítási fájl betöltésére, illetve egyéni fájlba menteni.

xconfig

Az *xconfig* használata megegyezik a *menuconfig*éval, a különbség a grafikus felület, és az egérrel való vezérlés.

Itt lehetőségünk van végig lépkedni az összes ablakon, még azokon is, amik amúgy nem lennének elérhetőek *menuconf* alatt, illetve ennek megfelelően láthatjuk a nem elérhető beállításokat is.

Fordítás

Fordítás előtt adjuk ki a *make dep* parancsot, ami leellenőrzi, hogy minden a helyén van-e, és erről egy információs fájlt is készít, ami a fordítást felgyorsítja.

A fordításhoz adjuk ki a *make bzImage* parancsot. Ha minden rendben lefutott az *arch/i386/boot/bzImage* nevű fájl lesz az új kernelünk.

Ha engedélyeztük a modulok támogatását, adjuk ki a *make modules* majd a *make modules_install* parancsot. Az előbbi lefordítja a modulok, míg az utóbbi feltelepíti a */lib/modules/(kernelverzió)/* könyvtárba. Mint láthatjuk, itt van nagyon fontos szerepe az egyedi verziószámozásnak.

A kernelhez tartozik még a *System.map* nevű fájl is.

10.4. Munka a kernellel

Most megismerkedhetünk a kernel feltelepítésével, és a modulok használatával.

A lefordított kernel feltelepítése

A kernel fájlunkat másoljuk be a */boot* könyvtárba valami egyedi névre, mondjuk *vmlinuz-verzió* (pl. *vmlinuz-2.4.21-200307150725*). Ugyanígy a *System.map* és a *.config* nevű fájlokat is másoljuk oda rendre hasonló névválasztással (azaz *System.map-200307150725* és *config-200307150725* legyen a nevük). Ha létezik */boot/System.map* nevű szimlink, akkor azt módosítsuk át az új *System.map* fájlunkra.

Ezek után már nincs más dolgunk, mint testreszabni */etc/modules.conf* fájlt, és beállítani a boot managerünket (erről bővebben a rendszerindítási részben olvashatunk).

Ha más helyre, máshogyan szeretnénk telepíteni a kernelt, a lehetőségeink adottak.

Munka a modulokkal

A modulok olyan eszközvezérlők, amiket nem fordítottunk bele alaphoz a kernelbe, ezáltal lehetőségünk van menet közben betölteni, vagy letiltani (memóriából kidobni) őket. Ez lehetőséget ad külső forrásból készült, vagy frissített modulok használatára is (mondjuk egy éles rendszeren futó gépet nem kell leállítanunk, ha frissíteni szeretnénk rajta pl. a hálózati kártya vezérlőjét, vagy pl. a hálózatot lekezelő belső algoritmusokat, mert egy pár parancs kiadásával ezt semmi pillanat alatt megtehetjük, szemben az újraindítás hosszú perceivel).

A modulok a */lib/modules/(kernelverzió)/* könyvtárban helyezkednek el. A modulok között függőségek vannak meghatározva, ez a gyakorlatban azt jelenti, hogy a tömörített CD fájlrendszer vezérlőjét nem fogjuk tudni betölteni a sima cdfs fájlrendszer megléte nélkül (mivel az előbbi az utóbbi egy kiterjesztése). De mondhatnánk sokkal fontosabb problémát is, pl. a SCSI eszközvezérlőket nem lehet betölteni az alapvető SCSI támogatás nélkül.

Egy modul betöltése és eltávolítása

A modulok használata elég egyszerű. A legtöbb esetben ez automatikus, a kernel démon automatikusan betölti a megfelelő modult, illetve eltávolítja azt, ha már senki sem használja. Ha kézzel töltünk be egy modult, azzal a kernel démon nem fog foglalkozni, azaz sosem távolítja el, azt is nekünk kell megtenni. Ez a lassan inicializálható eszközöknél (pl. SCSI merevlemez) előnyös.

A modulok betöltésére és eltávolítására három alapvető parancsot használhatunk:

```
modprobe  
insmod  
rmmod
```

illetve a betöltött modulok kilistázására az *lsmod*.

Az *insmod* betölt egy konkrét modult, amit név szerint megadtunk neki. A *modprobe* annyival jobb az *insmod*-nál, hogy betölti a modul működéséhez szükséges egyéb modulokat is, de ezen kívül több más hasznos funkcióval is rendelkezik. Ezek közül az egyik, hogy képes kilistázni az elérhető modulokat (*modprobe -l*), vagy csak egy adott típushoz tartozó modulokat (például a fájlrendszer modulok listázása: *modprobe -l -t fs*). Erről és az összes parancsról bővebben a parancsok kézikönyv lapjain olvashatunk.

Sok modulnak paramétereket kell megadnunk. Ezt az *insmod* vagy *modprobe* parancssorában vagy a */etc/conf.modules* fájlban tehetjük meg. Parancssorban egyszerűen a *modprobe modulnév opciók* formátumot kell használnunk.

Ha egy modult a *modprobe* vagy az *insmod* nem akar betölteni valami okból, de biztosak vagyunk benne, hogy az működik a kernelünkkel, megpróbálhatjuk betölteni a *-f* opcióval, azaz kényszeríthetjük (force) a betöltését (*insmod -f modulnév*). Természetesen ez is csak akkor sikerül, ha a modul működőképes az aktuális kernellel.

A modulok listázása és eltávolítása

Az *lsmod* kilistázza az betöltött modulokat, és függőségeiket. Az első oszlop a modul nevét, a második a méretét (kernellapok számában) a harmadik az őt használó más modulok számát és esetlegesen a nevüket adja meg. Az (autoclean) jelentése: a modult a kernel démon eltávolítja, ha már nincs rá szükség. Ha ez nem szerepel a sor végén, a modul addig marad betöltve, amíg kézzel el nem távolítjuk.

Az *rmmod*-al van alkalmunk van eltávolítani egy modult, ha nem kapcsolódik hozzá semmi sem. Az *rmmod* használata:

```
rmmod modulnév
```

A modulok beállítása

A modulok opcióit beírhatjuk a */etc/modules.conf* fájlba, ahol mind a kernel démon, mind a *modprobe* és az *insmod* program megtalálják azt. Ezen felül itt adhatjuk meg azt, hogy egyes eszközök használatához mely modulok szükségesek. Azaz itt állíthatjuk be, hogy például a hálózati kártya NE2000 kompatibilis, azt az *ne* modullal kell kezelni, aminek az opciói *io=0x300 irq=10*. Ezt az alábbi módon tehetjük meg:

```
alias eth0 ne
options ne io=0x300 irq=10
```

Két kártya esetén ugyanez (a fenti példát használva):

```
alias eth0 ne
alias eth1 ne
options ne io=0x300,0x340 irq=10,5
```

Ezen felül ha szükséges, megadhatunk programokat is, melyek a modul betöltése előtt vagy után futnak le, de modulbetöltés helyett is futtathatunk programot. Ezekre ritkán van szükség, így itt nem ejtünk több szót róla.

Másik példánk esetén a */etc/modules.conf* fájlhoz az alábbiakat kell hozzáadni. Mivel az Iomega Zip vezérlője egy SCSI eszközt emulál, nem mindegy, hogy van-e már SCSI eszköz a gépünkben. Az alábbiakban feltételezzük, hogy nincs. Amennyiben van, akkor a SCSI eszközök neve ennek megfelelően változik. Ahhoz, hogy a kernel démon tudja, melyik lemez eléréséhez kell betölteni a ppa modult, meg kell azt is adni. A kernel démon ezt a device típusa és sorszáma alapján tudja megállapítani. A linuxos eszközökre kivétel nélkül a */dev/* könyvtár alatti fájlokkal lehet hivatkozni, melyek speciális "device file"-ok. Ezek megadják az eszköz típusát (esetünkben ez block) és kódját (a SCSI lemezek major kódja 8). Azaz a SCSI lemez "neve" block-major-8 (*/dev/sd**). A ppa modul beállítása tehát:

```
alias block-major-8 ppa
options ppa ppa_base=0x278
post-install ppa /sbin/insmod -k sd_mod
```

A *post-install* opció azt a célt szolgálja, hogy a ppa modul betöltése után betöltsük a SCSI lemezkezelő modult is, hiszen szükségünk van rá. (Ami azt illeti, nem teljesen világos, miért nem tölti be a kernel démon automatikusan. Hagyományos SCSI eszközök esetén megteszi.). Az *insmod -k* opciója azt jelzi, hogy a kernel démon (kernel) kezelje a továbbiakban a modult, azaz amikor már nem kell, vegye azt is ki a ppa-val együtt.

A kernel démon a fenti elnevezési rendszer szerint keres minden modult, aminek az eszközére hivatkoztunk. Gyakran előfordul, hogy egy ilyen eszköznek a modulja nem elérhető, mert nincs rá szükségünk, és a kernel fordításakor azt kikapcsoltuk.

Ilyen például az *Appletalk protokoll* net-pf-5 nevű drivere. Ez azon protokollok közé tartozik, amit a kernel démon minden új hálózati interface konfigurálásakor megpróbál betölteni (logikusan, hiszen lehetséges, hogy használni akarjuk az adott interface-en. Ugyanilyen az IPX drivere is). Ha azonban ez a modul nem létezik, minden alkalommal kapunk egy üzenetet, hogy nem találja. Ez azon kívül, hogy nem szép, más gondot nem okoz. De ezt a szépséghibát is orvosolhatjuk, igen egyszerűen. Az alábbi sort kell betenni a */etc/modules.conf* fájlba:

```
alias net-pf-5 off
```

Azaz egyszerűen kikapcsoljuk ezt a modult. Ugyanígy kapcsolhatunk ki más modulokat is, függetlenül attól, hogy azok elérhetők vagy sem.

Az elején már szóba került, hogy a modulok a `/lib/modules/(kernelverzió)/` könyvtárban találhatók telepítésük után. Ezen belül a típusuknak megfelelően alkönyvtárakba kerülnek. A hálózati modulok - például az `ne` - a `/lib/modules/2.4.21/kernel/drivers/net` alatt találhatók. Ezek az alapbeállítás szerinti könyvtárak, ahol a kernel démon keresi a modulokat.

A modulok függőségei és további információk

A modulfüggőségek egy fájlban (`/lib/modules/(modulverzió)/modules.dep`) vannak eltárolva.

Ez az információs fájl a rendszerbeinduláskor automatikusan frissítődik, de kézzel is megtehetjük a `depmod -a` parancs kiadásával.

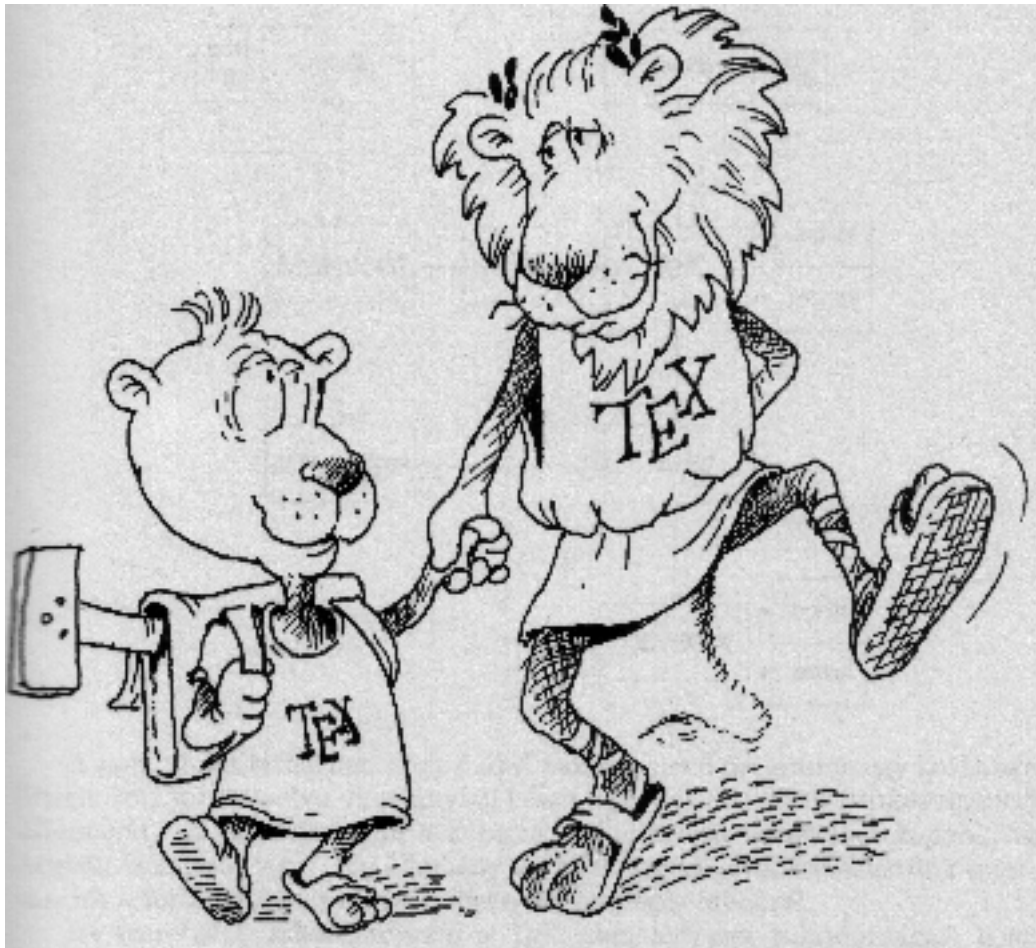
Egy modullal információkat a `modinfo` parancssal kérhetünk.

10.5. Felhasznált irodalom

- Linux oktató anyag by Kósa Attila
- A Linux (eddig) története (0.2 változat) by Horváthné Harmati Szilvia és Horváth András <horvatha@rs1.szif.hu>
- Debian specifikus kernelcsomag készítése by Balázs Tibor <co-vek@tux.linux.hu>
- Kernel modulok by Slapic <slapic@linux.co.hu>
- Kernel konfigurálás, fordítás ismeretlen szerzőtől

11. fejezet

A Linux környezet mélyebb ismerete



A merevlemezek (Hard disk, Winchester, stb.) olyan eszközök, amikben (a porosodás ellen hermetikusan lezárva) egymás alatt több – mágneses réteggel ellátott – korong és hozzájuk tartozó olvasó–író fejek (valamint léptetőmotor és sok minden más) találhatóak. A korongokon koncentrikus körökben felírva található az információ. Egy ilyen kört sávnak (track) nevezünk, az egymás alatti korongok azonos sugarú sávjait pedig együttesen cilindernek. Egy sáv tovább van szeletelve szektorokra, ez a lemez hozzáférés alapegysége. Egy szektor általában 512 byte "hasznos" adatot tartalmaz. (A szektoron belüli adattárolás részletei most nem érdekesek.) Egy koronghoz 2 olvasó–író fej (head) tartozik, a korong mindkét oldalához egy. Néha e miatt fej helyett oldalt (side) szoktak emlegetni.

Egy szektorhoz úgy férünk hozzá (úgy címezzük meg), hogy megadjuk, melyik cilinderen (milyen sugarú körön), melyik fejjel olvasva/írva, a sáv hányadik szektorában van az adat. Ezt a címezést C/F/S (angol rövidítéssel C/H/S) címzésnek nevezzük. A cilinder és a fej cím 0 kezdetű, míg a szektor cím 1-től indul. A lemezek ilyen (C/F/S) mértékegységben megadott méretét lemez geometriának nevezzük. Ezt az értéket (illetve – mint majd később látni fogjuk – valami hasonlót) a szoftver (például diszk partícionáló szoftver) a BIOS-tól tudhatja meg.

AT buszos lemezek

A kezdeti MFM (Modified Frequency Modulation) illetve RLL (Run Length Limited) lemezek is, valamint illesztőkártyáik is egyszerű felépítésűek voltak. Az illesztőkártya és a lemez között soros vonalon közlekedtek az adatok, azokat az illesztőkártya szedte össze byte-okba. (Ez nagy gátja volt a sebességnövekedésnek is.) Az illesztőkártya néhány parancsot ismert, ezekben a szektor címezését 10 bites cilinder, 4 bites fej, és 6 bites szektor cím adta.

Az IBM-PC és klónjainak BIOS-a ezért (int 13h-s hívások, Disk I/O) egy szavas regisztert (CX) használ a cilinder+szektor cím megadására (10 bites cilinder és 6 bites szektorcímek), és egy byte-os regisztert (DH) a fej címezésére (mivel kisebb nem volt :). Ez, mint látni fogjuk, sok problémához vezetett a későbbiekben.

Az adatátviteli sebesség növelése érdekében később az IDE (Integrated Device Equipment, mások szerint Integrated Drive Electronics) más néve ATA (AT bus Attachment) meghajtókat kezdték alkalmazni. Ezekben már egy beépített processzor végzi a feladatok nagy részét, az egyszerű felépítésű illesztőkártya felé 16 bites párhuzamos adatvonalakon megy az információ. Így még nagyobb kábelhosszak (kisebb kábelsebesség) esetén is jóval nagyobb sebesség érhető el, mint az MFM lemezek soros átvitele esetében.

Az IDE lemezek illesztőkártyája a számítógép (a szoftver) felől nézve kompatibilis maradt a korábbi MFM kártyákkal. A lemez mellé épített processzor viszont egyéb dolgokra is használhatóvá vált, pl. eltakarhatja a lemez fizikai kezelését: A lemezen

belül változó szektorszámmal dolgozhat a kapacitás növelése érdekében (a külső, nagyobb kerületű sávokban több információ fér el ugyanis, mint amit a belső, rövidebb sávokra megállapított szektorszám lehetővé tenne), a külvilág felé viszont mondhat egy állandó szektorszámot. Az újabb lemezeknél szokás az is, hogy a lemezterület egy részét fenntartják hibajavításra, és amikor egy szektor bizonytalankodni kezd, a beépített processzor automatikusan lecseréli egy jóra. Így a korábbi szoftverekkel való kompatibilitás megtartásával lehetett a biztonság növelése mellett még tárolókapacitást és átviteli sebességet is növelni.

Tehát IDE lemezek esetében már eléggé bizonytalan dolog geometriáról beszélni, ennek ellenére a lemezek szimulálnak kifelé ilyen adatot. Sőt, lekérdezhető a lemezek geometriája. Az IDE vezérlő kicsit meg is növelte a lehetséges kezelhető lemezméretet: 16 biten címzi a cilindert, továbbra is csak 4 biten a fejet, és 8 bitre nőtt a szektorok címe is.

Ezek szerint IDE lemez esetében a lehetséges maximális kapacitás ($16 \text{ bit} + 4 \text{ bit} + 8 \text{ bit} =$) $65536 \text{ cylinder} * 16 \text{ fej} * 255 \text{ szektor} * 512 \text{ byte} = 127.5 \text{ GigaByte}$, a címzés néhány évig még elegendőnek tűnik (Megj: a szektor cím 1-től indul, ezért 8 bitből csak 255 lehetőség lesz.)

Amennyiben BIOS-t használ az operációs rendszer, akkor annak korlátait is figyelembe kell venni ilyen maximumok számításánál: $(10 \text{ bit} + 8 \text{ bit} + 6 \text{ bit} =)$ $1024 \text{ cylinder} * 255 \text{ fej} * 63 \text{ szektor} * 512 \text{ byte} = 7.8 \text{ GigaByte}$, ez sem olyan rossz méret azért, bár lesznek sokan, akiknek jövőre már szűk lesz. (Megj: a fej cím 0..254 lehet csak, ez 255 lehetőség, a szektorcím 1-ről indul, ez 63 lehetőség.)

Ha viszont összevonjuk a kétféle geometriát, abból maximális kapacitásként 10 bit cylinder, 4 bit fej és 6 bit szektorcím adódik, ez viszont csak $1024 * 16 * 63 * 512 \text{ byte} = 504 \text{ MegaByte}$, ami már ma mindenkinek szűk keresztmetszet! Ennek feloldására tett lépésekről kicsit később lesz még szó.

Megjegyzés: Van, aki 528 mega / 8.4 giga / 136.9 gigáról beszél 504 mega / 7.8 giga / 127.5 giga helyett: attól lett "nagyobb" a kapacitás, mert megelégedett kilobyte-onként 1000 byte-tal 1024 helyett, és hasonlóképpen kerekített a mega illetve giga váltáskor is. Ilyen könnyű a semmiből néhány tíz/száz/ezer MegaByte-ot csinálni! :)))

SCSI buszos lemezek

SCSI busz esetébe első közelítésben nincsenek ilyen problémák, mert nem C/F/S rendszerben, hanem egyetlen lineáris szektorcímként kell címezni a lemezt. Mégis, BIOS-t használva (tehát boot-oláskor illetve DOS alatt végig) az int13 paraméterei továbbra is C/F/S adatokat várnak, így valamit mégiscsak tudnunk kell a geometriáról (és mellesleg a 7.8 GigaByte-os korlát így ide is bejön).

A lemez fizikai geometriáját le lehet kérdezni a SCSI buszon keresztül, viszont nem igazán ez az adat az, ami a BIOS szempontjából érdekes (mert így gyakran 1024-nél jóval

több cylinder jönne ki). A geometria itt teljesen a BIOS belügye (itt BIOS alatt az illetestőkártyán lévő BIOS bővítést kell érteni, amennyiben nem alaplapra integrált SCSI meghajtóról van szó), hiszen az int13–ba bejövő C/F/S címet a BIOS maga átalakítja lineáris szektorcímmé és már csak azt küldi tovább a lemezegység felé. Ezért gyakran 64 fejjel és 32 szektorral számolnak a BIOS–ok. Ennek "szépészeti" előnye van, ugyanis így egy cylinder mérete kerekén 1 MegaByte, így a partíciós táblában a partíció annyi megabyte–os, ahány cylindert lefoglal. Hátránya viszont, hogy csak 1 GigaByte–ig használható. Ennél nagyobb kapacitású lemezeknél gyakori választás a 255 fej és 63 szektor, amikkel a BIOS által elérhető majd 8 GigaByte–os teljes kapacitás kezelhető. Néhány SCSI csatolónál (pontosabban az abban lévő BIOS bővítésnél) engedélyezni kell, hogy hajlandó legyen 1024 MegaByte fölé menni, vagyis $\max 1024/255/63$ –as geometriát szimulálni.

Nagy kapacitású AT buszos lemezek

Térjünk most vissza az IDE lemezekhez: Ha a BIOS–on keresztül érjük el a lemezt, 504 MegaByte–ra korlátozódik a címezhető kapacitás (emlékeztető kicsit korábban). Ennek feloldására kétféle megoldást találtak ki. Az egyik azonos a SCSI lemezeknél alkalmazottal: a modernebb IDE lemezeknél bevezettek új parancsokat, amikkel nem C/F/S rendszerben, hanem lineáris szektorcímmel érhető el a lemez. Ezt LBA–nak (Linear Block Addressing) szokás nevezni, 28 bites lineáris címekkel lehet így dolgozni, tehát 128 GigaByte–ot lehet kezelni. Persze itt is át kell alakítani ezt a lineáris címet C/F/S címezéssé akkor, amikor a BIOS int13 hívását kell használni (boot–oláskor illetve DOS alatt). Ilyenkor is a SCSI csatolóknál leírtak szerint járnak el a BIOS–ok, tehát szimulálnak valamilyen szimpatikus geometriát, és egyben lecsökkentik a használható méretet 7.8 GigaByte–ra.

A másik megoldásnál is azt használja ki a BIOS, hogy a fej címezésére a DH regiszterben 8 bit áll rendelkezésre 4 helyett: az eredeti C/F/S geometria helyett egy C'/F'/S geometriát szimulál a BIOS befelé (a szoftver felé), ahol $C' = C/n$ és $F' = F*n$. Az n konstans pedig úgy állapítja meg, hogy $C' \cdot 1024$ –nél ne legyen nagyobb. (n elvileg bármilyen egész szám lehetne, de kettő hatványt szoktak csak használni a BIOS–ok, így szorzás helyett shift is elegendő.) Kifelé, a lemezegység felé természetesen a fizikai C/F/S–re alakít vissza mindent a BIOS. Ezzel a megoldással tehát nincs szükség arra, hogy a lemez tudja az LBA–t. Ilyenkor is 7.8 GigaByte a lehetséges maximális lemezméret.

A BIOS setup–ban általában az eredeti, tehát nem a szimulált geometriát kell beállítani, illetve automatikus lemez–felismeréskor is azt írja ki a BIOS. Azt, hogy melyik szimulációs megoldást használja a BIOS, tipikusan a setup–jában lehet beállítani "LBA" illetve "Large" néven. Mindkét esetben majdnem 8 GigaByte kapacitásig kezelhető így (tehát BIOS–on keresztül) egy lemez.

Régebbi BIOS-ok nem tudtak ilyen szimulálást, ezért léteznek olyan disk manager-ek (pl. OnTrack vagy EZ-Drive), amik a BIOS helyére lépnek már boot-oláskor, maguk kezelik a diszkeket és elvégzik ezt a geometria-átalakítást. Ezekről nem tudok sokat, de remélhetőleg ki is halnak idővel (ha ki nem haltak már:). Ha mégis ilyenekről akarsz olvasni, valamennyi információ van róluk a John Wehman és Peter den Haan által karbantartott ATA-FAQ-ban (magyarországon például az <ftp.vein.hu/pc/doc/atafaq191.zip> címen található).

Ugyanebben a FAQ-ban találtam két érdekes BIOS bug leírást:

Néhány régebbi (1996 előtti) BIOS rosszul kezeli a 4096 (mások a 8192) cilindernél (2016 illetve 4032 MByte-nál) nagyobb kapacitású lemezeket. Ezeknél meg lehet próbálni a BIOS setup-jában kézzel átállítani a lemez kapacitását 4095 (illetve 8191) cylinderre, a többi terület DOS számára elérhetetlen (Linuxból azért az is használható).

Más BIOS-ok 8192 cilindernél nagyobb kapacitás esetén $n=16$ -tal (shift 4) számolnak, hogy a cilinderszám 1024 alá kerüljön (lásd néhány paragrafussal feljebb). Ez viszont az eredeti 16 fejből $16 \cdot 16 = 256$ fejet csinál, ami már nem ábrázolható 8 biten! Ha a BIOS-ban ilyen hiba van, azt lehet tenni, hogy a BIOS setup-jában kézzel átállítjuk a fejszámot 16-ról 15-re, a cilinderszámot pedig x -ről $x \cdot 16/15$ -re (lefelé kerekítve). (Az ATA-FAQ szerint ilyen módosítást mindig megtehetünk, de nekem hiányzik egy közbülső lépés, hogy megértssem. Esetleg valaki ki tud segíteni?)

[Egy általam (is?) talált DOS-Boot hibáról, ami szintén 2GByte környékén jön elő, később lesz szó.]

Partícionálás

A PC-k merevlemezén (ha pl. DOS lemezként van formázva,) 2 boot szektor található, szemben pl. a hajlékony lemezzel, ahol csak egy. A fő boot szektor a lemez fizikailag első szektora (0-ás cylinder, 0-ás fej, 1. szektor, a továbbiakban 0/0/1), szokás MBR-nek is nevezni (Master Boot Record). Ez az 512 byte-os szektor tartalmaz egyrészt egy Intel x86 gépi kódú betöltőprogramot, másrészt a szektor végén egy 4-szer 16 byte-os táblázatot, amit két byte, 0x55 és 0xAA zár le (Megjegyzés: a 0x__ jelölés 16-os számrendszerben leírt számot jelent). Ez a 64 byte-os táblázat a partíciós tábla.

A partíciós táblázat segítségével a merevlemez eredetileg maximum 4 tartományra, partícióra lehetett osztani. Ezek a partíciók tartalmazhatnak akár egyforma, akár különböző operációs rendszerekhez tartozó lemezterületet. A lemezt akkor is érdemes lehet több részre partícionálni, ha csupán egyetlen operációs rendszert használ rajta az ember, erről esetleg később még írok. (Illetve ha valakinek van kedve írni, vagy már létező link—je van, szóljon)

A partíciós bejegyzések felépítése

A táblázat egy 16 byte-os bejegyzése a következő adatokat tartalmazza:

- F[1]: boot flag (boot—olható—e a partíció)
- B[3]: a partíció első szektora (cilinder/fej/szektor címként megadva)
- T[1]: a partíció típuskódja (durván: az operációs rendszer kódja)
- E[3]: a partíció utolsó szektora (cilinder/fej/szektor címként megadva)
- R[4]: a partíció első szektora a partíciós tábla címéhez képest, szektorokban
- S[4]: a partíció mérete szektorokban

Megjegyzések:

1. A táblázatban az adatok a fenti sorrendben találhatók
2. A zárójelben lévő számok az adat méretét adják byte—ban
3. A cilinder/fej/szektor (C/F/S) cím fej,szektor,cilinder sorrendben szerepel a táblázatban, 8,6,10 bitként kódolva (ahol a 10 bites cilindercím alsó 8 bitjét a harmadik byte adja, felső 2 bitjét pedig a második byte legnagyobb helyiértékű két bitje)

Első ránézésre látszik, hogy a táblázat meglehetősen redundáns: Ugyanaz az adat, a partíció kezdete, kétféleképpen is meg van adva: C/F/S címként, illetve (relatív) szektorcímként is. A partíció végének C/F/S címe is egyértelműen meghatározza a partíció méretét, ami ennek ellenére meg van adva. Ezeknek a mennyiségeknek ugyanazt az adatot kell eredményezniük, egyébként a bejegyzést a különféle operációs rendszerek illegálisnak veszik, és figyelmen kívül hagyják.

C/F/S címmel csak az első 1024 cilinderen lévő (pl. 504 MegaByte—ig, illetve módosított geometria esetén tipikusan 1 GigaByte—ig, maximum 7.8 GigaByte—ig terjedő) partíciókat adhatunk meg, viszont a partíciós táblában szintén szereplő szektorcímmel ennél sokkal nagyobb a korlát: 2 TeraByte (2048 GigaByte). Ezért az OS/2 és a Linux csak a szektorcímet használja ezekből az adatokból (hiszen semmi sem köti őket a BIOS C/F/S korlátaival), és a "messzi" partíciók kezdő illetve befejező C/F/S címként a partíciós táblában 1023/xx/yy (az utolsó C/F/S-ként címezhető szektor) szerepel. (Régebbi Linux fdisk még nem figyelt erre, a címet moduló 1024 tette a partíciós tábla C/F/S bejegyzéseibe, ez az OS/2-nek nem mindig tetszik:) A boot-olandó partíciónak viszont az első 1024 cilinderen belül kell lennie (legalábbis azon file-oknak, amikre a boot-olás közben szükség van), hogy a BIOS el tudja azt indítani. Hogy ez megabyte-ban mit jelent, az már a BIOS által használt (illetve szimulált) diszk geometriától függ. (pl. 504 mega, 1 giga, 7.8 giga)

Erre a témára még visszatérünk, most viszont maradjunk az eredeti, C/F/S és szektorcím redundanciáján alapuló partíciós bejegyzéseknél.

A partíció helye

A redundáns információk kiszámításához (meg egyébként is:) tudnunk kell a lemez geometriáját, tehát hogy a lemezegységen egy cylinder hány fejet, illetve egy fej hány szektort tartalmaz, nevezzük ezt rendre FPC-nek illetve SPF-nek. A legtöbb operációs rendszer elvárja még, hogy egy partíció cylinderhatáron kezdődjön, illetve végződjön. Ettől csak a partíciós táblát közvetlenül követő partíció kezdőcíme térhet el, ami fej határon kell kezdődjön. (Ebből következik egyébként, hogy a partíciós tábla egyetlen szektorja után következő SPF-1 darab szektor kihasználatlan marad; néhány boot manager képes ide költözteti magát.)

Szóval:

A partíciós tábla helye PC cylinder, PF fej és PS szektor:

PC = 0 (legalábbis az MBR esetében)

PF = 0

PS = 1

Egy partíció kezdete BC cylinder, BF fej és BS szektor:

BC = szabadon választható

BF = 0 vagy 1 (partíciós táblát követő partíciónál lehet 1)

BS = 1

A partíció vége EC cylinder, EF fej és ES szektor:

EC = szabadon választható

EF = FPC-1

ES = SPF

A partíció kezdetének R relatív szektorcíme:

$$R = ((BC-PC) * FPC + BF) * SPF$$

A partíció S mérete szektorban:

$$S = ((EC+1-BC) * FPC - BF) * SPF$$

Ezek a képletek csak akkor alkalmazhatóak, ha a partíció kezdete és vége "jólnevelt", cylinderhatáros, ahogy nem sokkal fentebb írtam. A képletben azért szerepel PC értéke, mert (majd később látni fogjuk) van olyan partíciós tábla is, ami nem a 0/0/1-es címen

található, hanem PC/0/1-en. Az ezt követő legelső partícióra is igaz az, hogy elég, ha fej határos címen kezdődik.

Lehet egyébként kísérletezni vele, hogy nem hagyunk lyukat az első partíció előtt (én még nem tettem ilyet:). Ha szereti valaki a veszélyeket, próbálkozhat azzal is, hogy nem cylinderhatáron fejez be egy partíciót. Ekkor a fenti képletek picit bonyolódnak. Valahol azt olvastam, hogy ami ezt biztos nem szereti, az a DOS és az OS/2. Megzavarhat más operációs rendszereket is, például akkor, hogyha egy SCSI csatolón nincs engedélyezve a BIOS bővítés: Ekkor a partíciós táblában szimulált diszk geometriát a megfelelő fdisk program (pontosabban a diszk vezérlő (driver) program) magából a partíciós táblából próbálja kitalálni, több-kevesebb sikerrel – ha nem cylinderhatáros a partíció, inkább kevesebb sikerrel..

A FreeBSD rögtön kezdődhet az MBR után, nem kell feltétlenül cylinderhatárra tenni. Installálásnál választható, hogy a 2-es szektorra kerüljön, vagy pedig cylinderhatárra.

Szerencsére ezeket a képleteket nem kell sűrűn használnunk; az fdisk jellegű partíció módosító programok ezt a számítást automatikusan elvégzik. Ami személy szerint nem tetszik bennük (sem a dos-os, sem a linux-os fdisk-ben), az az, hogy magát a partíció kezdőcímét is elrejtik (bár linux alatt a cfdisk-kel van lehetőség "table" formátumú használatra is). A kedvenc partíció nézegető – módosító programom a Norton féle DiskEditor (DE), amivel a partíciós táblában található minden információhoz hozzá lehet férni, e mellett pedig ad egy jó eszközt is, amivel a redundáns adatok egyikét automatikusan kiszámíthatjuk az után, hogy a másikat beírjuk. Persze tudom, ízlések és pofonok...

A linux fdisk esetén arra még figyelni kell, hogy a cilinderek számozását ez a program nem 0-tól, hanem 1-től kezdi!

A boot flag

Ennek a byte-nak az értéke 0 (inaktív) vagy 0x80 (aktív) lehet. Az MBR-ben lévő boot program (illetve programocska) ebből tudja azt, hogy a 4 közül melyik az a partíció, amit el kell indítania. Pontosabban az eredeti (egyszerűsítve fogalmazva: DOS-os) boot program viselkedik így. Ha lecseréljük valamilyen boot manager-re, vagy egy másik operációs rendszer (pl. Linux) betöltő programjára, az szedheti ezt az információt máshonnan is, így nem kell a boot flag-ben feltétlenül 0x80-as értéknek lennie a bootolandó partíciónál. A DOS viszont nem csak bootoláskor használja ezt a flag-et, hanem később is ebből tudja, hogy melyik is az a partíció, ahol saját maga van, és amiből a C: drive kell legyen, tehát DOS-os partíciónál (ha azt akarjuk boot-olni) akkor is be kell állítani ezt a flag-et, ha maga a boot program (pl. a linuxos lilo) ezt nem kívánja.

Az előzőek azt is jelentik, hogy lehetséges a partíciós táblába több DOS partíciót is tenni (ezek közül az lesz a C: drive, amelyiknek a boot flag-je aktív), illetve nem feltétlenül az első partíciónak kell lennie a DOS-os partíciónak. Nagy kapacitású lemezek

esetében van azonban egy korlát, aminél nagyobb címen nem kezdődhet olyan DOS partíció, amiről boot-olni akarunk: A DOS partíció első szektora (az úgynevezett boot record) szintén tartalmaz egy adatstruktúrát, és azon kívül egy kis programocskát. Ennek a programocskának része egy szubrutin, ami lineáris címből C/F/S címet számít ki. Ebben a rutinban van egy "bug" (programhiba), aminek eredményeként nem boot-ol ("Non-System disk" hibát jelez) a rendszer, hogyha a betöltendő szektor (IO.SYS első 3 szektora) címében C*F értéke 65535-nél nagyobb lenne. Ha például nagy kapacitású SCSI lemezünk van (tipikusan XX/255/63-as geometriával), akkor az IO.SYS kezdete legfeljebb a 257-es cilinderen lehet, tehát 2015 MegaByte-nál (1.96 GigaByte-nál) korábban kell a DOS partíciót kezdeni. (IDE lemez esetén más geometriával, de hasonló címre (2016 MegaByte) adódik a határ.)

A probléma megoldására csináltam egy patch-et, hogyha szükséged van rá, szóljál. Ezen patch lehetővé teszi 2 GigaByte-nál magasabb címen kezdődő DOS partíció boot-olását is.

(A Windows 95 installálása módosítja a boot programot is, és abban más módszerrel számítják ki a C/F/S címet, itt már nincs az előbb említett programhiba.)

A partíció típuskódja

Ez a byte írja le azt, hogy az adott partíció milyen adatokat (milyen operációs rendszert, pontosabban milyen file rendszert) tartalmaz. Nulla értékű kód jelzi azt, hogy a partíciós bejegyzés nincs használva, a többi közül pedig a legfontosabbakat itt összesítem: (A 0x__ írásmód 16-os számrendszerbeli számot jelez)

- 0x01: DOS, 12 bites FAT filerendszer
- 0x04: DOS, 16 bites FAT filerendszer
- 0x06: DOS, 16 bites FAT filerendszer, >32 MByte (max 2GByte) partícióméret
- 0x07: OS/2, HPFS filerendszer vagy Windows NT, NTFS filerendszer
- 0x0A: OS/2 boot manager
- 0x63: GNU Hurd
- 0x64: Novell Netware 2
- 0x65: Novell Netware 3
- 0x81: Linux, Minix filerendszer
- 0x82: Linux, Swap partíció

- 0x83: Linux, EXT2FS filerendszer

Újabb kódokról később lesz szó.

Valójában egy normális operációs rendszer esetében ezek a kódok nem kellenének nagy fontossággal bírónak lenniük, ott csak azért hasznosak, mert az fdisk ezekből ki tudja írni, hogy az adott partíciónak mi a neve. Azért van néhány kivétel: az 5-ös kód (ami fentebb nem is szerepel) kitüntetett szereppel rendelkezik, erről külön fejezet szól. A DOS csak a saját partícióival hajlandó foglalkozni, úgyhogy ott fontos, hogy a partíció típusa 6-os legyen (ma már az 1-es és 4-es érték nem használatos). (Hallottam még egy 0xF2-es DOS partíciókódról is, SecondaryDOS néven, de nem tudom, mit takar...)

Az Extended partíció

Az 5-ös típusú partíció kitüntetett szereppel rendelkezik: Ennek segítségével lehet négy-nél több partíciót csinálni.

Mint eddig láttuk, a partíciós tábla 4 bejegyzés számára biztosít helyet. Ez hamar kevésnek bizonyult. Megoldásként sok minden szóba jöhetett volna (pl. még egy szektorral kibővíteni az MBR-t, hiszen úgyis lyuk van mögötte), de a Microsoft (vagy IBM?) mást választott: Az egyik partíciót a 4 közül ki lehet nevezni bővítő partíciónak, amit aztán elvileg tetszőlegesen sok logikai partícióra lehet vágni (mindjárt szó lesz róla, hogy hogyan). Ezeket eléggé félrevezető módon logikai DOS drive-oknak szokta a DOS-os fdisk nevezni, pedig egyáltalán nem csak DOS partíciókat lehet ezen a módon létrehozni.

Csak egyetlen partíció lehet bővítő (a továbbiakban angol elnevezéssel: extended) a 4 közül. Ha akár kézzel megpróbálunk csinálni több 5-ös kódút, akkor is csak az első ilyen használja minden operációs rendszer, a többit figyelmen kívül hagyják.

A partíciók lánc

Az 5-ös típusú bejegyzést valójában egy pointerként kell felfogni, ami egy újabb partíciós táblára (szektorra) mutat. Ott a 0x1BE címen szintén 4 bejegyzésnyi hely van, amit az MBR-hez hasonlóan 0x55 és 0xAA zár le (ez fontos!). Viszont a szektor eleje nem tartalmazza feltétlenül a gépi kódú programocskát. A négy bejegyzésből csak kettő lehet kitöltve (bármelyik kettő), mégpedig úgy, hogy az egyik bejegyzés egy normális partíció leírása, a másik pedig (ha van másik) egy újabb 5-ös típusú Extended partíció bejegyzés, tehát egy újabb pointer.

Néhány elnevezésbeli kérdés (előre elnézést kérek az angol-magyar keveredésért): Azokat a partíciókat, amiket az MBR partíciós táblája ír le, primary (elsődleges) partícióknak szokás nevezni. Ezek közül az az egy, aminek 5-ös a kódja, az extended partíció.

Az extended partíció részeit logikai partícióknak (néha secondary (másodlagos) partícióknak) nevezzük. Ezen logikai partíciókat megelőző és definiáló partíciós tábláknak nem szoktak nevet adni, de nevezhetjük őket másodlagos partíciós tábláknak. Mindezek viszonyát a következő ábrák segítik megérteni:

A szokásos elrendezés pl. 4 adatpartíció esetén egy primary és három logikai partíció:

hda							
MBR	Első partíció	Extended partíció					
Part1	data1 hda1 C:	Part2	data2 hda5 D:	Part3	data3 hda6 E:	Part4	data4 hda7 F:
Ext1		Ext2		Ext3		Üres	
Üres		Üres		Üres		Üres	
Üres		Üres		Üres		Üres	

Itt Ext1 mutat Part2 partíciós táblájára, Ext2 mutat Part3 partíciós táblájára, Ext3 pedig Part4 táblájára.

Ha pedig pl. az utolsó adatpartíciót nem az extended-be rakjuk, hanem külön primary partícióként kezeljük:

hda						
MBR	Első partíció hda1	Extended partíció hda2				Utolsó partíció hda3
Part1	data1 hda1 C:	Part2	data2 hda5 D:	Part3	data3 hda6 E:	data4 hda3 F:
Ext1		Ext2		Üres		
Part4		Üres		Üres		
Üres		Üres		Üres		

A táblázatokba bejelöltük, hogy a lemez egyes részeit Linux alatt milyen /dev/hdaX device-on érhetjük el (feltéve, hogy az első IDE hard disk-ről van szó), illetve azt is, hogy DOS alatt milyen drive betű rendelődik hozzájuk. (A kiválasztott példában véletlenül mindkét esetben ugyanaz a DOS drive betű rendelődik a partíciókhoz, de ez más esetekben bonyolultabban alakul.)

A partíciók utólagos módosításánál szükség lehet arra, hogy a lánc belsejébe szúrjunk bele egy új partíciót. Ezt tapasztalatom szerint nyugodtan megtehetjük, sem DOS, sem Linux alatt nem kell a fizikai sorrendnek követnie a logikait. Még az sem okoz gondot, ha a logikai partíciók közül némelyik "kilóg" az extended partícióból. Ilyenkor viszont már nagyon vigyázni kell arra, nehogy utólag kézi módosítás helyett fdisk-kel próbálkozzunk, ami esetleg rosszul mérheti fel, hogy maradt-e üres (particionálatlan) hely a lemezen. Mindenesetre a Linux-os fdisk warning-ot mond olyan esetben, hogyha ilyesmire utaló dolgot tapasztal, pl. ha a lánc visszafelé mutat. Ha tudjuk, hogy mit csinálunk, ilyen warning mellett is nyugodtan alhatunk azért :-)

A másodlagos partíciós táblák sajátosságai

Összegezve tehát: A logikai partíciókat nem az MBR partíciós táblája, hanem a másodlagos partíciós táblák írják le. Pontosabban minden másodlagos partíciós tábla egyetlen logikai partíciót ír le, ezen kívül egy pointert tartalmaz(hat) egy újabb másodlagos partíciós táblára. Ha több bejegyzés is lenne benne, azt az operációs rendszerek figyelmen kívül hagyják, ennek következtében a partíciókból nem lehet pl. egy "fát" szerkeszteni. A partíció helye című fejezetben a partíció kezdtének relatív kezdőcímét megadó kép-letben szerepelt a partíciós tábla cilindercíme (PC). Ennek igazán most van szerepe, hiszen a másodlagos partíciók esetében lesz csak ezen érték nullától különböző. Tehát a "relatív szektorcím" a partíciót definiáló partíciós táblához képest relatív. Amire figyelni kell, az az, hogy a redundánsan megadott partíció kezdetnek csak a lineáris szektorcíme relatív a megfelelő másodlagos partíciós táblához, a C/F/S címe nem, az továbbra is abszolút cím! Persze erre figyelni csak akkor kell, ha valaki kézzel (nem fdisk-kel) akarja a partíciós táblákat módosítani. Én ezt úgy szoktam, hogy a C/F/S címeket írom csak be, aztán a Norton féle DiskEdit-tel kiszámíttatom a lineáris címet.

A partíciók sorrendje (elnevezése)

Ebben a fejezetben arról lesz szó, hogy DOS (és az ezzel ekvivalens Windows) illetve Linux alatt különbözőképpen elrendezett partíciók esetén milyen néven férhetünk hozzá azokhoz.

Linux esetében könnyű a dolgunk: Az első IDE csatoló Master vonalán lévő lemez esetében /dev/hdaX lesz a partíciók neve, a Slave vonalán pedig /dev/hdbX, stb. SCSI diszkek esetén ugyanezek /dev/sdaX illetve /dev/sdbX, stb. X értéke pedig az első négy partíciós bejegyzésben található partícióknál 1-től 4-ig terjed. Tehát ha a /dev/hda első és utolsó partíciós bejegyzésében van csak adat, akkor a partíciókhoz /dev/hda1 és /dev/hda4 néven férhetünk. Ha valamelyik partíció extended, akkor az ott indított lánc partíciói /dev/hda5-től kezdve kapnak nevet, a láncolás sorrendjében növekvő számokkal. A lánc méretére egy (azt hiszem) 64-es praktikus limit van csak.

DOS esetében (a Caldera OpenDOS egy kicsit máshogy csinálja, lásd később) a drive betűk kiosztása a következők szerint alakul:

1. Azok a partíciók, amiknek nem DOS-os típuskódjuk van, nem kapnak betűjelet.
2. C:-től kezdve kapnak betűjelet a lemezegek MBR-jeiben lévő legelső DOS partíciók. Ha több elsődleges DOS partíció is van valamelyik lemez MBR-jében, azok ebben a lépésben nem kapnak nevet. Az első lemezegegyes esetében van egy kis változás ehhez képest: ott a boot flag mondja meg, hogy melyik primary DOS partíció legyen a C: nevű. (Azt nem próbáltam ki, hogy mi van másik lemezeről bootolás esetén.) Ez a kiosztás akkor is így alakul, hogyha az extended (5-ös)

partíció bejegyzése megelőzné a partíciós bejegyzések között az elsődleges DOS partíció bejegyzését.

3. Ezek után következnek az esetleges extended partíció logikai partíciói először az első lemezezen, aztán a másodikon, stb.
4. Végül az MBR-ben még megmaradt elsődleges DOS partíciók kapnak betűjelet, először az első lemezezen, aztán a másodikon, stb. Ezek a partíciók akkor is a sor végére kerülnek, hogyha a partíciós bejegyzések között fizikailag megelőzik is az extended partíció bejegyzését!

Az előző algoritmusnak egy furcsa következménye az, hogy ha az ember egy új lemezzel bővíti a gépét, akkor az azon allokált DOS partíció alapesetben az előzőleg már esetlegesen meglévő C: és D: drive-ok közé fog furakodni (D: néven). Ezen úgy tudunk segíteni, hogy a második lemezen eleve csak extended partíciót allokalunk. Ezt viszont kénytelenek vagyunk kézzel csinálni, mert a (dos-os) fdisk nem hajlandó ilyenre. Remélem, hogy ez a leírás elegendő segítséget ad egy ilyen manipuláláshoz annak, akinek szüksége lesz rá.

A Caldera OpenDOS (illetve a Novell DOS is) a fentebbiektől kicsit eltérően működik: a lemezen található összes primary DOS partíciót (ha van egyáltalán több) leképzi első lépésben C: D: stb. betűkre, és utána következnek az extended partíció drive betűi. Itt a boot flag nem játszik szerepet, mindig a legelső DOS partíció lesz a C: drive. Azt még nem próbáltam, hogy hogyan alakul ez a kiosztás akkor, ha több lemezezés is van a gépben...

Bár nem kapcsolódik szorosan a témához, de azért megjegyzem, hogy néhány software (pl. diszk kompresszálo program) csereberéli a drive betűket, tehát a fentebb leírt kiosztás módosulhat. Másik furcaság, hogy sok DOS program (pl. néhány BIOS, az FDISK, stb.) az egyszerűség kedvéért C: és D: drive betűkkel hivatkozik a lemezezésekre is pl. disk1 és disk2 helyett: Ez tipikus esetekben még sok partíció esetén is jó megfeleltetés, de vannak olyan helyzetek is, amikor nem fedi a valóságot.

Egyéb információk

Újdonságok a BIOS kapcsán

1995-ben a Microsoft kiadott egy specifikációt a BIOS int 0x13-as API-jának (API=Application Program Interface) bővítésére (azt viszont nem tudom, van-e már BIOS, ami támogatja ezt). Itt már nem 16 bites regiszterekben történik a C/F/S paraméterek átadása a BIOS és az applikáció (adott esetben a boot-oló program) között, ezzel lehetővé válik, hogy 1024 cilindernél (illetve 8 GigaByte-nál) nagyobb lemezeket is

kezelni lehessen BIOS hívásokkal. Az új függvények magukban foglalják szektorok olvasását (int13/42) és írását (int13/43), és még egyéb, kisebb jelentőségű hívásokat. Van lehetőség a lemez geometriájának lekérdezére is, de tipikusan erre nincs szükség, mivel a címzés ezen új függvényeknél 64 bites lineáris szektorcímmel történik – néhány évig bizonyára elég lesz ebből a címből az alsó 32 bit is, azzal is 2048 GigaByte-ra nőtt fel a felső kapacitáshatár :-)

Az említett függvényekről alapvető információkkal a következő lapok szolgálnak: Check Extensions Present, Extended Read, Extended Write és Data Structure.

A BIOS bővítésével egyidőben bevezettek néhány új partíció típuskódot is:

- 0x0B: DOS, 32 bites FAT (FAT32) filerendszer, max 7.8 GB partícióméret
- 0x0C: DOS, 32 bites FAT (FAT32) filerendszer, max 2048 GB partícióméret, INT13EXT
- 0x0E: DOS, 16 bites FAT filerendszer, max (talán) 2048 GB partícióméret, INT13EXT
- 0x0F: Extended partíció (mint az eddigi 5-ös típuskód), INT13EXT

A fentebbiek közül azoknál a típuskódoknál, ahol INT13EXT szerepel, a BIOS bővítésen keresztül férhet hozzá a boot-oló program a lemezegységhez. A partíciós bejegyzésben ilyenkor figyelmen kívül vannak hagyva a C/F/S stílusú címek, csak a lineáris (relatív) szektorcím és a szektorméret az érdekes. Természetesen a logikai partíciók láncolójának (extended partíció) is kellett új típuskód, így már két speciálisan kezelendő kód is van (0x05 és 0x0F).

A partíciós bejegyzésben a lineáris szektorcím méretét ezen INT13EXT stílusú típuskód esetében sem növelték meg 32 bitről 64 bitre (bár az új BIOS függvények 64 bites címekkel képesek dolgozni), így 2048 GigaByte (2 TeraByte) méretű lemezt lehet kezelni. Ez jó ideig elegendőnek tűnik még. Megjegyzendő viszont, hogy ez a korlát csak az elsődleges partíciók esetében szerepel így. Extended partíció láncnál minden egyes új partíciót a hozzá tartozó másodlagos partíciós táblához képesti relatív szektorcíme címez (lásd a A másodlagos partíciós táblák sajátosságai című fejezetet). Ezért a logikai partícióknak a méretére igaz csak a 2 TeraByte-os korlát, összes kapacitásuk lehet nagyobb 2 TeraByte-nál. Persze ilyenkor az MBR-ben lévő Extended partíció nem tudja lefedni a lánc teljes méretét, de ez valószínű nem okoz majd gondot.

Néhány szó a lilo-ról

A lilo a Linux Loader rövidítése, egy többféle operációs rendszert betölteni képes program. Fő célja persze az, hogy a Linuxot betöltse. (A Linuxról olvashatsz többek között

a Magyar Linux Alapítvány honlapján.) Nem óhajtok a lilo-ról sem kimerítően beszélni (pl. a fontosabb dolgok közül nem beszélek az `/etc/lilo.conf` file tartalmáról, a `-r` opcióról, stb.), csak néhány érdekességet írok itt le.

A boot-olás szempontjából a legfontosabb információ az, hogy a Linux nem BIOS hívásokkal kezeli a hardware-t, így a merevlemezt sem. Boot-oláskor viszont a lilo-nak nincs más választása, hisz olyankor még nincs bent a memóriában a kernel. Továbbá a kernelt magát (általában) már a Linux file-rendszeréről kell betöltenie, aminek felépítését, így a benne lévő file-ok helyét is maga a kernel tudja.

Ezt a dilemmát a lilo úgy oldja meg, hogy valójában két részre bomlik:

- Van egy 16 bites, real módú része (neve `/boot/boot.b`), ezt indítja el az MBR-be költöztetett kis programocska. (Az MBR-en kívül máshová is lehet a lilo-t installálni, de ezt hagyjuk most.)
- Van aztán egy natív Linux futtatható része, ennek a file-nak a neve valójában a `lilo (/sbin/lilo)`.

Az `/sbin/lilo`-t Linux alatt futtatva az lekérdezi a kerneltől, hogy a betöltendő file-ok (pl. maga a kernel (tipikusan `/boot/vmlinuz`)) blokkjai hol (vagyis milyen C/F/S címen) találhatóak a lemezen, és ezeket az adatokat beírja a `/boot/map` file-ba. Persze ennek a file-nak a helyét is lekérdezi, ezt az adatot magába az MBR-ben lévő programocskába írja bele. Így aztán boot-oláskor BIOS hívásokkal tud hozzáférni mindenhez, ami számára fontos.

Ezek után nyilvánvaló, hogy minden kernelfordítás után újra kell futtatni az `/sbin/lilo`-t, hogy az új kernel elhelyezkedését a `map` file-ba írhasa. Ha ezt elmulasztjuk, könnyen lehet, hogy a régi kernel indul el, még ha le is töröltük a lemezeről :-)

Minden file, amire a boot-olás során szükség van, és amiket BIOS hívásokkal kell elérni, a `/boot/` könyvtárban található. Ezek tehát azok a file-ok, amiknek a lemez első 1024 cilinderén belül kell lenniük, hisz a BIOS (az előzőleg említett BIOS bővítés nélkül) csak ezeket képes kezelni. Nagy kapacitású lemez esetén elegendő egy kis méretű partíciót kreálni a lemez elején és a `/boot/` könyvtárat ebbe helyezni, ezzel elérhetjük, hogy a kritikus file-ok BIOS-ból olvashatóak legyenek.

Gondot okoz viszont az, hogyha a kernel a BIOS-tól eltérő geometriát feltételezve adja meg a file-ok helyét, hiszen így boot-oláskor teljesen fals helyről beolvasott adatokat próbál a processzor utasításként végrehajtani. Ennek eredménye tipikusan az, hogy a kezdeti LILO feliratnak csak a fele (LI) jelenik meg a monitoron, aztán a gép lefagy.

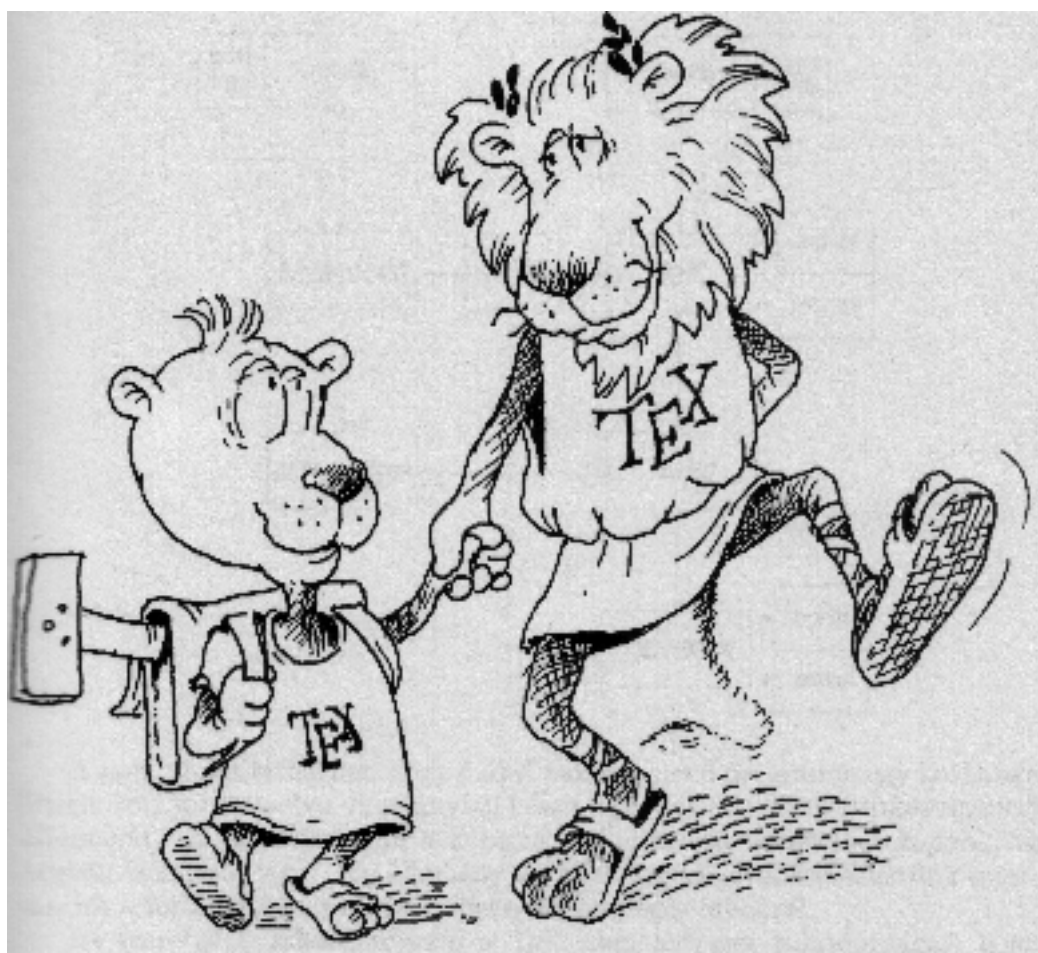
Ilyen esetben általában elegendő megoldás az, hogyha bekapcsoljuk a lilo linear opcióját az `/etc/lilo.conf`-ba írt `linear` kulcsszóval, vagy az `/sbin/lilo -l` opcióval. Ennek hatására a `map` file-ba nem a C/F/S cím, hanem a lineáris szektorcím kerül bele, és ezek alapján a

betöltő program a boot-olás során közvetlenül a BIOS-tól lekérdezett geometria szerint alakítja ki a C/F/S címeket.

Tulajdonképpen nem értem, hogy miért nem ezt a megoldást használja alapértelmezésben a lilo. Ha valaki tudja ezt, vagy van, akinél a linear opció ellenére sem működik a boot-olás, ossza meg velem a tapasztalatait.

12. fejezet

Biztonság



12.1. Biztonsági kérdések

„Számítógép-biztonság (computer security) alatt szűkebb értelemben az adatok illetéktelenek hozzáférésétől való védelmét, elsősorban a titkosságot (secrecy, confidentiality) értik.

Tágabb értelemben az alábbi hármat értjük rajta:

- titkosság és hozzáférési kontroll (access control),
- integritás (sértetlenség - integrity, pontosság - accuracy, hitelesség - authenticity),
- elérhetőség (availability),

és gyakran még egy negyedik szempontot:

- megbízhatóság (reliability), melyen a hardver, a szoftver s a szolgáltatások üzembiztonságát értjük, azaz, hogy rendszerünk azt, és úgy végzi-e, amint az elvárható/elvárt, elfogadható karbantartási igény és meghibásodásszám mellett. Ez több, mint a rendszer elérhetősége.”

12.1.1. Fizikai biztonság

Ez több részből tevődik össze.

A számítógép lezárása. Szinte minden modern PC-be bele van építve a lezárás lehetősége. A számítógép elején vagy hátulján van egy kulccsal elfordítható zár, ez megakadályozza a gép kinyitását, így meggátolja, hogy idegen hardvert téve a gépbe, használjuk azt (ez lehet akár egy floppymeghajtó is).

BIOS biztonság A BIOS a legalacsonyabb szintű szoftver, ami hozzáfér az x86 alapú számítógépek hardveréhez. A LILO és egyéb Linuxos boot programok használják a BIOS-t a rendszer elindításához. A rosszfiú képes a BIOS-on keresztül manipulálni a rendszert, ha el tudja indítani azt. A PC-k mindegyike rendelkezik azzal a lehetőséggel, hogy jelszavakat képes rendelni a rendszer elindításához és a BIOS behívásához.

Boot betöltő A különböző linux boot betöltő programnál lehetőségünk van jelszavakat rendelni az indulási folyamathoz. (Például a LILO restricted és password beállításai.)

xlock és vlock Ha otthagysz a számítógépet egy kis időre, de nem akarsz kikapcsolni, akkor jó választás a képernyő zárolása. Két program van erre a célra: konzolon a *vlock* (??), grafikus felületen pedig az *xlock*. Természetesen a zárolás nem befolyásolja a munkádat, de ha a zárolás közben újraindítják a gépet, akkor az természetesen leállít minden futó programot, beleértve a te zároló programodat is.

A fizikai biztonság ellenőrzése Először meg kell nézni, hogy mitől indult újra a számítógép. (Ez a rendszer nagyon stabil és robosztus, csak akkor muszáj újraindítani, ha hardvert cserélsz, új kernelt szeretnél bootolni, vagy valami hasonló nagyságrendű dolog miatt.) Azután ellenőrizni kell a log-fájlokat. Amiket vizsgálni kell:

- rövid vagy nem teljes log-fájlok,
- a log-fájlok időbélyegei,
- sikertelen hozzáférések,
- szolgáltatások újraindulása,
- hiányzó log-fájlok,
- su próbálkozások és sikertelen bejelentkezések.

A megnézendő log-fájlok a */var/log/* könyvtárban és alkönyvtáraiban találhatóak.

12.1.2. Helyi biztonság

Új felhasználók létrehozásakor csak a szükséges privilégiumokat adjuk meg. Ha lehetséges, akkor korlátozzuk, hogy mikor használhatják a számítógépet, ellenőrizzük a bejelentkezéseiket, és hogy mennyi időt töltöttek a számítógépen. Ha szükséges, akkor naplózhatjuk a gépen végzett tevékenységüket is. A megszűnő accountokat töröljük, az ideiglenesen nem használt accountok használatát akadályozzuk meg (a */etc/passwd* fájlban a jelszó helyére (vagy elé) tegyünk egy * (csillag) karaktert, ezzel megakadályozva, hogy be lehessen jelentkezni).

Nagyon fontos, hogy a rendszer fájljait nem kell tudnia minden felhasználónak olvasni, végrehajtani; írni pedig különösen nem! Ha lehetséges egy szolgáltatást korlátozni, akkor a lehető legtöbb korlátozást használjuk is ki. Szerintem jó hozzáállás az – a rendszer biztonságának szempontjából –, hogy alapesetben minden tiltott, és csak azt engedélyezzük, amire feltétlenül szükségünk van.

Hozzáférési jogosultságok

A fájlokhoz tartozó hozzáférési jogosultságok meghatározzák, hogy melyik felhasználó melyik fájlra hajthat végre műveletet, és még a műveletet is meghatározza. A Linux a felhasználókat három csoportra osztja, amikor a fájlokhoz való viszonyukat vizsgálja:

- a fájl tulajdonosa (user),
- csoport (group),
- egyéb (others).

A „csoport” segítségével csoportosíthatjuk a felhasználókat – lehetőleg – valamilyen logika szerint, hogy meg tudják osztani egymással a megfelelő fájljaikat.

Erről részletesebben olvashatunk itt – ?? . fejezet.

A root biztonsága

Mivel a *root* (rendszeradminisztrátor) a rendszer korlátlan ura, az ő jogainak megszerzésére indul a legtöbb támadás. Ezért kell nagyon odafigyelnünk, amikor rootként dolgozunk – ezt csak az elengedhetetlenül szükséges esetekben tesszük (például a rootnak érkező leveleket át lehet irányítani másik felhasználónak, így a leveleket sem kell rootként olvasgatnunk). Kisebb feladatok (például fájlrendszerek becsatolása) végrehajtására használhatjuk a *sudo* parancsot.

Jelszavak

Az általam használt rendszer (Debian) alapértelmezésben megköveteli, hogy a jelszavak hossza 5 és 8 betű közé essen, legyen benne kisbetű, nagybetű és szám. A */etc/login.defs* és a */etc/login.access* fájlok a *login* (bejelentkező program) csomag konfigurációs fájljai. Ezekben lehet meghatározni mindent, ami a bejelentkezésre vonatkozik. Egy magyar kommentekkel ellátott példány található belőlük és a ?? . fejezetben.

A *shadow* csomag segítségével biztonságosabbá lehet tenni rendszerünket, mert így a jelszavakat nem a */etc/passwd* fájlban tároljuk, hanem a */etc/shadow* fájlban – ami csak a *root* és a *shadow* csoport számára olvasható. A */etc/passwd* fájl mindenki számára olvasható, tehát ha abban tárolnánk a jelszavakat, akkor mindenki el tudná érni a kódolt jelszavakat. Vannak olyan programok, amelyek segítségével fel lehet törni a jelszavakat, legalábbis azokat, amik nem jól vannak megválasztva.

12.1.3. Az X Window System biztonsága

Ezzel a kérdéssel a ?? . fejezetben foglalkozom.

12.1.4. SVGA biztonsága

Az SVGAlib program tipikusan `suid root` program, hogy közvetlenül hozzá tudjon férni a gép videóhardveréhez. Ezért nagyon veszélyes program. Ha összeomlik, akkor újra kell indítanod a számítógépedet. Jobb, ha nem futtatsz ilyen programokat.

12.1.5. Biztonsággal kapcsolatos programok

Egy pár programról szeretnék szólni, amelyek megkönnyíthetik a dolgunkat, amikor ellenőrizni akarjuk a rendszerünk beállításait biztonsági szempontból.

Tripwire

Nem foglalkozik a rendszer biztonsági vizsgálatával, csak fájlok és könyvtárak integritásának vizsgálatát teszi lehetővé.

A kijelölt fájlokról (kívánság szerint több módszerrel) úgynevezett digitális ujjlenyomatot készít. Az ujjlenyomat vételénél a fájl minden egyes bájtja számít. Beállíthatjuk azt is, hogy a fájlnak melyik tulajdonságának megváltozásáról nem akarunk értesítést kapni.

Először megfelelően be kell állítanunk a `/etc/tripwire/` könyvtárban található `tw.config` konfigurációs fájlt. A

```
tripwire -initialize
```

parancs kiadása hozza létre az adatbázist. Az elkészült adatbázist olyan helyre kell elraknunk, ahol nem férhetnek hozzá illetéktelenek. Az adatbázis, a konfigurációs fájl és a programfájlok módosításával érhető el a program olyan módosítása, hogy az már a rendszer biztonságát veszélyeztesse.

A legcélszerűbb megoldás (amit a konfigurációs fájlban is javasolnak), hogy egy floppy-lemezre mentjük el az adatbázist, amit aztán eltávolíthatunk a rendszerből.

Satan

A név egy rövidítés, a Security Analysis Tool for Auditing Networks-nek a rövidítése. Távoli gépek és hálózatok szolgáltatásainak biztonságtechnikai szempontból történő ellenőrzésére szolgál.

Rosszul beállított hálózati szolgáltatásokat vizsgál, ismert hibák után kutat a rendszerben és a hálózati programokban. A potenciális biztonsági problémákat vizsgálja.

Az eredményt valamilyen böngészőprogrammal analizálhatjuk, használhatunk lekérdezéseket is.

Sniffit

Ez a program demonstrálja, hogy a TCP protokoll mennyire nem biztonságos. Lehetőséget nyújt különböző protokollok által szállított anyagok megfigyelésére.

Használhatjuk parancssorból, de akár interaktívan is. Rengeteg parancssori kapcsolója van, de konfigurálható fájlon keresztül is (a konfigurációs fájl nevét meg kell neki adni). Különböző hálózati eszközök megfigyelésére is képes.

Fájlokba tudja menteni a megfigyelés eredményeit (logolás), és több logolási szintet ismer.

Néhány fontos dolog még róla:

- csak a root tudja futtatni,
- kívülről (másik gépről) nem lehet észlelni a működését,
- belülről (a futtató gépen) a processzlistában látszik,
- a kernelben benne kell lennie a System V IPC támogatásnak.

12.1.6. Biztonság a hálózatokon

Szólni szeretnék azokról a programokról, amik lehetővé teszik a biztonságos kapcsolat-tartást nyilvános hálózatokon keresztül is.

Az SSH magyar nyelvű leírásának copyright információi megtalálhatóak itt – ???. fejezet.

SSH - a biztonságos alternatíva az RSH helyett

[?]

Miért használjunk ssh-t az rsh helyett?. A (világ)hálózaton dolgozva nap-mint-nap kell bejelentkeznünk egyik számítógépről egy másikra. Ehhez leggyakrabban a *telnet* és *rsh/rlogin* parancsok valamelyikét használjuk, noha azok biztonsági szempontból még csak nem is kielégítőek. Például közbülső gépek lehallgathatják az adatforgalmat, beleértve a password-öket is – a *telnet*, *rsh* nem titkosít –, de sok más módszer is létezik jogosulatlan hozzáférés megszerzéséhez.

A finn Tatu Ylönnen által írt *ssh* programcsomag (*ssh*, *slogin*, *scp*) funkcionálisan az *rsh* (*rsh*, *rlogin*, *rcp*) helyettesítője úgy, hogy biztonságos, erős autentikációval ellenőrzött titkosított kapcsolatot hoz létre két, „egymásban” nem bízó gép között, amelyeket a nem-biztonságosnak tekintett hálózat köt össze. Az *ssh* a következő típusú támadások ellen védett:

- IP cím hamisítás, amikor egy távoli gép olyan IP csomagokat küld, mintha azok egy másik gép csomagjai lennének.
- IP source routing, amikor egy gép úgy tesz, mintha egy távoli gép IP csomagja rajta keresztül érkezne.
- DNS hamisítás, amikor name server bejegyzéseket hamisítanak
- az adatforgalom közbülső gépek általi lehallgatása ellen
- az IP csomagok közbülső gépek általi megváltoztatása ellen
- X autentikációs adat lehallgatása és meghamisítása ellen

Az *ssh* rhosts-RSA autentikációs módszerét használva semmiben sem különbözik a felhasználók számára az *rsh*-től – akár *rsh/rlogin/rcp*-ként is installálható. Használható *ssh*-t nem ismerő gépekhez kapcsolódva is, mivel az *ssh*, amennyiben nem találja az *ssh* szervert a másik gépen, elindítja az *rsh*-t önmaga helyett.

Az *ssh* természetesen semmifajta védelmet nem tud nyújtani akkor, ha valaki hozzáfér az adatainkhoz a számítógépünkön, vagy ha valaki már betört a gépre, és azon „root” jogokkal rendelkezik – a hálózaton keresztüli támadások kivédésére készült.

Az *ssh* szerver és kliens Unix rendszereken működik és ingyenesen használható; kliens létezik OS/2 és MS-Windows-ra is, az utóbbira a hivatalos változat azonban kereskedelmi termék.

Hogyan működik? Az *ssh* RSA kulcsokon alapul. Minden *ssh*-t használó gépnek van egy host-azonosító RSA kulcsa (alapértelmezésben 1024 bit). A szerver gépen az *sshd* démon ezen kívül generál egy szerver RSA kulcsot is (alapértelmezésben 768 bit), amelyet óránként frissít, és amit soha nem tárol a merevlemezen.

Amikor egy kliens (*ssh*) hozzákapcsolódik a szerverhez (*sshd*), először a szerver azonosítása történik meg. A szerver elküldi a host- és szerver-kulcsok publikus részét a kliensnek. A kliens összehasonlítja a host-azonosító publikus kulcsot az adatbázisában lévővel és ellenőrzi, hogy az változatlan-e. Ezután a kliens generál egy 256 bites véletlenszámot, amit a szerver host- és szerver-kulcsával egyaránt titkosít, majd ezt visszaküldi a szervernek. A szerver az RSA kulcsai ismeretében vissza tudja fejteni a titkosított véletlenszámot, amit a továbbiakban a két oldal a forgalom titkosító kulcsául (session key) fog használni – ettől a ponttól kezdve minden titkosított, IDEA, DES, 3DES, ARC-FOUR (RC4) vagy TSS algoritmust használva.

A következő lépés a kliens-azonosítása (autentikáció). Több módszer áll ehhez a rendelkezésünkre:

- *rhosts* autentikáció: Ha a kliens gép szerepel az */etc/hosts.equiv* fájlban és a felhasználói azonosítók megegyeznek a két gépen, vagy a kliens gép és a felhasználó azonosítója szerepel a *~/.rhosts* vagy *~/.shosts* fájlok valamelyikében, akkor a bejelentkezés engedélyezett. (Ez az autentikációs módszer nem használatos, mert az *ssh* ekkor az *rsh*-nál nem nyújtana nagyobb biztonságot.)
- *rhosts* és *host-RSA* autentikáció (az elsődlegesen használt autentikációs mechanizmus): Amennyiben a bejelentkezés engedélyezett lenne *rhosts* autentikációval, a szerver még ellenőrzi a kliens *host*-azonosító RSA kulcsát, úgy, ahogy a kliens ellenőrizte a szerverét. Ez a mechanizmus kivédi az IP hamisítás, DNS hamisítás és source routing támadási módszereket.
- *user-RSA* autentikáció: az *ssh* támogatja azt, hogy a felhasználók definiáljanak saját RSA kulcsokat, amelyeket azután távoli gépekre *ssh*-val való bejelentkezés-kor önmaguk azonosítására használhatnak.
- *password* alapú autentikáció: ha az előző módszerek egyikével sem sikerült azonosítani a felhasználót, az *ssh* kéri a felhasználó *password*-jét. A *password* titkosítottan kerül át a szerverre, így lehallgatni nem lehet.

Ha a kliens sikeresen azonosította magát, akkor a kapcsolat előkészítéseként különböző szolgáltatásokat kérhet a szervertől: pszeudo-terminál allokálását, az X11-es kapcsolatok átirányítását a biztonságos csatornára, tetszőleges TCP/IP kapcsolatok átirányítását a biztonságos csatornára vagy az úgynevezett authentication agent kapcsolat átirányítását a biztonságos csatornára.

Végül a kliens vagy egy shell indítását (*slogin*, *ssh*) vagy egy parancs végrehajtását (*ssh*, *scp*) kérheti a szervertől.

A fázisokat jól illusztrálja az *ssh* verbose módban való indítása:

```
kadlec@blackhole:~$ ssh -v sunserv
SSH Version 1.2.13 (i486-unknown-linux), protocol version 1.3.
Standard version. Does not use RSAREF.
Reading configuration data /home/kadlec/.ssh/config
Applying options for *
Reading configuration data /usr/local/etc/ssh_config
ssh_connect: getuid 648 geteuid 0 anon 0
Connecting to sunserv [148.6.0.5] port 22.
Allocated local port 1022.
Connection established.
Remote protocol version 1.3, remote software version 1.2.13
Waiting for server public key.
Received server public key (768 bits) and host key (1024 bits).
```

```

Host 'sunserv' is known and matches the host key.
Initializing random; seed file /home/kadlec/.ssh/random_seed
Encryption type: idea
Sent encrypted session key.
Received encrypted confirmation.
Trying rhosts or /etc/hosts.equiv with RSA host authentication.
Remote: Accepted by .shosts.
Received RSA challenge for host key from server.
Sending response to host key RSA challenge.
Remote: Rhosts with RSA host authentication accepted.
Rhosts or /etc/hosts.equiv with RSA host authentication accepted by server.
Requesting pty.
Requesting X11 forwarding with authentication spoofing.
Requesting authentication agent forwarding.
Requesting shell.
Entering interactive session.
Last login:
No news.
You have mail.
kadlec@sunserv:$ exit
Connection to sunserv closed.
Transferred: stdin 14, stdout 1109, stderr 31 bytes in 40.4 seconds
Bytes per seconds: stdin 0.3, stdout 27.5, stderr 0.8
Exit status 0

```

Mit tud még az ssh?.

X11 átirányítás. Az egyik legkényelmesebb tulajdonsága az *ssh*-nak, hogy X Window grafikus felület használata esetén (a lokális gépen a `DISPLAY` változó be van állítva) a távoli X11-es kapcsolatokat automatikusan átirányítja a biztonságos titkosított csatornára. Továbbá, hogy az X11 szerver autentikációs adatai ne hagyják el a szervert, az *ssh* véletlenszerű „cookie”-t generál, és azt küldi át a hálózaton, természetesen titkosítottan:

```

-----
| X11 kliens, amely az |
| ssh szerveren fut    |
-----

```

```

|
|

```

```

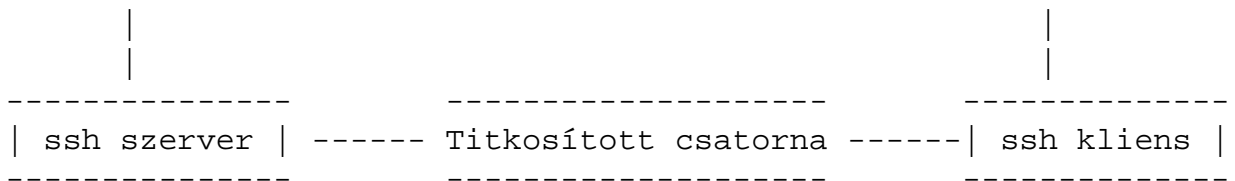
-----
| A valódi, fizikai   |
| X display           |
-----

```

```

|
|

```



Nem kell bajlódniunk sem a `DISPLAY` változó beállításával, sem `xhost` vagy `MAGIC-COOKIE X Window` autentikációkkal, mert az `ssh` mindent elvégez helyettünk. Egyszerűen végrehajthatjuk a következő parancsot,

```
kadlec@blackhole:$ ssh sunserv xterm
```

és az X terminál a képernyőnkön minden további nélkül megjelenik.

TCP/IP port átirányítás. Az X11-es kapcsolatok automatikus átirányítása a TCP/IP portok átirányításának egy speciális esete. Például az *ftp* parancsot használhatjuk úgy, hogy a password-ünk ne titkosítatlanul haladjon át a hálózaton a gépünk és a távoli gép között, hanem az *ssh* által titkosítottan. Ehhez grafikus felületen csak a következőt kell tennünk:

- Egy ablakban végrehajtjuk a következő parancsot (a lokális port száma tetszőleges):

```
kadlec@blackhole:~$ ssh -L 1234:sunserv:21 sunserv
```

Ezzel a paranccsal bejelentkeztünk a sunserv gépre és ugyanakkor a sunserv 21-es portját, amely az *ftp* parancs-portja, átirányítottuk a lokális gép 1234-es portjára a titkosított csatorna fölött.

- Egy másik ablakban pedig a következőt:

```
kadlec@blackhole:~$ ftp localhost 1234
Connected to localhost.
220 sunserv FTP server (Version wu-2.4(6)
Fri Jun 16 11:25:25 MET DST 1995) ready.
Name (localhost:kadlec):
```

Kész! Az *ftp* parancs-portja így titkosított, password-ünk nem-lehallgathatóan fog áthaladni a hálózaton.

Mail-szervert használva levelező programjaink minden egyes kapcsolatnál titkosítatlanul küldik át password-ünket a szervernek. Az *ssh* port-átirányítását használva a titkosítás ezeknél is megoldható. Stephane Bortzmeyer Perl-scripje *POP*, míg Holger Trapp program-csomagja *pine* és *IMAP* esetén oldja meg a problémát *ssh* segítségével; sem a mail-szerver, sem a kliens nem igényel semmilyen módosítást:

```
ftp://ftp.pasteur.fr/pub/Network/gwpop/  
ftp://ftp.kfki.hu/pub/packages/security/ssh/contrib/imap_sec.tgz
```

Tömörítés. Lassú vonalak fölött igen hasznos lehet az *ssh* azon képessége, hogy képes tömöríteni a teljes forgalmat, beleértve az átírányított X11 és TCP/IP portok forgalmát is.

Felhasználói RSA kulcsok. Az *ssh* esetében felhasználói RSA kulcsok az *ssh-keygen* paranccsal hozhatók létre. Egy RSA kulcshoz „password” helyett tetszőleges hosszúságú „passphrase” rendelhető. Az *rhost* és RSA autentikációhoz képest az RSA autentikáció a felhasználótól több (de egyszeri) konfigurációs munkát igényel:

rhost autentikáció. A távoli gépen az *./rhosts* vagy *./shosts* fájlhoz megfelelő

```
host user
```

bejegyzéseket kell adni.

RSA autentikáció.

- A lokális gépen:
 - Az *ssh-keygen* paranccsal RSA kulcs(ka)t kell létrehozni
 - Módosítani kell a bejelentkezési folyamatot úgy, hogy az az *ssh-agent* programmal kezdődjön és minden más program (shell) abból induljon: Például X session (*xdm*) esetén:
~/.xsession fájl:

```
#!/bin/sh  
exec ssh-agent $HOME/.xsession.real
```
 - Vagy a prompt-nál, vagy az *./xsession.real*-ből, vagy menüből stb. végrehatjuk az *ssh-add* parancsot, amellyel a lokális gépen futó *ssh-agent*-hez hozzáadjuk az általunk használt RSA kulcsot a megfelelő passphrase megadásával.

- A távoli gép(ek)en az `./ssh/authorized_keys` fájl(ok)hoz hozzáadjuk az RSA kulcsunk publikus részét, amely a lokális gépen az `./ssh/identity.pub` fájlban van.

Az elképzelés az RSA autentikáció mögött a következő: az *ssh-agent* a felhasználó lokális gépén fut, és kezdetben semmilyen RSA kulcsot nem ismer - azokat az *ssh-add* programmal kell hozzáadni. (Tetszőleges számú kulcs használható.) Mivel az RSA kulcsokhoz a kulcsmondatokat a lokális gépen kell megadni, azok soha nem haladnak át a hálózaton. A távoli gépek az RSA kulcsunk publikus részével azonosítják magukat és a kapcsolatok az *ssh-agent*-hez automatikusan a titkosított csatornára irányítódnak.

Az RSA autentikáció különösen akkor hasznos, amikor az *ssh*-t olyan gépen szeretnénk használni, amelyen nincs *ssh* és a gép adminisztrátora valamiért nem hajlandó azt installálni. Az *ssh* ekkor közönséges felhasználóként lefordítható, installálható és RSA valamint password autentikációval használható. (Ahhoz, hogy az rhost-RSA autentikáció működjön, az *ssh*-t „root”-ként kell telepíteni.)

Honnan tölthető le Magyarországon? <ftp://ftp.kfki.hu/pub/packages/security/ssh>

Hol található a témáról további információ az Interneten?

- Ssh Home Page <http://www.cs.hut.fi/ssh/>
- Ssh FAQ <http://www.uni-karlsruhe.de/ig25/ssh-faq/>
- RSA FAQ <http://www.rsa.com/rsalabs/faq/>

Pretty Good Privacy – Nyilvános kulcson alapuló titkosítás a tömegek számára (PGP felhasználói leírás)

Philip Zimmermann (szerző)

Czakó Krisztián (fordító) (1993/06/14)

Alapvető fontosságú dolgok . PGP Version 2.3a - 1 Jul 93 Programozta: Philip Zimmermann Branko Lankester, Hal Finney, és Peter Gutmann segítségével

Kivonat . A PGP e-mail levelek illetőleg adatfájlok nyilvános-kulcsos titkosítását teszi lehetővé. Ismeretlen, soha nem látott emberekkel való biztonságos levelezést anélkül, hogy azokkal előzőleg biztonságos csatornán kulcsot kellene cserélni. A PGP széleskörű szolgáltatásokkal van ellátva és gyors, magában foglalja a kulcsfájlkezelést, az elektronikus aláírást, az adatok tömörítését is.

A programra és a dokumentációra ©Copyright 1990-1992 Philip Zimmermann.

A PGP forgalmazásával, terjesztésével, használatával, szabadalomjogi kérdésekkel, felelősségkorlátozással és az exporttal kapcsolatos kérdésekkel a dokumentáció II. részében foglalkozik egy fejezet.

Gyors áttekintés . A Phil's Pretty Good Software által írt Pretty Good(tm) Privacy (PGP) egy nagybiztonságú titkosító program, mely MSDOS, Unix, VAX/VMS, és sok más operációs rendszer alatt fut. Lehetővé teszi adatok és üzenetek autentikus, titkos és irányított célbajuttatását. Az irányított ebben a szóhasználatban azt jelenti, hogy kizárólag azok férnek hozzá az adatok tartalmához, akiknek azt szántuk. Az autenticitás azt jelenti, hogy a fogadó bizonyos lehet abban, hogy az adatokat ki adta fel, és hogy az adatok nem módosultak. Mindehhez nincs szükség arra, hogy a felek hagyományos titkosítási eljárásoknál szokásos titkos kulcsokat cseréljenek egymással, és azokat megbízható csatornán állandóan szintentartsák. A PGP-nél egyáltalán nincs szükség biztonságos csatornára a kulcsok terítéséhez. Mindez azért van, mert a PGP egy új, úgynevezett „nyilvános kulcsos” titkosítási eljárás alapul.

A PGP-ben együtt vannak a Rivest-Shamir-Adleman (RSA) algoritmuson alapuló nyilvános kulcsos titkosítás előnyei a hagyományos eljárások sebességével, a kivonatokon (message digest, MD) alapuló digitális aláírással, a titkosítás előtti tömörítéssel, valamint az ergonomikus környezettel és kulcs-adatbázis kezelő funkciókkal. A PGP a széles tömegek számára készült.

A PGP nem tartalmaz beépített modem-szolgáltatásokat. Erre a célra más szoftvert kell igénybe venni.

A dokumentáció ezen része csak a használathoz nélkülözhetetlenül szükséges információkat tartalmazza. Ezt minden használónak el kell olvasnia, és meg kell értenie. A II. rész foglalkozik a speciálisabb szolgáltatásokkal, és bővebb betekintést ad a technikai kérdésekbe is. Magukat a felhasznált algoritmusokat egyik dokumentáció sem írja le, csak hivatkozik rájuk.

Miért kell neked a PGP? . Személyes dolog. Magánügy. Senkinek semmi köze hozzá. Politikai kampányt folytatsz, átgondolhatod az adóbevallásodat, üzleti terveidet. Vagy valami olyat, ami szerinted normálisan legális lenne, de nem az. Vagy bármilyen hasonló esetben az az igényed, hogy a privát levelezésedet vagy bizalmas adataidat ne olvashassa el más. Semmi rossz nincs abban, hogy véded a privát adataidat. Ez a magánélethez való jogok egy része, amelyek az Alkotmányban is le vannak fektetve.

Esetleg úgy gondolod, hogy semmi takargatnivalód nincs, és e-mailed nem szorul titkosításra. De ha ilyen teljesen törvénytisztelő polgár vagy, a postai levelezésedet miért nem nyílt levelezőlapon bonyolítod? Úgy gondold, hogy amennyiben borítékot használsz, az arra mutat, hogy rejtegetsz valamit? Különben talán magadra vonnád a rendőr-

ség figyelmét? Akkor rögtön rejtőzködő kábszerkereskedő vagy? Vagy paranoiás fickó? Szóval szüksége lehet egy törvénytisztelő polgárnak a PGP-re?

Mert mi lenne akkor, ha valóban mindenki csak levlapot használna? Akkor az a külön, aki borítékba rakja a postáját nyilván rögtön gyanús lenne. A hatóságok talán fel is bontanák a küldeményét, hogy megvizsgálják, mit is rejteget. Szerencsére nem ilyen világban élünk, hiszen szinte mindenki borítékot használ, így senkire nem vetül gyanú pusztán azért, mert privát levelezését eltakarja a nyilvánosság elől. A biztonság a széles elterjedésen múlik. Hasonlóan, ha mindenki elkezd titkosítva küldeni az elektronikus postáját, legyen az valóban bizalmas, vagy nem, ez ugyanúgy nem fog feltűnést kelteni.

Manapság, ha a hatóság úgy dönt, hogy behatol valakinek a magánéletébe, komoly erőforrásokat és munkát kell áldoznia arra, hogy felbontsa és elolvassa az illető papír alapú postáját, lehallgassa és értelmezze telefonbeszélgetéseit. Az ilyen jellegű megfigyelés nagyon drága és kis hatékonyságú, ha emberek széles csoportja ellen akarnák alkalmazni. Ezért ilyen eszközökkel csak a valóban indokolt esetekben élnek.

Ugyanakkor a kommunikáció jelentős része lassanként áttevődik az elektronikus formára. Az e-mail átveszi a hagyományos posta szerepét. És az így küldött üzeneteket nagyon egyszerű monitorozni, és kulcsszavakra kereső programok segítségével kiszűrni a valóban megfigyelni kívánt részeket. Embertömegek levélforgalmát lehet ilyen módszerekkel teljesen rutinszerűen, automatikusan és láthatatlanul átvizsgálni. Az NSA már ma is megfigyelés alatt tartja a nemzetközi levelezőhálózatokat.

A jövőben a hálózatok egyre nőni fognak, míg szinte minden ember személyi számítógépe valamilyen formában egy nagy közös hálózat részévé válik. Az e-mail minden ember szokásos kommunikációs eszköze lesz. A kormány majd saját fejlesztésű titkosító eljárásával fogja titkosítani a magánemberek levelezését. És az emberek nagy többsége bízni fog ezekben az eljárásokban. De biztosan lesznek olyanok, akik inkább bíznak saját eljárásaikban.

1991-ben a Szenátus 266. sz. beadványa, ami a bűnmegelőzésről szólt, érdekes dolgokat vetett fel. Amennyiben ezek törvényerőre emelkednének, akkor a gyártók csak olyan kommunikációs berendezéseket forgalmazhatnának, melyek titkosítása beépített „hátsó bejárat” rendelkezik. És a kormány szükség esetén visszafejthetné bárki rejtett levelezését. Az ominózus szöveg így szól: „A Kongresszus azon a véleményen van, hogy az elektronikus kommunikációs szolgáltatást nyújtóknak és az ezzel kapcsolatos berendezések gyártóinak lehetőséget kell biztosítaniuk arra, hogy a Kormány – törvényben körülhatárolt esetekben – hozzájuthasson a berendezéseken áramló információkhoz eredeti, titkosítatlan formában.” A beadvány végül is a civil szabadságjogok védői valamint a gyártók éles tiltakozásának hatására kisebbségben maradt.

1992-ben került a Kongresszus elé az FBI digitális telefonok lehallgatásáról szóló (törvény)javaslata. Ennek alapján a gyártóknak speciálisan a lehallgatást lehetővé tevő portot kellene építeni minden készülékbe, ezzel lehetővé téve, hogy az FBI az irodájából

hallgasson le bárkit, akit akar. Annak ellenére, hogy a javaslat egyáltalán nem talált támogatókat, 1993-ban újra be fogják nyújtani.

A legfélelmetesebb pedig a Fehér Ház új titkosításokról szóló irányelv-tervezete, melyet az NSA készített elő négy év alatt, és 1993. április 16-án hozták nyilvánosságra. A javaslat központi témája egy, a kormány által tervezett eszköz, az ún. „Clipper” chip, mely az NSA által újonnan kidolgozott titkosítási eljárást alkalmazná. És a Kormány jelentősen támogatja mindazon gyártókat, akik a kódolt kommunikációs berendezésekben (telefonok, faxok, stb.) mindenütt ezt a chipet alkalmazzák. Az AT&T most ezt építi minden „biztonságos” készülékébe. És ami a lényeg: a gyártáskor minden egyes chipbe beleégetik az egyedi rejtjelkulcsot, de erről a kormány egy másolatot őriz elzárva. De semmi ok az aggodalomra – hiszen a Kormány becsületszavát adja, hogy ezt a kulcsot kizárólagosan a törvényben meghatározott esetekben fogja üzeneteid lehallgatásához felhasználni. Természetesen ahhoz, hogy a dolog igazán effektív legyen, a következő logikus lépés az lesz, hogy betiltanak minden alternatív titkosítási módszert.

Amennyiben a magánélet törvényen kívülre szorul, csak a törvényen kívülieknek lesz magánélete. A hírszerző szervezetek természetesen hozzájuthatnak a jó titkosító eljárásokhoz. Hasonlóképpen a fegyver- és drogkereskedők. Meg a hadiipari cégek, az olajmultik és hasonló gigászok. Ezzel szemben a közönséges polgárnak nincs lehetősége, hogy tisztességes, „military grade” nyilvános kulcsos titkosító eljáráshoz jusson. Egészen eddig a pillanatig.

A PGP azt a lehetőséget nyújtja az embereknek, hogy saját kezükbe vegyék a saját magánéletüket. Egyre nagyobb erre az igény. Ezért írtam meg ezt a programot.

Hogyan működik . Ez a pont valószínűleg inkább csak azoknak segít, akik már egyébként is tisztában vannak a titkosító eljárásokkal, különös tekintettel a nyilvános kulcsosakra. Mindazonáltal itt van egy kis összefoglaló a témában.

Először is az elemi szakszavak: Tegyük fel, hogy én akarok küldeni neked egy üzenetet, és azt akarom, hogy senki más ne olvashassa azt el. Ekkor először is „rejtjelzem”, avagy „kódolom”, ami által egy reménytelen zagyvaság lesz belőle, amit senki nem tud elolvasni, kivéve természetesen téged. A kódoláshoz egy „kulcsot” használok, amit neked is használnod kell, amikor az üzenetet visszafejted, vagyis „dekódolod”. Legalábbis így működik a hagyományos rejtjelzés.

A hagyományos kriptográfiában, mint amilyen pl. a DES algoritmus (US Federal Data Encryption Standard), ugyanazt a kulcsot kell használni a kódoláshoz és a dekódoláshoz. Ami viszont azzal jár, hogy a kulcsot először egy megbízható csatornán el kell juttatni a fogadó félhez, csak utána lehet nekikezdeni a titkosított üzenetek megbízhatatlan csatornákon való küldésének. Ez sok esetben kényelmetlen lehet. Másrészt pedig, ha a kulcsküldéshez van biztonságos csatornád, tulajdonképpen miért is van olyan nagy szükség a titkosításra?

A nyilvános kulcsos kriptográfiában mindenki két, egymással szorosan összefüggő kulccsal rendelkezik. Az egyik kulcs nyilvános, míg a másikat szigorúan titokban kell tartani. Mindkét kulcsot a másik által kódolt üzenetek dekódolására lehet felhasználni. A nyilvános kulcs nem nyújt lehetőséget a titkos kulcs kitalálásához. Ezért a nyilvános kulcsot fenntartások nélkül széles körben el lehet terjeszteni a kommunikációs hálózatokon. Ez az eljárás mindennemű biztonságos csatorna nélkül képes szavatolni a titkosságot.

Egy ember nyilvános kulcsával bárki kódolhat üzeneteket. Ezeket a fogadó a saját titkos kulcsával tudja kibontani. Ő és senki más, hiszen a titkos kulccsal senki más nem rendelkezik. A kódolt üzenetet még a feladó sem tudja dekódolni.

Meg van oldva az üzenetek biztonságos sértetlenség- és eredetigazolása (autentikációja) is. A küldő saját titkos kulcsával „aláírja” az üzenetet. Az aláírás eredetiségét bárki ellenőrizheti az illető nyilvános kulcsának felhasználásával. A fogadó a dekódolás után bizonyos lehet a küldő személyében, és abban is, hogy az üzenet tartalma nem változott meg. Mindezt azért, mert ehhez a feladó titkos kulcsa szükséges, mellyel rajta kívül senki nem rendelkezik. A hamisítás ki van zárva, és a küldő nem tagadhatja le magát utólag.

A két dolgot egybe is lehet kötni, amikor is az eredmény biztonság és autentikusság egy flakonban. Ehhez először alá kell írni az üzenetet a saját titkos kulccsal, majd kódolni a fogadó nyilvános kulcsával. A fogadó pedig először a saját titkos kulcsával dekódol, majd a küldő nyilvános kulcsával ellenőrzi az eredetet. És ezt a két lépést a program önműködően megteszi.

Miután a nyilvános-kulcsos titkosítás algoritmusa sokkalta lassabb, mint a hagyományos eljárások, célszerű egy nagy megbízhatóságú hagyományos eljárással kódolni az üzenetet. {Az eredeti, kódolatlan üzenetet nyersszövegnek (plaintext) nevezik legalábbis én – jobb híján :-)}. A titkosítás első lépésében egy hagyományos kódoláshoz használható véletlen kulcs generálódik, ami kizárólag abban az egy küldésben kerül alkalmazásra. Ezzel kódolják a nyersszöveget egy hagyományos algoritmussal. A felhasznált kulcsot pedig a fogadó nyilvános kulcsával rejtjelzik, és a rejtjelzett kulcsot a kódolt üzenettel (a továbbiakban „kódszöveg”) együtt küldik el. A fogadó a saját titkos kulcsával először dekódolja az ideiglenes kulcsot, majd ennek segítségével a gyors algoritmussal dekódolja a kódszöveget.

A kulcsok speciális struktúrában vannak tárolva, mely tartalmaz egy azonosítót (userId – a személy neve), a kulcspár generálásának dátumát, és magát a kulcskódot. A nyilvánoskulcs-struktúra tartalmazza a nyilvános kulcsot, míg a titkoskulcs-struktúra a titkos kulcsot. A titkos kulcs ráadásul még egy saját jelszóval is védve van ellopás esetére. A kulcsfájl („key ring”) egy vagy több kulcs-struktúrát tartalmaz. Az előbbieknek megfelelően vannak titkos és nyilvános kulcsfájlok.

A program egy azonosító számmal, a kulcs ID-vel hivatkozik az egyes kulcsokra, ami a teljes kulcs egy rövidítése, nevezetesen a nyilvános kulcs utolsó 8 bájta. Amikor ez

megjelenítésre kerül, a további rövidítés kedvéért csak az utolsó 3 bájt tartalma kerül kijelzésre. Amíg több különböző kulcs felhasználó-azonosítója megegyezhet, (gyakorlatias szempontok miatt) két kulcs ID-je nem lehet azonos.

A PGP kivonatokat (message digest) használ az aláíráshoz. A kivonat egy 128 bites szám, amit egy kriptográfiai szempontból megbízható, szigorúan egyirányú hash függvény állít elő az üzenet alapján. Ez a kontroll-összeggel vagy a CRC értékkel analóg abban a tekintetben, hogy jelzi az üzenet megváltozását. Ugyanakkor a kivonatot létrehozó eljárás kizárja, hogy a kalóz olyanformán módosítsa az üzenetet, hogy az eredetivel megegyező kivonat jöjjön létre (ami az összeg és a CRC esetében lehetséges). Ezt a kivonatot kell a küldő titkos kulcsával kódolni, ami ezáltal az üzenet elektronikus aláírását adja ki.

Az aláírás folyamán az üzenet szövege mellé kerül az aláírás struktúra. Ez az aláíró kulcs-ID-jéből, az aláírás időpontjából és a kódolt kivonatból áll. A kulcs-ID azért kell, hogy fogadáskor a program kiválassza a kulcsfájlból azt a nyilvános kulcsot, amelyikkel az ellenőrzést el kell végezni.

A kódolt üzenet is kiegészül a kódoló kulcs ID-jével. Ez lehetővé teszi, hogy a program automatikusan kiválassza a dekódoláshoz szükséges titkos kulcsot a titkos kulcsfájlból.

A kulcsfájl-konceptió sarkalatos pontja a programnak. Az egyes kulcsok nem egyenként vannak sok-sok fájlban tárolva, hanem egyetlen kulcsfájl (key ring) gyűjti az összes azonos típusút. Ez sokkal gyorsabb előkeresést tesz lehetővé – akár a személy, akár a kulcs azonosítója alapján. Minden használónak két kulcsfájlja van: a nyilvános és a titkos. A nyilvános kulcsfájl esetén lehetőség van az egyedi kulcsok kinyerésére, exportjára, így azokat elküldhetjük ismerőseinknek, akik felvehetik az új kulcsokat saját nyilvános kulcsfájljukba.

A PGP installálása . A PGP MS-DOS alá készült 2.3-as verziója egy sűrített archív fájlként kerül terjesztésre.

Ennek neve PGP23.ZIP {naná, hogy PGP23A.ZIP} (x.y verzió mindig PGPxy.ZIP-ként kerül kiadásra). Ezt MS-DOS alatt a shareware PKUNZIP-vel lehet kibontani, Unix alatt pedig az unzip utility-vel. A PGP csomagban mindig van egy README.DOC fájl, amit feltétlenül el kell olvasnod. Ebben mindig benne vannak az „utolsó-perces” hírek, a verzióval kapcsolatos új dolgok, valamint itt van felsorolva, hogy a többi fájl mire jó.

Ha MS-DOS alá való 1.0-s verziójú PGP programod van, azt jobb ha letöröld, lévén hogy ezt már senki nem használja. Ha mégsem akarod kitörölni, akkor nevezd át a PGPEXE-t mondjuk PGP1.EXE-nek, hogy ne legyen azonos a neve az új verzióval.

Az MS-DOS alatti installáció nem is áll többől, mint az archív fájl kicsomagolásából, bár ezenkívül még célszerű lehet az AUTOEXEC.BAT-odat is módosítani. Erről a dokumentáció más részében szólnunk (SETUP.DOC). De ez a változtatás nem létfontosságú, később is sort keríthetsz rá, amikor már megismerkedtél a PGP-vel, és főként miután

többet is elolvastál ebből a leírásból. Ha még sohasem használtad a PGP-t, az installálás (és a dokumentáció elolvasása) után az első lépés saját kulcspárod legyártása lesz a „PGP -kg” paranccsal.

Ha Unix vagy VAX/VMS alá installálsz, akkor ez hasonló az MS-DOS-éhoz, de lehet, hogy le is kell fordítanod a programot az operációs rendszered alatt. Ehhez a forrásfájlokat tartalmazó archívban találsz egy makefile-t.

Az installálással kapcsolatos további részleteket a Setup.Doc fájlban találod meg. Ott részletesen leírjuk, hogy hogyan állítsd fel a PGP könyvtárát, és hogy módosítsd az AUTOEXEC.BAT fájlot, és azt is, hogy hogyan kell használni a PKUNZIP programot azok részére, akiknek ez is gondot okoz.

A PGP használata .

A használati útmutató. Az opciók egysoros összefoglalását a

```
>PGP -h
```

paranccsal lehet megnézni.

Üzenet kódolása (titkosítása) . Ahhoz, hogy egy üzenet nyersszövegét a fogadó nyilvános kulcsával letitkosítsd {-e = encrypt}, a

```
>PGP -e textfile f_userid
```

parancsot használd. Ennek hatására textfile.PGP néven létrejön egy kódszöveg-fájl. Gyakorlati példa:

```
>PGP -e letter.txt Alice
```

vagy:

```
>PGP -e letter.txt „Alice S”
```

Az első példa a nyilvános kulcsfájlodban (pubring.gpg) azokat keresi, akiknek a userId-je tartalmazza az „Alice” szöveget. A másodikban azokat, akiknek az userId-jében az „Alice S” szöveg szerepel. A parancssoron az egyes paraméterekben csak akkor használhatsz szóközt, ha az egész paramétert idézőjelbe teszed. A keresés nem érzékeny a kis-nagybetű különbségre. Amennyiben a pubring.gpg-ben talált megfelelő userId-t, a hozzá tartozó kulccsal letitkosítja a letter.txt-t, és ezáltal létrehozza a letter.gpg fájlt.

A PGP mindig először megpróbálja tömöríteni a nyersszöveget, ezáltal az még jobban ellenáll az esetleges kriptanalízisnek. Ne csodálkozz, ha a titkosított fájl rövidebb, mint az eredeti.

Ha a kódolt levelet e-mail segítségével akarod továbbítani, ahhoz ASCII-(radix-64) formába kell konvertálni. Ehhez a parancssort kódoláskor ki kell egészítened a -a opcióval. (Lásd később). {-e helyett használj -ea opciót}.

Üzenet kódolása több fogadó részére . Ugyanazt az üzenetet egyszerre több személy számára is kódolhatod. Ilyenkor a felsorolt fogadók bármelyike képes lesz kibontani az üzenetet. A művelethez ugyanúgy járd el mint az előző pontban, és sorold fel az összes fogadó userId-jét.

```
>PGP -e letter.txt Alice Bob Carol
```

A PGP létrehozza a kódszöveget (letter.pgp), amit Alice, Bob és Carol egyaránt kibonthat. Akárhány fogadót fel lehet sorolni (bár a parancssor hosszát az op. rendszer korlátozza).

Üzenet aláírása . Ahhoz, hogy egy üzenet nyersszövegét a saját titkos kulcsoddal aláírd (-s = sign) a

```
>PGP -s textfile [-u a_te_useridd]
```

parancsot használd. A szögletes zárójel arra utal, hogy a saját userId megadása nem kötelező. Ha megadod, ne írd ki a zárójeleket.

A parancs hatására létrejön az aláírt textfile.pgp fájl. Gyakorlati példa:

```
>PGP -s letter.txt -u Bob
```

Ez a titkos kulcsfájlban (secreting.pgp) kikeresi a „Bob” szöveget tartalmazó felhasználót, és az ő titkos kulcsával írja alá a letter.txt fájlt, és az eredményt a letter.pgp fájlba írja ki. A keresés figyelmen kívül hagyja a kis/nagybetű különbséget.

Amennyiben nem adsz meg userId-t, akkor a program a legelső kulcsot veszi elő, és azt használja.

Aláírás majd titkosítás . Ahhoz, hogy egyszerre aláírd, majd titkosítsd az üzenetet, kombinálni kell az előző pontban írottakat:

```
>PGP -es textfile f_userid [-u a_te_useridd]
```

A szögletes zárójel arra utal, hogy a saját userId megadása nem kötelező. Ha megadod, ne írd ki a zárójeleket.

A program kikeresi a megfelelő titkos kulcsodat, aláírja az üzenetet, majd előkeresi a megfelelő nyilvános kulcsot és elvégzi a titkosítást. Ha nem adod meg a fogadó userId-jét, akkor a program interaktívan rákérdez.

Ha a titkos kulcshoz tartozó id-t nem adod meg, akkor a program az első kulcsot használja.

Vedd figyelembe, hogy a PGP először mindig megpróbálkozik az üzenet tömörítésével.

Ha a kódolt levelet e-mail segítségével akarod továbbítani, ahhoz ASCII-(radix-64) formába kell konvertálni. Ehhez a parancssort kódoláskor ki kell egészítened a -a opcióval. (Lásd később). (-esa opció a -es helyett).

Ha több fogadónak kódolsz, egyszerűen sorold fel userId-jüket ott, ahol az előbb csak egy állt.

Titkosítás konvencionális módon . Időnként szükség lehet arra, hogy egy-egy fájlt hagyományos egykulcsos módszerrel kódoljunk. Ez akkor lehet hasznos, ha olyan anyagaink vannak, amit védeni akarunk, de nem küldjük el senkinek. Miután itt a kibontó ugyanaz a személy, mint a titkosító, nincs szükség a kétkulcsos titkosításra.

Egy fájl hagyományos kódolásához a

```
>PGP -c textfile
```

parancsot kell használni. Hatására a program kódolja a textfile-t, miközben nem használja sem a nyilvános, sem a titkos kulcsokat. A titkosításhoz egy jelszót kér interaktívan. Ennek a jelszónak nem kell (sőt, nagyon célszerűtlen) azonosnak lennie azzal a jelszóval, mellyel a titkos kulcsunkat védjük. Vedd figyelembe, hogy a PGP kódolás előtt sűríti az adatfájlt.

Ha ugyanazt az adatfájlt ugyanazzal a jelszóval többször is kódolod, minden esetben más eredmény fog kijönni.

Kibontás és aláírás-ellenőrzés . A kódolt üzenet kibontásához, valamint az automatikus aláírás-ellenőrzéshez a

```
>PGP kódszövegfájl [-o nyersszövegfájl]
```

parancsot kell használni. A szögletes zárójel opcionális paramétert jelöl, ha megadod, ne írd ki a zárójeleket.

A kódszövegfájl kiterjesztését a program automatikusan .pgp-nek veszi, ha nem adod meg. A kibontott fájl alapértelmezésben a kódszövegfájl nevét kapja kiterjesztés nélkül. Ezt lehet felülbírálni a -o opcióval. Ha a szöveg alá van írva, akkor a program automatikusan ellenőrzi az aláírást, és kiírja az aláírás gazdájának teljes userId-jét.

A kibontás teljesen automatikus, és a PGP felismeri, hogy a fájl csak kódolva van, csak alá van írva, vagy mindkettő. {Ehhez hasonlóan felismeri a Radix-64 alkalmazását, a ClearSig-et, és bármit, amit a programmal elő tudsz állítani ;)}. A PGP a kódszöveg fejlécéből automatikusan megállapítja, hogy melyik titkos kulcsodat kell a kifejtéshez használni. Aláírt üzenet esetén az aláírás fejlécében levő kulcsId alapján tudja kiválasztani az ellenőrzéshez szükséges nyilvános kulcsot. Ha mindkét Id-hez tartozó kulcs szerepel a kulcsfájljaiban, a program csak a titkos kulcsod jelszavát fogja megkérdezni. Ha a kódszöveg hagyományos kódolással készült, a PGP nem nyúl a kulcsfájlokhoz, hanem a dekódoláshoz szükséges jelszót kérdezi.

Kulcskezelés . Julius Caesar idejétől fogva a titkosítások legproblémásabb pontja a kulcskezelés. A PGP program egyik legfőbb szolgáltatása az, hogy a kulcsokat nagyon kényelmesen kezeli.

Az RSA kulcs készítése . Ahhoz, hogy létrehozod saját nyilvános és titkos kulcsodat, add ki az alábbi parancsot: (-kg = kulcsgenerálás)

```
>PGP -kg
```

A PGP egy menüben felajánlja a különböző kulcsméreteket (alkalmi szintű = casual grade, ipari szintű = commercial grade, katonai szintű = military grade), és rákérdez, hogy mekkora kulcsot akarsz (kb. 1000 bites határig). Minél nagyobb a kulcs, annál nagyobb a biztonság, de a nagyobb kulccsal való munkához több idő kell.

Utána megkérdezi az userId-det, ami többnyire a neved. Célszerű a teljes nevedet használni, mert így kisebb az esélye, hogy valaki eltéveszti az azonosítót, amikor a te számodra kódol. A userId tartalmazhat szóközőket és írásjeleket is. Ha van e-mail címed, azt célszerű <> jelek között a neved után fűzni, mint az alábbi példában:

```
Robert M. Smith <rms@xyzcorp.com>
```


Ha nincs e-mail címed, használd a telefonszámod, vagy találj ki valamit, amitől az `userId`-d igazán egyedi lesz, és bizonyosan téged azonosít.

A PGP ezután egy jelszót kér, ami a titkos kulcsodat védi. A titkos kulcsfájlod a jelszó nélkül nem használható. A PGP-ben a jelszó nem kell, hogy egy szóból álljon. Lehet több szó, vagy akár egy egész mondat. Szerepelhet benne bármilyen karakter. Jól jegyezd meg a jelszavadat. Ha elfelejtetted, nincs semmiféle mód a megfejtéséhez. A jelszóra a későbbiekben mindig szükség lesz, ahányszor csak szükség lesz a titkos kulcsodra. A jelszóban különbözőnek számítanak a kis- és nagybetűk. A jelszó ne legyen túl rövid, és ne lehessen könnyen kitalálni. A jelszó sohasem jelenik meg a képernyőn. Sohase írd le, és ne tárold sehol a számítógépeden. Ha úgy döntesz, hogy nem véded jelszóval a titkos kulcsodat (te bolond!) egyszerűen üss ENTER-t a kérdésre.

A kulcs generálásához a program nagy, valódi véletlenszámokat használ. Ezeket úgy kapja, hogy a billentyűlételek között eltelt időt méri egy nagyon gyors órával. A program először egy véletlenszerű szöveget kér, amit néhány véletlen bit inicializálására használ. Utána a billentyűlételek közötti időt használja. Nyomkodd a billentyűket véletlenszerűen, amíg a program azt nem mondja, elég. Egyes helyeken a program a billentyűket is nézi, ezért ne ugyanazt a sorozatot ismételd.

Az RSA-kulcsok generálása időigényes. Lassú gépen a nagyméretű kulcs létrehozásához több perc is szükséges lehet.

A létrehozott kulcsok automatikusan bekerülnek a nyilvános és titkos kulcsfájlodba. A későbbiekben a `-kx` parancs segítségével létrehozatsz egy olyan üzenetfájlt, amely tartalmazza a nyilvános kulcsodat. Ezt a fájlt azután elterjesztheted ismerőseid körében, vagy e-mail hálózaton. A titkos kulcsodat a titkos kulcsfájlodban kell tartanod. Ha több titkos kulcsfájlod van, akkor ezek mindegyike önálló jelszóval rendelkezik.

A titkos kulcsodat sose add ki a kezedből. Ugyanezen okból sose készíts más számára titkos kulcsot. Ezt mindenkinek magának kell megtennie. Gondoskodj arról, hogy a tied legyen a titkos kulcs feletti fizikai kontroll. Ne hagyd a titkos kulcsot időosztáson alapuló többfelhasználós rendszeren, vagy olyan hálózaton, amelyen mások is dolgoznak. Tárold a saját személyes számítógépeden.

Kulcs importálása a kulcsfájlba . Ha hozzá akarsz adni egy kulcsot a nyilvános vagy a titkos kulcsfájlodhoz, használd az alábbi parancsot (a `[]`-ban levő paraméter opcionális):

```
>PGP -ka kulcs [kulcsfájl]
```

A „kulcs” természetesen az az üzenetfájl, amiből a kulcsot importálni akarod. A `[kulcsfájl]` automatikusan `.PGP` kiterjesztést kap, ha nem adod meg. Ha nem adod meg ezt a paramétert, az alapértelmezés a „`pubring.pgp`”, vagy a „`secring.pgp`” attól függően,

hogyan a <kulcs> nyilvános vagy titkos kulcsot tartalmaz. Ha szükséges, használhat más kulcsfájl-nevet.

Ha a kulcs már szerepel a kulcsfájlban, a PGP nem adja hozzá még egyszer. Ha az importfájlban több kulcs is szerepel, mindegyik feldolgozásra kerül – természetesen a duplikátok kihagyásával. Amennyiben a kulcsokat aláírták, az aláírások is bekerülnek a kulcsfájlba. Ha a kulcs már szerepel, de új aláírások is vannak rajta, csak ezek kerülnek kigyűjtésre.

Kulcs vagy userId törlése a kulcsfájlból . Ha ki akarod törölni valamelyik kulcsot a kulcsfájlodból, használd az alábbi parancsot:

```
>PGP -kr userId [keyring]
```

A parancs hatására a program megkeresi a kulcsfájlban az userId-hez tartozó kulcsot és kitörli. Emlékezz rá, hogy nem kell a teljes userId-t megadnod, hanem a program megelégszik azzal, ha valamelyik userId tartalmazza az általad a parancssoron megadott szöveget. Ha nem adsz meg kulcsfájl-nevet, az alapértelmezés a „pubring.pgp”. Ha a kulcsfájl-nevet kiterjesztés nélkül adod meg, a program .PGP kiterjesztést használ.

Ha a kiválasztott kulcshoz több userId is tartozik a kulcsfájlban, a program rákérdez, hogy a kulcsot kell-e törölnie, vagy csak a megadott userId-t. Az utóbbi esetben a kulcs a kulcsfájlban marad az összes fennmaradó userId-jével.

Kulcs exportálása a kulcsfájlból . Kulcs exportálásához a

```
>PGP -kx userId kulcs [kulcsfájl]
```

parancsot kell használni. A kulcsfájl érintetlenül hagyása mellett egy olyan fájlt hozunk létre, amely alapján a kulcs egy másik (saját, vagy másnál levő) kulcsfájlba importálhatjuk. A saját nyilvános kulcsunk közzétételénél is ezt a parancsot kell alkalmazni.

Ha a kulcsfájlban a kulcs alá volt írva, akkor ugyanígy, aláírva kerül exportálásra.

Ha e-mail rendszeren akarod közzétenni a kulcsodat, használd a -a opciót is (-kxa).

A kulcsfájl tartalmának megtekintése . A

```
>PGP -kv[v] [userId] [kulcsfájl]
```

parancsra a program kilistázza a kulcsfájl tartalmát. Ha megadtad az `userId`-t, akkor minden olyan kulcsot kiír, amelynek `userId`-jében szerepel a megadott szöveg. Ha nem adod meg a kulcsfájl nevét, az alapértelmezés a „`pubring.pgp`”. A titkos kulcsfájl tartalmának megtekintéséhez meg kell adni a „`secring.pgp`”-t a parancssor végén. A kulcsfájl alapértelmezett kiterjesztése a `.PGP`.

A `-kvv` opció hatására a kulcsok mellett megjelennek a rajtuk levő aláírások is:

```
>PGP -kvv [userId] [kulcsfájl]
```

Ha egy alternatív kulcsfájlban szereplő összes kulcsot akarsz kilistázni, akkor a parancssor:

```
>PGP kulcsfájl
```

Ha nem adsz meg opciókat a parancssoron, a PGP kilistázza a kulcsfájl.PGP-ben szereplő összes kulcsot, és egyben megpróbálja hozzáadni őket a nyilvános kulcsfájlodhoz, ha még nem szereplnének benne.

Hogyan védj kulcsfájljaidat a módosítástól? . A nyilvános-kulcsos szisztémában a nyilvános kulcsokat nem kell attól féltetni, hogy eljutnak valakihez. Sőt, az a jó, ha terjednek. Viszont meg kell óvni a kulcsot az esetleges módosítástól. És a kulcs fogadásának meg kell bizonyosodnia arról, hogy a kulcs valóban ahhoz a személyhez tartozik, akit mi gondolunk. Ez a nyilvános-kulcsos titkosítási rendszer legsebezhetőbb pontja. Először is tekintsünk át egy potenciális katasztrófát, utána tárgyaljuk, hogy hogyan előzhető ez meg a PGP segítségével.

Tegyük fel, hogy privát üzenetet akarsz küldeni Anikónak. Letöltöd Anikó nyilvános kulcsát a BBS-ről. Kódolod az üzenetet Anikó kulcsával és e-mail segítségével elküldöd.

Ugyanakkor, anélkül hogy te, vagy Anikó tudnátok róla, Béla generált egy kulcsot, felruházta Anikó `userId`-jével, és feltöltötte a BBS-re, felülírva Anikó eredeti kulcsát. Miközben minden jónak tetszik, a privát üzenetedet Anikó helyett Béla tudja kibontani, hiszen nála van a megfelelő titkos kulcs. Sőt, a kibontás után megteheti, hogy az üzenetet lekódolja Anikó eredeti kulcsával, és továbbküldi Anikónak. Ilyenkor senki nem sejt semmit, Béla pedig vígan olvassa a privát üzeneteket. Még rosszabb, hogy Béla aláírhat Anikó nevében, hiszen mindenki a hamisított nyilvános kulcsot használja az ellenőrzéshez.

A probléma megelőzéséhez meg kell akadályozni a kulcsok módosítását. Ha Anikó személyesen adja át a kulcsát, akkor persze nem lehet gond. No de mi van akkor, ha Anikó tőled párezer kilométerre lakik, vagy egyéb okból épp elérhetetlen?

Például beszerezheted Anikó kulcsát barátodtól, Dávidtól, aki a saját (titkos kulcsával való) aláírásával igazolja a kulcs eredetét és eredetiségét. Az aláírás egyrészt jelzi, hogy Dávid kezekszik a kulcsért, másrészt megakadályozza annak módosítását. Az ellenőrzéshez persze rendelkezned kell Dávid megbízható nyilvános kulcsával. Hasonló módon Dávid eljuttathatja a te kulcsodat (saját aláírásával) Anikóhoz. Ebben a folyamatban Dávid mintegy közvetítőként szerepel.

Ha a BBS-ről úgy töltöd le Anikó kulcsát, hogy rajta van Dávid aláírása (amit megbízható kulccsal ellenőrizhetsz), bizonyos lehetsz benne, hogy amit letöltöttél, valóban Anikó kulcsa. Ilyenkor Béla hiába fáradozik, hiszen Dávid aláírását nem tudja előállítani az Anikó nevére írt hamis kulcson.

Egy széles körben megbízhatónak tartott ember megteheti, hogy arra szakosodik, hogy ezenformán „bemutatja” egymásnak az embereket, aláírva a nyilvános kulcsukat. Ez a megbízható ember „key server”-ré válhat. Minden kulcs, amelyik az ő aláírását viseli, megbízhatónak tekinthető. Ilyenkor az összes résztvevőnek egyetlen kulcsot kell megbízhatóan beszerezniük: a „key server”-ét, amivel az aláírását ellenőrizhetik.

Nagy szervezeteknek központosított „key server”-ekre van szüksége. Egyes környezetekben a „key serverek”-nek megvan a kiépített hierarchiája is.

Kevésbé centralizált, széleskörű hálózatokban a központi „key serverek”-nél jobban működhet az a rendszer, hogy mindenki a közvetlen ismerőseit tekinti megbízható aláírónak. A PGP különösen itt nyújthat nagy segítséget. Működése jól illeszkedik a szokásos humán kapcsolatokhoz.

Ez a nyilvánoskulcs-védelemmel kapcsolatos tevékenység az egyetlen nehézség a nyilvánoskulcsos titkosítási rendszer gyakorlati alkalmazásában. Az egész dolognak ez az Achilles-sarka. Éppen ezért a program elég nagy része ennek a problémának a megoldására van kihegyezve.

Csak akkor vehetsz használatba egy nyilvános kulcsot, ha meggyőződted arról, hogy azt nem módosították, és hogy valóban ahhoz a személyhez tartozik, akire az `userId`-je utal. Ezt csak akkor veheted bizonyosnak, ha a kulcsot közvetlen úton a tulajdonostól kaptad, vagy ha általad megbízhatónak elismert ember aláírását viseli, és az aláírás ellenőrzéséhez használt nyilvános kulcs megbízhatóságáról már előzetesen megbizonyosodtál. A kulcs `userId`-jének a tulajdonos teljes nevét tartalmaznia kell, nemcsak a keresztnévét.

Bármennyire csábító is – sose bízz meg egy BBS-ről letöltött kulcsban, kivéve, ha a fentiek alapján megbízható kulccsal alá van írva. A BBS-en a kulcsokat elvileg bárki módosíthatja, például a BBS üzemeltetője.

Ha arra kérnek, hogy írd alá egy kulcsot, kizárólag akkor tedd meg, ha teljesen bizonyos vagy abban, hogy kihez tartozik, és a kulcs az ő nevét viseli az `userId`-ben. Az aláírásoddal leteszed a nagyescüt a kulcs valóságára mellett. Más emberek azért fogják valódinak elfogadni a kulcsot, mert a te aláírásod van rajta. Nem szabad pusztán hallomás alapján

aláírnod egy kulcsot. Csak ha első kézből meg tudsz győződni a valódiságról. A legjobb, ha kizárólag olyan kulcsot írsz alá, amit személyesen a gazdájától kaptál.

A kulcs aláírásához sokkal jobban kell bíznod benne, mintha csak üzenetkódoláshoz használnád. Ahhoz, hogy használj egy kulcsot elég lehet néhány rajta levő aláírás. Ahhoz, hogy magad aláírd a kulcsot, közvetlen, első kézből szerzett bizonyítékkal kell rendelkezned az eredetiségre vonatkozóan. Például felhívod a tulajdonost – akit a telefonhang alapján azonosíthatsz – és beolvastatod vele a kulcsot – vagy az azonosításhoz szükséges darabját. Erre vonatkozóan olvasd el a „Kulcsok telefonon való azonosítása” c. fejezetet a dokumentáció haladó témákkal foglalkozó II. részében.

Egy kulcs aláírásával csak a kulcs megbízhatóságát igazolod, nem a kulcs tulajdonosának megbízhatóságát. Nyugodtan aláírhatod megbíz- hatatlan barátod kulcsát, ha tudod, hogy az övé. Ha az emberek bíznak benned, használni fogják a kulcsot. Ugyanakkor barátod aláírásának elfogadhatóságát majd mindenki egyénileg bírálhatja el.

A „megbízhatóság” nem feltétlenül továbbadható. Van egy barátom, akinek a szavát készpénznek veszem. Ő meg mondjuk hisz abban, amit a miniszterelnök mond. Ez még nem jelenti azt, hogy én is megbízom abban, amit a miniszterelnök mond. Hasonló a dolog az aláírásoknál is. Ha én elfogadom Anikó aláírását, Anikó meg Béláét, attól én nem feltétlen fogom Béla aláírását elfogadni.

Saját nyilvános kulcsodat célszerű minél több emberrel aláíratni. Annál inkább valószínű, hogy egy potenciális használó talál valakit az aláírók között, akiben bízik. Az aláírt kulcsot terjesztheted BBS- rendszereken keresztül is. Ha aláírod valakinek a kulcsát, juttasd el a kulcsot a gazdájához. Ezzel bővítheti a kulcsa aláírásgyűjteményét.

A PGP nyilvántartja, hogy a kulcsfájlodban szereplő kulcsok közül melyek viselik megbízható ember aláírását. Ehhez mindössze meg kell adnod a PGP számára a megbízható emberek listáját és alá kell írnod az ő kulcsukat a saját (teljesen megbízható) titkos kulcsoddal. A PGP a lista alapján bármelyik kulcsról el tudja dönteni, hogy megbízhatónak tekinthető-e. De természetesen magad direkt módon is aláírhatsz kulcsokat. (A későbbiekben még visszatérünk a témára.)

Gondoskodj róla, hogy a nyilvános kulcsfájlodat ne módosíthassa senki. Egy új kulcs ellenőrzése attól függ, hogy sikerül-e a már összegyűjtött megbízható kulcsok alapján validálni. Őrizd meg a fizikai kontrollt a nyilvános kulcsfájlod felett is. Lehetőleg ne tartsd közös rendszeren, csak a saját számítógépeden. A nyilvános kulcsot csak a belepiskálástól kell megvédeni, a nyilvánosságra kerüléstől (szemben a titkossal) nem. Mindig legyen megbízható (és sérthetetlen) másolatod a kulcsfájljaidról.

Miután minden ellenőrzés valamelyik lépésben a saját nyilvános kulcsod sértetlenségét feltételezi, ennek módosíthatatlanságáról különös gonddal kell gondoskodnod. Lehetőséged van a PGP-t úgy beállítani, hogy a nyilvános kulcsodat minden induláskor összevesse egy csak olvasható médiumon levő másolattal. A részleteket olvasd el a „Haladó témák” dokumentációban (II. rész).

A PGP feltételezi, hogy te gondoskodsz a megbízható körülmények között való működéséről. A kulcsfájlokat és a programot egyaránt olyan gépen célszerű tartani, amely fizikailag is a te kezében van. Ha valaki belenyúlhat a kulcsfájljaidba és a PGP-be, megteheti, hogy a PGP programot úgy módosítsa, hogy ne jelezze a kulcsok megváltozását.

A nyilvános kulcsfájl védelmének egyik – talán kicsit körülményes – módja az, hogy aláírod az egész fájlt a titkos kulcsoddal. Ehhez a -sb opciót kell használnod (bővebben az „Az aláírás leválasztása az üzenetről” c. fejezetben a haladó témák között). Sajnos az ellenőrzéshez szükség van egy másik, megbízhatónak elismert másolatra a saját nyilvánoskulcsból (emlékezz: a titkos kulccsal aláírt dolgok ellenőrzéséhez a nyilvános kulcs kell). A benne levő kulcs erre nem használható, hiszen akkor a potenciálisan megbízhatatlan, ellenőrizendő dolgot használnád mint megbízható ellenőrző információt.

Hogy követi nyomon a PGP a megbízható kulcsokat? . Mielőtt ezt a fejezetet elolvasnád, feltétlenül olvasd végig az előző (Hogyan védj kulcsfájljaidat a módosítástól? c.) fejezetet.

A PGP a kulcsokon levő aláírások alapján tudja eldönteni, hogy az adott kulcs megbízhatónak tekinthető-e. A te egyetlen ezzel kapcsolatos dolgod az, hogy egy listát készítesz, amiben felsorolod az általad megbízhatónak tartott embereket, valamint alá kell írnod az ő kulcsukat a sajátoddal. Ettől kezdve a PGP a felsorolt emberek által aláírt kulcsokat megbízhatónak fogja jelezni. Természetesen arra is van lehetőség, hogy egyéni elbírálás alapján magad írd alá további kulcsokat.

A PGP két független szempont alapján osztályozza a kulcsokat. Ne keverd a kétféle osztályozást:

- Valódinak tekinthető-e a kulcs? Valóban ahhoz a személyhez tartozik, akinek a nevét viseli? Ezt a rajta levő megbízható aláírás alapján lehet elbírálni.
- Olyan ember kulcsa-e, akinek az aláírását bizonyító erejűnek fogadod el egy kulcs valóságának tekintetében.

Az első kérdést a PGP saját hatáskörben el tudja bírálni. A másodikat csak a te útmutatásod alapján. Természetesen az 1. kérdésre adott választ a 2.-ra adott határozza meg.

Még egyszer összefoglalva: a megbízhatónak deklarált emberek által aláírt _kulcsokat_ tekinti a PGP megbízhatónak. A megbízható _személyek_ körét neked kell meghatároznod. Az ő kulcsukat pedig vagy elsődleges forrásból kell beszerezned, vagy más megbízható emberek aláírása van rajtuk.

A PGP-ben az embereket több csoportba sorolhatod megbízhatóság szempontjából. A besorolás nemcsak az humán megbízhatóságot kell hogy tükrözze, hanem a PGP használatában elért jártasságot is. Azt a képességüket, hogy jól el tudjanak bírálni egy kulcsot, mielőtt aláírnák. Az alábbi kategóriákat használja a PGP: ismeretlen (unknown),

megbízhatatlan (untrusted), aránylag megbízható (marginally trusted), tökéletesen megbízható (completely trusted). Ezt a besorolást a kulcsfájlodban őrzi az egyes kulcsok mellett. Amennyiben exportálsz a kulcsot, a megbízhatóságra vonatkozó információ nem kerül a kivonatba, mert ez a te kizárólagos személyes ügyed.

A kulcs megbízhatóság szerinti elbírálásánál a PGP tekintetbe veszi az aláírások számát és „értékét”. Pl. két többé-kevésbé megbízható aláírás ér annyit, mint egy tökéletesen megbízható. A pozitív elbíráláshoz szükséges aláírás-mennyiséget te is tudod szabályozni, pl. megadhatod, hogy két tökéletesen, vagy három többé-kevésbé megbízható aláírás legyen a megbízhatónak elfogadott kulcson.

A saját kulcsod definíció szerint tökéletesen hiteles. A saját kulcsaidat a titkos kulcsfájlban tárolt kulcsok alapján ismeri fel a program. Hasonlóképpen tökéletesen hitelesnek számítnak a te kulcsoddal aláírt kulcsok.

Ahogy az idő halad, mindenki gyűjti a kulcsokat, és kiválasztja a hitelesítéshez a megbízható embereket. És időnként mindenki kiad egy- két maga által aláírt „holtbiztos” kulcsot, természetesen a rajta levő többi aláírással együtt. Ahogy a kulcsokon gyűlnek az aláírások, egyre valószínűbb, hogy egy új fogadó talál az aláírások között szerinte megbízhatót. A helyi bizalmi gócból így létrejön egy nagy háló, amely lehetővé teszi a kulcsok biztonságos azonosítását.

Ez a séma pont ellentétes a kormány által támogatott gyakorlattal. Ott központi előírások vannak a bizalmi kategóriákra, felülről mondják meg, hogy melyik aláírásban mennyire kell hinned. A PGP decentralizált kulcs-kezelése neked hagyja meg a döntést ezekben a kérdésekben, ezáltal téged helyezve az előző gyakorlatban használt piramis legtetéjére. Egy hasonlattal élve: a PGP azoknak az embereknek kedvez, akik maguk szeretik összehajtogatni az ejtőernyőjüket. {Az, hogy az ejtőernyő ki tud-e nyílni, leginkább az összehajtogatástól függ. :-)}

Hogyan védj titkos kulcsodat? . Nagyon vigyázz a titkos kulcsodra és az ezt védő jelszóra. Ha a kulcs egyszer „kicsúszott a kezedből”, jobb ezt mielőbb a világ tudtára adni. Lehetőleg még mielőtt valaki a te nevedben kezd aláírásokat produkálni. A titkos kulcsod birtokában pl. kulcsokat tudnak hamisítani, és ez sok embernek okozhat komoly kellemetlenségeket. Különösen akkor, ha sokan bíznak a te aláírásodban. És természetesen az, aki megszerezte a titkos kulcsodat el tudja olvasni a neked szánt üzeneteket.

Először is a kulcsfájl és az ezt tartalmazó médium mindig legyen a te felügyeleted alatt. Tartsd az otthoni gépeden, vagy azon a notebook-on, amit magaddal szoktál hordani. Ha a munkahelyeden levő géphez más is hozzáfér (vagy pl. egyik pillanatról a másikra elvihetik), a kulcsfájljaidat tartsd inkább egy írásvédett lemezen, amit magaddal hordasz. Ne másold fel a számítógépedre. Mindenképpen rossz gyakorlat a kulcsfájlokat közös rendszeren elhelyezni (amilyen pl. egy felhívható, Unix alapú rendszer). Ha valaki képes lehallgatni a modemedet, elcsípheti a jelszavadat, a kulcsodat meg egy-

szerűen kimásolja a közös rendszer háttértárolójából. A titkos kulcsot csak kizárólagos használatú gépen használd!

A jelszót sose tárold azon a számítógépen, amelyen a kulcsfájl. (Semmilyen számítógépen ne tárold.) Az együtt-tárolás pontosan annyira helytelen, mint pl. a PIN-edet együtt tárolni az Automatic Teller Machine bankkártyával. Ha valaki történetesen hozzáférne a gépedhez, legalább ne kaphassa kézbe a kulcs mellett a jelszavadat is. A leghelyesebb gyakorlat, ha a jelszó sehol nem szerepel leírva vagy elektronikus formában, kizárólag a te memóriádban. Ha szükségesnek érzed, hogy leírd, akkor vigyázz rá még jobban, mint a kulcsfájlra.

És legyen mindig másolatod a titkos kulcsodról! Emlékezz, a titkos kulcsod csak neked van meg. És ha elveszíted, a nyilvános kulcsod a világban elterjedt számtalan példánya azon nyomban használhatatlanná válik.

A PGP decentralizált kulcs-elbírálási rendszere a gyakorlatban komoly előny, viszont mivel nincsenek hivatalos, érvényes kulcslisták, egy titkoskulcs-kiszivárgás esetén csak abban bízatsz, hogy az erről szóló információ elég gyorsan elterjed.

Amennyiben megtörténik a legrosszabb: valaki hozzájut a titkos kulcsodhoz ÉS a jelszavadhoz is, (de ez legalább a tudomásodra jut), akkor egy „kulcs érvénytelenítési” üzenetet kell generálnod, és ezt kell terjesztened. Ez az üzenet arra figyelmezteti a fogadókat, hogy ne használják többé a(z adott kulcshoz tartozó) nyilvános kulcsot. A PGP a -kd opcióval csinál ilyen üzenetet. Ezután el kell juttatnod a Föld minden emberéhez – de legalább a barátaidhoz, és azok barátaihoz, stb. – azokhoz, akik szokták a nyilvános kulcsodat használni. Az ő PGP-jük feldolgozza az érvénytelenítési üzenetet, és az érvénytelenítés ténye bekerül a nyilvános kulcsod mellé a kulcsfájljukba. Ez a jövőben megakadályozza a kulcsod véletlen használatát. Az eset után új kulcspárt kell generálnod, és az új nyilvános kulcsot szintén el kell terjesztened. A két üzenet (érvénytelenítés + új kulcs) egyszerre is terjeszthető.

A nyilvános kulcs bevonása . Tegyük fel, hogy titkos kulcsod a használatához szükséges jelszóval együtt idegenek kezére került. Ilyenkor értesítened kell mindenkit, hogy a nyilvános kulcsod többé nem használható. Ehhez kulcs „érvénytelenítési” vagy más szóval „visszavonási” parancsot kell kiadnod. Ez a parancs a

```
>PGP -kd saját_userId
```

A visszavonás ugyanazzal a kulccsal van aláírva, amit visszavonsz (ezért a titkos kulcsod nélkül nem hamisítható). A visszavonási üzenetet minél szélesebb körben terítened kell. A fogadónál ez az információ beépül a kulcsfájlba és megakadályozza a visszavont kulcs további használatát. A visszavonás után készíthetsz magadnak egy új kulcspárt.

Egy kulcsot az említettől eltérő ok esetén is visszavonhatsz. A mechanizmus ebben az esetben is ugyanez.

Mi a helyzet, ha elveszíted a titkos kulcsodat? . Normális esetben az érvénytelenítő üzenet alá van írva a titkos kulccsal (lásd az előző két fejezetet).

De abban az esetben, ha elveszíted (pl. letörlőd) a titkos kulcsod, erre nem lesz lehetőség. A PGP jövőbeli verzióiban az ilyen esetekben történő visszavonás némileg biztonságosabbá lesz téve. Az érvénytelenítő határozatot majd ugyanúgy megbízható emberekkel kell aláírni, mint normál esetben a kulcsokat. A jelenlegi rendszerben ez még nincs implementálva, ezért mindössze egy olyan üzenetet küldhetsz ki, amely megkér mindenkit, hogy saját maga érvénytelenítse a kulcsodat a saját kulcsfájljában a

```
>PGP -kd userId
```

paranccsal. A PGP a -kd parancsnál először a titkos kulcsok között keres, s ha ott nem szerepel az Id, akkor a nyilvános kulcsfájlban folytatja. A megtalált kulcsot „letiltott” (disabled)-ként jegyzi be. A letiltott kulccsal nem lehet kódolni, és nem lehet az illetet exportálni sem. Aláírás ellenőrzéséhez továbbra is használható, de ilyenkor mindig egy figyelmeztető üzenet is jár az ellenőrzés eredménye mellé. A kulcs újonnan való importálása sem lehetséges, hiszen az adott userId már foglalt. Ezen funkciók együttesen (többé- kevésbé) gátat vetnek a letiltott kulcs további terjedésének.

Amennyiben a -kd parancsot alkalmazod és a kulcs már tiltott, a program rákérdez, hogy engedélyezni akarod-e.

Haladó témák . A haladó témakörök nagyobb része a dokumentáció II. részében kapott helyet. De néhány fontos opciót itt is leírunk.

Kódszöveg küldése e-mail csatornán . A legtöbb e-mail hálózat csak 7-bites ASCII kódolású szöveg terjesztését teszi lehetővé, azt a 8-bites kódszöveget, amit a PGP generál, nem tudod eredeti formában elküldeni. A probléma megoldásához a PGP a PEM rendszerhez hasonlóan lehetővé teszi a küldendő üzenet Radix-64 szerinti kódolását. Ez a speciális formátum kizárólag ASCII karaktereket alkalmaz, ezért gond nélkül küldhető. Ez a formátum mintegy pajzsként szolgál az üzenet körül, megakadályozva, hogy a továbbítói állomásokon az üzenet az átviteli technológia miatt módosuljon (és tönkremenjen). A formátum része egy CRC érték, ami a továbbítási hibák detektálására szolgál.

A Radix-64 formátumban 3 db. 8 bites értékből 4 db. 6 bites, az ASCII kódtartományba eső kód keletkezik. Ezáltal az üzenet mérete mintegy 33%-kal megnő. De ha tekintetbe

veszed, hogy titkosítás előtt a PGP tömörítette az eredetit, lehet, hogy a kódolt output mégis rövidebb lesz, mint az eredeti.

Ahhoz, hogy a PGP bármelyik opciójával nyert output ne bináris, hanem Radix-64 kódolású legyen, a parancssori opciót ki kell egészíteni egy „a” betűvel. Pl.

```
>PGP -esa message.txt f_userid
```

Ez a parancs az aláírt és kódolt üzenetet Radix-64 formátumban írja ki, amit biztonságosan feladhatsz egy e-mail rendszerre.

A Radix-64 formátumban érkezett üzenetek dekódolása nem tér el a binárisakétól. A PGP automatikusan észleli a két formátumot.

```
>PGP message
```

parancsra a PGP először message.ASC fájlt keres, ha ez nem létezik, akkor message.pgp-t. Az ASC először belsőleg a .pgp-hez hasonló bináris formátumúvá alakul. Innen már egységesen történik meg a kibontás majd az aláírás-ellenőrzés. A végeredmény pedig ugyanaz a message.TXT.

Némely Internet levelező rendszer tiltja az 50000 bájt nál hosszabb üzenetek átvitelét. A hosszabbakat több darabra kell tördelni. A tördelést a PGP automatikusan elvégzi. Az egyes darabok .AS1, .AS2, stb. kiterjesztést kapnak. A kibontás előtt ugyanezek a fájlok egységes egésszé állnak össze. Az üzenet-fájlokban levő, nem rá tartozó szöveget (címezés, tagline, stb.) a PGP automatikusan figyelmen kívül hagyja, ezért a leveleket nem kell előzetesen kozmetikázni.

A Radix-64 formátum nemcsak üzeneteknél, hanem kulcs-export esetén is használható. Az így exportált kulcs e-mail hálózaton terjeszthető.

A konverzió utólagosan is elvégezhető. A

```
PGP -a file
```

a megadott fájl egyszerű bináris -> Radix-64 kódolását végzi.

Amennyiben el akarsz küldeni egy olyan üzenetet, ami nincs rejtjelezve, csak alá van írva, és ezen alkalmazod a Radix-64 formátumot, az üzenet emberi szemmel olvashatatlanná válik.

Lehetőség van arra, hogy amennyiben az üzenet eredetileg szöveges volt, csak a szignatura, vagyis az újonnan keletkező bináris részek lesznek radix-64 formátumúak, az üzenet többi része változatlan marad. Ilyenkor teljesül az, hogy az üzenet e-mail útján

küldhető, Közvetlenül olvasható, ugyanakkor az aláírás által ellenőrizhető az autenti-
kussága.

Az -a opció alkalmazásával kapcsolatos további tudnivalók a speciális témák (doku-
mentáció, II. rész) között találhatók. Az „A paraméterek külső konfigurálása: CON-
FIG.TXT” fejezet CLEARSIG paraméterrel foglalkozó részében.

Környezeti változó a path megadásához . A PGP számos fájlt használ a működé-
séhez. Ilyenek a nyilvános és a titkos kulcsfájlok („pubring.pgp”, „secreing.pgp”), a vé-
letlenszámok generálásához használt „randseed.bin”, a konfigurációt leíró „config.txt”,
és az idegennyelvű üzenet-konverzióhoz használt „language.txt”. Ezek a fájlok tetsző-
leges könyvtárban lehetnek, ha beállítod a „PGPPATH” változót, hogy erre mutasson.
Pl. MS-DOS alatt a

```
>SET PGPPATH=C:\PGP
```

végrehajtása után a program a C:\PGP\pubring.pgp fájlt keresi nyilvános kulcsfájlként.
Ezt célszerű lehet az AUTOEXEC.BAT-ban beállítani. Ha a változó értéke nincs meg-
adva, a PGP mindig a kurrens directoryban keresi a működéséhez szükséges fájlokat.

A konfiguráció beállítása: CONFIG.TXT . A CONFIG.TXT fájlban be lehet állí-
tani sok PGP-paramétert, amelyeket egyébként a parancssoron kellene megadni minden
indításkor. A CONFIG.TXT egy egyszerű szövegfájl, amit kedvenc ASCII- szövegszer-
kesztőddel szerkeszthetsz. A program a PGPPATH által beállított (vagy ennek hiányá-
ban a kurrens) könyvtárban keresi a CONFIG.TXT-t.

A konfigurációs fájlban lehet olyan opciókat definiálni, mint azt, hogy hol tárolja a
PGP az ideiglenes fájlokat, hogy melyik idegen nyelven jelenítse meg a promptokat és
a diagnosztikai üzeneteket, vagy itt lehet beállítani a kulcsok validálásához szükséges
aláírások súlyát és mennyiségét.

A konfigurálással kapcsolatos részleteket a dokumentáció II. részében találod, egy kü-
lön erről szóló fejezetben.

Sebezhetőség . Minden biztonsági rendszerbe be lehet hatolni. A PGP-t is számos
úton ki lehet játszani. Az erre szolgáló módszerek (melyeket ki kell védened): pl. a titkos
kulcs és jelszavának megszerzése, a nyilvános kulcs módosítása, hamisítása, a számí-
tógépen vagy a diszken rosszul letörölt eredeti fájlok, vírusok és trójai faló programok,
közvetlen behatolás, az elektromágneses emisszió monitorozása, „lefigyelés” a többfel-
használós rendszerben, adatforgalom-vizsgálatok, esetleg még direkt rejtjel-fejtés is.

A felsorolt dolgok részletes leírása a dokumentáció II. részében a Sebezhetőség fejezet-
ben található.

A hamis biztonság . Egy titkosító szoftver esetén mindig felmerül az a kérdés, hogy miért kellene bíznod benne. Még akkor is, ha magad vizsgáltad át a forráskódot, kriptográfiai ismeretek hiányában nem feltétlenül tudod jól megítélni a biztonság fokát. És ha jól értesz is a kriptográfiához, előfordulhat, hogy az algoritmus apró hiányossága esetleg elkerülik a figyelmedet.

Amikor gimnáziumba jártam, még a hetvenes évek elején, kitaláltam egy zseniális titkosítási módszert. Ebben egyszerűen pseudorandom számokat kellett vegyíteni a nyersszövegbe, így jött létre a kódszöveg. A bekevert számok lehetetlenné teszik a frekvenciaanalízisen alapuló fejtést, és még a kormányzók se bírnák visszafejteni az üzenetet. Nagyon elégedett voltam a felfedezésemmel. És rendkívül magabiztos.

Évekkel később megtaláltam a módszeremet számos kriptográfiába bevezető könyvben. Príma. Más kódolóknak is eszébe jutott ez a dolog. Csak az volt a baj, hogy a módszer azért került bemutatásra, hogy néhány elemi kriptoanalízis-technikát demonstráljanak az így kódolt üzenetek visszafejtésénél. Na ennyit a zseniális módszeremről. . .

Az esetből kénytelen voltam levonni azt a tanulságot, hogy a rejtő- algoritmus tervezésénél mennyire könnyű beleesni a hamis biztonság érzetébe. Az emberek többsége nem érti, hogy mennyire nehéz egy olyan titkosító algoritmust találni, mely ellenáll egy erőforrásokban bővelkedő ellenfél tartós feltörési kísérleteinek. Jónéhány szoftver- fejlesztő megtette, hogy teljesen naiv séma alapján írt titkosító algoritmust (akár többen is ugyanazt.) Egyeseket bele is építettek a kereskedelmi termékekbe és jó pénzt csináltak belőle, amikor eladták sokezer gyanútlan felhasználónak.

Ez azzal analóg, hogy olyan biztonsági övet árul valaki, amely ránézésre jónak tűnik, viszont a legcsekélyebb karambol esetén is elszakad. Az ilyenek használata még veszélyesebb, mint teljesen biztonsági öv nélkül autózni. Hiszen senki sem gondolja, hogy rosszak egy valódi karambol bekövetkeztéig. A gyenge algoritmuson alapuló titkosítás használata hasonló veszélynek teszi ki féltett adatainkat, amelyek titkosítás nélkül esetleg nem kerülnének ugyanilyen helyzetbe.

Egyes kereskedelmi csomagok a DES algoritmust használják, ami egy elég jó hagyományos titkosítási eljárás, amelyet saját köznapi adatainak védelmére kormány is előír (de nem úgy kiemelt titkosságú adatokra – hmm.). A DES többféle módon futhat, melyek közül egyes módok jobbak, mint a többiek. Így a kormány előírja, hogy ne alkalmazzák a legegyszerűbb ECB (Electronic Codebook) módot, hanem az erősebb CFB (Cipher Feedback) vagy CBC (Cipher Block Chaining) módot kell alkalmazni.

Ennek ellenére az általam látott DES-t alkalmazó programokban az ECB-t alkalmazzák. Beszéltem néhány ilyen program készítőjével, és kiderült, hogy egyesek nem is hallottak a CBC és CFB módokról, sem az ECB ismert gyengeségeiről. Már az a tény is felháborító, hogy az ilyen témában programozónak halvány sejtelve sincs a kriptográfia alapjairól. És ezek a szoftverek ráadásul szoktak egy alternatív algoritmust is tartalmazni, melyet a lassú DES helyett javasolnak használni. A szerző talán abba a hitbe ringatja magát (és a fogyasztókat), hogy az alternatív rejtjelzés ugyanolyan biztonságos

mint a DES lenne, de kikérdezve gyakran tapasztaltam, hogy az én előzőekben ismertett „zseniális” algoritmusom egy változatáról van szó. Vagy a programozók másik csoportja egyszerűen semmit nem hajlandó elárulni a titkosítás módjáról, ugyanakkor elvárja, hogy feltétel nélkül megbízzak benne. Elhiszem, hogy meg van győződve, hogy az alkalmazott módszere zseniális, de hogyan lehetnék ebben biztos, ha nem láthatom? A tárgyilagosság érdekében le kell szögezmem, hogy a fentiekben tárgyalt szoftverek nem olyan társaságoktól származnak, melyek kriptográfiára specializálódnak.

Van egy AccessData nevű társaság (87 East 600 South, Orem, Utah 84058, tel: 1-800-658-5199), mely 185 dollárért árul egy olyan programot, mely megfejt a WordPerfect, Lotus 1-2-3m MS Excel, Symphony, Quattro Pro, Paradox és MS-Word 2.0 programok beépített titkosítását. És nem egyszerűen kitalálja a jelszót, hanem valódi kriptoanalízist használ. Az emberek jó része saját – elfelejtett jelszóval védett – adatainak visszafejtéséhez vásárolja meg az említett programot. Meg törvényes végrehajtással foglalkozó társaságok, amikor a lefoglalt fájlokba szeretnének beleolvasni. Beszéltem Eric Thompsonnal, a program szerzőjével, ő mondta, hogy a program a másodperc tört része alatt végez a megfejtéssel, csak beleépített jópár lassító ciklust, hogy a vásárló szemében a megfejtés komoly munkának tűnjön. Azt is közölte, hogy a PKZIP jelszavas védelme ugyanúgy könnyen feltörhető és a végrehajtók máris hozzájutnak ehhez a szolgáltatáshoz egy másik cég által.

A rejtjelzés bizonyos tekintetben hasonló a gyógyszerekhez. Az integritása a döntő. A rossz penicillin kívülről pont ugyanolyan mint a jó penicillin. A táblázatszámoló szoftverről ránézésre meg tudod mondani, hogy rossz, de hogy állapítod meg a titkosítóról, hogy az? A rossz titkosító által létrehozott kódszöveg pont olyan jónak tűnik mint a jó titkosítóé. Ez jó terepet kínál a „kuruzslóknak”. De szemben a gyógyszert hamisítókkal, a rossz titkosítók készítői még csak nincsenek is tisztában azzal, hogy amit eladnak, az a hamis biztonság. A készítők általában jó programozó szakemberek, de életükben nem láttak még kriptográfiáról szóló irodalmat. És meg vannak győződve róla, hogy jó titkosítót tudnak készíteni. Miért is ne? Hisz a feladat nem tűnik olyan nehéznek. És a program látszólag jól is működik.

Az az ember, aki kitalált egy szerinte megtörhetetlen titkosítást vagy egy rendkívül ritka zseni, vagy naiv és tapasztalatlan a területen.

Emlékszem, hogy Brian Snow, az NSA (USA biztonsági szolgálat) egyik sokra tartott előljáró kriptográfusa azt mondta: sose hinne egy olyan ember által kitalált algoritmusban, aki előzőleg nem szerzett komoly tapasztalatokat a rejtjelfejtés területén. Az ilyen tapasztalat nagyon sokat ér. Ahogy megfigyeltem, a titkosításra szakosodott cégeknél csupa ilyen tapasztalt embert látni. „Igen”, folytatta Snow, „ez jelentősen megkönnyíti a munkánkat itt az NSA-ban”. Magam is osztom ezt a véleményt.

A hamis biztonsággal maga a kormány is kereskedik. A II. világháború után pl. eladták a németektől zsákmányolt Enigma rejtjelző készülékeket a fejlődő országok kormánynak. De azt nem közölték, hogy a szövetséges hatalmak már a háború alatt megfejtették

az Enigma rejtjelezést. Ezt a tényt a háború után sokáig titokként őrizték. A ma elterjedt Unix rendszerek közül elég sok ma is az Enigma kódolást használja a fájlok titkosítására, részben azért, mert a kormány akadályokat gördített a jobb sémák alkalmazásának útjába. Az RSA algoritmus 1977-ben való publikálását is szerették volna megakadályozni. És azóta is megakadályozzák a gyártók minden olyan törekvését, melyek valóban biztonságos telefonokkal látnák el a köznépet.

Az USA biztonsági hivatalának (NSA) is az a legfőbb munkája, hogy információkat gyűjtsön. Elsősorban a privát telefonbeszélgetések lehallgatásával (lásd James Bamford „The Puzzle Palace” c. könyvét). Az NSA-nál jelentős tudás és erőforrás áll rendelkezésre a különféle titkosítások megfejtéséhez. Ha az emberek nem jutnak jó titkosítóprogramhoz, az csak öröm az NSA-nak. Ugyanez a szervezet foglalkozik a titkosító algoritmusok minősítésével és előírásával is. Egyesek szerint ez komoly érdekkonfliktust jelent. Olyan, mintha a kecskére bíznák a káposztát. Az NSA a saját algoritmusának használatát erőlteti, melynek működésébe nem lehet betekinteni (lévén titkos), ugyanakkor elvárja, hogy bízzunk benne és használjuk. Ezzel szemben bármelyik kriptográfus megmondhatja, hogy egy jó titkosító algoritmust nem kell titokként őrizni, a nyilvánosságra hozatal semmiben nem jelent támpontot a megfejtéshez. Csak a titkosító kulcsokat kell titokban tartani. Az NSA emberein kívül ki tudhatja, hogy az algoritmus megbízható? És mindenki tudja, hogy nem nehéz egy olyan algoritmust tervezni, amelynek alapján azután csak ők tudnak kódot visszafejteni, amíg nem hozzák nyilvánosságra. Elképzelhető, hogy szándékosan hamis biztonságot árulnak?

Én nem vagyok annyira bizonyos a PGP megbízhatóságában, mint annak idején a saját „zseniális” algoritmusomban hittem. Ha így lenne, azt rossz jelnek tartanám. Viszont abban biztos vagyok, hogy az RSA nem tartalmaz semmilyen könnyen felfedhető gyengeséget. A PGP-ben alkalmazott algoritmusokat a civil szféra kiemelkedő kriptográfusai tervezték, és ezek egyenként komoly gyakorlati próbáknak voltak alávetve. A program forráskódja széles körben hozzáférhető, tehát bárki beletekinthet. A program sok éven át készült, és ezalatt rengetegen áttekintették. És én nem az NSA-nak dolgozom. Remélem, hogy ennyi elegendő ahhoz, hogy valaki megbízhasson a PGP-ben.

PGP - gyors áttekintés . Íme a PGP parancsainak gyors áttekintése:

Nyersszöveg kódolása a fogadó nyilvános kulcsával:

```
>PGP -e textfile f_userId
```

Nyersszöveg aláírása a saját titkos kulcsoddal:

```
>PGP -s textfile [-u s_userId]
```

Nyersszöveg aláírása a saját titkos kulcsoddal, majd kódolása a fogadó nyilvános kulcsával:

```
>PGP -es textfile f_userId [-u s_userId]
```

Nyersszöveg kódolása hagyományos kriptográfiával:

```
>PGP -c textfile
```

Kódszöveg kibontása, és aláírás ellenőrzése (illetve kibontás/ ellenőrzés bármilyen kombinációja):

```
>PGP ciphertextfile [-o plaintextfile]
```

Nyersszöveg kódolása több fogadó nyilvános kulcsával:

```
>PGP -e textfile f_userId1 f_userId2 f_userId3 f_userId4 ...
```

Kulcskezelő funkciók . Kulcspár generálása:

```
>PGP -kg
```

Kulcs importálása egy fájlból a kulcsfájlba:

```
>PGP -ka file [kulcsfile]
```

Kulcs exportálása a kulcsfájlból:

```
>PGP -kx[a] userId file [kulcsfile]
```

A kulcsfájl tartalmának listázása:

```
>PGP -kv[v] [userId] [kulcsfile]
```

A nyilvános kulcs kivonatának megjelenítése (pl. telefonon való egyeztetéshez):

```
>PGP -kv[c] [userId] [kulcsfile]
```

A kulcsfájl listázása az egyes kulcsok aláírásaival együtt:

```
>PGP -kc [userId] [kulcsfile]
```

A titkos kulcs userId-jének vagy jelszavának módosítása:

```
>PGP -ke userId [kulcsfile]
```

Kulcs vagy userId törlése a kulcsfájlból:

```
>PGP -kr userId [kulcsfile]
```

Más kulcsának saját kulccsal való aláírása a kulcsfájlban:

```
>PGP -ks f_userId [-u s_userId] [kulcsfile]
```

Egyes aláírások eltávolítása egy kulcsról:

```
>PGP -krs userId [kulcsfile]
```

Saját kulcs végleges érvénytelenítése:

```
>PGP -kd s_userId
```

Kulcs letiltása/engedélyezése a saját nyilvános kulcsfájlodban:

```
>PGP -kd userId
```

Még néhány parancs az ingyenceknek. Kódszöveg dekódolása az aláírás rajtahagyásával:

```
>PGP -d ciphertextfile
```

Aláírás létrehozása külön fájlba:

```
>PGP -sb textfile [-u s_userId]
```


Más opciókkal kombinációban használható opciók . Csak ASCII kódokat fogadó terjesztőhálózaton való továbbításhoz a Radix-64 kódolást eredményező -a opciót kell használni. Kulcs-export vagy kódszöveg létrehozásakor a parancssor:

```
>PGP -kxa userId file [kulcsfile]
```

```
>PGP -sea textfile f_userid
```

Ahhoz, hogy a nyersszöveg fájl a sikeres kódolás után a PGP felülírja és letörölje, a -w opcióval kell kiegészíteni a parancssort.

```
>PGP -sew textfile f_userid
```

Ha a nyersszöveg nem bináris, hanem szövegfájl, és a fogadó oldalon át kell esnie a helyi szövegformázáson, a parancssoron a -t opciót kell alkalmazni:

```
>PGP -seat message.txt f_userid
```

Ahhoz, hogy a kifejtett szöveget ne fájlba írjuk, hanem csak a képernyőre és ott képernyőnkénti megállás (more) mellett olvassuk:

```
>PGP -m ciphertextfile
```

Ahhoz, hogy kódoláskor megakadályozzuk a kibontáskor a fájlba írást (a kibontott szöveget kizárólag a képernyőn lehet olvasni), a kódoló sorban kell a -m opciót alkalmazni a többi mellett.

```
>PGP -steam message.txt f_userid
```

Ahhoz, hogy a kibontáskor az eredeti nyersszöveg-fájlnevet kapjuk vissza, a -p opció szükséges:

```
>PGP -p ciphertextfile
```

Unix filterként való viselkedéshez a -f opció szükséges. A program ilyenkor a standard inputot olvassa, a standard outputra ír.

```
>PGP -feast f_userid <input >output
```

Jogi kérdések . A témával kapcsolatos részleteket a dokumentáció második részében olvashatod ugyanilyen című fejezet alatt.

A PGP-ben használt nyilvános-kulcsos eljárás az USA területére a #4,405,829 patent alatt van bejegyezve. A szabadalom USA területén való használatának kizárólagos joga a Kaliforniai Public Key Partners (PKP) nevű társaságé. Amennyiben a PGP-t az USA területén kívül használod, ez jogsértő lehet. Az ezzel kapcsolatos részletekről a dokumentáció II. részében olvashatsz.

PGP ún. „gerilla” freeware, így szabadon beszerezheted és terjesztheted. Csak ne engem kérj, hogy küldjek neked egy példányt. Szerezd be magad valamelyik BBS-ről, Internet FTP helyről vagy ismerőseidtől.

Köszönetnyilvánítás . Köszönetet szeretnék mondani az alábbi embereknek, akik segítettek nekem a PGP program létrehozásában. A PGP 1.0 verzióját még én magam írtam, a későbbi verziókban nagy részeket több más szerző implementált, nemzetközi összefogásban, az én felügyeletem alatt.

Branko Lankester, Hal Finney és Peter Gutmann rengeteg munkát fektetett a PGP 2.0 sok funkciójának megírásába és a szoftver számos Unix variáns alá való adaptációjába. Hal és Branco herculesi erővel dolgozott az új kulcskezelési sémám megvalósításánál. Branco több időt töltött a PGP írásával, mint bármely más segítőtársam.

Hugh Kennedy írta át VAX/VMS alá, Lutz Frank Atari ST gépre, Cor Bosman and Colin Plumb pedig Commodore Amigára.

A program különféle nyelvekre fordítását és nemzeti adaptációját az alábbi emberek végezték: Jean-loup Gailly – Franciaország, Armando Ramos – Spanyolország, Felipe Rodriguez Svensson és Branko Lankester – Hollandia, Miguel Angel Gallardo – Spanyolország, Hugh Kennedy és Lutz Frank – Németország, David Vincenzetti – Olaszország, Harry Bush és Maris Gabalins – Lettország, Zygimantas Cepaitis – Litvánia, Peter Suchkow és Andrew Chernov – Oroszország, és Alexander Smishlajev – eszperantó. Peter Gutmann felajánlotta az új-zélandi angolba való fordítást, de végül úgy döntöttünk, hogy az USA angol elég lesz.

Jean-loup Gailly, Mark Adler és Richard B. Wales publikálta a ZIP tömörítő rutinokat és engedélyezte ezek felhasználását a PGP-ben. Az MD5 rutinokat Ron Rivest írta és adta át public domain-be. Az IDEA(tm) kódoló eljárást Xuejia Lai és James L. Massey fejlesztette ki az ETH- ban, Zurich-ben. Az IDEA PGP-ben való alkalmazásához az Ascom-Tech AG adott engedélyt.

Charlie Merritt tanította nekem eredetileg a RSA-ban szükséges nagy pontosságú számítások kivitelezését, Jimmy Upton pedig egy gyors szorzó/modulus-számító eljárást készített. Thad Smith készített egy még gyorsabb modulus-számító algoritmust. Zhaihai Stewart számos ötlettel járult hozzá a PGP fájlformátum és hasonlók tervezéséhez, ideértve azt, hogy egy kulcshoz több userId tartozhasson. A kulcs- aláírókra vonatkozó

ötletet Whit Diffie vetette fel. Kelly Goen a PGP 1.0 elektronikus úton való elterjesztésén dolgozott.

Colin Plumb, Derek Atkins, és Castor Fu a kódolásban segített. Szinkén a kódolásban, valamint sok egyéb dologban segített még Hugh Miller, Eric Hughes, Tim May, Stephan Neuhaus és sok-sok más ember, akiknek a neve éppen nem jut eszembe. Zbigniew Fiedorwicz és Blair Weiss irányítása alatt két Macintosh project is fejlesztés alatt áll.

A PGP 2.0 publikálása óta számos programozó küldött patch-eket és hibajavításokat, valamint segítséget az új rendszerekre való átírá- sokhoz. Túl sokan vannak ahhoz, hogy mindegyiküknek személyesen mondjak köszönetet.

A PGP projekt önálló életre kelt. Nagy politikai jelentőségére való tekintettel számos önkéntes programozót vonzott szerte a világban. Emlékszel a kőlevesről szóló mesére? Az egyre sűrűbb levesen keresztül már alig látszik az a kő, amelyet én dobtam bele annak idején, útjára indítva a PGP-t.

A szerzőről . Philip Zimmerman egy 19 éves gyakorlattal rendelkező programozó. Kiemelt területei a real-time rendszerek, a rejtjelezés, az autentikáció és az adattovábbítás. Gyakorlata kiterjed a pénzügyi információs hálózatokban alkalmazott adatbiztonsági rendszerek, kulcskezelési protokollok, valós idejű multitaszk rendszerek, operációs rendszerek és helyi hálózatok tervezésére és megvalósítására.

Ez egy rövid használati útmutató a PGP 2.6.2i parancsaihoz.

Új kulcspár generálás `pgp -kg`

Kulcs hozzáadás `pgp -ka kulcsfájl [kulcskarika]`

Kulcs kimásolás `pgp -kx[a] azonosító kulcsfájl [kulcskarika]`

Kulcs(ok) megnézése `pgp -kv[v] [azonosító] [kulcskarika]`

Ujjlenyomat megnézése `pgp -kvc [azonosító] [kulcskarika]`

Részletes ellenőrzés `pgp -kc [azonosító] [kulcskarika]`

Kulcs vagy azonosító eltávolítása `pgp -kr azonosító [kulcskarika]` (Ismételd, ha több azonosítója van egy kulcsnak)

Új azonosító hozzáadása `pgp -ke azonosító [kulcskarika]`

Jelszó megváltoztatása `pgp -ke azonosító [kulcskarika]`

Megbízhatóság módosítása • `pgp -kx azonosító kulcs.tmp [kulcskarika]`

- `pgp -kr` azonosító [kulcskarika]
- `pgp -ka` kulcs.tmp [kulcskarika]

Nyilvános kulcs aláírása `pgp -ks` más_azonosítója [-u aláíró_azonosítója] [kulcskarika]

Aláírás eltávolítása `pgp -krs` azonosító [kulcskarika]

Visszavonás, ki-/bekapcsolás `pgp -kd` azonosító [kulcskarika]

Lekódolás `pgp -e[a]` textfájl TO_id [TO_id2 TO_id3...]

Aláírás `pgp -s[a]` textfájl [-u MY_id]

Aláírás és lekódolás `pgp -se[a]` textfájl TO_id [TO_id2 TO_id3...] [-u MY_id]

Különálló igazolás `pgp -sb[a] [+clearsig=on]` mainfájl [-u MY_id] (Bináris fájlhoz is) (clearsig=on beállítható a CONFIG.TXT fájlban is)

Lekódolás csak IDEA-val `pgp -c` textfájl

Visszakódolás vagy az aláírás ellenőrzése `pgp [-d] [-p]` sifre (-d megtartja a pgp adatot, -p az eredeti fájl-névhez)

Különálló igazolás ellenőrzése `pgp igazoló-fájl [ellenőrizendő-fájl]` (Ha a fájl-nevek azonosak, elhagyható az [ellenőrizendő-fájl])

Tedd hozzá az [a]-t ha ASCII kimenetet szeretnél

Tedd hozzá a [-o kimeneti-fájl]-t, hogy megadd a kimeneti-fájl-t

Tedd hozzá a [+batchmode]-ot, hogy ERRORLEVEL-t kapj vissza

Tedd hozzá az [f]-et, hogy a ki-/bemenet átirányítható legyen (`pgp -f[ARGS] <bemenet> >kimenet`)

Tedd hozzá a [w]-t, hogy felülírja az eredeti fájl-t (lekódolásnál)

Tedd hozzá az [m]-et, hogy az eredetit csak megnézni lehessen (nincs fájl-ba írás)

Tedd hozzá a [t]-t, hogy módosítsa a sorvégeket (unix, stb.)

A kódolásnál további azonosítók fájl-ból történő beolvassáshoz használd a -@ opciót: `pgp -e textfájl egy_azonosító -@további_azonosítók.txt`

Secure Hypertext Transfer Protocol (Secure-HTTP)

„S-HTTP néven is ismert. Draft Internet szabvány, léteznek megvalósításai, gyakorlati alkalmazásai. Az S-HTTP nem önálló protokoll, hanem a szabványos HTTP kiterjesztése. Az S-HTTP képes az adatforgalom mindkét irányú titkosítását, digitális aláírás alkalmazását és hitelesítést biztosítani a kliens és szerver között. A választható titkosítási

eljárásokra nem ad megkötést (támogatja a nyilvános kulcsok használatát is), megengedi nem-S-HTTP tudatú kliensek alkalmazását is (sőt kényesebb igényeknek is eleget tesz).” [?]

Secure Socket Layer (SSL)

„A Netscape által kifejlesztett és támogatott SSL nem a HTTP kiterjesztése, nem is kizárólag Web-specifikus, más Internet alkalmazásokhoz is használható (így az S-HTTP-vel is kompatibilis). Az SSL a HTTP-nél alacsonyabb szintű protokoll, biztonságos csatornát képes létrehozni két végrendszer között: end-to-end titkosítást, digitális aláírást, kliens és szerver azonosítást stb. támogat. Az SSL külön URL használatát követeli meg ún. biztonságos szerverek esetén (»https://« kezdetűt a »http://« helyett). A Netscape és számos más kliens program képes kezelni az SSL-t, s a kliens grafikus felülete jelzi, hogy biztonságos, vagy nem biztonságos szerverhez csatlakoztunk.” [?]

Kerberos

A hálózaton a felhasználók és a szolgáltatások eredetiségét vizsgálja. Ez egy megbízható, három oldalú szolgáltatás. Ez azt jelenti, hogy van egy harmadik fél (a kerberos szerver), akiben mindenki megbízik a hálózaton (felhasználók és szolgáltatások, amiket „megbízóknak” nevezünk).

DES alapú titkosítást használ, és a titkosító algoritmus „jósa” miatt exportkorlátozás alá esik az Egyesült Államokban. Ezért az Egyesült Államokon kívüli társaság tartja karban és fejleszti.

Sok dolgot magában foglal, a telnet-től a POP3 szerveren keresztül, egészen az X kapcsolatok titkosításáig.

„Amikor a felhasználó bejelentkezik valamelyik számítógépbe, és már beírta a nevét a login: üzenet után, a login program (aki a bejelentkezéskor fellépő feladatokat végzi el) egy üzenetet küld a Kerberos illetékesség-vizsgáló szervernek. Az üzenet tartalmaz két szöveget: az egyik szöveg a felhasználó azonosítója, a másik szöveg pedig az ún. jegykiadó szerver (ticket-granting szerver, TGS) nevét tartalmazza.

Az illetékesség-vizsgáló szerver a fent említett adatbázisból kikeresi a kapott üzenetben lévő két azonosítóhoz tartozó kulcsokat (a felhasználói azonosítóhoz tárolt kulcs megegyezik a felhasználó titkosított jelszavával). Ezután a szerver egy válaszüzenetet küld vissza, ami tartalmaz egy titkos ún. TGS-kulcsot, és egy ún. jegyet. Ez a TGS-jegy tartalmazza a felhasználói azonosítót, a jegykiadó szerver (TGS) nevét, annak a számítógépnek az Internet címét, amelyen a felhasználó dolgozik, és a fent említett TGS-kulcsot. A TGS-jegy az üzenetben titkosítva van a felhasználó által nem ismert kulccsal (a jegykiadó szerver azonosítójához tartozó kulccsal), és a teljes üzenet titkosítva van a felhasználó titkosított jelszavával, hogy más ne férjen hozzá a tartalmához.

A felhasználó jelszavának beadása után az azonosító szervertől visszakapott üzenetet a login program a felhasználó titkosított jelszavával dekódolja, és a TGS-kulcsot valamint a (titkosított) TGS-jegyet eltárolja.

Ezután ha a felhasználónak valamilyen hálózati szolgáltatásra van szüksége, akkor a jegykiadó szervertől (TGS-től) kérnie kell egy „jegyet” a kívánt szerverhez, és a szervernél ezzel a jeggyel azonosíthatja magát.” [?]

12.1.7. Linux szerverek biztonsági kérdései

Ez a rövid lélegzetű íromány főként kezdő rendszergazdáknak íródott, ennek ellenére nem szeretnék minden - máshol is könnyen hozzáférhető vagy a felsorolt szoftverek dokumentációjában olvasható - megoldást részletezni. Ha nem írtam le részletesen valamit, akkor annak egyszerűen utána lehet nézni (man, howto, stb.) Szóval RTFM :). Később, ha lesz rá igény, részletesebben kifejtek egyes témákat, de szívesen venném, ha más is hozzászólna, esetleg részt vállalna az adott téma kidolgozásában. Bármilyen építő jellegű hozzászólást szívesen fogadok.

Bevezető

Csak a rendeltetésszerű működéshez szükséges csomagokat telepítsük. Célszerű legalább a */tmp*, a */home* és a */var* könyvtárakat külön partícióra tenni. Ez megnehezíti néhány program hibáinak kihasználását (pl. állományok jogosulatlan átírása az adott file-ra mutató link segítségével, bár a symlink-es exploitok ellen nem véd), lehetővé teszi a disk quota hatékonyabb beállítását illetve megakadályozza, hogy a syslogd üzenetei megtöltsék a root partíciót, megnehezítve ezzel a rendszer normális működését.

Ha még óvatosabbak akarunk lenni, a */var/spool*-t is külön partícióra tehetjük. Ezzel megakadályozható, hogy a */var* valamilyen módon való megtöltése esetén (pl. DoS attack) se álljanak le a *spool*-t használó processzek, illetve viszont: ha a *spool* meg is telik, a */var* nem. A */tmp*-t és a */home*-ot érdemes nosuid paraméterrel mountolni. A */home* mountolható noexec paraméterrel is, ha a felhasználók saját programokat nem futtatnak. A */tmp* nosuid mountolása megelőzi a setuid tmp file-ok race condition jellegű hibáinak kihasználását. Szóval ez az */etc/fstab*-ban valahogy így nézzen ki:

```
/dev/hda2 /tmp ext2 defaults,nosuid 1 2
/dev/hda4 /home ext2 defaults,nosuid,noexec 1 2
```

A felhasználók által foglalható maximális merevlemez területet is érdemes korlátozni, így a userek nem tudnak buta dolgokat (pl. „cat /dev/zero > nagyfile”) művelni. Ennek beállításához ajánlott olvasmány a 'quota howto'. Quota kell a */tmp*-re (a nagy tmp file-ok létrehozásának megakadályozására illetve, hogy az elvetemültebbek ne itt tárolják a

dolgaikat), a */var*-ra a mail spool méretének limitálása miatt, illetve a */home*-ra a home könyvtárak méretének korlátozása érdekében.

Telepítés után ellenőrizzük, nincsenek-e már ismert biztonsági hibák rendszerünkben. Az ellenőrzést kezdjük az adott disztribúció hibalistájával, majd nézzük át a biztonsággal foglalkozó site-okat (pl. Rootshell), listák archívumait (pl. BUGTRAQ). Ha valamilyen már találtak biztonsági hibát, azonnal cseréljük le. A legtöbb helyen a javított csomagok elérhetőségét is megadják. Ha még nincs javított verzió, próbálkozhatunk a közreadott patchekkel is. Ha patch sincsen, az adott szolgáltatást célszerű ideiglenesen leállítani. Javítás után minden esetben teszteljük le a rendszert, valóban kijavítottuk-e a hibát. A fent említett levelezési listára feliratkozni is érdemes, így előbb értesülhetünk a számunkra fontos hibákról és patchekről.

Rossz hozzáállás, hogy „itt nincs semmi, ami bárkinek is kellene, nem éri meg betörni”. Igenis megéri. Általában nem a célgépre törnek be először, hanem keresnek egy könnyen törhető, de gyors kapcsolattal rendelkező gépet, amit ugródeszkeként használnak. Így a betörő valódi címe rejtve marad, mivel az ugródeszkeként szolgáló gépről általában mindent törölnek, ami a nyomukra vezethetne. Nem igazán jó megoldás az sem, ha egy jól működő, hibátlan programot minden megfontolás nélkül lecserélünk csak azért, mert megjelent egy új verziója. Jusson eszünkbe, hogy az új verzió új hibákat is eredményezhet, ezért upgrade előtt mindig olvassuk el a README/CHANGES/FEATURES/stb.-t. Azért vannak.

Külső behatolási kísérletek elleni védelem

Hálózati kommunikáció védelme. Ne használjunk *telnet*-et és *ftp*-t a rendszeradminisztrációhoz – ha lehet, mindig kerüljük a használatukat – mivel a kommunikáció ebben az esetben titkosítás nélkül folyik; egy sniffer-rel a közepesen képzett próbálkozó is sok hasznos információt (pl. jelszavak) szedhet össze. Ugyanez igaz az *rsh*, *rcp*, stb. szolgáltatásokra, melyek még veszélyesebbek. Épp ezért javasolt az *SSH* vagy *SRP*-s *telnet/ftp* (Secure Remote Password) használata, amelyek a hálózati kommunikációt biztonságos mértékben titkosítják.

Sniffer-ek. Aki a gépünk és a célgép közötti forgalmat figyelni képes, értékes információkhoz juthat a titkosítatlan kapcsolatokról. Ehhez persze az kell, hogy a hálózaton valahol legyen egy root accountja. Ha ez megvan, több program is rendelkezésére áll a *tcpdump*-tól (minden Linux disztribúcióban megtalálható) az intelligensebb programokig (pl. *sniffit* vagy *tcpflow*).

Működésük a hálózati interface promiscuous módba kapcsolásán alapszik. Detektálásuk is erre épül, mivel létezik program a promiscuous mód-ban levő ethernet interface-ek keresésére. Ezzel általában meghatározható, milyen címről fut a sniffer (feltéve, hogy az elkövető nem olyan operációs rendszert használ, mely a detektálást megakadályozza). A

detektorok működési elve valami ilyesmi: végigpingeli a lokális hálózatot, így megkapja minden interface hardver címét. Ezután egyenként minden IP-hez tartozó hardver (MAC) címet véletlenszerűen megváltoztat az ARP cache-ben és újra pingel. Amelyik gép erre válaszol, ott az interface promiscuous módban van. Persze ez még nem jelenti, hogy sniffert futtat, mert néhány más program is átkapcsolja az interface-t ebbe a módba.

Portscan-ek. A portscan kapcsolat kezdeményezése több portra, egymás után. Azt jelenti, hogy valaki kíváncsi arra, milyen szolgáltatások futnak a gépen. Sok rendszerellenőrző program is használja ezt a technikát az elemzéshez. A scannelő valószínűleg arra kíváncsi, melyik szolgáltatást kihasználva tudna betörni gépünkre. Védekezésként log-olhatjuk a TCP/UDP/ICMP scan-t ipp-vel vagy más hasonló programmal.

A rendszeresen visszatérő próbálkozók gépének adminisztrátorait célszerű tájékoztatni az esetekről. Ha ők nem tesznek semmit a helyzet megváltoztatására, érdemes a firewall-on kifilterezni ezeket a címeket.

Address spoofing. Ez már a haladóbb technikák közé tartozik. . . eléggé komoly hálózati ismeretek szükségesek hozzá.

A felállás a következő: adott A és B gép, illetve C a támadó, aki át akarja venni B szerepét.

Az első esetben az ICMP redirect-tel foglalkozunk. C, amennyiben képes figyelni B szegmensét (mert pl. már betört valamelyik lokális hálózaton lévő gépre), a megfelelő pillanatban A routerének küld egy ICMP redirect csomagot B routere nevében. Ezután A routere úgy tudja majd, hogy B – akivel kommunikálni akar – a C routerének irányában van.

Nem ismerek konkrétan kidolgozott védelmet a támadás ellen, de firewall használata a helyi és a külső hálózat között sokat segíthet. Így akár ki is tiltható a kívülről érkező ICMP forgalom.

A következő eset, amikor C (a támadó) nem tudja figyelni a hálózat forgalmát.

Ilyenkor valahogy meg kell bolondítani A routerét, hogy az úgy tudja, hogy B routere C irányában van, tehát A C felé fogja a kapcsolatot kezdeményezni. C gépe természetesen B gépének IP címét veszi fel. Ez általában a routing táblák valamilyen módon való átírását jelenti. Ez lehetséges úgy, hogy a támadó A routerébe tör be először és ott írja át, de ha A routere SNMP-vel menedzsel, (hibás vagy rosszul beállított SNMP szoftver esetén) lehetséges átkonfigurálni betörés nélkül is. Egy jól beállított router az ilyen jellegű támadások nagy részét megfogja.

A harmadik esetben B A szegmensén van.

Ekkor C, miután sikeresen kiiktatta B-t (azaz valamilyen formában elérte, hogy B leszakadjon a hálóról), átveszi B hardware címét is. Ezután A B helyett C-vel fog kommunikálni.

Nem tudok megfelelő védelemről, de ha ez a veszély fennáll, hasznos lehet a helyi hálózatot firewall-lal elválasztott részekre szabdalni. A kritikus hálózatrészekre csak megbízható host-ok kerülhetnek. Így C csak akkor érhet el eredményt, ha B szegmensén van. Ekkor viszont sokat segíthet, ha B jól karbantartott gép, azaz nem könnyű kiiktatni. Mindenesetre érdemes *arpwatch*-ot használni. Ez a program figyeli a hálózati forgalmat és a detektált IP és hadver címeket kiírja. Így észlelhető, ha valaki a lokális hálózaton egy eszköz hadver címét állítgatja.

DNS spoofing. Ismét az előző felállás: A és B egymással akarnak kommunikálni, de ott van C, aki át akarja venni B helyét.

C tehát megnézi B name server-ének címét és betör rá (ha ez nem sikerül, akkor itt a vége), átállítja B IP címét a sajátjára, majd újraindítja a name servert. Ezután A, ha B hostnevét használja a kapcsolat felvételéhez és nem ellenőrzi annak valódiságát (azaz nem kéri B reverse name server-től a nevéhez tartozó IP címet), akkor B helyett C-vel fog kommunikálni.

Elég jó védelmet nyújt ellene a *tcp-wrapper*, illetve a megfelelő kliens és szerver programok használata.

TCP/UDP spoofing. Hibásan tervezett vagy implementált protokoll stack esetén vagy a protokollok hiányosságaiból fakadó okok miatt lehetséges a már felépült vagy kezdeményezett kapcsolatok „elrablása” vagy a kapcsolatba idegen csomagok „becsempészése” harmadik személy által. Ehhez persze komoly hálózati ismeret, programozási tudás és a TCP/IP protokollok részletes ismerete szükséges.

100%-os védelem nincs ellene, de a felfedezett hibák gyors kijavítása és egy jól konfigurált firewall sokat tud segíteni.

man-in-the-middle attack. A felállást már ismerjük: A és B kommunikációjába C be szeretne avatkozni. Ha a csatorna végpontjai védettek, a csatorna A és B közötti szakaszának „megcsapolása” lehet a megoldás. Ilyenkor C rákapcsolja a saját terminálját a csatornára és az ott folyó kommunikációt lehallgatja. Amikor lehetősége van, a csatornát észrevétlenül átvágja, majd mindkét végét a saját termináljára kapcsolja, így az egyfajta átjáróként szolgál A és B között, megadva ezzel C-nek a beavatkozás lehetőségét. Tehát C küldhet A-nak és B-nek is a másik nevében csomagokat, a választ pedig elfogja. A számára érdektelen kommunikációba nem avatkozik be.

Védekezni csak a teljes kapcsolat kódolásával lehetséges, de még ez sem véd 100%-ig az olyan esetekben, amikor C a kapcsolat kezdetétől képes beavatkozni a kommunikációba. Az utóbbi eset ellen bizonyos protokollok (pl. *SSL*) implementációja megfelelő védelmet biztosít ún. certificate-ek használatával.

Denial of Service (DoS). Ez a támadások egy speciális fajtája, amikor a támadó nem betörni akar, hanem egy adott szolgáltatás működését szeretné megbénítani. Hogy pontosítsunk: az adott kiszolgáló program vagy az operációs rendszer hibája segítségével a program rendeltetésszerű működését megakadályozza. A hiba lehet például egy távolról kihasználható buffer overflow, amelynek következtében a program illegális műveletet hajtana végre vagy számára nem hozzáférhető memóriaterületre írna, így az operációs rendszer a futását megszakítja. Elképzelhető olyan hiba is, amit felhasználva a program egy része vagy egésze lefagyasztható, vagy hibás működésre bírható. Lehetséges olyan támadás is, amely az egyszerre kiszolgálható kapcsolatok számának a határérték fölé növelésével teszi használhatatlanná az adott szolgáltatást. Védekezni a szerver programok karbantartásával és helyes beállításával lehet.

Resource starvation. Ez többé-kevésbé a Denial of Service attack-ek kategóriájába tartozik. Ez esetben a támadó a gép erőforrásainak (memória, processzor, háttértár) kapacitását igyekszik olyan mértékben kihasználni, hogy ezzel más felhasználó számára a rendszert használhatatlanná tegye. Ennek több módja ismert, lehetséges lokálisan (azaz a gépre bejelentkezve), (pl. fork bomba) illetve kívülről végrehajtani a támadást (pl. a syslogd üzeneteit felhasználva megtölthető a partíció vagy a régebbi *Apache httpd*-hez intézett kérdésben megfelelő számú „../..” hivatkozást elhelyezve a *httpd* által okozott processzorterhelés ugrásszerűen megnövelhető).

Ide tartozik még a hálózati sávszélesség kihasználása is. Ekkor a TCP/IP protokollok hiányosságait vagy egy rosszul beállított hálózati eszközt felhasználva olyan mértékben megnövelhető a hálózati forgalom, hogy a felhasználók által kezdeményezett kapcsolatok timeout-tal lépjenek ki. Erre példa a közelmúltban kitalált „smurf” attack, amikor a támadó a célhálózat broadcast címére állított forráscímű csomagokkal bombázza a hálózati eszközöket. Az eredmény elképzelhető. E támadások elleni védekezés megvalósítási módja általában csak a támadás megvalósítása után válik közismertté.

Web browser attack. A népszerű web böngészők (Netscape, Internet Explorer) méretének növekedésével egyre több kritikus hiba kerül napvilágra. A kliens oldali Javascriptekkel kapcsolatos hibákat sem szabad elhanyagolni. A Java biztonsági modellje elég jó, az előforduló hibák általában a hibás vagy hiányos implementációból adódnak. Bizonyos hibák kihasználásával a gépünkön tárolt és számunkra olvasható file-ok hozzáférhetővé válnak a látogatott oldal készítője számára. Ezért ajánlott letiltani a Java support-ot a böngészőben, ha nem megbízható site-okat látogatunk.

Belső támadások elleni védelem

Fork bomba. Ez egy kis programcska, amely a CPU időt és memóriát eszik. Védekezni ellene erőforrás limitek beállításával lehet. Az újabb Linux disztribúciók már alapból PAM-ot használnak, így ez nem probléma.

Buffer overflow (stack overflow). Ha programozási hiba folytán egy adott buffer méretén 'túlírható', akkor a tömböt tartalmazó függvény visszatérési címe (a stack-en a buffer után vége után tárolódik) átírható. Ennek oka az, hogy a C compilerek fordításakor nem figyelik a tömbhatár-túllépés lehetőségeit, illetve léteznek olyan függvények, melyek a bufferbe írás során nem ellenőrzik annak méretét. Ez nem lenne gond, mert minden ilyen függvénynek létezik biztonságos (a tömbhatárokat ellenőrző) változata, csakhogy könnyű a rosszabb megoldást választani.

Így azokon a rendszereken, ahol a stack futtatható, érdekes dolgokat művelhetünk (pl. a futó program jogaival shell hívható a visszatérési cím egy 'exec(„./bin/sh”...)'-ra állításával). Ha a program `setuid/setgid`-es, akkor az kód, amire a visszatérési cím mutat, öröklí a futó program jogait, tehát pl. az előbbi esetben egy `setuid`-es shell-t kaphatunk. Súlyosabb a helyzet, ha az adott program egy hálózati kiszolgáló, mert bizonyos esetekben a hiba távolról is kihasználható, lehetőséget adva ezzel bárkinek a jogosulatlan hozzáféréshez.

A programozói gondatlanság ellen védelmet nyújt a StackGuard, amely egy GCC (GNU C Compiler) kiterjesztés. Érdemes használni, főként, ha valaki `setuid`-es programot fejleszt, de ha nem bízunk egy más által fejlesztett programban, az is újrafordítható vele.

Solar Designertől származik egy kernel patch, amely korlátozott mértékben védelmet nyújt a stack futtathatóságát kihasználó támadások ellen, de létezik módszer a megkerülésére.

A legjobb védekezés tehát a `setuid/setgid`-es programok ellenőrzése, valóban kell-e az nekik. Vannak programok, amelyeknek a korrekt működéshez kell, de néhánynak nem. Ezekről vegyük le. Néhány program futtatása biztonságosabb, ha valamilyen wrapper-t használunk, amely megnehezíti a hibák kihasználását.

Symlink attack. Known tmp filename attack-nak is nevezik. A probléma az, hogy egyes programok world-writable könyvtárba (pl. `/tmp`) írt tmp file-jainak neve kitalálható, tehát lehetséges a program indítása előtt létrehozni azon a néven egy linket, amely egy másik file-ra mutat. Ez különösen `setuid/setgid` bites program, illetve több privilégiummal rendelkező felhasználó vagy a rendszer által futtatott program esetén veszélyes, mivel így a symlink segítségével olyan file módosítható, amelyhez a linket létrehozó felhasználónak normális esetben nem lenne joga. Az értelmesebb programok használják a `TMPDIR` környezeti változót is, ennek beállításával (pl. `TMPDIR=„$HOME/tmp”`)

a */etc/profile*-ba vagy a */etc/security/pam_env.conf*-ba), csökkenthető a veszély. Ha egy programról kiderül, hogy hibás, upgradeljük vagy cseréljük le.

Race condition. Ez a symlink attack kiterjesztése arra az esetre, amikor a program ellenőrzi, hogy létezik-e már az adott tmp állomány, illetve nem symlink-e, de nem megfelelő módon nyitja meg (open használatakor az *O_EXCL* flag nélkül) vagy rosszul állítja be a hozzáférési jogokat. Ekkor a file ellenőrzése és megnyitása közötti időben létrehozhatunk ugyanazon a néven egy symlinket, amely egy általunk nem, de a program tulajdonosa/groupja (setuid/setgid program esetén) vagy használója számára olvasható/írható file-ra mutat, így már nekünk is jogunk lesz azt módosítani, esetleg még a hozzáférési jogai is átíródnak. Solar Designer erre is írt patchet, bár ilyen jellegű hibák ellen hatásos védelem csak a megfontolt programírás és telepítés, illetve a symlink attack-nél említett *TMPDIR* környezeti változó beállítása lehet.

IFS (Inter Field Separator, mezőhatároló) megváltoztatása. Az *IFS* az egymás után következő karaktorsorok (utasítások, paraméterek, stb.) elválasztására szolgál. Ennek átírása setuid/setgid-es programok esetén használható ki, ha a program írója nem kellő körültekintéssel használta a *system()* vagy az *exec()* függvényhívásokat. Egy *system(„./bin/akarmi”)* függvényhívás, ha kiadtuk az „export IFS=’/’” utasítást, a következőképp hajtódik végre: „bin akarmi”. Így tehát bárki elhelyezheti saját „bin” nevű programját a *PATH*-ban. Szerencsére ez nem egyszerű, mivel a *PATH*-ban lévő könyvtárak egy mezei felhasználó számára általában nem olvashatók¹.

Jelszavak megválasztása. Ne használjunk rövid vagy szótárból kikereshető szavakat és szóösszetételeket. Ugyancsak kerülendő a usernév használata a jelszó részeként vagy az 'Egyszerűszó', illetve az 'egyszerűszó+egy_két_szám' megoldás. Ezeket a variációkat percek, de legrosszabb esetben órák alatt dobja ki egy jelszófejtő program (amit érdemes néha lefuttatni), ha a password file rossz kezekbe kerül. A legtöbb jelenlegi rendszer már alapértelmezésben használ szótárakat és különféle algoritmusokat a megfelelően bonyolult jelszó kiválasztásához, így általában nem is fogadja el a felsoroltak egyikét sem. Felhasználóinkat is figyelmeztessük jelszavaik védelmére és arra, hogy accountjukat nem adják kölcsön senkinek.

A fent említett okok miatt ajánlott a shadow password használata. Így */etc/passwd* file – amely mindenki számára olvasható – nem tartalmaz jelszavakat, azokat a */etc/shadow* file-ban tárolja a rendszer, hozzáférési joga pedig csak a root-nak van.

File-ok változásainak figyelése. A legkomolyabb óvintézkedések mellett sem lehetünk biztosak, hogy nem lehet betörni szerverünkre. A behatoló viszont – ha kellően

¹Ez elírás a szerző részéről, „nem írható”-ra gondolt

ügyes és nem rombolni akar – nem hagy feltűnő nyomokat, viszont készít magának egy kiskaput, hogy később könnyedén hozzáférjen a gépünkhöz. Ez lehet egy megváltoztatott 'passwd' file, esetleg egy átírt 'login', stb. Ezeket szűrhetjük ki, ha *tripwire*-t használunk, amely a file-ok változásaira figyelmeztet. Ajánlott az általa generált file-t lemezre menteni, és biztonságos helyen tárolni. Ha bármi gyanús dolog történik, elővehető, és a gépen lévővel összehasonlítható.

PAM (Pluggable Authentication Module). Ez egy egységes autentikációs rendszer, feladata a felhasználók bejelentkezésével, memória és filerendszer használatával, valamint az autentikációs szolgáltatásokat nyújtó programokkal kapcsolatos beállítások kezelése.

A felhasználókra vonatkozó beállításokat a */etc/security/* könyvtárban lévő file-ok segítségével végezhetjük, a programokkal kapcsolatos config file-ok */etc/pam.d/* könyvtárban vannak. A részletes dokumentáció a */usr/doc/pam*/* alatt található.

- */etc/security/access.conf*: formátuma: permission : users : origins

permission „+”, ha engedélyezés, „-”, ha tiltás

users user vagy group neve

origins hostnév, domain név („.”-tal kezdődik), IP cím (alhálózati címnél „.” a végződés) vagy terminál azonosító

Ezen kívül a 2. és a 3. mezőben használhatók a következő operátorok:

ALL mindent helyettesít

EXCEPT kivételkezelés (pl. ALL EXCEPT root vagy ALL EXCEPT server.akarmi.hu)

LOCAL mindent helyettesít, ami nem tartalmaz „.” karaktert például:

- -: lamer: ALL EXCEPT .akarmi.hu (azaz lamer csak az akarmi.hu domain-ből léphet be)
- +: lamer: ALL EXCEPT .akarmi.hu (lamer mindenholon beléphet, kivéve az akarmi.hu domaint)

- */etc/security/limits.conf*: formátuma: domain type item value

domain user neve, group neve (@group formában) vagy * (azaz minden)

type soft (azaz a limit valamekkora túllépése megengedett) vagy hard (a limitet nem lehet túllépni)

item core (core file max. mérete KB-ban), data (adatterület max. mérete KB-ban), fsize (max. file méret KB-ban), memlock (a locked-in-memory címterület max. mérete), nofile (egyszerre megnyitható file-ok max. száma), rss (a tárban maradó rész max. mérete KB-ban), stack (a stack max. mérete KB-ban), cpu (max. CPU idő percben), nproc (processzek max. száma), as (címtérület mérete), maxlogins (egydőben ugyanazzal a névvel bejelentkezett felhasználók száma)

value az item értéke például:

- * hard core 0 (azaz a core file-ok max. mérete 0 KB)
- @student - maxlogins 4 (a student group-ba tartozó felhasználók egyszerre 4-en léphetnek be)

- /etc/security/pam_env.conf: formátuma: VARIABLE [DEFAULT=[value]] [OVERRIDE=[value]]

VARIABLE a környezeti változó neve

DEFAULT alapértelmezett érték beállítása VARIABLE változóra

OVERRIDE felülírja DEFAULT értékét, ha meg van adva

Események naplózása. A gépen történő események (belépés, kilépés, különböző kiszolgáló programokhoz intézett kérések) naplózása a rendszerbiztonság fontos eleme.

A számunkra fontos log file-okat általában a /var/log/ könyvtár alatt találjuk. Ezek közül a 'messages', a 'secure', a 'spooler' és a 'maillog' az, ami alapértelmezésben létezik. Tartalmuk általában a syslog() függvényhívás segítségével keletkezik.

A /etc/syslog.conf-ban beállítható, mit és hova szeretnénk logolni. A „mit” nagyrészt csak attól függ, mennyi információt akarunk kapni a rendszer működéséről. A „hova” azért fontos, mert egy betörés esetén a betörő megváltoztathatja a logokat, ha azok a gépen vannak tárolva. Ha logjainkat biztonságban szeretnénk tudni, irányítsuk át őket nyomtatóra vagy egy másik gépre, ahol nincsenek helyi felhasználók és nem fut semmilyen szolgáltatás (mail, httpd, telnet, ftp, stb.), amit felhasználva a gép feltörhető. Az átirányításra a syslogd is kínál lehetőséget, de érdemes a kapcsolatot ssh-val forwardolni, így egy sniffer elől is biztonságban vannak.

Kiszolgálók

tcp-wrapper. Ezzel a wrapper-rel szűrhetők és monitorozhatók az inetd-ből indított (pop2, pop3, imap, r*, talk, telnet, ftp, tfp, finger, stb.) és a hozzá fordított szerverprogramok által nyújtott hálózati szolgáltatások. A legtöbb Linux disztribúcióban megtalálható.

Használata egyszerű: a */etc/hosts.allow* és a */etc/hosts.deny* file-okban beállítható, hogy egy adott szolgáltatást honnan vagy honnan ne lehessen igénybe venni.

Melegen ajánlott először tiltani (azaz „ALL: ALL” a */etc/hosts.deny*-be) aztán engedélyezni azt, amit és akinek szükséges (a */etc/hosts.allow*-ban):

```
ftp: .engedelyezett_domain.hu  
pop-3: levelező.engedelyezett_domain.org
```

Ha tiltásnál nem az „ALL: ALL” opciót használjuk, érdemes használni az „ALL: UNKNOWN PARANOID” beállítást.

A PARANOID beállításakor, ha a host neve nem egyezik a címével, a kapcsolatot eldobja. Ennek segítségével csökkenthető a DNS spoofing veszélye.

Az UNKNOWN használatakor a tcpd ellenőrzi, hogy egy ident kérdésre mi volt a kliens válasza. Ha UNKNOWN@akarmi, akkor megtagadja a szolgáltatás használatát. Ha a kliens nem használ *ident*-et, akkor az opció figyelmen kívül marad.

SSH (Secure Shell). Jelenleg a legfrissebb verzió az 1.2.26 vagy a 2.0.12 (ssh2 protokoll). Letölthető szinte minden Linux-os ftp szerverről, illetve a készítő, az SSH Communications honlapjáról.

Érdemes használni már csak a szolgáltatásai miatt is. Képes kapcsolatokat forwardolni, így a távoli felhasználók több szolgáltatás használata esetén sem kommunikálnak kódolatlanul. Például:

```
ssh -a -f -L port2:tavoli.gep:port1 tavoli.gep -l user perl -e 'sleep'
```

A távoli gépen X programokat indítva a *DISPLAY* változó értékét megváltoztatja a saját gépünk X konzoljának címére és automatikusan forwardolja a kapcsolatot.

Az *ssh* helyettesítheti az *rsh/rlogin* parancsokat. Fileátvitelre *ftp* vagy *rcp* helyett használhatjuk az *ssh scp* parancsát is:

```
scp forráskönyvtár/file célhost:célkönyvtár/
```

vagy több file és könyvtárak másolása esetén:

```
scp -r forráskönyvtár/* célhost:célkönyvtár/
```

Kiválthatjuk vele az *rexec*-et is a következő módon:

```
ssh hostnév parancs
```

Lehetőség van a kulcsok központi elosztására, amely egy nagyobb cég hierarchikus szabályainak betartását is lehetővé teszi.

Minden kapcsolat kezdetekor azonosítja a másik oldalt, így csökken a DNS vagy az address spoofing veszélye. Korlátozott védelmet nyújt a man-in-the-middle jellegű támadások ellen, mivel a kapcsolat kezdetétől kódoltan folyik a kommunikáció. A kapcsolat elrablása ellen ugyan nem véd, de ha az elkövető nem a kapcsolat kezdetekor lép be a vonalba, akkor érdemi információ birtokába nem juthat, hacsak nem ismeri mindkét fél kulcsait, amelyek a kódolás feloldásához szükségesek. Ezek azonban óránként újragenerálódnak és az *ssh* nem tárolja azokat file-okban.

A kapcsolat kezdetekor jelez, ha az adott címmel még nem folytattunk kommunikációt, illetve szól, ha valami nincs rendben vele (pl. a host nevéhez tartozó IP cím nem egyezik a tárolttal). Ha szeretnénk biztosak lenni, hogy illetéktelenek nem használják a szolgáltatást, a */etc/ssh/sshd_config*-ban (vagy a */etc/hosts.allow* / *hosts.deny*-ben, ha az *sshd tcp-wrapper*-hez lett fordítva) azt is beállíthatjuk, hogy honnan vagy honnan ne lehessen használni.

X biztonság. Egy X-et futtató gépre X alól bejelentkezve lehetőségünk van ott programokat indítani a *-display gépünk_címe:0* paraméter segítségével, így az alapértelmezett display-t (ami a 0:0, azaz a localhost-on az a terminál, ahol elsőnek indítottunk X-et) átírjuk a saját gépünk X konzoljára. Előtte azonban ki kell adnunk gépünkön egy *xhost + szervernév* parancsot, hogy gépünk a szerver által kezdeményezett kapcsolatot elfogadja. Régebbi – és épp ezért nem biztonságos rendszereken – az *xhost +* volt az alapértelmezés, azaz bárki kezdeményezhetett kapcsolatot gépünkre az engedélyünk nélkül, és így bármit írhatott/olvashatott a terminálunkról. Épp ezért ne használjuk így ezt a parancsot! Csakis az először említett verzió a biztonságos.

Azonban még így is kódolatlanul megy át az információ a hálózaton, ezért célszerű a kapcsolatot SSH-val forwardoltatni.

Inetd. Csak a szükséges szolgáltatásokat engedélyezzük. A */etc/inetd.conf* átnézése melegen ajánlott, itt mindent kommentezzünk ki, ami nem kell. Amit mindenképp ajánlott letiltani: *finger*, *netstat*, *systat*. Ezek a cracker-eknek fontos információkkal szolgálhatnak gépünkről. Ha nem használunk *inetd*-ből futó szolgáltatást, ne indítsunk *inetd*-t. Ha mégis szükséges, használjunk inkább *xinetd*-t helyette, amely kisebb, biztonságosabb, és kínál 1-2 hasznos plusz szolgáltatást is.

WWW kiszolgáló. Eléggé elterjedtek Roxen Challenger vagy az Apache alapú szerverek. Mindkettőt megfelelően sokan használják ahhoz, hogy a felmerülő hibák gyorsan kijavításra kerüljenek, a hibák leírását és a patcheket a biztonsággal foglalkozó listákon

és fórumokon közölni szokták. A Roxen alapértelmezésben, az Apache a `mod_ssl` kiegészítéssel képes SSL3 használatára. Ezt csak akkor tudjuk kihasználni, ha SSL3 képes böngészőnk van, ezért Netscape-hez célszerű a Fortify használata. Egy jó tanács: soha ne futtassuk root jogokkal a webszervert! Egy rosszul beállított rendszeren vagy hibás CGI scriptek használata esetén ez komoly gondokat okozhat. Az Apache alapértelmezésben `nobody` vagy `wwwuser` néven fut, Roxen használatakor a 'Global Variables' menüpont alatt beállítható.

CGI scriptek. Gondoljuk meg, mit és hogyan használunk. Egy rosszul megírt script komoly veszélyeket hordoz rendszerünkre nézve.

Ha mások által írt scripteket használunk, mindig ellenőrizzük, mit is csinál az adott script valójában. Ha ez nem lehetséges, nézzünk át néhány biztonsággal foglalkozó site-ot (lásd fent), nincs-e az adott scripttel kapcsolatban valami probléma.

Ha saját magunk írunk scripteket, érdemes elolvasni előtte egy-két kapcsolódó dokumentumot (lásd alább). Pár dolog, amire érdemes figyelni:

- ha Perl-ben írunk scripteket, használjuk a `-T` opciót (`taint`), ez figyelmeztet a biztonsági hiányosságok egy részére;
- form outputja ne legyen file vagy könyvtár név;
- form-ból ne vegyünk át olyan adatokat, amelyek később utasításként használhatók;
- form adatait ellenőrzés nélkül ne adjuk át paraméterként (pl. mail-nek);
- a `system()` és az `exec()` és `open()` hívásokkal körültekintően járjunk el;
- file jogosultság beállításokkal vigyázzunk.

DNS (Domain Name Service). Feladata a kiosztott hálózati (IP) címekhez rendelt nevek (reverse name server funkciók), és az IP címekhez tartozó nevek (name server funkciók) leképezése. Használjuk a `bind` 8-as változatából a legfrissebbet (ez jelenleg a 8.2). Ez a hagyományos (4.* verziójú) `bind`-hez képest nyújt jó néhány hasznos funkciót.

Ha valaki paranoid, használhat `chroot`-olt `bind`-et is. Ezzel megelőzhető, hogy ártó szándékú egyének egy frissen felfedezett hibát kihasználva adatokat szerezzenek meg, vagy kárt okozzanak szerverünkön.

Levélkezelő (MTA, Mail Transport Agent). Levélkezelőnek használjuk a Wietse Wenema által írt *postfix*-et vagy a *qmail* legújabb verzióját. Az előbbi két MTA megfelelően biztonságos és jól konfigurálható. Az utóbbi mellett szól, hogy elég sokan használják, így a hibákra előbb fény derül. A *postfix* még új, de felépítéséből eredően a biztonsági hibák veszélyét komolyan csökkenti.

POP3 szerver. Lehetőleg olyan legyen, amelyik tudja az *APOP*, *SSL* vagy más kódolt átvitelt (pl. *cucipop*), mert enélkül a kommunikáció titkosítatlanul folyik. Persze ehhez megfelelő kliens használata szükséges (pl. *fetchmail*).

lpd (Line Printer Daemon). Csak akkor fusson, ha szükség van rá, de ebben az esetben korlátozzuk a használatra jogosult gépeket a */etc/hosts.lpd* vagy a */etc/hosts.equiv* beállításával. A legújabbat tegyük fel, régebbi verziókban buffer overflow-t találtak. Helyettesíthetjük *LPRng*-vel, ami az lp csomag egy újraírt, biztonságosabb változata.

FTP szerver. Csak indokolt esetben használjuk (az nem igazán indok, hogy a tisztelt felhasználó csak azt ismeri).

Egész jó a *pro-ftp*d legújabb verziója (a régebbiekben buffer overflow volt. Értelmes módon be lehet állítani, ki mihez férjen hozzá, könnyű *chroot*-olni és egyebek.

Firewall szoftverek. Segítségükkel lehetőségünk van a hálózati forgalmat szabályozni. Korlátozhatjuk szerverünk egyes portjainak vagy éppen a teljes szerver elérését, lehetőségünk van kitiltani egyes protokollokat (pl. ICMP) is, de lehetséges a csomagok tartalma vagy a használt alkalmazási rétegbeli protokoll alapján való szűrés is.

Packet filter firewall A Linux-os firewall szoftverek közül eléggé elterjedt az *ipfwadm*, bár ez a 2.2.* kernellel már nem működik, helyette az *ipchains* használatos. Az utóbbi sokkal több lehetőséget biztosít a forgalom szűréséhez, de mindkettő a Linux kernel packet filter (csomagszűrő) funkcióit használja ki, azaz protokollok, IP címek, illetve portok alapján filterezhetjük a forgalmat, illetve forwardolhatunk hálózati interface-ek között.

Application level (proxy) firewall Az ilyen funkciókat megvalósító programokkal (pl. *tis*) lehetséges az alkalmazási rétegben szűrni a csomagokat, azaz megoldható az adott alkalmazások által használt protokollok sajátosságain alapuló szűrés. A proxy firewall nem forwardol az interface-ek között, a megfelelő portokon az adott alkalmazási protokoll kezelésére írt speciális programok fogadják a kéréseket. Minden forgalom keresztülmegy rajtuk, így lehetőségük van a csomag tartalmának megváltoztatására is.

Rendszerellenőrző programok (security scanner-ek). Használatukkal az ismert hibák azonosíthatók a rendszerben. Figyelmeztet a hiányosságokra, hibás programokra. Részletesebb információt az adott program dokumentációjában találhatunk. Hogy csak néhányat említsek: *NESSUS*, *ADMhack*, *mscan*, *SATAN*, *COPS*. Megtalálhatók a COAST archívumban.

Használatukkor vigyázzunk, legtöbbjük portscan-t is csinál.

Egyéb fontos dolgok

A legjobb védekezési technikák sem érnek sokat, ha nem vagyunk elég körültekintőek napi munkánk során. Hogy csak néhány rossz példát említsek: root-ként vagy több jogosultsággal rendelkező felhasználó nevében futtatott *X*, *netscape*, *lynx*, *irc*, stb. . . Érdemes megfogadni a alábbi tanácsokat:

- csakis a legszükségesebb munkákat végezzük root-ként, a napi munkára hozzunk létre egy privilégiumokkal nem rendelkező accountot;
- ismeretlen programokat soha ne installáljunk vagy futtassunk root-ként, amíg nem vagyunk biztosak, hogy nem tartalmaz kártékony részeket;
- a root jelszót soha ne adjuk meg senkinek, ne írjuk fel;
- jelszavakat és egyéb fontos információt soha ne küldjünk kódolatlan levélben, használjunk *PGP*-t (Pretty Good Privacy) vagy *GPG*-t (GNU Privacy Guard) a titkosításhoz;
- root konzolt ne hagyjunk ott, jelentkezzünk ki vagy lock-oljuk (pl. *vlock -a*);
- több gépen ne használjuk ugyanazokat a jelszavakat, de legfőképp a root password ne legyen egyforma;
- mindig olvassuk a logokat! Ha valaki próbálkozik, annak általában nyoma marad. Persze ezeket fel is kell ismerni. Nyom lehet pl. a többszöri belépési kísérlet adott hostról vagy accounton, daemonok érdekes hibaüzenetei, kapcsolódási kísérletek gyanús címekről, stb. Ha ilyet észlelünk, a próbálkozó hostot azonnal tiltsuk ki minden szolgáltatásból! A legjobb megoldás, ha a firewall-ban tiltjuk ki, ha ez nem lehetséges, akkor tegyük az */etc/hosts.deny*-be a címet. Ha ez megvan, érdemes tájékoztatni a kitiltott gép adminisztrátorát.

Ha már megtörtént a baj

A betörés észlelése után azonnal húzzuk le a hálózatról a gépet! Aztán jöhet a rendszerfile-ok átnézése a *tripwire* log alapján. Minden megváltozott binárist le kell cserélni, config file-okat, firewall rule-okat felülvizsgálni.

Gyakran megváltoztatott file-ok: binárisok: *login, su, netstat, ps, who*, stb. config file-ok: */etc/passwd, /etc/group, /etc/shadow, /etc/securetty, /etc/ssh/sshd_config, /etc/hosts**, name server config file-ok, stb.

Töröljünk minden gyanús, nem a rendszerhez tartozó file-t. Ajánlott az összes olyan könyvtár átnézése, ahonnan program futtatható. (A */dev*-et se hagyjuk ki!) Ellenőrizzük az összes *setuid/setgid* jogosultságot használó programot.

Minden log-ot mentsünk. Később ezekből talán visszanyerhető némi információ a betörő kilétéről vagy a támadó gép címéről, esetleg a feltört szolgáltatásról vagy a törés módjáról. Ezeket az információkat célszerű a security-l listán közölni, hogy másoknál ne tudjon próbálkozni az illető. A lista zárt, feliratkozni a *majordomo@sunserv.kfki.hu* címen lehet 'subscribe security-l sajátnev' szöveggel a levél törzsében. Ezután jöhet az összes jelszó megváltoztatása, az userek értesítése.

A log-ok elemzése során a gyanús címeket azonnal tiltsuk ki (*/etc/hosts.deny* vagy *firewall*), majd tájékoztassuk adminisztrátoraikat, mivel lehetséges, hogy az ő gépüket is ugródeszkaként használta valaki.

Ha már biztosak vagyunk abban, hogy mindent átnéztünk, nézzük át még egyszer a rendszert. Ha még mindig rendben van, visszakapcsolhatjuk a hálózatra.

Amit érdemes elolvasni

- Üzembiztonság – <http://ludens.elte.hu/szanyi/uzem.html>
- Linux Security-Audit FAQ – <http://www-jcr.lmh.ox.ac.uk/security/index.txt>
- WWW Security FAQ – <http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html>
- Security Code Review Guidelines – <http://www.homeport.org/adam/review.html>
- FAQ: Network Intrusion Detection Systems. – <http://www.shake.net/misc/network-intrusion-detection.htm>
- Linkek néhány érdekesebb oldalra – <http://www.choraii.com/linkek.html>

Használt fogalmak definíciója

ARP Address Resolution Protocol. Feladata az adott címhez tartozó hardver cím lekérdezése a másik géptől a következő módon (kicsit egyszerűsítve): küld egy broadcast üzenetet, amelyben elküldi azt a címet, amelyhez tartozó hardver címet tudni akarja. A broadcast üzenet lényege, hogy minden interface elfogadja a helyi hálózaton, de csak az fog válaszolni, akinek a címét az üzenet tartalmazza.

ARP cache az ARP által lekérdezett hardver címek egy előre definiált ideig itt tárolódnak, ezért nem kell minden alkalommal újra lekérdezni őket.

certificate a hálózaton erre felhatalmazott szerverek által kiadott tanúsítvány. Ha egy kliens kapcsolatot kezdeményez, a szerver elküldi a certificate-jét, amit a kliens ellenőriz. Ha rendben találja, megkezdődhet az információ átvitele.

chroot használatával egy program bezárható egy adott könyvtárba. Ha jól állítjuk be, nincs lehetősége, hogy a megadott könyvtáron kívül mást is lásson.

exploit biztonsági hiba kihasználása, általában a kihasználására írt programot illetik e névvel.

hardver cím a hálózati interface fizikai címe. Ethernet hálózatok esetén egy 32 bites hexa szám, amely alapesetben minden ethernet interface-nél különböző.

ICMP Internet Control Message Protocol. A hálózatba kapcsolt intelligens eszközök – a hálózattal kapcsolatos – üzenetcserejére találták ki.

ident a kapcsolatot kezdeményező felhasználó nevének azonosítására szolgál.

noexec a mount által használt opció. Az így felmountolt filerendszeren lévő programok nem futtathatók.

nosuid a mount által használt opció. Az így felmountolt filerendszerben nincs hatása a setuid/setgid bitnek.

promiscuous mód az ethernet interface olyan módja, amelyben nem csak a neki címzett csomagokat fogadja el, hanem mindent.

routing tábla megadja egy adott célcím esetén az eléréséhez szükséges útvonalat.

rsh, rcp, rexec, rlogin a kernel RPC (Remote Procedure Call) szolgáltatását használják. Az első egy shell-t hív a távoli gépen, az második file-okat másol a két gép között, a harmadik távoli programfuttatást tesz lehetővé, az utolsó beléptet a távoli gépre. A hálózaton kódolatlanul kommunikálnak, így az átvitt információ lehallgatható.

setuid/setgid a futtatható binárisokon lévő „s” bit. Ha a file tulajdonosának (setuid), illetve group-jának (setgid) jogai között van, akkor az adott program futtatója a file userének/groupjának jogaival rendelkezik a program futásának ideje alatt. A programok betöltése előtt a linker kitisztítja az environment veszélyes részeit, lefutásuk után pedig az általuk használt memóriaterületeket.

SNMP Simple Network Management Protocol. Az intelligens hálózati eszközök távoli managementjének megkönnyítésére készült protokoll.

spoof(ing) általában a hálózati kommunikáció valamely részébe harmadik fél részéről történő beavatkozás.

syslogd a rendszer és az egyes programok üzeneteinek kezelésére írt daemon.

TCP Transmission Control Protocol. A TCP réteg felelős a csomagok nagy részének veszteségmentes átviteléért. Kapcsolat-orientált protokoll, azaz várja az elküldött csomag nyugtázását. Ha ezt nem kapja meg egy meghatározott időintervallumon belül, újraadja a csomagot.

UDP User Datagram Protocol. A feladata hasonló, mint a TCP-nek, de nem kapcsolat-orientált.

visszatérési cím (return address) a következő végrehajtandó utasítás címe.

world-writable a „t” (sticky) bit be van állítva a könyvtáron vagy bárki számára írható/olvasható. A „t” bit jelentése: mindenki számára írható/olvasható, de mindenkinek csak a saját tulajdonú file-okhoz vannak jogai.

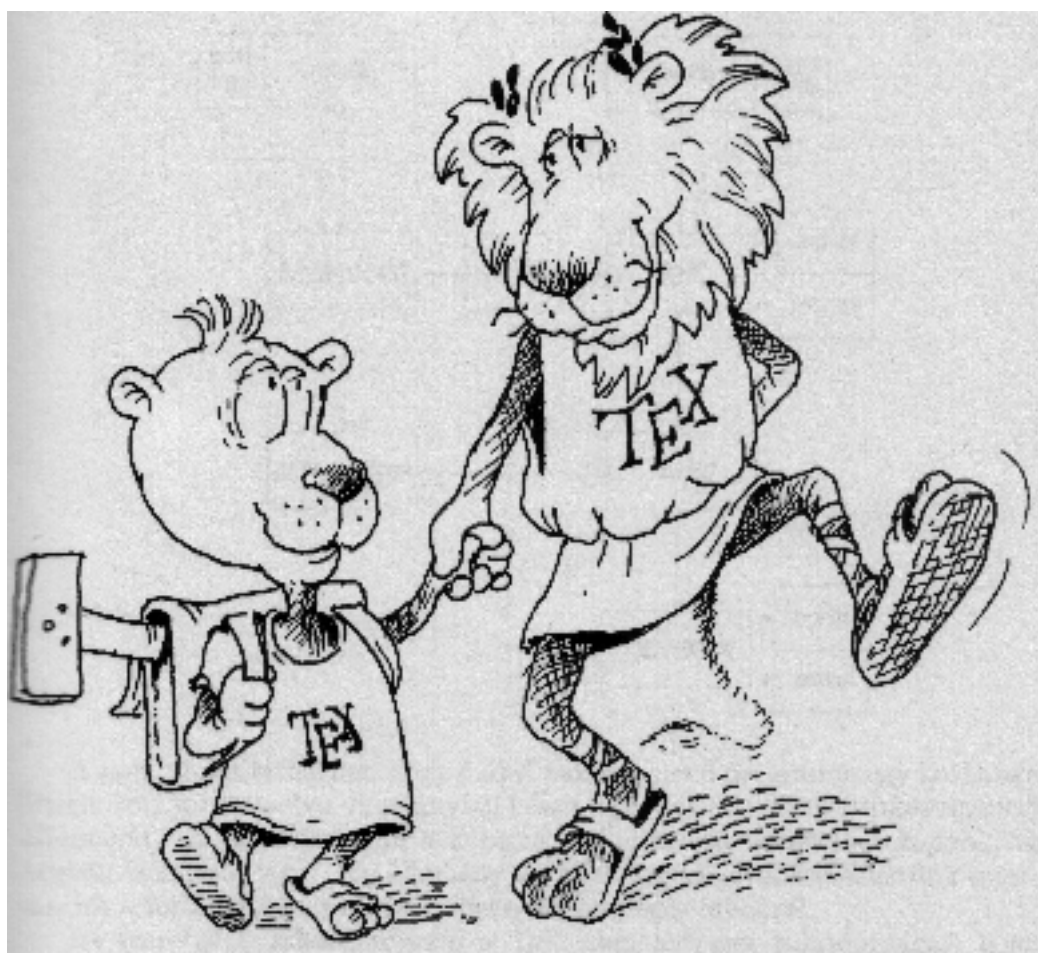
Copyright Srágli Attila (sragli@choraii.com)

Köszönet Bedő Sándornak (bsanyi@valerie.inf.elte.hu) építő jellegű hozzászólásaiért.

A dokumentum eredeti formájában szabadon terjeszthető a copyright megjelölésével.

13. fejezet

Segítségkérés, dokumentációk



13.1. Segítség, dokumentációk

A Linux fejlődésének kezdeti szakaszában a programozók meglehetősen csekély gondot fordítottak munkájuk dokumentálására. Szerencsére hamar felfedezték, hogy ez az igazán egyenes út a káosz felé. Mára talán az egyik legjobban dokumentált operációs rendszer lett a Linux.

Hazánkban a felhasználók nagyobbik része idegenkedik a dokumentációk olvasgatásától. Ennek csupán az az oka, hogy az írásos anyagok ritkán találhatók meg magyar nyelven. A Linuxban ezen a téren is komoly előremozdulás tapasztalható, köszönhető a sok lelkes felhasználónak, akik szabad idelyüket fordításra szánják, illetve a különböző fordító projekteknek. Természetesen ez nem zárja ki az angol nyelv ismeretének szükségességét, mivel a dokumentáció nagy része csak angol nyelven érhető el, valamint a legfrissebb információk is csak ezen a nyelven találhatók meg.

13.2. Segítség helyben

Már a meglévő rendszerünkön is komoly segítségeket kaphatunk. Komplettdokumentumok a `/usr/share/doc/` alatt találhatók.

A man

Fontos segítséget kaphatunk a kézikönyv lapokon keresztül. Ezek az úgynevezett man oldalak. Elérésükhöz használjuk a következő parancsokat:

```
man man           (közvetlenül a man-ról kapunk információt)
man parancsnév    (parancsokról kapunk információt)
```

Minden kézikönyv elején, NAME szó alatt, található az adott parancs egy soros leírása - az úgynevezett permutált index. Ez akkor tehet jó szolgálatot, ha konkrét feladat megoldásához keresünk egy programot. Az index minden parancsismertető sor minden szava szerint rendezett, így pillanatok alatt választ kaphatunk kérdésünkre. A megfelelő parancs `man -k keresett_szó`, ahol a `keresett_szó` helyére írhatjuk be az általunk gondolt kifejezést. Esetleg érdemes lehet valamilyen oldalanként listázó programot is használni (`more`, `less`), hogy ha túl sok találatot kapunk, akkor is végig tudjuk őket böngészni. Bővebb információ: `man man`

whereis

E parancs segítségével azt tudhatjuk meg, hogy egy program megtalálható-e rendszerünkön, ha igen, akkor hol van a futtatható változata, és a hozzá tartozó kézikönyv. Használata a következő:

```
whereis parancsneve
```

Bővebb információ: *man whereis*

whatis

Egy parancsról kérhetjük le a parancs egy soros leírását. Csak a parancs nevével alkalmazható, kulcsszavakat nem lehet használni.

Bővebb információ: *man whatis*

apropos

Ez a parancs a kézikönyvlapok NÉV és LEÍRÁS mezőiben keres az általunk megadott kulcsszó alapján. Megadhatunk több szót is, de nem azokat listázza ki, amelyekben mindegyik benne van. Alkalmazhatunk helyettesítő karaktereket is. Bővebb leírás található róla: *man apropos*

13.3. Segítség az interneten

Sok parancs, ha rossz paraméterezéssel, paraméterek nélkül vagy a (-)-help opcióval hívjuk meg, akkor kiír a képernyőre egy rövid útmutatást arról, hogy hogyan kell használnunk az adott parancsot, illetve hogyan kérhetünk további segítséget a parancstól.

Segítséget találhatunk az interneten is. Információ beszerzésre használhatóak a különböző Linuxos internetes portálok, ahol rendszerint fórum is üzemel. A teljesség igénye nélkül álljon itt néhány ismertebb:

13.3.1. Magyar nyelven

```
http://www.linux.hu/  
http://www.linuxportal.hu/  
http://linux.index.hu/  
http://www.uhulinux.hu/
```

```
http://www.szabilinux.hu/  
http://mylinux.hu/  
http://portal.fsn.hu/  
http://www.prog.hu/  
http://www.debian.hu/  
http://sulix.homelinux.net/  
stb.
```

13.3.2. Egyéb nyelveken

```
http://www.linuxportal.com/}  
http://www.debian.org/  
http://www.redhat.com/  
http://www.suselinux.de/  
stb.
```

Hathatós segítséget kaphatunk levelező listákon keresztül:

```
http://www.lme.hu/levlista.html  
http://mlf.linux.rulez.org/mailman/listinfo/  
https://lists.uhulinux.hu/
```

13.4. Mit illik és mit nem a linux és más levelező listákon?

Az alábbi kis illemtan a Linux listákon való könnyebb és hatékonyabb eligazodás céljából született. Az illemtan megsértéséért eltiltás járhat.

1. Általában a linux listákon nem illik:

- csúnya szavakat használni
- mások nem szakmai véleményét/tulajdonságait kritizálni
- egy bizonyos gyártótól különböző gyártók eszközeit kritizálni
- flamelni (ez alól a linux flame kivétel, ott erősebb szavak is elfogadhatóak...)

2. A linux listákon folyó támogató/support tevékenység ingyenes. Tehát ha kérdezünk, valaki jó szándékú megpróbál segíteni, ingyen, bérmentve, szabadideje függvényében. Előfordulhat az is, hogy a kérdés olyan területre vonatkozik, amit

mások még nem próbáltak. E két indok miatt nagyon helytelen, a lista szemére vetni, hogy miért nem foglalkoztak a probléma megoldásával. Ugyanezen okok miatt a kérdést megismételni sem helyes. Nem illik firtatni, hogy ki milyen célból teszi fel a kérdést, neki milyen formában hasznos a válasz.

3. Sajnos nem rendelkezik mindenki megfelelő sávszélességgel. Ezért helytelen:

- 4 sornál hosszabb aláírásfájlt használni
- forrásprogramokat, bináris programokat beküldeni (tegyük ki webre, ftp-re, vagy ajánljuk fel, hogy szívesen elküldjük annak, akit érdekel), kivéve – az l-code-l és a shell-l profilja miatt erre a két listára lehet rövid programot küldeni, de inkább a *web/ftp* javasolt
- kettőnél több csoportba postázni ugyanazt a kérdést
- feleslegesen teljes leveleket idézni a válaszban
- kellő indok nélkül *PGP*-t használni
- magánlevélben csak magánügyben illik választ kérni (pl. szavazás, tesztek, CD írás, bögre, stb), ekkor magánlevélben is válaszoljunk, s az illető majd beküldi vagy elérhetővé teszi az összesítőt
- közérdekű kérdésre nem illik magánlevélben választ kérni/küldeni
- offtopic levélre nem illik válaszolni a listán. Ha ilyet látunk, legegyszerűbb figyelmen kívül hagyni, vagy értesíteni a lista adminisztrátorát

4. Hazánkban a listák hivatalos nyelve a magyar, emellett elfogadható, ha más listákról angol nyelvű hírt másolunk be. A hivatalos magyar nyelv azt is jelenti, hogy a listára írhatunk:

- angol abc betűivel, ékezet nélkül
- szabványos ISO-8859-1 kódolással
- szabványos ISO-8859-2 kódolással

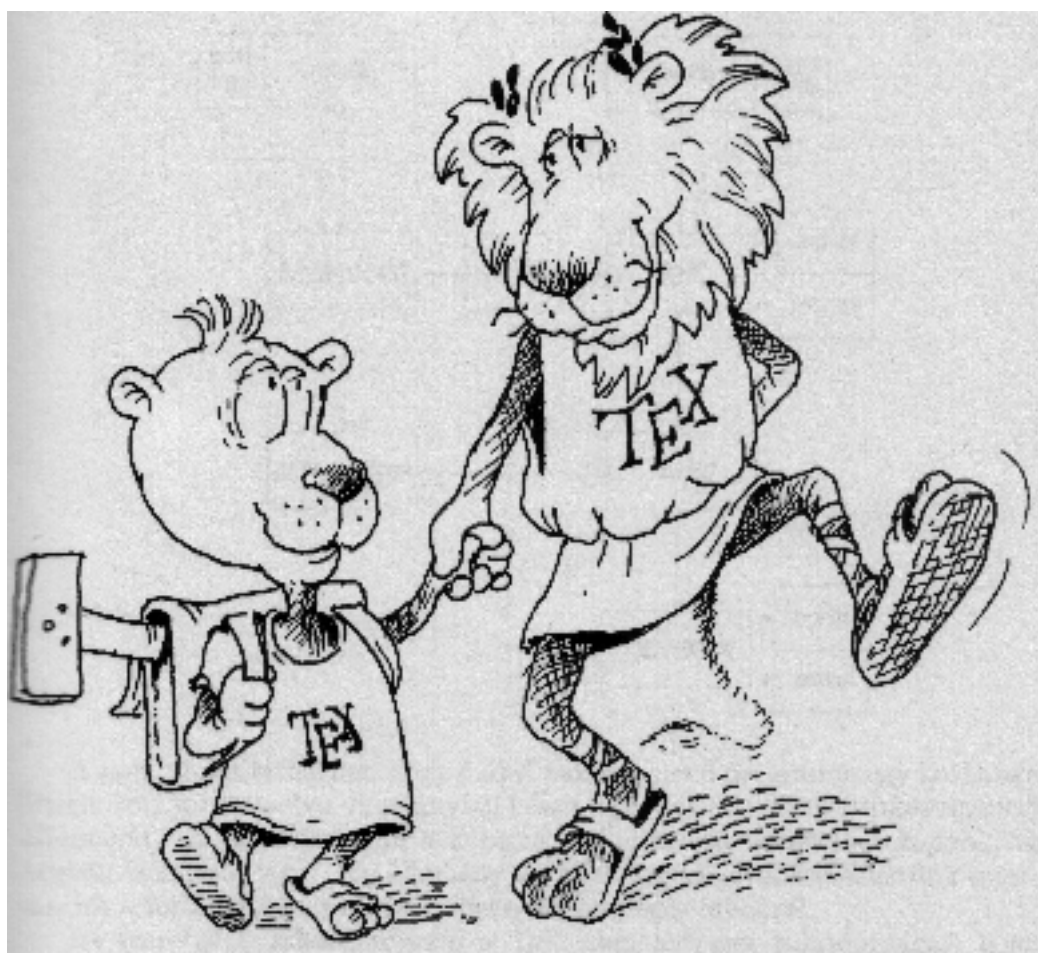
Mivel a magyar ékezetek az *ISO-8859-2*-es szabványos kódtáblában vannak benne, ezért elsősorban ez a támogatott. A listákon hosszú vita és program javítgatás után alakult ki az a helyzet, hogy tudjunk terjeszteni szabványos karakterkódú, szabványosan csomagolt leveleket. Ettől eltérő kódolású/csomagolású leveleket ne küldjünk a listára (pl. HIX féle 123 kódolás, repülő ékezetek, egyéb kacifánt).

5. A levél formátuma nem szabadon választható, hanem minden esetben plain ASCII vagy 8 bites ASCII, egyszerű szövegszerkesztővel (pl. unix more parancs, vagy vi/joe) elolvasható állomány. A sorok hossza optimális esetben 65, 70 körüli hosszra elfogadjuk, de a 75-nél hosszabb sorokat inkább tördeljük.
6. A levelek tartalmáról:
 - a subject a tartalomról szóljon
 - amikor a téma kezd elkanyarodni a subjecttól, akkor változtassuk meg azt
 - a levél is a tartalomról szóljon, vagyis legyen benne minden fontos részlet, amely szükséges a válaszhoz, pl.:
 - milyen disztribúció
 - milyen kernel
 - libc, ld.so verzió
 - hivatalos vagy nem hivatalos patcheket használunk-e
 - ha sejteni lehet, hogy hardver közeli a kérdés, akkor milyen processzor, memória, merevlemez, vezérlőkártya van a gépben
 - milyen hálózat/modem
 - milyen verziójú programmal van probléma
 - rövid, lényegre szorítókozó log file kivonat
 - a program vagy a kernel által küldött pontos hibaüzenet
 - megfelelő kivonat a program futásának nyomkövetéséből
 - ne lepődjünk meg, ha nem a kezdő listára küldött triviális kérdésre egy rövid, olvasd el ezt-azt stílusú választ kapunk (pl. RTFM akármi)
 - mielőtt kérdezzünk, nézzünk bele az archívumokba, esetleg más már találkozott a mi problémánkkal, így a válasz is megtalálható. Ne feledjük, a listák forgalma igen nagy, ne foglaljuk a vonalat és mások szabadidejét gyakran feltett kérdésekkel. Ne lepődjünk meg, ha valaki ingerülten válaszol ilyen kérdésre.
 - számítsunk rá, hogy ha kifogásolható stílusú volt a levelünk és nem a flame listára küldtük, akkor nyilvánosan vagy magánlevélben figyelmeztetnek
 - minden listának van tulajdonosa, aki egyfajta felügyelő szerepet is betölt (amellett, hogy a saját eszközeiből áldoz a lista számára). A tulajdonos szava parancs, amit kér, azt kötelesek vagyunk megtartani. Amikor például a tulajdonos úgy ítéli meg, hogy egy vita nem felel meg a lista követelményeinek, akkor megkéri a feleket, hogy fejezzék be, amit a felek örömmel teljesítenek.

- a szövegben igyekezzünk helyesen, megfelelő szóhasználattal fogalmazni. Bátran használhatunk angol szakkifejezéseket, de jó, ha helyesen írjuk. Ez nem csak azért fontos, mert illik helyesen fogalmazni, hanem azért is nagyon fontos, mert a kereső szerverek szavak alapján keresnek, tehát ha nem helyesen írunk valamit (vagy nem az általánosan elfogadott rövidítést/becenevet/argót használjuk), akkor a szerverek nem fogják megtalálni a levelet.
7. A listán folyó támogató tevékenység publikus. Ezért nem helyes a lista tárgyához tartozó kérdésben magán emailben kérni a választ, mivel előfordulhat, hogy más is éppen kíváncsi rá.
 8. A listán folyó minden tevékenység publikus, a világ számos szerverén archiválják a hozzászólásokat. A levelek alapján az egész lista megítélése változhat. Ezért igyekezzünk megfelelő kifejezéseket használni. A téma legyen elfogadható. Különös tekintettel a szerzői jogokra, Lake Success-i megállapodásra, VÁM és ÁFA törvényre, törvénytelen tartalmú cikket ne küldj a listákra.
 9. A listák non-profit jellegűek, ezért gazdasági haszonszerzés céljából tilos igénybe venni őket. Ezt szigorúan kezeljük, figyelmeztetés nélküli három hónapos eltiltás jár a meg nem tartásért. Ezzel párhuzamosan linux-business lista nyílt ezen kérdések tárgyalására, ahol viszont nem szerepelnek szakmai kérdések és flame.
 10. Fontos, hogy létező, jól működő email címről iratkozzunk fel. Amennyiben postafiókunk kapacitása szűkös, gondoskodjunk leveleink rendszeres letöltéséről/letörléséről. Szabadságotól esetén, inkább iratkozzunk le a listákról. Ezek azért fontosak, mert ha megtelik a postafiókunk vagy a gép nem tudja fogadni a leveleket, akkor egyrészt a lista szerveren foglalja a helyet, lassítja a lista működését, másrészt a hibaüzeneteket az egész lista megkaphatja, de az admin biztosan.
 11. Alapfokú angol nyelvtudás sokat segíthet.
 12. A listák forgalmának csökkentése érdekében a következők ajánlottak:
 - először olvassuk el a dokumentációkat
 - olvassuk el még egyszer
 - olvassuk el a *FAQ*-t (Frequently Asked Questions = Gyakran Feltett kérdések)
 - keressünk rá a témára a lista archívumokban
 - keressünk rá a témára a weben
 - ha még mindig nincs meg a keresett információ, akkor kérdezzünk a listán

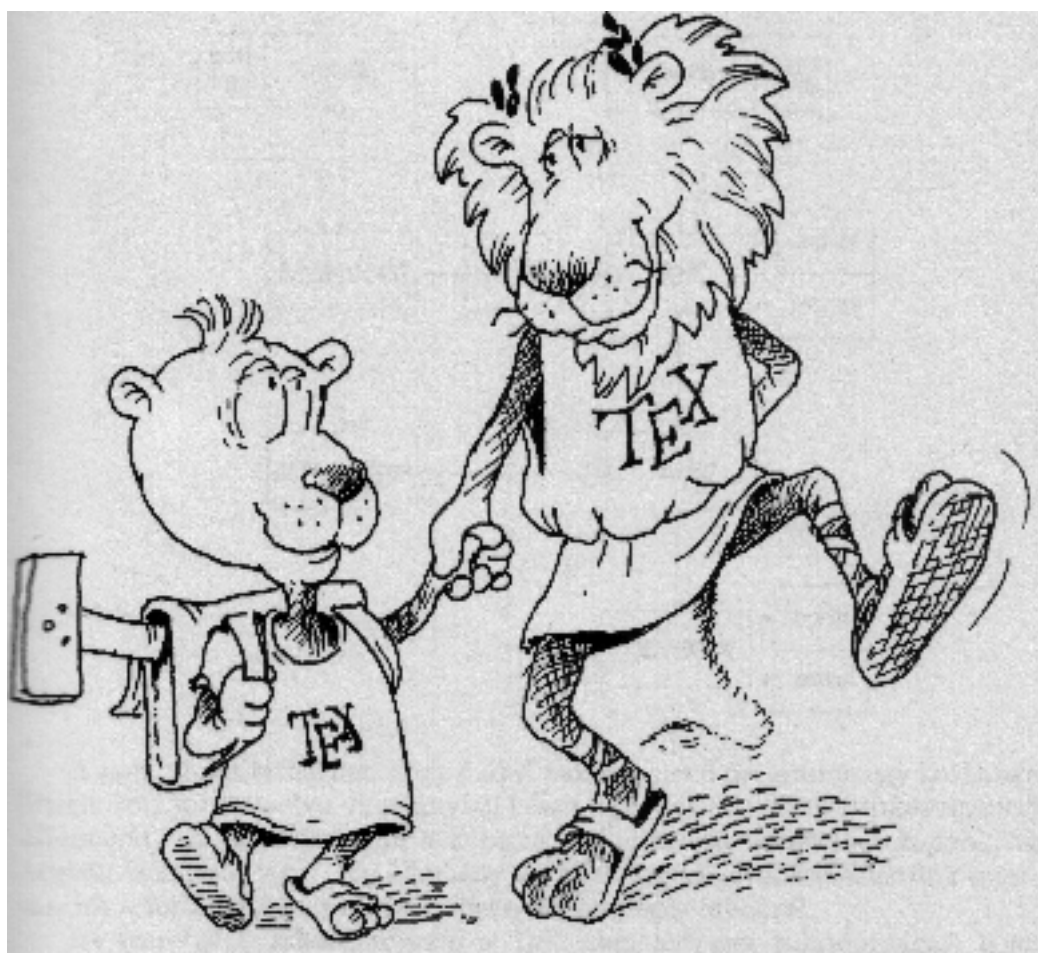
14. fejezet

Szövegfeldolgozás



15. fejezet

Szerver programok UHU-Linux alatt



15.1. Az APACHE 2 és PHP4 beállítása UHU-Linux alatt

15.1.1. Bevezetés

Úgy gondoljuk, hogy a címben szereplő 2 dolgot nem kell különösebben bemutatnunk, hiszen aki az utóbbi időben használta az Internetet, az nagy valószínűséggel találkozott már e két fogalommal:

- az Apache a világ vezető webservere alkalmazása, nyílt forrású, csakúgy, mint a Linux
- a PHP (PHP: Hypertext Preprocessor) pedig egy HTML-be ágyazható szerveroldali programnyelv (de nem csak HTML-be ágyazva használható, hiszen van már parancssori, ill. GTK-t támogató változata is). Szintén nyílt forráskódú

A telepítés során egy konzolra lesz csupán szükségünk. Mindenképpen rendelkezünk kell rendszergazdai jogosultságokkal! A parancsok előtt kétféle jel állhat:

\$ a parancsot "mezei" felhasználóként is kiadhatjuk

a parancsot csak rendszergazdai jogosultságokkal adhatjuk ki (root)

Mivel mindkét program forrásból kerül lefordításra, ezért fontos, hogy a rendszeren megtalálhatóak legyenek a fejlesztői csomagok (telepítésnél a "Fejlesztői programok" összetevő).

15.1.2. Beszerzés

Mindkét programcsomagot legcélszerűbb a honlapjukról letölteni:

#Gibzo: (A krix féle UHU csomagok nem játszanak? Apache mintha a 2. CD-n is lenne, a PHP modul szintén létezik már UHU csomagban. Beleírtam? Látom a 9. fejezetben már szerepel, akkor legyen itt 1 utalás rá.)

- Apache <http://httpd.apache.org/>
- PHP <http://www.php.net/> (magyar mirror <http://hu.php.net/>)

Az Apache oldalán keressük meg a 2.0-s verzió legfrissebb forráskódját. A fájlnev így néz ki: *httpd-2.0.44.tar.gz* (kb. 5,3 MB)

A PHP honlapján a download linkre kattintva az oldal tetején találhatjuk a forráskódot tartalmazó fájlt: *php-4.3.1.tar.gz* (kb. 4,5 MB)

15.1.3. Előkészítés

Ebben a fázisban csak annyi a feladatunk, hogy kicsomagoljuk a letöltött fájlokat (Feltételezzük, hogy a fájlok a felhasználó "home" könyvtárába kerültek letöltésre).

```
$ cd ~
$ tar -xzf httpd-2.0.44.tar.gz
$ tar -xzf php-4.3.1.tar.gz
```

Ezzel meg is volnánk...

15.1.4. Az APACHE telepítése

Elsőként az Apache kerül telepítésre. Ennek menete nagyvonalakban a következő :

```
$ cd httpd-2.0.44/
$ ./configure --enable-layout=UHU --enable-so \
  --enable-mods-shared=all --disable-suexec
$ make
# make install
```

Következzék egy kis magyarázat.

./configure --enable-layout=UHU Ez az opció arra szolgál, hogy a könyvtárszerkezetet ne kelljen könyvtáranként megadni a *./configure*-nak, hanem a *config.layout* fájlból olvassa be az adatokat. Ahhoz, hogy működjön, ki kell egészíteni a *config.layout* fájlt néhány sorral, amelyek a következők:

```
# UHU-Linux layout
<Layout UHU>
prefix:           /usr
exec_prefix:      ${prefix}
bindir:           ${prefix}/bin
sbindir:          ${prefix}/sbin
libdir:           ${prefix}/lib
libexecdir:       ${prefix}/lib/apache2
mandir:           ${prefix}/share/man
sysconfdir:       /etc/apache2
datadir:          /usr/share/apache2
installbuilddir: /etc/apache2/build
errordir:         ${datadir}/error
```

```

iconsdir:      \${datadir}/icons
htdocsdir:     /var/www
manualdir:     \${datadir}/manual
cgidir:        \${libdir}/cgi-bin
includedir:    \${prefix}/include/apache2
localstatedir: /var
runtimedir:    \${localstatedir}/run
logfiledir:    \${localstatedir}/log/apache2
proxycachedir: \${localstatedir}/cache/apache2
</Layout>

```

–**enable-so** Ez azt teszi lehetővé, hogy használni tudjuk a dinamikus modulokat.

–**enable-mods-shared=all** Az összes elérhető modult dinamikusan betölthetővé fordítja le.

–**disable-suexec** Ez egy olyan modul, melynek helytelen használatából kifolyólag hatalmas biztonsági réseket lehet nyitni a szerveren, így talán jobb, ha most ezzel még nem foglalkozunk...

make Ezzel a paranccsal megkezdődik a webservert lefordítása, amely gépünk teljesítményétől függően 5-10 percig tart (régibbi gépek esetén akár több óráig is).

make install Ezt a parancsot root-ként kell futtatni, ezzel végső helyükre kerülnek a fájlok.

15.1.5. A PHP telepítése

Felvázoljuk a teendőket, majd rövid magyarázattal szolgálunk:

```

$ cd ~
$ cd php-4.3.1/
$ ./configure --with-layout=GNU --prefix=/usr \
  --sysconfdir=/etc/php4 --with-config-file-path=/etc/php4 \
  --with-apxs2 --without-mysql
$ make
# make install

```

./configure --with-layout=GNU Hasonló a funkciója, mint az Apache esetében, de itt nem kell szerkesztgetni...

–**prefix=/usr** Így nem a */usr/local* alá lesznek másolva a fájlok, hanem oda, ahol a többi program is tartja őket (*/usr*).

- sysconfdir=/etc/php4** Ebben a könyvtárban lesznek eltárolva a konfigurációs fájlok.
- with-config-file-path=/etc/php4** A *PHP* itt fogja keresni a *php.ini* fájlt, amely a beállításokat tartalmazza.
- with-apxs2** Ez az opció mondja meg a *PHP*-nek, hogy az *Apache 2.0*-hoz lesz fordítva...
- without-mysql** Erre csak akkor van szükség, ha esetlegesen a “make” egy *MySQL* fájlal kapcsolatos hibaüzenettel leállna (sajnos ez előfordulhat). Így nem kerül bele a *PHP*-ba a *MySQL* támogatás...

15.1.6. A PHP beállítása

Miután lefordult mindkét összetevő, nincs más hátra, mint beállítani a *PHP*-t, hogy együttműködjön az *Apache*-al.

Első teendőnk az, hogy az *Apache* által használt és értelmezett MIME típusok közé felvegyük a *PHP* fájlokat is. Adjuk hozzá a következő sorokat a */etc/apache2/mime.types* fájlhoz:

```
# PHP mime types
application/x-httpd-php      php
application/x-httpd-php-source phps
application/x-httpd-php3     php3
application/x-httpd-php4     php4
```

Ezek után rá kell bírunk arra az *Apache*-ot, hogy töltsse be indításkor a *PHP* modult is. A */etc/apache2/httpd.conf* fájlban a sok *LoadModule* sor után írjuk be a következőt:

```
# Load PHP module
LoadModule php4_module lib/apache2/libphp4.so
```

Így már elvileg minden működik.

15.1.7. Az APACHE indítása, leállítása

Erre a legalkalmasabb egy *init.d* szkript, amely a gép minden indulásakor elindítja, ill. minden leállításakor leállítja a webszervert. Módosítsuk az előre elkészített mintát (*/etc/init.d/SKELETON*), majd mentjük el */etc/init.d/daemons/httpd* néven.

```
---=[ /etc/init.d/daemons/httpd ]---
#!/bin/sh
#
# /etc/init.d/daemons/SKELETON
#
# (C) 2001-2002
# Pozsar Balazs <pozsy@uhulinux.hu>
#
# This file is originally part of the UHU-Linux distribution.
# Distributable under GPL v2.
#
# Ez a fájl eredetileg az UHU-Linux disztribúció része.
# A GPL v2 licenc szerint terjeszthető.
#

# -----
DAEMON=/usr/sbin/httpd
NAME="httpd"
DESC="Apache 2.0 Webszerver"

# -----
PATH="/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin"
PIDFILE="/var/run/$NAME.pid"

[ -x "$DAEMON" ] || exit 0

[ -f /etc/init.d/common ] && . /etc/init.d/common || {
    echo "Hiba a /etc/init.d/common beolvasása közben" >&2
    exit 1
}

loaddefaults "$NAME"

function error() {
    echo "$ERR"
    exit 1
}

case "$1" in
    start)
        echo -n "$DESC indítása ($NAME)"
        try start-stop-daemon --oknodo --start --quiet --pidfile "$PIDFILE" \
            --exec "$DAEMON"
        echo "$OK"
        ;;
    stop)
        echo -n "$DESC leállítása ($NAME)"
        try start-stop-daemon --oknodo --stop --quiet --pidfile "$PIDFILE" \
            --exec "$DAEMON"
```

```

echo "$OK"
;;
    reload|force-reload)
echo -n "$DESC konfigurációs fájljainak újraaktiválása... ($NAME)"
try start-stop-daemon --oknodo --stop --signal 1 --quiet --pidfile \
    "$PIDFILE" --exec "$DAEMON"
echo "$OK"
;;
    restart)
"$0" stop
sleep 1
"$0" start
;;
    *)
echo "Használat: $0 {start|stop|restart|reload|force-reload}" >&2
exit 1
;;
esac

exit 0

---=[ /etc/init.d/daemons/httpd ]=---

```

Ahhoz, hogy a gép minden indulásakor/leállításakor lefusson ez a szkript, a következő sort illesszük be a */etc/runlevel.conf* fájlba:

```
20 0,1,6 2,3,4,5 /etc/init.d/daemons/httpd
```

Már csak futtathatóvá kell tenni az indító szkriptet:

```
# chmod +x /etc/init.d/daemons/httpd
```

Mostmár kipróbálhatjuk web szerverünket!

```
# /etc/init.d/daemons/httpd start
```

A parancs végrehajtása után a következő üzenetet kell megkapnunk:

```
Apache 2.0 Webszerver indítása (httpd) [ OK ]
```

Készítsünk gyorsan egy fájlt, és mentjük el */var/www/phpinfo.php* néven:

```
<?php phpinfo(); ?>
```

Most navigáljunk böngészőnkkel a `verb+http://localhost/phpinfo.php` címre, ahol ha mindent jól csináltunk, akkor egy PHP logóval díszített információs oldal jelenik meg.

15.2. Az Apache finomhangolása

Ebben a részben a httpd.conf fájl alapvető opcióit fogjuk sorra venni.

15.2.1. Listen

Azt állíthatjuk be, hogy milyen IP címen, ill. milyen porton várja a kapcsolatokat az Apache.

Példák:

```
Listen 80 - a 80-as porton várja a kapcsolatokat  
Listen 10.0.0.1:8000 - a 10.0.0.1-es IP címen és a 8000-es porton figyel
```

15.2.2. User/Group

Beállíthatjuk, hogy milyen felhasználó-, ill. milyen csoportjogosultságokkal fusson a szerver. Célszerű erre a célra külön felhasználót és csoportot létrehozni.

Példák:

```
User nouser  
Group nogroup
```

vagy

```
User www-data  
Group www-data
```

15.2.3. ServerAdmin

A szerver adminisztrátorának, karbantartójának az e-mail címét adhatjuk meg. (A hi-baoldalakon fog megjelenni)

Példa:

```
ServerAdmin webmaster@domain.hu
```

15.2.4. ServerName

Azt az Internet vagy IP címet adhatjuk meg, ahol a szerver található (akár a portot is megadhatjuk).

Példák:

```
ServerName hidraulika.uhulinux.hu
ServerName 10.0.0.1
ServerName webserver.cegem.hu:80
ServerName 192.168.0.1:8000
```

15.2.5. DocumentRoot

Ez a webszerver gyökérkönyvtárát jelöli. Tehát ha például a böngészőben a *http://localhost/phpinfo.php* oldalt hívom be, akkor valójában a */var/www/phpinfo.php* fájlt nyitom meg. Ha ezt megváltoztatjuk, akkor a többi utalást is cseréljük le!

Példa:

```
DocumentRoot /var/www
```

15.2.6. UserDir

Ez a beállítás annak a könyvtárnak a nevét tartalmazza, amelyben a webszerver keresen, ha *+/felhasznalo+* kezdetű kérést kap.

Például meghívom a *+http://localhost/ferike/index.html+* oldalt, akkor az Apache a *+/home/ferike/public_html/index.html+* fájlt fogja megnyitni.

Ha ezt a funkciót használni kívánjuk, akkor engedélyeznünk kell a felhasználók könyvtárának olvasását a webszervert futtató felhasználó számára is:

```
# chmod 755 /home/*
```

Példa:

```
UserDir public/_html
```

15.3. Az Apache telepítése csomagból

Az UHU-Linux fejlesztői szerencsére már elkészítették az Apache csomagot, így azt nem szükséges forrásból lefordítani.

A csomagokat a következő címen találjuk meg:

ftp://ftp.uhulinux.hu/uhu/1.0/cd2/packages/

Innen a következő csomagokat kell letöltenünk ahhoz, hogy a PHP-t rendesen telepíthessük: *apache*, *apache-dev*, *apache-mod-accs*, *apache-mod-alias*, *apache-mod-autoindex*, *apache-mod-dir*, *apache-mod-mime*, *apache-mod-negotiation*, *apache-mod-userdir*, *libapr-dev*.

Egy alternatív megoldás, hogy az apt-get segítségével töltjük le a csomagokat. Ehhez az alábbi parancsokat kell rendszergazdaként (root) kiadnunk:

```
# apt-get update
# apt-get install apache apache-dev apache-mod-accs \
  apache-mod-alias apache-mod-autoindex apache-mod-dir \
  apache-mod-mime apache-mod-negotiation \
  apache-mod-userdir libapr-dev
```

Miután telepítettük a csomagokat, néhány helyen módosítsuk a konfigurációs fájlokat, hogy ne legyen probléma.

/etc/apache/httpd.conf • a Namevirtualhost sorok elé írjuk egy #-et

/etc/apache/main.conf • a 120. sort javítsuk erre: *<IfModule mod_dir.c>*

- a 107-118. sorok előtt kivehetjük a #-et
- a többi beállítást is módosíthatjuk.

Ezek elvégzése után fordítsuk le a PHP-t az ismertetett módszerrel.

Amikor ezzel is végeztünk, itt is adjuk hozzá a /etc/apache/mime.types fájlhoz a PHP-ra vonatkozó dolgokat.

Az Apache UHU csomag különleges felépítése miatt itt máshogy kell megoldani a PHP modul betöltését: létre kell hozni egy php4_module fájlt a /etc/apache/mods/loaders/ könyvtárban az alábbi tartalommal:

```
LoadModule php4_module /usr/lib/apache/libphp4.so
```

Ezek után már csak annyi a dolgunk, hogy elindítjuk az Apache-ot:

```
# /etc/init.d/daemons/apache start
```

Itt is tesztelhetjük a PHP-t a már említett módon, de figyeljünk oda, hogy itt az alapértelmezett könyvtár a `/var/www/html`, tehát ide kell másolnunk a `phpinfo.php` fájlunkat.

Copyright (c) 2003 by Szilveszter Farkas (Phanatic) <linux@psoftwares.hu>

All rights reserved.

15.4. A Postfix

15.5. A proftpd

15.6. A MySql

A szabad szoftverek megjelenésével egyre nagyobb igény keletkezett olyan adatbázis kezelőre, ami könnyen kezelhető, a modern adatkezelési elveknek megfelel és megfelelően gyors is.

A MySQL SQL alapú adatbáziskezelő. Az SQL (Structured - egyes ellenzői szerint Simple - Query Language, azaz strukturált lekérdező nyelv) első változatát az IDM az 1970-es évek elején fejlesztette ki. Alapelve a lekérdezések szabatos (angol) mondatokban való megfogalmazhatósága, így viszonylag kevés számítástechnikai ismeretet igényel. A kialakítás olyan sikeres volt, hogy az ORACLE megjelentette első kereskedelmi megvalósítást. Jelenleg az SQL2 számít szabványnak (másnéven SQL-92, mert 1992-ben jelent meg), de nemrég megjelent az SQL3 is.

A MySQL fejlesztői kifejezetten a gyorsaságra és az interaktivitásra törekedtek. Ez természetesen háttérbe szorított olyan szolgáltatásokat, amelyek hatékonyak ugyan, de megvalósításuk lassulást okozott volna (pl. tranzakció kezelés). Más kérdés, hogy a mai erős hardvereknél ez már nem okoz gondot, így néhány kényelmes lehetőség a megjelenés előtt álló 4.1-es változatba már be lett építve (pl. a beágyazott SELECT, vagy a tranzakció kezelés; ez a 3.23-as változattól létezik, de ezekben még nem alapértelmezés), azonban a tárolt eljárások kezeléséig várni kell az 5.0 verzióig.

1979-ben "Monty" Michael Widenius egy belső használatra szánt adatbázist fejlesztett ki, az UNIREG-et, amelyben lévő indexelési megoldás alapjául szolgált az MySQL 2-nek.

A MySQL első változata 3.11 verziószámmal 1995. májusára készült el. A tulajdonos TcX AB egyik üzletfele szorgalmazta az interneten való kiadást. Ez alapul szolgált a későbbi GPL alkalmazására. Időközben a TcX vállalattá alakult át MySQL AB néven

(internetes honlapja <http://www.mysql.com>). Saját becslésük szerint 4 millió felett van a felhasználók száma (ebbe természetesen az összes platformon - Windows, Macintosh, a Unix különböző változatai - futó változatot beleértik).

15.6.1. Telepítés

Mint minden programot, a MySQL többféle módon telepíthető:

- binárisból (ún. tarball)
- forrásból
- csomagokból

A bináris és forrás alapján való telepítést csak érintőlegesen tárgyaljuk, tekintettel arra, hogy minden disztribúció, így az UHU-Linux is tartalmazza a megfelelő csomagokat.

Telepítés bináris alapján

A MySQL honlapján mindig megtalálható a legfrissebb változat bináris tarball-ja. A változat névadási konvenciója úgy lett kialakítva, hogy mindig tartalmazza azt a platformot, amire készítették.

Első lépésként létre kell hozni a megfelelő felhasználót és csoportot (nyilvánvalóan mindkettő mysql). Ezt követően ki kell bontani a gunzip és tar programok kombinációjával a tarball tartalmát a megfelelő könyvtárba (pl. */usr/local/mysql*). Ezzel lényegében kész is a telepítés, de még nem használható, ugyanis az adatbázist is létre kell hozni. Ezt a script könyvtárban található `mysql_install_db` szkript végzi el. Most már indítható az adatbázis szerver:

```
# bin/safe_mysqld -user=mysql &
```

Ennél ésszerűbb egy indító szkriptet készíteni. Ehhez jó támpontot nyújt a */etc/init.d/daemons/mysqld* szkript (UHU 1.0), de általában minden disztribúció tartalmaz olyan szkriptet, ami a betöltési folyamatba becsatolható. Ezek bármelyikének némi átalakításával automatizálható a szerver indítása

Telepítés forrásból

Amíg a tarball-ból való telepítés egy előre elkészített binárist helyez a megadott helyre, addig a forrásból való telepítés lehetőséget nyújt az elkészítendő bináris testreszabására.

A folyamat megfelel más programok forrásból való telepítésének:

Először le kell tölteni a forrást (újabbban ez a MySQL honlapjának "letöltés" szekciójának alján található), amit egy ideiglenes könyvtárba kell kibontani. Ezt követően a `./configure` parancsot le kell futtatni a megfelelő opciókkal. Ehhez segítséget nyújt a `--help` kapcsolóval való indítás, ami a lehetőségeket kilistázza. A választandó opciók meghatározásához mindenképpen el kell olvasni a dokumentációs fájlokat, ugyanis verzióként eltérő lehet az alapértelmezés.

A sikeres konfigurálást követően a `make` parancs végzi el a binárisok előállítását. Ez hosszabb időt vesz igénybe. Lefutását követően root-ként kiadva a `make install` (pl.: `su -c 'make install'`) elvégzi a binárisok megfelelő helyre való bemásolását.

A beállításokat ekkor is el kell végezni (pl. az adatbázis inicializálását), majd ezt követően ugyanolyan elv szerint lehet az adatbázist elindítani, mint a tarball-ból való telepítés esetén.

Telepítés csomagokból

Az előző esetekben a telepítés nem vizsgál bizonyos összefüggéseket. Ez - szélsőséges esetben - azzal is járhat, hogy további összetevőket kell az adatbázis szerver működéséhez telepíteni, bár a MySQL AB fejlesztői igyekeztek minden eshetőségre felkészülni.

Az egyes disztribúciókban megjelenő csomagok ettől megkímélik a felhasználót. Az UHU-Linuxban használt megoldás installáláskor a megadott hely(ek)ről megkísérli letölteni a szükséges csomagokat, és csak akkor telepíti a MySQL-t, ha az sikeres volt. Erről részletesebben a csomagok telepítéséről szóló fejezetben írtunk.

A mysql telepítésekor a következő csomagok kerülnek a rendszerbe:

mysql-common minden olyan fájl, ami a mysql általános működéséhez szükséges,

mysql-server az adatbázis szerver futtatásához szükséges fájlok, segédprogramok,

mysql-client kliens működtetéséhez szükséges programok,

libmysql<xx> (<xx>: jelenleg 10) olyan dinamikus könyvtárak, amelyek a szerver és/vagy kliens működtetéséhez elengedhetetlenül szükségesek.

Amennyiben elegendő winchester kapacitás áll rendelkezésre, célszerű a teljes mysql programcsomag választékot telepíteni a következő paranccsal:

```
# apt-get install mysql*
```

Ez egyrészt a `mysql-doc` csomagot is feltelepíti, ami egy igen részletes dokumentációt is tartalmaz (beleértve a SQL utasítások magyarázatát, szintakszisát is), másrészt a `libmysql<xx>-dev` csomagot, ami saját funkciók megírásakor szükséges. A dokumentáció erre vonatkozóan viszonylag kevés információt tartalmaz, azonban a saját funkció készítéséhez kellő információ található a MySQL honlapján.

Külön megemlítjük a `mysqlcc` nevű programot, ami külön csomagban települ. Bár még csak fejlesztési szakaszban van, érdemes legalább kipróbálni.

Amennyiben Apache szervert kívánunk úgy üzemeltetni, hogy a php felületen a mysql hívások is működjenek, szükséges telepíteni a `php-mysql` csomagot is és a szükséges beállításokat elvégezni. Mivel ezt a `php.ini` fájlban, illetve az Apache beállító fájljában kell elvégezni, a megfelelő fejezetekben van erre utalás.

Ha programot szeretnénk készíteni, ami a mysql szerveren lévő adatokat kezeli, akkor - programnyelvtől függően - kiegészítő csomagokra lehet szükség. C illetve C++ esetén a `libmysql<xx>-dev` elegendő, de perl használatához több kiegészítő modul (DBI, mysql) installálása is elengedhetetlen. Szerencsére nem kell "kutatómunkát" folytatni, mivel a `perl-mysql-modules` csomag ezeket tartalmazza, illetve ennek telepítése során a függőségeket letölti.

A telepítést követően további feladatokra nincs szükség, mert a telepítő létrehozza az alap adatbázist is. Ezen felül egy szkriptet is elhelyez a `/etc/init.d/daemons` alá (UHU 1.0 alatt) `mysqld` néven, amit root felhasználóként el lehet indítani. Ha a rendszerbe-töltés során automatikusan akarjuk indítani, akkor az uhu-control-centerben ezt be lehet állítani.

15.6.2. Beállítások

Ahhoz, hogy az adatbázis kezelőt használatba vehessük, mind a szerver oldalon, mind a kliensnél beállításra lehet szükség. Általános szabály az, hogy bármilyen beállításról is van szó, a MySQL a különböző helyeken található beállító értékeket egy meghatározott sorrendben dolgozza fel és a később érkező beállítási érték mindig felülírja az előző értéket. Ez a következőkben érthetőbbé válik.

Amikor maga az adatbázis szerver elindul, vagy egy ügyfélfolyamatot indítunk el, a következő sorrendben történik a beállítások értékeinek vizsgálata:

- környezeti változók,

- konfigurációs fájlok,
- parancssori kapcsolók

Ez azt jelenti, hogy bármilyen beállítás is szerepel bárhol, mindig az utolsó megtalált érték lesz az érvényes. Például amennyiben mindhárom helyen szerepel jelszó, akkor a parancssorban megadott lesz az érvényes (mert ezt vizsgálta legutoljára).

A környezeti változók a szokásos módon adhatóak meg: `<változónév>=<érték>`. Megjegyezzük, hogy ez a legkevésbé használatos mód, éppen a vizsgálati sorrend miatt. Ezen felül minden környezeti változó megadható konfigurációs állományban, ezért ez a célszerűbb.

A konfigurációs állományok több helyen lehetnek, ezért az ezzel kapcsolatos fontosabb ismereteket külön részletezzük.

A parancssori kapcsolók az indításkor adhatóak meg. Ez is kevésbé népszerű megoldás (egy-két esetet kivéve), mivel az adatbázis szerver jellemzően a rendszer betöltésekor elindul, a sűrűn használt kapcsolók, beállítások pedig a konfigurációs állományban szoktak szerepelni.

A konfigurációs állományok vizsgálati sorrendje

Már a rendszer telepítése során is keletkezik konfigurációs állomány. Alapértelmezésben ez a `my.cnf`, bár a fordítás során ez felülírható. Ez azonban nem célszerű!

A `my.cnf` több helyen is lehet. A szerver, illetve a kliens oldali programok a következő sorrendben olvassák be:

- `/etc/my.cnf`; itt szerepelnek a rendszerszintű beállítások. Ez általánosságban itt van, de a fordítás során más elérési út is megadható. Az UHU-Linux pl. `/etc/mysql/my.cnf` útvonalat használja.
- `$DATADIR/my.cnf`; mivel a MySQL ismeri az instance (példány) fogalmát, szükség lehet a példányonkénti egyedi beállításra. Ez szerepel ebben az állományban. A `$DATADIR` az a könyvtár, ahol a MySQL az adatállományt tárolja (ez általában a `/var/lib/mysql`).
- A `--default-extra-file=<elérési útvonal>` parancssori kapcsolóval megadott fájl; ez tulajdonképpen azt a lehetőséget biztosítja, hogy az állandó beállításokat ideiglenesen felülírjuk (nyilvánvalóan tesztelési jelleggel).
- `$HOME/.my.cnf`; ebben szerepeltethetők a felhasználó egyedi beállításai.

Fontosnak tartjuk kiemelni, hogy a parancssori kapcsolóban megadott fájl tartalmát a `$HOME/.my.cnf` felülírja! Ha kísérletezni szeretnénk, akkor ezt a fájlt érdemes ideiglenesen eltávolítani.

A konfigurációs állomány formátuma, tartalma

A MySQL fejlesztői abból indultak ki, hogy a konfigurációs állomány önmagában is jól érthető legyen. Ehhez azt a formátumot választották, ami már régebb óta ismert volt. Ennek lényege az, hogy csoportok vannak, ezen belül egyes paraméterek értéke egyenlőségjellel van hozzárendelve. A gyakorlatban ez így néz ki:

```
# The following options will be passed to all MySQL clients
[client]

# password = your_password

port = 3306
socket = /var/run/mysqld/mysqld.sock

# Here follows entries for some specific programs

# The MySQL server

[mysqld]
port = 3306
socket = /var/run/mysqld/mysqld.sock
skip-locking

[mysqldump]
quick
set-variable = max_allowed_packet=16M

[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
# safe-updates
```

A csoport megjelölése "[" és "]" jelek között szerepel. Ebben a sorban más nem lehet (csak megjegyzés). Az egyes paraméterek pedig <megnevezés>=<érték> formában adható meg. Ügyelni kell arra, hogy az adott paraméternek megfelelő szintaxist alkalmazzuk, ellenkező esetben nem maghatározható értéket kap az adott paraméter, ami nem kívánt következménnyel járhat. Ezen felül nincs lehetőség beágyazott érték megadására (pl. kapu=3306, majd port=\$kapu hibás!). Bár a dokumentáció nem említi, de általános paraméter (azaz az első csoport előtt) nem adható meg.

A példából is kiderül, hogy bárhol elhelyezhető megjegyzés jel: a "#". Ami ezután szerepel, a programok figyelmen kívül hagyják.

Az egyes csoportok alkalmazás specifikusak. Ez azt jelenti, hogy a [client] csoportban megadott paraméterek vonatkoznak a kliens oldalon indított programokra (hacsak

az adott programra vonatkozó csoportban ezt nem írjuk felül), ugyanakkor a [mysqld] csoportban megadott paramétereket nem biztos, hogy figyelembe veszi.

Tartalmilag minden programot azonosító csoportban olyan paramétereknek lehet értéket adni, ami parancssori kapcsolókkal is megadható (kivéve természetesen azokat, amelyek csak parancssorban definiálhatók). Ennek felsorolása messze túllépi e dokumentáció kereteit, viszont könnyen szerezhető segítség az adott program `--help` kapcsolóval való indításával (ami viszont nem adható meg konfigurációs állományban :-)), illetve program "man" oldalán.

A beállítások tanulmányozásához megfelelő segítséget nyújtanak még a `/usr/share/doc/mysql/server/examples` alatt található példa beállító fájlok. Érdeemes átnézni azokat és a gépünk adottságainak megfelelőiből (itt elsősorban az operatív memória mérete a mérvadó) kiindulni.

15.6.3. Jogosultságok, felhasználók felvétele

Ebben a témakörben számtalan dokumentáció született, mert maga a jogosultság témakör egy külön könyvet is meg tud tölteni. Ebben a dokumentációban csak azokat az ismereteket közöljük, ami az elinduláshoz szükséges.

A jogosultságok kezelése

Ugyanúgy, mint a beállítások, konfiguráció esetén, a jogosultságok vizsgálata is meghatározott sorrend szerint történik. Ehhez tudni kell, hogy a MySQL külön adatbázist tart fent a jogosultságok kezeléséhez, a "mysql"-t. Bár az ebben lévő táblák egyszerű SQL utasításokkal is kezelhetők, ezt nagyon nem tanácsoljuk, inkább a GRANT és REVOKE parancsok alkalmazását részesítjük előnyben.

A jogosultságokat objektum szinten lehet értelmezni. Az objektumok:

oszlop egy-egy tábla egy-egy mezője,

tábla az adatbázis ugyanolyan struktúrájú rekordjait összefogó adatok,

adatbázis logikailag összetartozó táblák összessége,

szerver az adatbáziskezelő kiszolgáló feladatait ellátó bázis szoftver.

(A definíciók önkényesek)

A jogosultságok:

USAGE a szerverhez való kapcsolódás joga,

SELECT táblá(k)ból adat(ok) olvasása,

UPDATE adatok módosítása (törléshez új adat létrehozásához nem elegendő),

INSERT új rekord(ok) létrehozása,

DELETE rekord(ok) törlése,

ALTER tábla struktúrájának, jellemzőinek megváltoztatása, de ez nem vonatkozik az indexekre,

CREATE új adatbázis/tábla létrehozása,

DROP adatbázis/tábla eltávolítása,

INDEX tábla indexeinek létrehozása, eltávolítása, módosítása,

FILE magával a MySQL szerverrel azonos jogokkal lehet a szerveren tárolt fájlokhoz hozzáférni; a **LOAD DATA INFILE** és **SELECT INTO OUTFILE** parancsok végrehajtásához szükséges,

RELOAD adatok újra betöltéséhez; ez a jog szükséges a **FLUSH** utasítás végrehajtásához,

SHUTDOWN az adatbázis szerver leállításához való jog.

PROCESS a MySQL folyamatokhoz való teljes hozzáférés, beleértve a **KILL MYSQL** utasítást is,

A két fogalom kapcsolatát a következő táblázat szemlélteti:

Jog	Oszlop	Tábla	Adatbázis	Kiszolgáló
USAGE	-	-	-	-
SELECT	X	X	-	-
UPDATE	X	X	-	-
INSERT	X	X	-	-
DELETE	X	X	-	-
ALTER	X	X	-	-
CREATE	-	X	X	-
DROP	-	X	X	-
INDEX	-	X	-	-
FILE	-	-	-	X

RELOAD	-	-	-	X
SHUTDOWN-	-	-	-	X
PROCESS	-	-	-	X
=====				

Az ismertett információk alapján könnyen beállíthatóak a jogok. A GRANT és REVOKE parancsok használatát, szintaxisát a dokumentáció tartalmazza.

15.6.4. Felhasználók felvétele

A felhasználók létrehozásához tudni kell, hogy alapértelmezésben létezik egy "root" nevű felhasználó, aminek nincs jelszava. **NAGYON FONTOS, HOGY AZ INSTALLÁLÁS UTÁN EZT AZONNAL ÁLLÍTSUK BE!** Erre a GRANT parancs alkalmas a következőképp:

```
GRANT ALL PRIVILEGES ON *.* TO root IDENTIFIED BY 'ez_a_jelszavam' \
WITH GRANT OPTION;
```

(értelemszerűen az "ez_a_jelszavam" helyére a megfelelő titkos jelszó írandó. Arra viszont ügyelni kell, hogy ez egyszeres idézőjellel - aposztróffal - legyen körülvéve).

A felhasználók felvételéhez tudnunk kell, hogy lehetőség van általános karakterek (wildcards) használatára. Amíg az adatbázis, azok tábláinak jelölésénél a "*" karaktert lehet használni, addig a felhasználók esetében a '%' (százalékjel) használható, pl: zsebibaba@"%"; ez bármilyen gépről az "zsebibaba" nevű felhasználót jelzi. Ez egyben az azonosítás módszerének az ismeretéhez vezet.

Amikor egy felhasználó kapcsolódik az adatbázishoz, megadja a felhasználó nevét, esetleg a hoszt nevet. A MySQL megkezdi az azonosítási folyamatot:

- megvizsgálja hoszt nevét. Minél jobban meghatározott, annál nagyobb a valószínűsége az egyezésnek, azaz először a pontos egyezést keresi, majd a helyettesítő karakter is tartalmazó értéket, végül a csak helyettesítő karaktert tartalmazó bejegyzést. A határozottabb az egyezés számít találatnak,
- a felhasználói nevek vizsgálatánál a megtalált hosztnévből indul ki. Itt is a meghatározottságot veszi alapul. Ha nem talál egyezést, akkor az üres felhasználó nevet tartalmazó rekordot veszi alapul.

A jobb megérthetőség kedvéért tételezzük fel a következő felhasználó-bejegyzéseket:

```
root@localhost
@localhost
(ezek alapértelmezetten már installáláskor szerepelnek)
micimacko@localhost
tigris@100holdas_pagony.aam
zsebibaba@%"
nyuszi@%" .100holdas_pagony.aam
```

Most pedig vizsgáljuk meg a megkeresések szerint:

- A saját gépről belépve root-ként 100%-os egyezés lesz, mert a localhost automatikusan adódik hozzá, ilyen rekord pedig van.
- A micimacko@localhost bejelentkezés szintén meghatározott, mert van ilyen bejegyzés,
- malacka@localhost esetén nincsen egyező sor, de létezik @localhost bejegyzés, ezért annak a jogait kapja meg,
- zsebibaba@100holdas_pagony.aam azonosításnak nincs pontos megfelelője, de létezik a zsebibaba@%" bejegyzés, így az itt szereplő jogok alkalmazása történik,
- zsebibaba@localhost azonosítás szokott a legtöbb félreértésre, hibára okot adni: 2 megfelelője is van, a zsebibaba@%" és a @localhost. Mivel az utóbbi számít meghatározottabbnak (ez lényegében a %"@localhost-nak fele meg), ezért az ebben a rekordban szereplő jogok lesznek a mérvadók,
- malacka@tilos_az_a.100holdas_pagony.aam bejelentkezése el lesz utasítva, mert semmilyen egyezés nem szerepel.

15.6.5. Hasznos segédprogramok

A MySQL egyszerű kezelhetősége ellenére számos olyan funkciót nem tartalmaz vagy csak körülményesen érhető el, amely a mindennapi munkában, a rendszer működőképességének fenntartásához elengedhetetlenül szükséges. Ebben a részben vázlatosan ismertetjük a csomagokban szereplő segédprogramokat.

Felhasználó kezelése: mysqlaccess

A kézikönyv oldal szerint ez egy felhasználó létrehozására alkalmas segédprogram, de ennél többet nem árul el. A beépített súgó is a lekérdezésre ad példát. Tulajdonképpen

ez a funkció a megfelelő, hiszen új felhasználó létrehozására magában a mysql-ben megfelelő lehetőség van.

A program meghívása:

```
mysqlaccess [kapcsolók] hoszt [felhasználó [db]]
```

A kapcsolóknál számos lehetőség van, ami elsősorban a belépő felhasználó azonosítását segíti.

Egy példa a localhost hoszton a kukkolo nevű felhasználó jogai a vegyes nevű adatbázison:

```
root@janUHU:~# mysqlaccess localhost kukkolo vegyes
mysqlaccess Version 2.06, 20 Dec 2000
By RUG-AIV, by Yves Carlier (Yves.Carlier@rug.ac.be)
Changes by Steve Harvey (sgh@vex.net)
This software comes with ABSOLUTELY NO WARRANTY.
```

Access-rights

for USER 'kukkolo', from HOST 'localhost', to DB 'vegyes'

Select_priv	Y	Shutdown_priv	N
Insert_priv	Y	Process_priv	N
Update_priv	Y	File_priv	N
Delete_priv	Y	Grant_priv	N
Create_priv	Y	References_priv	Y
Drop_priv	Y	Index_priv	Y
Reload_priv	N	Alter_priv	Y

NOTE: A password is required for user 'kukkolo' :-(

The following rules are used:

```
db      : 'localhost','vegyes','kukkolo', (->)
'Y','Y','Y','Y','Y','Y','N','Y','Y','Y'
host    : 'Not processed: host-field is not empty in db-table.'
user    :
'localhost','kukkolo','7a090e0058335e09', (->)
'N','N','N','N','N','N','N','N','N','N','N','N','N','N'
BUGs can be reported by email to Yves.Carlier@rug.ac.be
root@janUHU:~#
```

A (->) jel azt mutatja, hogy a következő sor ténylegesen ezzel a sorral egyben van.

Az adatbázis adminisztrációja: mysqladmin

Ez a program elsősorban az adatbázis adminisztrátorok eszköze. Hívása:

```
mysqladmin [kapcsolók] parancs [...]
```

A kapcsolók a szokásosak: a hoszt, felhasználó, jelszó megadásához alkalmasak. Ezen felül megadható az a port is, amin a távoli adatbázis kommunikál (ha ez nem az alapértelmezett 3306).

A parancsok a jellegzetes adminisztrációs parancsok: adatbázis létrehozása, eldobása, shutdown stb. Óvatosan kell vele bánni, mert visszavonhatatlan műveletek hajthatók végre és nem tesz fel biztonsági kérdést!

A mysqladmin paraméter nélküli meghívása megfelelő segítséget ad használatához, a man oldal alig részletesebb.

Adatok exportálása: mysqldump

Bár létezik a mysql-re jellemző adat exportálási lehetőség, a mysqldump arra használható, hogy adatbázisok, egyes táblák tartalmát SQL utasítások alakjában elmentse.

Hívásának több alakja is van, az alkalmazandó kapcsolóktól függően. Jellemző meghívása:

```
mysqldump -u <felhasználó> [--add-drop-tables] <adatbázis> [<táblák>]
```

Ez az alapértelmezett kimenetre (a képernyőre) dolgozik, ezért a fájlba irányítással lehet a lemezre készíteni export fájlt:

```
mysqldump ... >dumpfile
```

ami később lehetőséget ad az importra:

```
mysqldump ... <dumpfile
```

Adatbázis futás közbeni mentése: mysqlhotcopy

Amíg a mysqldump egy adatbázis exportot készít el, addig a mysqlhotcopy egy tényleges mentést végez: lényegében magát az adatbázist binárisan elmásolja a megadott területre, illetve a megadott helyre. Természetesen lehetőség van a másolat visszatöltésére. Hívása:

```
mysqlhotcopy [kapcsolók] <adatbázis neve>
```

Ha nem adjuk meg a mentési útvonalat, akkor a mysql adatbázis terében (ami alapértelmezésben a /var/lib/mysql) készít külön könyvtárban másolatot. Az alapértelmezett névkiegészítés (suffix): _copy.

Ennek a segédprogramnak is van man oldala.

Az adatbázis menedzselése: mysqlcc

Bár jónéhány menedzselő karbantartó program létezik, ezt azért említjük meg külön, mert a MYSQL AB készítette.

A "cc" kiegészítés a "control-center" rövidítése. A program leírása nem fér bele e dokumentum kereteibe, mert egy nagy tudású, mindenre kiterjedő szervíz programról van szó. Angol nyelvű segítséget tartalmaz, ami megfelelő eligazítást nyújt a felhasználónak. A többi itt említett programmal ellentétben ez grafikus felületet ad, így áttekinthetősége lényegesen jobb.

15.7. A PostgreSQL

15.7.1. Telepítés

Csomagok felrakása

A postgresql-server csomagot kell feltenni:

```
$ apt-get install postgresql-server
```

(ez telepíti a postgresql-client csomagot is)

A csomag felrakása az alábbiakat végzi el:

- felmásolja a szerver működéséhez szükséges fájlokat a */usr/lib/postgresql* könyvtárba illetve az egyéb fájlokat (man oldalak, indító szkript: */etc/init.d/daemons/postgresql*, konfigurációs minta fájlok: */usr/share/postgresql*)
- létrehozza a postgres grup-ot és postgres user-t.
- inicializálja az adatbázis rendszert (cluster) a */var/lib/postgres/data* könyvtárban.

A szerver csomag után ajánlott még feltenni a *postgresql-doc* csomagot ami a teljes szerver dokumentációt tartalmazza html formában. A csomag telepítése után a dokumentációk helye: */usr/share/doc/postgresql*

Adatbázis cluster inicializálása

Adatbázis clusternek nevezzük azt a szerver környezetet ami tartalmazza a konfigurációs állományokat és az induló minta (template) adatbázisokat. Illetve a későbbiek folyamán a user adatbázisokat. Az adatbázis cluster környezetet a csomag telepítése

automatikusan létrehozza, az itt leírtakat csak akkor kell elvégezni ha ez a környezet nekünk nem megfelelő. Az adatbázis cluster vagy rendszer környezet létrehozására az `initdb` program szolgál. Ez a fejezet először ismerteti a template adatbázisok szerepét majd részletesen leírja az `initdb` programot és paramétereit.

A minta (template) adatbázis. Az adatbázis clusterben az `initdb` futtatásakor létrejön két adatbázis a *template1* és a *template0*. A két adatbázis kezdetben teljesen egyforma a *template0* a *template1* adatbázis másolata, de read-only állapotban van. A *template0* adatbázis szerepe az, hogy a *template1* sérülése esetén a *template0* másolásával újra létre lehet hozni. A *template1* adatbázis szerepének megértéséhez a `CREATE DATABASE` parancs működését kell ismernünk. Postgresql-ben egy új adatbázis létrehozása egy már meglévő adatbázis másolásával történik. Azt hogy melyik adatbázis másolásával hozzuk létre az új adatbázist a `CREATE DATABASE` parancs-nak megadhatjuk de alapértelmezésben ez a *template1* adatbázis. A Postgresql szerver telepítése és elindítása után ez a *template1* adatbázis az alapértelmezett adatbázis, ha nem adunk meg semmit akkor ide jelentkeznek be a kliens programok is. De a fentiek miatt ebbe az adatbázisba ne hozzunk létre felhasználói táblákat mert azok ezután minden új adatbázisban megjelennek. Postgresql-ben a munkát kezdjük mindig egy `CREATE DATABASE` parancssal. A *template1* adatbázist abban az esetben módosítsuk ha olyan funkciót, csomagot telepítünk mint például a "PL/pgSQL procedural language" modul. Így az újonnan létrehozott adatbázisokban már ez a funkció automatikusan rendelkezésünkre fog állni.

Az initdb. Mint az már korábban a fejezet elején írtuk, az adatbázis cluster környezet létrehozására az `initdb` program szolgál. Az `initdb` program egyetlen kötelező paramétere az adatbázis cluster adatok könyvtára. Ezt két módon adhatjuk meg:

a. Paraméterként (-D vagy -pgdata=)

Az UHU csomag telepítése alapján:

```
$ /usr/lib/postgresql/bin/initdb \  
-D /var/lib/postgres/data
```

vagy:

```
$ /usr/lib/postgresql/bin/initdb \  
--pgdata=/var/lib/postgres/data
```

b. Környezeti változóban (PGDATA)

Az UHU csomag telepítése alapján:

```
$ export PGDATA=/var/lib/postgres/data  
$ /usr/lib/postgresql/bin/initdb
```

FONTOS!!!: Az `initdb` parancsot mindig annak a felhasználónak a nevében kell futtatni akinek a nevében a szerver folyamat futni fog. UHU esetén ez a `postgres` user. Ezért a fenti parancsok futtatása előtt adjuk ki az alábbi két parancsot:

```
$ su -  
$ su - postgres
```

Itt érdemes megemlíteni az adatbázis és adatbázis cluster közötti különbséget. Az adatbázis cluster magában foglalja a teljes adatbázis szerver környezetet a konfigurációs állományokkal adatbázis felhasználókkal és felhasználói adatbázisokkal együtt. Egy ilyen cluster környezet több felhasználói adatbázist is tartalmazhat ezért egy szerver gépen többnyire elég egy adatbázis cluster környezet létrehozása. De természetesen előfordulhat olyan eset is két vagy több teljesen különálló adatbázis szerver környezetre van szükségünk ugyanazon a gépen. Ezért van szükség arra, hogy az `initdb` parancsnak mindig egyértelműen megadjuk a cluster környezet könyvtárát. Nem árt tudni, hogy az egy gépen több cluster környezet eléggé erőforrás igényes, mivel ilyenkor az adatbázis szerver folyamatot annyi példányban kell elindítani ahány adatbázis cluster környezetünk van.

Az `initdb` paraméterei. Az előző fejezetben már említett könyvtár (directory) kötelező paraméter mellett az `initdb`nek van még néhány fontos paramétere. Itt most csak a legfontosabbakat ismertetjük, a teljes paraméter listát a "Reference Manual" "Server Application" fejezete tartalmazza.

Könyvtár (directory).

```
-D könyvtár  
--pgdata=könyvtár
```

Az adatbázis cluster könyvtára. Itt helyezkednek el fizikailag a konfigurációs fájlok és az adatbázis fájlok.

Karakter kódolás (encoding).

```
-E encoding  
--encoding=encoding
```

A minta (template) adatbázis karakter kódolási beállítását állíthatjuk be vele. Ez a beállítás lesz az alapértelmezett a felhasználói adatbázisok létrehozásakor de felülbírálható. A karakter kódolás paraméter azt adja meg hogy milyen karakter készletet használunk az adatbázisunkban. Ez a beállítás lehetővé teszi a multibyte-os karakterek használatát a reguláris kifejezésekben, a `LIKE` után és még néhány egyéb függvény esetén. Ha nem adjuk meg akkor az alapértelmezett karakter készlet: `SQL_ASCII`.

Magyar karaterkészlethez használható beállítások:

```
LATIN2    -> ISO 8859-2 ECMA-94 Latin Alphabet No.2
UNICODE   -> Unicode (UTF-8)
```

(A teljes listát lásd a dokumentációban: "Administrator's Guide" "Localization")

Nyelvi beállítás (locale).

```
--locale=locale
```

Az alapértelmezett nyelvi beállítás az adatbázis cluster számára. Ha nem adjuk meg akkor az `initdb` a beállított környezeti változók alapján állítja be. UHU-Linux esetén a csomag telepítő az `initdb`-t `locale=C` beállítással futtatja. Ha mi futtatjuk az `initdb`-t a postgres felhasználó alap beállítása mellett akkor a `locale hu_HU` lesz.

Felhasználónév (username).

```
-U username
--username=username
```

Az adatbázis "superuser" neve. Ha nem adjuk meg akkor az a felhasználó lesz akinek a nevében az `initdb` fut. Az adatbázis működése szempontjából nem lényeges ennek a felhasználónak a neve, de ajánlott a szokásos *postgres* név használata ha az `initdb`-t futtató operációs rendszer felhasználó neve más.

Jelszó bekérése (pwprompt).

```
-W
--pwprompt
```

Ha ezt a paramétert megadjuk, akkor az `initdb` futáskor bekéri a *superuser* felhasználó jelszavát. Ha nem tervezzük a jelszó ellenőrzés használatát a *superuser*-es bejelentkezéshez (lásd később) akkor ez a paraméter elhagyható. Ellenkező esetben viszont a jelszó ellenőrzés addig nem működik, amíg nem állítjuk be a jelszót. (A *superuser* jelszavát később is be tudjuk állítani.)

Példa új adatbázis környezet létrehozására

Ha a fentebb leírtak alapján úgy döntünk, hogy a csomag telepítése után létrejött cluster környezet nekünk nem megfelelő, az alábbi módon újat hozhatunk létre. Az alábbiakban leírtakat annak a felhasználónak a nevében kell futtatni, akinek a nevében a szerver folyamat futni fog. UHU-Linux esetén ez a *postgres* user.

a. Ha már fut állítsuk le az adatbázis szerveret (Lásd adatbázis szerver leállítása fejezet.)

b. Töröljük a */var/lib/postgres/data* könyvtár tartalmát vagy hozzunk létre egy új könyvtárat a nekünk megfelelő helyen.

FONTOS!!! A */var/lib/postgres/data* könyvtár tartalmát csak akkor töröljük, ha a Postgresql szervert még nem használtuk vagy pedig bizonyosak vagyunk abban, hogy az eddig végzett munkánkra nincs már szükség.

c. Futtassuk az *initdb* parancsot a nekünk megfelelő paraméterekkel. Például:

```
$ /usr/lib/postgresql/bin/initdb -D /var/lib/postgres/data \
--encoding=LATIN2 --locale=hu_HU
```

d. Indítsuk újra az adatbázis szerveret (Lásd adatbázis szerver elindítása fejezet.) Ha minden jó sikerült akkor az alábbihoz hasonló üzenetet kell kapjunk:

```
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
```

```
The database cluster will be initialized with locale hu_HU.
This locale setting will prevent the use of indexes for pattern matching
operations.  If that is a concern, rerun initdb with the collation order
set to "C".  For more information see the Administrator's Guide.
```

```
Fixing permissions on existing directory /var/lib/postgres/data... ok
creating directory /var/lib/postgres/data/base... ok
creating directory /var/lib/postgres/data/global... ok
creating directory /var/lib/postgres/data/pg_xlog... ok
creating directory /var/lib/postgres/data/pg_clog... ok
creating template1 database in /var/lib/postgres/data/base/1... ok
creating configuration files... ok
initializing pg_shadow... ok
enabling unlimited row size for system tables... ok
initializing pg_depend... ok
creating system views... ok
loading pg_description... ok
creating conversions... ok
setting privileges on built-in objects... ok
vacuuming database template1... ok
copying template1 to template0... ok
```

Success. You can now start the database server using:

```
/usr/lib/postgresql/bin/postmaster -D /var/lib/postgres/data
or
/usr/lib/postgresql/bin/pg_ctl -D /var/lib/postgres/data -l logfile start
```

15.7.2. Beállítás

Az adatbázis cluster konfigurációs állományai a cluster könyvtárban található, alap esetben a `/var/lib/postgres/data` könyvtárban. Itt három konfigurációs fájlt találunk:

postgresql.conf Az adatbázis cluster beállításait tartalmazza.

pg_hba.conf Kliens hitelesítési beállítások. Itt adhatjuk meg ki mely gépekről, milyen módon jelentkezhet be az adatbázisba.

pg_ident.conf Felhasználói azonosítók leképezési táblái. Itt azt adhatjuk meg hogy melyik Unix felhasználó, melyik postgresql felhasználónak feleljen meg.

postgresql.conf

A fájlban található összes opciók ismertetése túlmutat e fejezet keretein, részletes leírást találunk az "Administrator's Guide" "Server Run-time Environment" fejezetében. Itt csak néhány paramétert említünk meg.

pg_hba.conf

pg_ident.conf

15.7.3. Adatbázis szerver elindítása és leállítása

A Postgresql adatbázis szerveret `postmaster`-nek hívják, azaz ez a futtatandó szerver folyamat neve. A szerver folyamat elindításához, leállításához és állapotának lekérdezéséhez van egy remek kis segédprogram, aminek `pg_ctl` a neve, a legjobb ha ezt használjuk a szerver elindításához és leállításához.

Adatbázis szerver elindítása

Az adatbázis szerver elindításakor paraméterként az adatbázis cluster könyvtárat kell megadnunk. Ha nem adjuk meg, akkor a `PGDATA` környezeti változóban megadott helyen keresi. Ajánlott még a logfájl paraméter megadása. UHU-Linux esetén a logfájlok helye a `/var/log` könyvtár:

```
$ /usr/lib/postgresql/bin/pg_ctl \
-D /var/lib/postgres/data \
-l /var/log/postgres.log start
```

Adatbázis szerver leállítás

A Postgresql szervert a futó szerver folyamatnak küldött szabványos megszakítás üzenet küldésével tudjuk leállítani. Linux esetén erre a `kill` parancs szolgál. Mint minden rendes adatbáziskezelőnél, a postgresql-nél is több fajta leállítási módot különböztetünk meg:

Normal Shutdown SIGTERM

```
$ kill -TERM <postmaster folyamat azonosító>
$
```

A *SIGTERM* szignál után a postmaster nem engedélyez több új kapcsolatot létesíteni az adatbázis szerverrel, de megvárja, hogy a futó háttérfolyamatok normál módon véget érjenek. Ezután a postmaster folyamat is leáll.

Fast Shutdown SIGINT

```
kill -INT <postmaster folyamat azonosító>
```

A *SIGINT* szignál után a postmaster nem engedélyez több új kapcsolatot létesíteni az adatbázis szerverrel, és *SIGTERM* szignált küld minden futó háttérfolyamatnak, aminek hatására azok abortálják az éppen futó tranzakcióikat és azonnal leállnak. A postmaster folyamat megvárja a háttérfolyamatok leállítását és csak ezután áll le.

Immediate Shutdown SIGQUIT

```
kill -QUIT <postmaster folyamat azonosító>
```

A *SIGQUIT* szignál után a postmaster *SIGQUIT* szignált küld minden futó háttérfolyamatnak, majd azonnal kilép (nem hajt végre egy rendes leállítási folyamatot). A futó folyamatok szintén azonnal kilépnek a *SIGQUIT* szignál után. Ezután a leállítási mód után a szerver a következő elindításkor helyreállítást végez. Csak végszükség esetén használjuk!

A postmaster folyamat elinduláskor eltárolja az adatbázis cluster könyvtárban a *postmaster.pid* fájlban a folyamat azonosítóját. Így a fenti parancsokat ennek a fájlnak a felhasználásával is futtathatjuk. Példa egy normál shutdown-ra:

```
$ kill -TERM `head -1 /var/lib/postgres/data/postmaster.pid`
```

Az adatbázis szervert leállíthatjuk a `pg_ctl` program segítségével is:

```
$ /usr/lib/postgresql/bin/pg_ctl \  
  [-D <cluster könyvtár>] [-m <leállítási mód>]
```

Ahol a "leállítási-mód" lehet: *smart*, *fast*, *immediate*. A *smart* az alapértelmezett. Ha a "cluster könyvtár" nem adjuk meg akkor a *PGDATA* környezeti változóban megadott helyen keresi.

Automatikus indítás és leállítás

Beállíthatjuk azt is, hogy az adatbázis szerver automatikusan elinduljon és leálljon gépünk elindításakor és leállításakor. Ezt az UHU-Linux Vezérlőpult -> Szolgáltatások pontja alatt tehetjük meg. Itt keressük meg a *postgresql* szolgáltatást és állítsuk Igenre.

15.7.4. Munka a Postgresql-el

Felhasználói környezet beállítása

Az adatbázis szervert az elindítása után már használhatjuk saját felhasználói bejelentkezésünk alól. A programok könnyebb elérése és használata miatt érdemes a *PATH* és *PGDATA* környezeti változókat beállítani. Ehhez írjuk be az alábbi sorokat a *./profile* nevű fájlba (Ha a fájl még nem létezik, hozzuk azt létre.):

```
PATH=$PATH:/usr/lib/postgresql/bin  
PGDATA=/var/lib/postgres/data  
export PATH PGDATA
```

Bejelentkezés

Az adatbázis szerverbe a `psql` programmal tudunk bejelentkezni. A bejelentkezéshez meg kell adni az adatbázis és a felhasználó nevét. Kezdetben az adatbázis *template1*, a felhasználó *postgres* (feltéve hogy a cluster létrehozásakor nem adtunk meg mást):

```
$ psql template1 postgres
```

(A Postgresql kliens programok alapértelmezett felhasználó és adatbázisneve megegyezik a programot futtató Linux felhasználó nevével, azaz a saját felhasználói nevünkkel.) Sikeres bejelentkezés esetén kapunk egy adatbázis promptot:

```
template1=#
```

Itt most kétféle parancsot adhatunk meg:

- `psql` belső parancsokat (ezek `\` (backslash) jellel kezdődnek)
- SQL parancsokat

A parancsok listáját lekérhetjük a `\?` (belső parancsok) illetve a `\h` (SQL parancsok) parancsokkal. Az egyes SQL parancsokról részletesebb segítséget kaphatunk a `\h <parancs neve>` formával. A `psql` programból `\q` paranccsal tudunk kilépni. A `psql` program leírását a PostgreSQL dokumentáció "Reference Manual" "Client Application" fejezetében találjuk. Az SQL parancsok leírását pedig a "Reference Manual" "SQL Commands" fejezetében.

Felhasználó létrehozása

Adatbázis felhasználót a *CREATE USER* SQL paranccsal lehet létrehozni. Jelentkezzünk be az adatbázisba, majd hozzuk létre azt az adatbázis felhasználót, amit használni szeretnénk a munkánk során:

```
$ psql template1 postgres
template1=# CREATE USER lajos;
```

(Az SQL parancsokat `;` jellel kell lezárni. Ez azért jó, mert így egyszerre több parancsot is megadhatunk.) Ellenőrzéshez listázzuk ki az adatbázis felhasználókat:

```
template1=# \du
               List of database users
  User name | User ID |           Attributes
-----+-----+-----
   lajos   |    100 |
 postgres |     1 | superuser, create database
(2 rows)
```

Mint látjuk a felhasználó amit létrehoztunk egy normál felhasználó, jogosultságok nélkül. A *CREATE USER* (illetve már létező felhasználónál az *ALTER USER* paranccsal két rendszer jogosultságot tudunk kiosztani:

CREATEDB A felhasználó létrehozhat, törölhet adatbázisokat

CREATEUSER A felhasználó létrehozhat, törölhet, módosíthat felhasználókat (superuser jogosultság).

Jelszót is megadhatunk a felhasználóhoz a *PASSWORD* opcióval. Pl:

```
template1=# ALTER USER lajos WITH PASSWORD 'lajcsika' \
    CREATEDB CREATEUSER ;
template1=# \du
               List of database users
  User name | User ID |           Attributes
-----+-----+-----
    lajos   |    100 | superuser, create database
   postgres |     1 | superuser, create database
(2 rows)
```

(Felhasználókat létrehozhatunk a *createuser* Linux paranccsal is, de ez is a fenti SQL parancsot futtatja.)

Adatbázis létrehozása

Adatbázis létrehozására a *CREATE DATABASE* parancs szolgál. Próbáljuk ki:

```
$ psql template1 lajos
template1=# CREATE DATABASE lajosdb;
ERROR:  CREATE DATABASE: source database "template1" is \
    being accessed by other users
```

A létrehozás nem sikerült, mivel alapértelmezésben a *template1* adatbázist másolná a rendszer, de mi ide jelentkeztünk be. Mivel kezdetben a *template0* megegyezik a *template1* adatbázissal hozzuk létre az adatbázisunkat a *template0* másolásával:

```
CREATE DATABASE lajosdb WITH TEMPLATE=template0;
template1=# \l
               List of databases
   Name      |  Owner   | Encoding
-----+-----+-----
   lajosdb   | lajos    | SQL_ASCII
  template0  | postgres | SQL_ASCII
  template1  | postgres | SQL_ASCII
(3 rows)
```

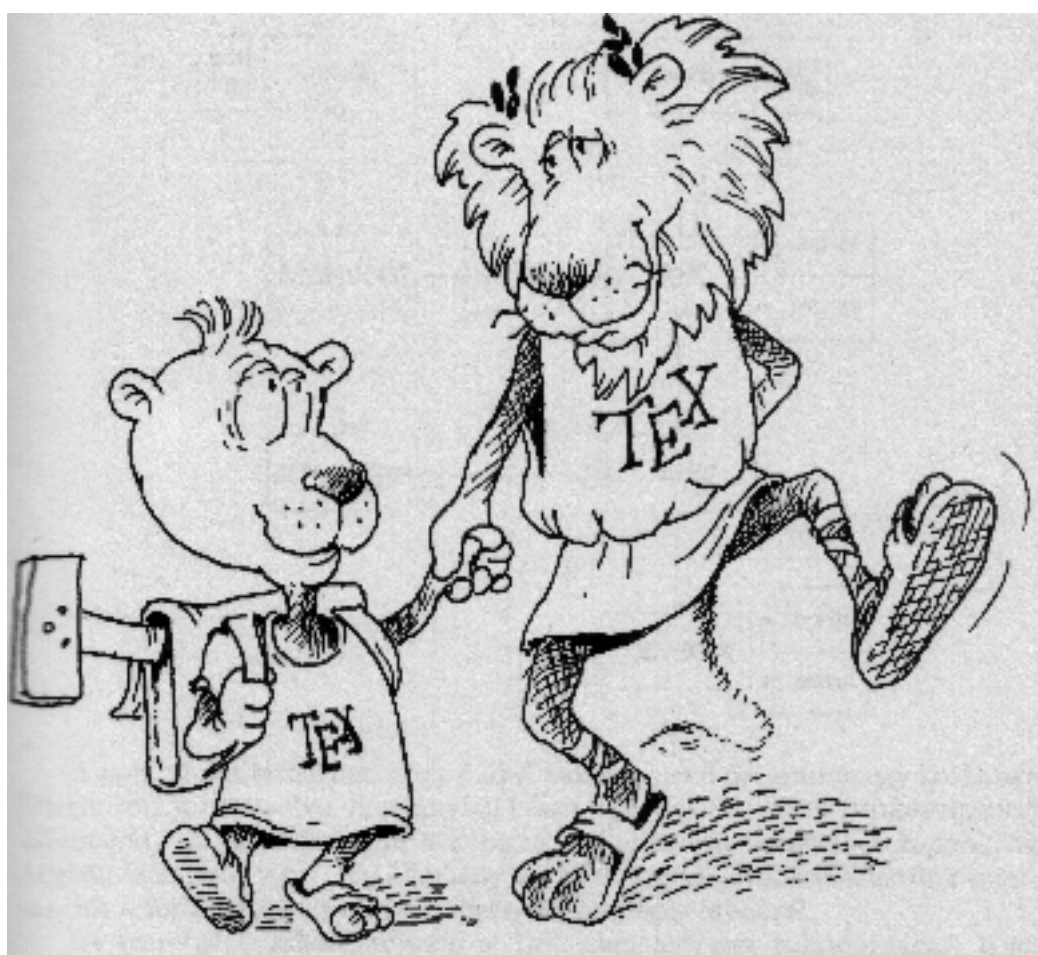
Az *Owner* és *Encoding* értéke szintén megadható a *CREATE DATABASE* parancsnak.

Már van felhasználónk, és adatbázisunk is, így elkezdhetjük a tényleges munkát!

Sajnos az SQL rejtelseinek a bemutatása nem célja e fejezetnek, ezeket az információkat a Postgresql "Tutorial" illetve "User's Guide" dokumentációiban találhatjuk.

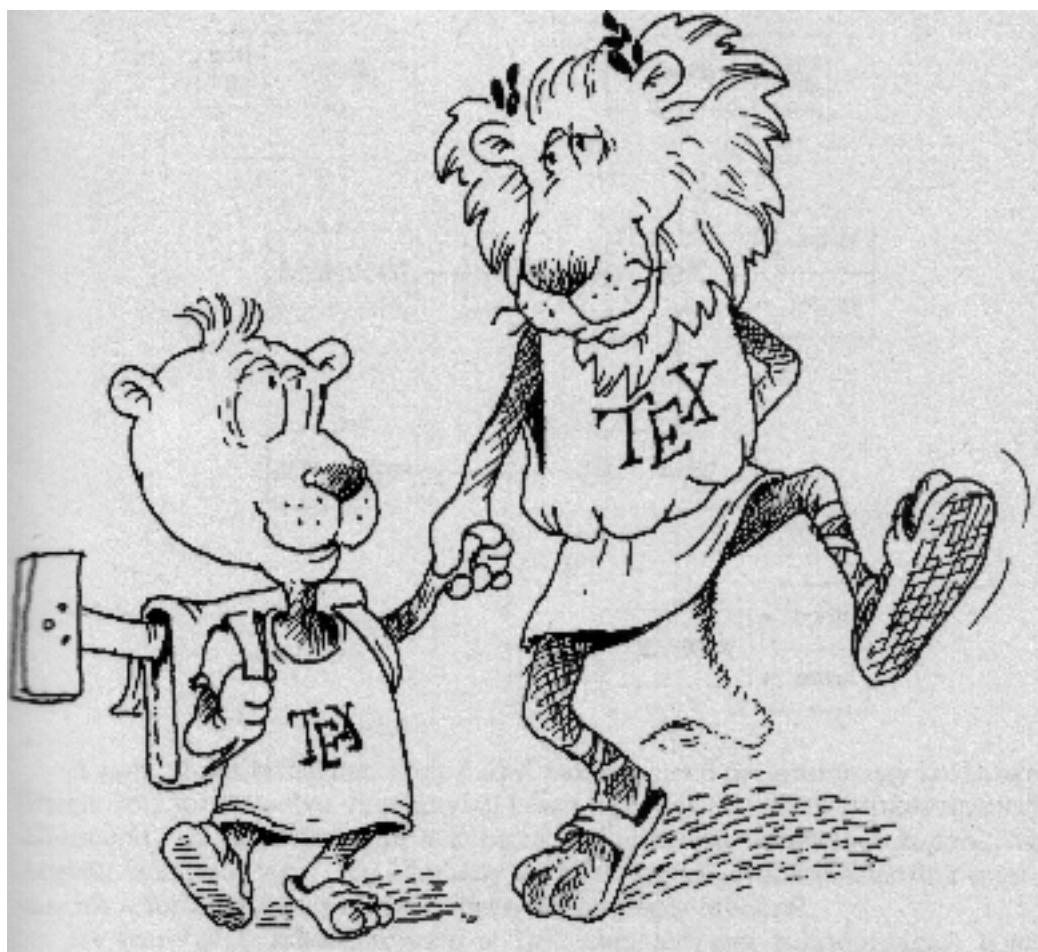
16. fejezet

Tűzfalak



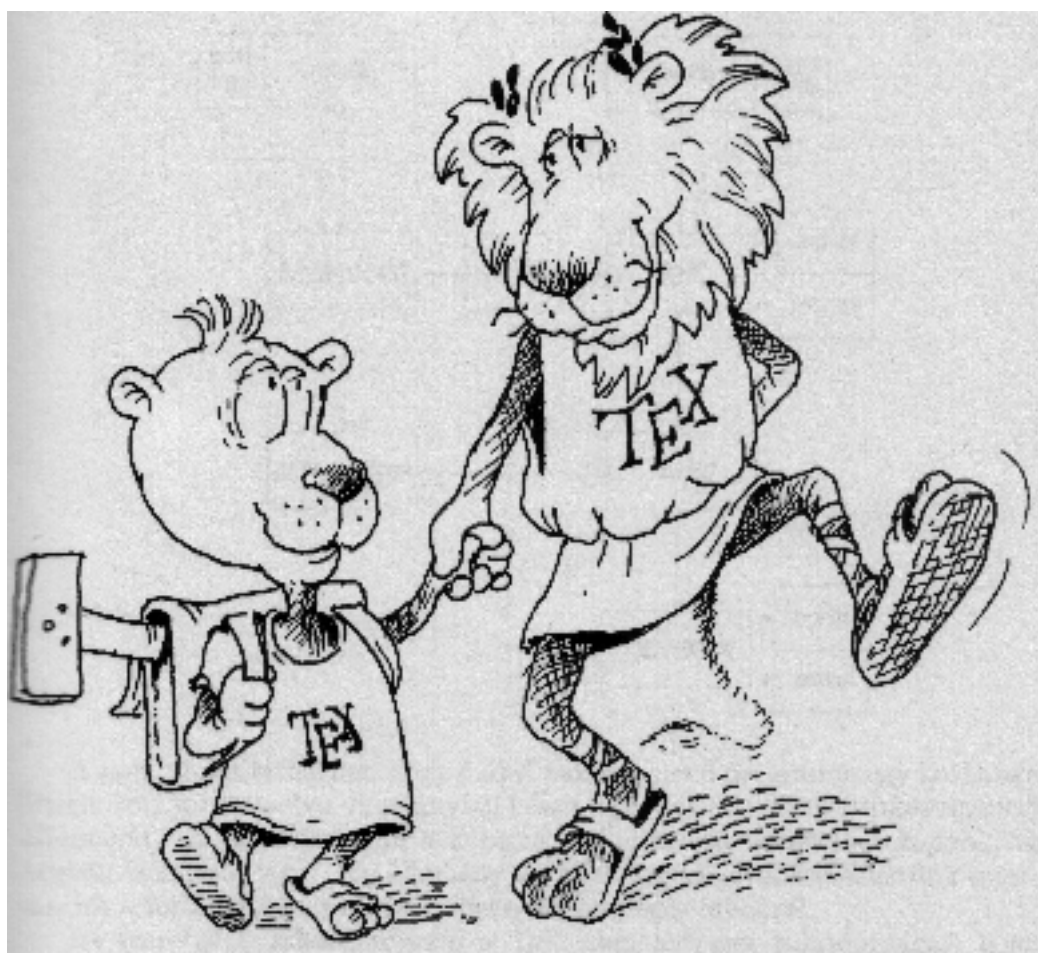
17. fejezet

Az SSH



18. fejezet

Fejlesztői eszközök, ismertetőik



18.1. Bash shell programozás

18.1.1. Bevezetés

Az elkövetkezendőkben a Bash shell scriptek írásának alapjait igyekszünk bemutatni. Némi túlzással talán úgy is utalhatnánk rá, mint shell programozásra, bár nyilvánvalóan a lehetőségek terén, a shell messze elmarad a valódi program nyelvektől, mégis nagyon hatékony eszköz.

Az, hogy milyen mértékben válik hasznossá számunkra, tulajdonképpen rajtunk múlik. Minél mélyebben megismerjük a bash shell-t, valamint minél több parancssoros programot megismerünk és tanulunk meg használni, annál hatékonyabbak lehetünk. Hiszen a shell scriptek egyik jellemzője, hogy a shell-en kívül különféle egyéb programokat hívhatunk meg segítségével. Ismertetőnkkel nem törekszünk a teljességre, hiszen a lehetőségek tárháza szinte végtelen. Célunk csupán megkönnyíteni a kezdeti lépéseket, valamint az egyéb operációs rendszert használóknak bepillantást engedni a Linux lelkivilágába. Nem titkolt célunk az, hogy felkeltsük az érdeklődést, motivációt adjunk, az alapvető parancsok és konzolos programok megismeréséhez. Szeretnénk átvezetni az érdeklődőket a grafikus felület burkán és bátorságot, némi önbizalmat adni a konzol használatához, a konfigurációs fájlok és scriptek átszerkesztéséhez, a programok forrásból történő lefordításához. Azaz egyfajta kontaktust szeretnénk teremteni az olvasó és a Linux valódi énje között.

A fentiek figyelembevételével megpróbálunk az alapoktól indulni, hogy azok is, akik még soha nem programoztak más nyelven (Basic, pascal, C, stb.), magukénak érezhesék az itt leírtakat, de nem kívánunk bonyolult programozástechnikai fogásokkal foglalkozni. A következő részeket érdemes elejétől fogva sorban olvasni, mivel megpróbálunk fokozatosan egyre több feladatot bízni a felhasználóra, feltételezve az előző részek ismeretét, a már megszerzett tudás használatát erőltetni, ezzel is elősegíteni az önálló Linuxos gondolkodás mielőbbi kialakulását.

Ezért, a példákat és a scripteket, érdemes elkészíteni, működésüket tanulmányozni. A részek után csak akkor lépünk tovább, ha az ott szereplő parancsok, scriptek működését megértettük. A későbbi részekben egyre több információ és példa scriptek találhatók, ahol nem ritka, hogy csak egy rövid összefoglaló leírás található a működésről. Aki megfelelő körültekintéssel végig követi az anyagot, annak ez nem okozhat problémát, sőt így könnyebben áttekinthetőek a példák, melyek kimondottan az anyag megértését szolgálják.

Természetesen egy-egy feladatot többféleképpen, más parancsokkal, rövidebb kóddal is meg lehet oldani. Ez is egy nagyszerű lehetőség az önképzésre és fejlődésre. Mivel a bash-script, nem más, mint parancssori utasítások sorozata, ezért néhány egyszerűbb esetben elég csak a konzolon, parancssorba írni a példát. Ezt a módszert - főleg eleinte - fogjuk használni, mivel így egyszerűbben és gyorsabban követhető az anyag. Ezeknél a

példáknál, egy "\$" jel látható a sor elején, mely a promptot jelenti, vagyis azt nem kell a terminálba begépelni. Amelyik sor pedig az előzőleg begévelt parancs eredményét tartalmazza, ott nincs előtte "\$" jel. Ezzel jól meg lehet őket különböztetni. A scripteket bármilyen text fájl kezelő szövegszerkesztővel elő lehet állítani, de erre a célra érdemesebb egyszerű szerkesztőt, vagy direkt erre a célra kifejlesztett editort használni. A legegyszerűbb az mc editorát használni (mcedit). Amikor már a fájl elején megnyitáskor szerepel a "#!/bin/bash" bejegyzés, akkor a "bash shell" szabályainak megfelelően különböző színekkel emeli ki a script egyes elemeit. Természetesen használhatunk grafikus felületű programot is, hiszen a "kwrite" is tudja a bash-nak megfelelően színekkel kiemelni az elemeket. Egy kisebb és gyorsabb lehetőség lehet a "nedit", ami szintén tudja színezní a szöveget a bash szabályinak megfelelően. Érdekes lehetőség még a "kate", mely a "kwrite"-hoz hasonló, de könnyű vele egyszerre több fájl szerkeszténi, kezelni, a képernyőt meg tudja osztani két szerkesztendő fájl között, a fájl csoportokat azaz fájl-listákat is lehet vele menteni, megnyitni. Valamint a "kate"-n belül használhatunk egy parancsértelmezőt, azaz terminált, ami dinamikusan az éppen kiválasztott dokumentum könyvtárára vált. A dokumentum egyes részeinek végén egy-egy "Xdialog"-os példa található.

18.1.2. Mi is azaz Xdialog?

Egy grafikus dialógus ablakokat megjelenítő program, amely a scriptekből hívható meg. Scriptbeli hívásakor megadott kapcsolók, paraméterek, argumentumok határozzák meg a működését és a típusától függően értékeket is képes visszaadni. Aki foglalkozott Windows API-val, Visual Basic-el, Delphi-vel, Kilyx, vagy bármilyen egyéb hasonló vizuális fejlesztőeszközzel, azok grafikus építőelemeire kell gondolni.

Természetesen lényegesen kevesebb lehetőséggel. Az Xdialog-gal, könnyen és gyorsan készíthetünk, "látványos" és felhasználóbarát scripteket. Az alábbi oldalokról lehet letölteni:

<http://xdialog.dyns.net/>
<http://freshmeat.net/projects/xdialog/>

vagy UHU-Linux csomag formátumban:

<http://ubk.uhulinux.hu/>

18.1.3. Bash Shell Programozás 1. "Hello World !"

Mivel a legtöbb új érdeklődő, Windows-os vagy DOS-os rendszereken nőtt fel, először hasonlítsuk a shell scriptet a legtöbbünk által ismert "batch" fájlokhoz. Ezek a Windows-

os vagy még inkább a DOS-os rendszerekből ismert futtatható fájlok, amikből programokat és DOS parancssorozatokat lehet futtatni. A hasonlat kedvéért tekintsük a DOS-t egy shell-nek. Ezeknél a "batch" fájl voltát, a *.bat kiterjesztés jelölte. Viszont, mivel a Linux esetében többféle shell is futhat, ezért fontos, hogy a script-ben meg legyen határozva, milyen shell-re írták. Ezt a fájl első sorában kell jelezni, valahogy így:

```
#!/shell/elérési/utvonala
```

Amikor ez nincs benne, a rendszer alapértelmezett shell-jével próbálja azt futtatni. Ez a legtöbb mai Linux-ban alapból a bash shell. A shell scriptet tartalmazó fájlnál nem kell különleges kiterjesztést használni. Csupán futtathatóvá kell tenni a fájlt.

```
$ chmod +x scriptfajl
```

Közelítsük meg a shell scripteket egy másik hasonlatból. A programkódokat alapvetően kétféleképpen lehet futtatni. Vagy bináris kóddá fordítja azt egy compiler (fordító), vagy futásidőben értelmezi egy interpreter. A modern programozásban, lehetőség van mindkettőre. Már a QBasic tervező programjában is, a program írása közben, magát a kódot is le lehet futtatni és ha jól működik, akkor binárisra is fordítható. Az azóta megjelent fejlesztőeszközökben pedig alapvető funkció ez. A shell scriptek esetében sajnos nincs lehetőség binárisra fordítani. A shell scripteket a shell soronként, parancsonként értelmezi és hajtja végre. Úgy is tekinthetünk a scriptekre, mint konzolon, egymás után begépelte parancsokra. A shell-t héj-programnak is szokták nevezni. Tulajdonképpen ez burkolja be a kernelt. A shell a saját nyelvezetében kapott parancsokat a kernel számára értelmezhető módon küldi tovább. Valamint visszafelé, a kernel üzeneteit továbbítja a felhasználónak, vagy a shell-t használó programnak. A shell scriptből, nem csak shell parancsokat adhatunk ki, innen bármilyen programot futtathatunk is. Sőt, a legtöbb esetben ez történik, ugyanis a "linuxos parancsok" legnagyobb része, önálló program. Legyen az első példánk, "kötelezően" egy "Hello World !" program. Rajta keresztül a bash-script több jellemzőjét is megvizsgáljuk és egy kis Xdialog-os példát is megnézünk. A parancs sorok előtti "\$" jel, a prompt-ot jelképezi, vagyis azt nem kell begépelni. Tehát nyissunk egy terminál ablakot és írjuk bele a lenti parancsokat ! Természetesen a sorok végén enter-t nyomva.

```
$ clear
```

```
$ echo Hello World !
```

Nem nehéz kitalálni mi történt. A "clear" törölte a képernyőt, az "echo" pedig megjelenítette az utána levő szöveget. Most tegyük a "Hello World !" szöveget egy változóba és így írassuk ki a képernyőre. A bash változói, alaphelyzetben szöveges változók. Bár a "declare" paranccsal lehet "integer" (egész numerikus érték) típust és tömb típusút is meghatározni. Az integer típusúba, ha szöveges adatot töltünk fel, akkor az nulla értéket ad vissza. Integer, szám típusú változó létrehozása:

```
$ declare -i $valt  
$ valt="szöveg"  
$ echo $valt 0
```

Tömb típusú létrehozása:

```
$ declare -a$valt[elemszam]
```

Értékadás:

```
$ valt[elemsorszám]="kifejezés"
```

Érték kinyerése:

```
$ echo ${valt[elemsorszám]}
```

A teljes tömbtartalom kiírása:

```
$ echo ${valt[*]}
```

Az értékkel feltöltött elemek számának lekérdezése:

```
$ echo ${#valt[*]}
```

Ez utóbbi dinamikus tömbként viselkedik, vagyis a deklarációnál meghatározott tömb elemszámnál nagyobb sorszámú elemmel is feltölthető. Viszont a többdimenziós tömbök nem támogatottak. Minden típus karakteresen tárolódik. A változót alapesetben nem kell külön létrehozni. Mikor értéket kap, akkor létrejön karakteres típusúként. A létrejövő változó, csak arra a shell-re érvényes, azaz ha közben becsukjuk a terminál ablakot és újat nyitunk, akkor újra értéket kell adni a változóknak. Nézzük meg a gyakorlatban is, a szöveges típusú változók használatát.

```
$ a=Hello World !
```

A fenti próbálkozás az alábbi hibaüzenetet eredményezi:

```
bash: World: command not found
```

Azt mondja, hogy a "World" parancsot nem találja. Azaz, az első szóköz utáni részt, újabb parancsnak értelmezi. Ennek kivédésére, a szöveget tegyük idézőjelek közé.

```
$ a="Hello World !"
$ echo $a
bash: !": event not found
```

Ez ismét hibaüzenettel tért vissza. Igen, mert a !" kettős karaktert a shell különleges esetekre használja. Tegyük egy szóközt közéjük.

```
$ a="Hello World ! "
$ echo $a
Hello World !
```

Most oldjuk meg, hogy a szöveg egy részét, dinamikusan változtathassuk. Ezt a szöveg adott részének változóval történő helyettesítésével érhetjük el.

```
$ b="World"
$ c="City"
$ echo "Hello $b ! "
Hello World !
$ echo "Hello $c ! "
Hello City !
```

Mint láthatjuk, a változó értéke behelyettesítésre került. De mi van, ha ezt nem akarjuk. pl.

```
$ echo "Adj értéket a $b-nak ! "
Adj értéket a World-nak !
```

Mivel a \$b változó értéke a "World" szó, ezért az echo ezt jeleníti meg a helyén. Ha nem akarjuk, hogy a változó behelyettesítésre kerüljön, akkor használjunk egyes-idézőjeleket.

```
$ echo 'Adj értéket a $b-nak ! '
Adj értéket a $b-nak !
```

Ekkor a felkiáltójel utáni szóköz is elhagyható. Az egyes-idézőjelek esetén nem történik semmi különleges dolog a végrehajtásban. De mivel a legtöbb esetre igenis kell a változó-érték behelyettesítése, leginkább a normál idézőjel használatos. Mit tehetünk akkor, ha egy szövegrészen belül, van olyan szakasz, ahol a változók egy részét szeretnénk, hogy az értékével helyettesítse a shell és van ahol konkrétan a változó nevét szeretnénk kiíratni.

```
$ szem="József"
$ echo "Kérlek kedves $szem adj értéket a $b változónak ! "
Kérlek kedves József adj értéket a World változónak !
$ echo 'Kérlek kedves $szem adj értéket a $b változónak ! '
Kérlek kedves $szem adj értéket a $b változónak !
```

Egyik esetben sem a kívánt eredményt kapjuk. Kétféle megoldás is kínálkozik.

```
$ echo "Kérlek kedves $szem1" 'adj értéket a $b változónak !'
Kérlek kedves Jóska adj értéket a $b változónak !
```

Ekkor kétfelé vettük az üzenetünket. Egyik felét kettős, a másik felét szimpla idézőjelekbe tettük. De használhatunk egy másik lehetőséget is, még pedig azt, hogy 1 db backslash, "\" jelet teszünk az elé a változó elé, amelyiket nem szeretnénk az értékével behelyettesíteni. Ez ugyanis az utána következő karakter, ami a shell számára a különleges jelentését hatástalanítja. Úgy is szokták mondani, "levédi" azt.

```
$ echo "Kérlek kedves $szem adj értéket a \$b változónak ! "
Kérlek kedves Jóska adj értéket a $b változónak !
```

Még egy probléma felmerülhet a változók értékének kiírása során, akkor, ha a kiírandó szövegben a változó után nem akarunk szóközt hagyni. Pl. egy férj nevéhez akarjuk a "né" jelzót fűzni:

```
$ nev="Kovács"
$ echo "$nevné"
```

Erre egy üres változót kapunk, mivel a "nevné" nevű változónk üres. A megoldás, hogy a változó nevét jelentő részt és a kiírandó részt a "" jelekkel különítjük el.

```
$ echo "${nev}né"
Kovácsné
```

Bár az eddigi példákat, parancssorba gépelgettük, de ha ezeket egy #!/bin/bash kezdetű fájlba helyezzük, majd erre futási jogot adunk, akkor ugyanezt az eredményt kapjuk. Tehát:

```
$ mcedit hw
```

Ekkor az mc fájlkezelő szövegszerkesztője megnyitja a hw nevű fájlt, vagy ha nem létezik még, akkor mentés során létre hozza. Ebbe begépeljük:

```
#!/bin/bash
a="Hello World ! "
echo $a
```

Majd mentés (F2) után az mcedit-ből kilépve, futási jogot adunk neki:

```
$ chmod +x hw
```

A "hw" parancs kiadásakor, azt a hiba üzenetet kapjuk, hogy nincs ilyen program. Ugyanis a \$PATH globális változóban lévő útvonalakon keresi. Általában: /usr/bin:/usr/local/bin:/usr/X11/bin:/home/user/bin Ha futtatni akarjuk, adjuk meg neki, hogy az aktuális könyvtárban is keresse, a "." karaktereket elé írva:

```
$ ./hw
Hello World !
```

Megoldás lehet, ha a hw elnevezésű fájlunkat áthelyezzük a /home/userneve/bin/ könyvtárba:

```
mv hw bin/
```

Xdialog példa

Most megnézzük hogy az Xdialog programot felhasználva, milyen egyszerűen és gyorsan tehetjük script-jeinket látványossá és felhasználóbaráttá. Ehhez szükséges, hogy az Xdialog program fel legyen telepítve. Az UHU-Linux csomagot itt keressük:

```
http://ubk.uhulinux.hu/
```

Most az Xdialog legegyszerűbb dobozát fogjuk használni, az msgbox-ot. Ez egy ablak, amiben a "kifejezés" szöveg olvasható és egy "OK" gomb van rajta. A két számmal az ablak mérete adható meg. 0 0 esetén automatikusan a szöveghez méretezi azt. A továbbiakért lásd az Xdialog jegyzetet. Ha feltelepítettük az Xdialog-ot akkor egy terminálban írjuk be:

```
$ Xdialog --title "Shell Programozás" --msgbox "Hello World !" 0 0
```

Természetesen nem fog működni, a következő hibával tér vissza:

```
bash: !": event not found
```

Nézzük meg mi a hiba. Az előző részben leírtakat nem jegyeztük meg, így kimaradt egy szóköz !-jel után. Próbáljuk még egyszer:

```
$ Xdialog --title "Shell Programozás" --msgbox "Hello World ! " 0 0
```

Így már működik. Egy másik lehetőség:

```
$ a="Hello World ! "  
$ Xdialog --title "Shell Programozás" --msgbox "$a" 0 0
```

Itt fontos az idézőjelek közé tenni a \$a változót, mert egyébként nem egységes egészként kezeli a kifejezést, hanem így értelmezi:

```
$ Xdialog --title "Shell Programozás" --msgbox Hello World ! 0 0
```

Ekkor csak a Hello szót írná ki. Ez érvényes más shell parancsoknál is.

18.1.4. Változók és argumentumok

Ebben a részben a változók értékadásának különféle módjairól lesz szó, beleértve a felhasználói adatbevitelt valamint egy parancssorozat eredményének változóba helyezését is. A változó="érték" és a változó='érték' típusú értékadásra itt már nem térnek ki, hiszen azt az előző részben megismertük.

let

Már volt róla szó, hogy a shell változók, alapesetben szöveges változók. Viszont a változóban gyakran numerikus, számértéket akarunk tárolni. Ezzel nincs is gond, hiszen:

```
$ a="12"  
$ echo $a  
12
```

Azonban ha az értékekkel, számolni is szeretnénk, nem ilyen egyszerű a dolog.

```
$ b="24"  
$ c=$a+$b  
$ echo $c  
12+24
```


A fenti módszer nem a kívánt eredményt hozta. Mind a változót értékét, mind az operációs jelet, szöveges adatként kezelte. Azt hogy a shell a változó értékét karakteres, vagy numerikus adatként, esetleg dátumként, vagy netán egy fájl neveként értékeli, azt a változó felhasználási módja határozza meg. Tehát a kellő eredmény érdekében meg kell mondanunk a shellnek, hogy hogyan kezelje a változók értékét. Ha azt akarjuk, hogy egy számítást végezzen el, akkor a "let" parancsot kell használnunk.

```
$ let c=$a+$b
$ echo $c
36
```

A következő formában is lehet számolni:

```
c=$(( $a+$b ))
c=$(( $a+$b ))
```

A "let" parancssal, összeadást, kivonást, szorzást és osztást végezhetünk és zárójeleket is használhatunk. De csak egész számokat tud kezelni. Azaz a:

```
$ let d=$c/5
$ echo $d
7
```

helytelen eredményt ad.

A törtszámokhoz, már külső programot kell használni pl. a "bc" programot.

```
$ echo $c/5 | bc -l
7.20000000000000000000
```

Bár meghatározható integer típusú változó, de ez csak annyit jelent, hogy ha szöveges stringgel töltjük fel az így deklarált változót, akkor annak az értéke 0 lesz. Számolni sajnos így sem lehet velük, csak a fent mutatott technikákkal.

```
$ declare -i szam
$ szam="szöveg"
$ echo $szam
0
```

Előfordulhat, hogy egy változó hosszára vagyunk kíváncsiak. Erre megfelelő a "wc" program használata, amely visszaadja, hogy a bemenetére érkező adat hány sorból, szóból és karakterből áll.

```
$ nev="Kovács Gáspár"
$ hossz=`echo $nev | wc -c`
$ echo "$nev neve $hossz karakterből áll a szóközzel együtt."
Kovács Gáspár neve 14 karakterből áll a szóközzel együtt.
```

read

Igen gyakori az is, hogy a felhasználótól szeretnénk, adatot bekérni. Erre a shell a "read" utasítást használja. Ekkor egy várakozó, kurzort kapunk. Az adatbevitel során, a kurzor-mozgató billentyűk is használhatóak. Az adatbevitelt az "enter" leütésével fejezhetjük be. Ekkor a shell folytatja a további utasítások végrehajtását. Feltételezve, hogy a read után pl. az "uhulinux" szót gépeljük be:

```
$read d
$ echo $d
uhulinux
```

Egyszerre több változóba is bekérhető adat. Ekkor a bevitel során a szóközöknél darabolja a begépelte szöveget. Azaz az első szóközig az első változóba, az első és második szóköz közötti részt a második változóba, a második és harmadik szóköz közötti részt a harmadik változóba helyezi, és így tovább. Pl. tételezzük fel, hogy a bevitel során a három adatot, nevet, lakóhelyet és e-mail címet akarunk bevinni.

```
$read a b c
$ echo $a
Gábor
$ echo $b
Kecskemét
$ echo $c
glindorf@mailbox.hu
```

Felmerül a kérdés, hogy miképpen kezelhetők helyesen a szóközöket tartalmazó kifejezések. Például, ha a névnél a vezetéknév és a keresztnév is meg akarjuk adni. Az eredmény ez lesz:

```
$ echo $a
Raffai
$ echo $b
Gábor
$ echo $c
Kecskemét glindorf@mailbox.hu
```

Látjuk, hogy amíg van újabb meghatározott változó a read sorban, addig minden szóköz után újba helyezi, a maradékot pedig elhelyezi az utolsóba. Hogyan lehet ezt kivédeni? Ezt a már ismert backslash "karakterrel tehetjük meg. Ez hatástalanítja a mögötte lévő vezérlő karakter különleges jelentését a shell számára. Így a szóközét is. A fenti példánál

maradva a következőképpen begépelve a read során, a helyes eredményt kapjuk: "Raffai Gábor Kecskemét glindorf@mailbox.hu" A linux parancsok során a szóközökre, és más különleges jelentésű karakterekre oda kell figyelni és a "megoldással, valamint az idézőjel típusok helyes alkalmazásával szabályozni kell, hogy a shell hogyan értelmezze. A kezdetekben ez egy gyakori és bosszantó hibaforrás lehet.

Az adat bekérése előtt gyakran tájékoztatni kell a felhasználót, milyen adatot is kérünk tőle. Ehhez egy echo parancsot használhatunk előtte.

```
$ echo "Írd be a neved, a várost ahol laksz és az e-mailcímed: "  
$ read adatok  
$ echo $adatok  
Gábor Kecskemét glindorf@mailbox.hu
```

Ilyenkor mint láttuk, a kurzor az adat-bekéréshez az echo parancs kiírása alatti sorba kerül. Ha azt szeretnénk, hogy a kurzor a tájékoztató szöveg után, vele egy sorba kerüljön, használjuk az echo parancsot "-n" kapcsolóval.

```
$ echo -n "Írd be a neved, a várost ahol laksz és az e-mailcímed: "  
$ read adatok  
$ echo $adatok  
Gábor Kecskemét glindorf@mailbox.hu
```

Előfordul, hogy egy fájl tartalmát szeretnénk egy változóba tenni. Vagy legalábbis fájlból szeretnénk a változónak értéket adni. Ehhez a "cat" parancsot használhatjuk. Tegyük fel, hogy fenti adatainkat, az adat.txt fájl tartalmazza.

```
$ cat adat.txt  
Gábor Kecskemét glindorf@mailbox.hu
```

A cat parancs ebben a formájában a képernyőre írja az adott fájl tartamát. Változóba, a balra dőlő szimpla idézőjelek segítségével tehetjük.

```
$ adatok=`cat adat.txt`  
$ echo $adatok  
Gábor Kecskemét glindorf@mailbox.hu
```

A balra dőlő idézőjelek esetén, (amelyet általában a magyar billentyűzeten az AltGr + 7 kombinációval érhetünk el,) a két idézőjel közötti részt, mint parancsot végrehajtja a shell és a kapott eredményt helyezi a változóba. Ez akár egy hosszú parancssorozat is lehet. Az olyan eset is gyakori, hogy egy-egy változó értékét, a script indításakor, argumentum formájában akarjuk megadni. Ezekre a scripten belül, speciális változókkal hivatkozhatunk. Nézzük meg egy példán keresztül, hogy működik ez. Hozzunk létre egy "probal" nevű fájlt:

```
$ mcedit probal
```

Töltsük fel az alábbi tartalommal:

```
#!/bin/bash
# 2.fej.1.script

clear
echo "Darab : $# "
echo "Név : $1 "
echo "Cím : $2 "
echo "E-mail : $3 "
```

Mentés után, adjunk neki futási jogot:

```
$ chmod +x probal
```

Ezután futtassuk három argumentummal:

```
$ ./probal Gábor Kecskemét glindorf@mailbox.hu
Darab : 3
Név : Gábor
Cím : Kecskemét
E-mail : glindorf@mailbox.hu
```

Mint látjuk a \$# különleges változó, a script futtatásakor megadott argumentumok darabszámát tartalmazza. Megemlítenék még három különleges változót:

\$0 A script nevét tartalmazza, pontosabban azt, amelyen néven meg lett hívva.

\$* Az összes parancssori argumentumot tartalmazza egyben, egyetlen egységként kezelve.

\$@ Az összes parancssori argumentumot tartalmazza, de külön-külön egységként kezelve.

Ezekén kívül a scriptekből elérhetőek a globális shell változók, mint a \$HOME a \$USER, a \$PATH és a többi.

Kicsit vizsgálódjunk még, a script milyen módon veszi át a parancssori argumentumokat. A scripten kívül létezik nekünk már az \$adatok nevű változó, a "Gábor Kecskemét glindorf@mailbox.hu" tartalommal. Adjuk át argumentumként ezt a változót a scriptnek, így:

```
$ ./probal $adatok
```

Ugyanazt az eredményt kapjuk, mert a script az \$adatok változó tartalmát, külön-külön argumentumként kezeli a szóközőknél darabolva. De most a változót tegyük idézőjelbe.

```
$ ./probal "$adatok"
Darab : 1
Név : Gábor Kecskemét glindorf@mailbox.hu
Cím :
E-mail :
```

Most a script, egyetlen argumentumként kezeli a változó tartalmát, mintha a szóközőket "="el levédtük volna. A teljesség kedvéért, próbáljuk még ki, szimpla idézőjellel is.

```
$ ./probal '$adatok'
```

A balra dőlő idézőjelet is megnézhetjük:

```
$ ./probal `cat adat.txt`
```

Gondolom a fentebbi sorok fényében ezeket már nem kell külön elmagyarázni.

shift

Felmerülhet az igény, hogy az átadásra kerülő argumentumokat sorban feldolgozzuk. Ezt a shift paranccsal tehetjük meg, mely eggyel balra lépteti az argumentumok értékét. De nézzük meg egy példán keresztül. probal nevű scriptünket, egészítsük ki az alábbi sorokkal.

```
#!/bin/bash
# 2.fej.2.script

clear
echo "Darab : $# "
echo "Név : $1 "
echo "Cím : $2 "
echo "E-mail : $3 "
echo "\$1:$1 \$2:$2 \$3:$3 "
shift
echo "\$1:$1 \$2:$2 "
shift
echo "\$1:$1 "
```

Mint láthatjuk ez sorban kiírja a kapott három argumentumot. Az első shift után a \$1-et, azaz a "Gábor"-t eldobja és a helyébe lép a második argumentum, azaz a "Kecskemét". A második helyre, (\$2) pedig feljött a harmadik argumentum, az emailcím. Ekkor már nincs \$3. A második shift után, a jelenlegi \$1-et, azaz a "Kecskemét"-et eldobja és a második helyről az e-mailcím feljön a \$1-be. A \$2 pedig megszűnik.

Xdialog példa. Befejezésül nézzük meg, miként kérhetünk be adatot egy változóba az Xdialog-gal.

```
$ adatok=`Xdialog --stdout --title "Shell Programozás" \
--inputbox "Írd be a neved, a várost ahol laksz és az \
e-mailcímed: " 0 0`
```

Valamivel elegánsabb így:

```
$ adatok=`Xdialog --stdout --title "Shell Programozás" \
--3inputbox "Írd be a személyes adataidat : " 0 0 \
"Neved: " " " "Városod: " " " "E-mail címed: " " "`
```

Ebben az esetben:

```
echo $adatok
Gábor;Kecskemét Március 15.u.;glindorf@mailbox.hu
```

Az Xdialog a visszatérési értékeit, még ha az több elemből is áll, mindig egyetlen "sor"-ban adja vissza, egy szeparátorral elválasztva őket. Ez alap esetben a "/". Más is meghatározható a --separator kapcsolóval.

Ajánlott a script műveleteivel nem ütköző szeparátort választani. Sajnos szeparátornak csak egyetlen karaktert fogad el.

```
$ adatok=`Xdialog --stdout --title "Shell Programozás" \
--separator ";" --3inputbox "Írd be a személyes adataidat : \
" 0 0 "Néved: " " " "Városod: " " " "E-mail címed: " " "`
```

Ebben az esetben:

```
echo $adatok
Gábor;Kecskemét Március 15.u.;glindorf@mailbox.hu
```

Ezután, ha szükséges, az egyes elemeket a cut programmal tehetjük külön-külön változóba, de erről még később lesz szó. Egy másik érdekes lehetőség az Xdialog inputbox-ainál, a jelszó szerű bevitel lehetősége. A -password kapcsoló használatakor, beviteli mezbe történ írás során, a bevitt betűk helyén, csak "*" karakterek látszanak. A kapcsoló egyszeri használata az utolsó beviteli mezőre érvényes, ha kétszer adjuk meg, akkor az utolsó kettőre, ha háromszor, akkor mindháromra.

Használható az -inputbox, az -2inputbox és a -3inputbox esetében. A dobozon megjelenik egy választó kapcsoló amellyel ki és be kapcsolható a "*" effektus, azaz a karakter rejtés. A részletekért és a jobb megértés végett, érdemes megnézni a Jegyzetek: Xdialog részt.

18.1.5. Szokásos be és kimenet, adatfolyamszerkesztés

Ebben a részben megismerkedünk, néhány a Windowsos és DOS-os világban nem ismert fogalommal, a szokásos bemenet, a szokásos kimenet, a szokásos hiba kimenet és a pipeline, azaz adatfolyam szerkesztés lehetőségével. Alaphelyzetben egy program szokásos bemenete, a billentyűzet, azaz onnan várja a utasításokat. A szokásos kimenet pedig a képernyő, ott jeleníti meg az eredményeket. A szokásos hibakimenet is a képernyő, a program ide küldi a hibaüzeneteit. Ezeket azért nevezzük "szokásosnak", mert ezek az alapértelmezettek, de ha a célunk úgy kívánja, megváltoztathatóak, átirányíthatóak. Nézzünk néhány lehetséges megvalósítást.

Az echo parancsot már ismerjük. Ez egy adott karaktersorozatot, vagy egy változó tartalmát írja ki a képernyőre. Most már fogalmazhatunk pontosabban úgy is, hogy a szokásos kimenetre küldi azt. De mi van, ha a változó tartalmát, nem a képernyőn szeretnénk megjeleníteni, hanem egy fájlba szeretnénk íratni. Ekkor a ">" jellel irányíthatjuk át a szokásos kimenetet, a fájlba. Használjuk az előző részben említett adat.txt nevű fájlt, helyezzük a már ismert módon, az "adatok" nevű változóba (ehhez természetesen abban a könyvtárban kell állnunk, ahol a fájl található). Majd a változó értékét, írassuk egy fájlba az echo parancs és a szokásos kimenet átirányításával.

```
$ adatok='cat adat.txt'
$ echo "$adatok" > uj-adat.txt
```

Ez ha már létezik az uj-adat.txt fájl, akkor felülírja azt, vagyis az előző tartalma elvész és beleírja az \$adatok változó értékét. Amennyiben nem ezt szeretnénk, hanem a fájlhoz hozzáfűzni új adatokat, akkor a ">>" jelet kell használnunk. Hozzunk létre egy "ujadat" nevű változót, töltsük azt fel és illesszük hozzá a uj-adat.txt fájlhoz, majd ellenőrizzük le az eredményt.

```
$ ujadat="András Budapest andras@freemail.hu"
$ echo $ujadat >> uj-adat.txt
$ cat uj-adat.txt
Glindorf Kecskemét glindorf@mailbox.hu András Budapest
andras@freemail.hu
```

grep

Nem csak fájlba lehet irányítani egy program kimenetét, hanem egy másik program bemenetére is. Erre a "|" jel szolgál. Ezt hívják adatfolyam szerkesztésnek. Ismerjünk meg itt egy új programot, a "grep"-et. Ez a bemenetére érkező sorokból, kiszűri azokat, amelyek tartalmazzák a megadott kifejezést és továbbküldi azt a szokásos kimenetén keresztül. De előbb vegyünk fel még egy személy adatait az uj-adat.txt fájlba, ellenőrizzük le sikerült-e, majd a cat programmal irányítsuk a fájl tartalmát a grep program bemenetére, azzal szűrjük meg a budapesti lakosokat és az eredményt jelenítsük meg a less programmal.

```
$ echo "Péter Baja peter@mailbox.hu" >> uj-adat.txt
$ cat uj-adat.txt
$ cat uj-adat.txt | grep Budapest | less
```

A cat program kimenetét a grep bemenetére küldtük, majd annak a kimenetét a less program bemenetére. A less pedig az eredményt a szokásos kimenetén, a képernyőn jelenítette meg. A less egy szöveges-fájl nézegető. A megtekintés állapotából, a "q" billentyűvel léphetünk ki (quit).

Egy program bemenetére, a "<" jellel is irányítható egy fájl tartalma. A fenti eredmény a következő módon is elérhető.

```
$ grep Budapest < uj-adat.txt | less
```

Amivel még nem foglalkoztunk, az a szokásos hibakimenet és annak átirányítása. A programok hibakimenete alaphelyzetben ugyanaz, mint a szokásos kimenet, azaz a képernyő. Tegyük fel, hogy nemlétező fájlt akarunk megnyitni a cat paranccsal.

```
$ cat személyes-adatok
cat: személyes-adatok: Nem létező fájl vagy könyvtár
```

Elfordulhat olyan, hogy a hibaüzeneteket, egy fájlba szeretnénk irányítani. A hibakimenetre ebben az esetben, a "2>" módon hivatkozhatunk. A normál kimenetre ez alapján így is lehet hivatkozni "1>". De ez azonos a ">" hivatkozással. Úgy is nevezhetjük őket,

hogya egyes és kettes kimenet. Vagy normál kimenet és hibakimenet. A szokásos bemenetet pedig egyszerűen csak bemenetnek. Lássunk egy példát arra, ha a hibaüzeneteket egy "hibák" nevű fájlba szeretnénk gyűjteni. Ekkor a meglévő "hibák" nevű fájl, felülíródik. Ha ezt el akarjuk kerülni és mintegy gyűjteni bele a hibaüzeneteket, akkor a már ismert módon tehetjük meg azt.

```
$ cat személyes-adatok 2>> hibák
```

Amikor a hibakimenetet a /dev/null -ba irányítjuk, ezzel nem íródik ki sehová sem.

```
$ cat személyes-adatok 2> /dev/null
```

Lehetőség van arra is, hogy a normál kimenetet és a hibakimenetet egyszerre kezeljük. Az első példa a normál kimenetet egy fájlba, a hibakimenetet pedig egy másik fájlba irányítja. A második példánál, pedig a hibákat egy fájlba, a normál kimenetet pedig egy következő, esetünkben a less program bemenetére adja tovább.

```
$ cat személyes-adatok > normál 2>> hibák  
$ cat személyes-adatok 2>> hibák | less
```

Az is előfordulhat, hogy egy kimenetet fájlba is szeretnénk irányítani és egy másik helyre is átirányítani. Erre a tee program használható. A következő sor az "eredmeny" nevű fájlba írja a normál kimenetet, de a less programmal is megjeleníti.

```
$ cat uj-adat.txt | grep Budapest | tee eredmeny | less
```

Amikor a tee utáni részt szabadon hagyjuk, akkor a fájlba is ír és a képernyőn is megjelenít.

```
$ cat uj-adat.txt | grep Budapest | tee eredmeny
```

Lehet két fájlba is íratni a kimenetet és a hibakimenetet pedig előtte egy harmadik fájlhoz hozzáfűzetni. Illetve ezeket lehet kombinálni.

```
$ cat uj-adat.txt 2>> hibafile | tee eredmeny > eredmeny2
```

Néha pedig arra lehet szükségünk, hogy a hibaüzenet is a normál kimenetre menjen. Erre is van lehetőség, hogy az adott kimenetet a másik kimenetre irányítsuk. Az általunk írt scriptek esetén ugyanígy használhatóak a szokásos ki és bemenetek átirányításai. Csúpan két dologgal kell tisztában lennünk. Egyik, hogy az echo parancs mindig a normál kimenetre ír, vagyis, ha a scriptünk futtatásakor annak normál kimenetét átirányítjuk, akkor a scripten belüli összes echo parancs arra a helyre ír. Az alábbi esetben a script összes echo parancsa a fájlba íródik.

```
$ sajatscript > fajl
```

A másik lehetőség a `read` parancs, ahol ehhez hasonlóan mindig a szokásos bemenetről olvas. Ami alaphelyzetben a billentyűzet. Ezért tudunk vele adatot bevinni. Visont ha a scriptünk bemenetére irányítunk valamit, akkor a scripten belüli összes `read` parancs onnan olvas. Amikor nincs ott több adat, akkor üres, 0 hosszúságú stringet olvas be. A lenti esetben a scriptbeli `read`-ek mind az adatfájl továbbító `cat` parancs kimenetéről várja az adatot.

```
$ cat adatfile | sajatscript
```

Természetesen ezek a szabályok eléggé megkötik a kezünket egy interaktív script be és kimenetének kezelése terén. Erre is nagyszerű gyógymódot kínál az `Xdialog` program, ha a dialógust az `echo` és a `read` parancsok helyett ezzel vezéreljük le. Ebben az esetben a kimenet-átírányítások szabadon használhatóak. A scriptünkben a hibakimenetre is tudunk írni. A normál kimenetet `"1>"`, a hibakimenetre `"&2"` irányítjuk. Az ilyen `echo`, a hibakimenetre ír.

```
$ echo "Hiba szöveg" 1>&2
```

Valamint mint minden programnak, scriptünknek is van egy visszatérési értéke, ami a `$?` változóban érhető el, közvetlenül azután, hogy a script befejezte futását. Az, hogy ennek az értéke mennyi legyen, a scripten belül az `exit <érték>` vagy `exit $x` utasítással határozhatjuk meg. De ez az érték, csak egy 0 és 255 közötti szám lehet.

18.1.6. Reguláris kifejezések, metakarakterek

Most ismerkedjünk meg a Windowsból bár ismert, de itt mégis másképpen működő dologgal, a shell által ismert reguláris kifejezésekkel.

Nevezhetjük őket metakaraktereknek is, vagy egyszerűen helyettesítő karaktereknek. Windows-ban és DOS-ban a `"*"` és a `"?"` karakterek ismertek. Linuxban több lehetőség is rendelkezésünkre áll. A `"$"` jellel a sor elejére, a `"^"` jellel pedig a sor végére hivatkozunk. A `"."` jellel, egy darab tetszőleges karakterre, (mint DOS és Windows alatt a `"?"` jel,) A `"*"` jellel pedig a megszokott módon, tetszőleges számú, tetszőleges karakterre hivatkozunk. A `"["` és `"]"` jellel, a karakterek egy csoportjára tudunk hivatkozni. A `[0-9]` bármilyen szám karaktert jelent. Az `[A-Z]` bármilyen nagybetűs karaktert jelent. Az `[skZ4o]` bármelyik karaktert jelenti a felsoroltak közül. Érdemes még megemlítenünk, hogy kifejezéseken belül a TAB-ra a `"\t"` a sorvégre pedig a `"\n"` módon hivatkozhatunk. Fontos megjegyezni, hogy egy-két program egy-egy reguláris kifejezést másképpen használ, vagy másképpen jelöl. Ezért ha valami nem jól működik, akkor nézzük át

az adott program man-ját. Ilyen pl. a grep program "*" metakarakter értelmezése, mely jelentése a shellben, "bármennyi számú, bármilyen karakter". A grep-ben pedig, "bármennyi számú, a "*" jel előtt álló karakter". Azaz az "a*" jelentése a shell-ben: egy "a" karakterrel kezdődő bármilyen string. Míg ugyanennek a jelentése grep-ben: Az "a" karakter bármennyiszer-i ismétlődése. Pl. "a" "aaa" vagy "aaaaaa". Mivel a grep a "." metakaraktert ugyanúgy értelmezi mint a shell, azaz egy darab bármilyen karakter, ezért a shell beli "*" reguláris kifejezés, grepes megfelelője a ".*", azaz a "bármilyen karakter bármennyiszer-i ismétlődése".

18.1.7. A kód formázása

Mivel rövidesen már valódi scripteket fogunk írni, ezért ismerkedjünk meg azokkal a lehetőségekkel, amikkel olvashatóbbá és áttekinthetőbbé tehetők scriptjeink. Az egyik alapvető dolog, hogy a logikailag összetartozó részeket megfelelő számú üres sorral választjuk el egymástól. Ez a függőleges tagolás. A másik, a vízszintes tagolás, ahol a programvégrehajtás során más-más mélységben lévő sorokat, bentebb kezdjük mint az előzőt. Ezzel az elágazások és ciklusok tehetők áttekinthetőbbé.

A harmadik dolog a kommentek használata. A sorban a "#" jel utáni rész, már nem hajtódik végre. Ha ez az első karakter, akkor értelemszerűen az egész sor kimarad a végrehajtásból.

```
read a; echo $a # a további szövegrész nem hajtódik végre.
# read a; echo "$a" ez a sor egyáltalán nem hajtódik végre.
```

Már ismerjük a "\" jelet és hatását. Ezzel a sor végén lévő sorvégjel is hatástalanítható, vagyis ha a sor végén egy "\" szerepel, akkor a shell úgy veszi, hogy nem történt újsor kezdése, vagyis a következő sort is az előző folytatásaként hajtja végre. Ebben az esetben fontos, hogy a "\" jel után már nem állhatnak karakterek, még szóközök sem, illetve abból egy igen. Ebből ered, hogy ha az ilyen eltört sorokban kommenteket akarunk elhelyezni, akkor a "\" és a "#" között, maximum egy space lehet. Ez akkor is hasznos lehet, ha olyan hosszú parancssorokat használunk, amik esetleg nem férnek ki egy sorba (mármint nem a terminálba, mert ott ez nem okoz gondot, hanem a dokumentumba, ahol a példák szerepelnek). A következő parancssor,

```
$ cat file.txt | grep Budapest | cut -d\; -f 2-4 | \
tr '[A-Z]' '[a-z]' ujfile.txt
```

alábbi módon történő leírása ugyanazt jelenti:

```
$ cat file.txt | \
```

```
grep Budapest |\n\ncut -d\\; -f 2-4 |\ntr '[A-Z]' '[a-z]'\n> ujfile.txt
```

Ennek az ellenkezője is megoldható. Több parancs is írható egyetlen sorba, a ";" jellel elválasztva.

```
$ echo -n "Mi a neve ?"\n$ read nev\n$ echo "Üdvözlöm $nev."
```

Ezt írhatjuk egyetlen sorba is.

```
$ echo -n "Mi a neve ?" ; read nev ; echo "Üdvözlöm $nev."
```

18.1.8. Adatfolyam szűrők és szerkesztők

Ebben a részben néhány egyszerűbb szöveg és szöveges fájl manipuláló lehetőséget ismerünk meg. Ezekre mint szűrőkre is szoktak hivatkozni. Mivel a programok egyébként csak a legszükségesebb mértékben vannak ismertetve, érdemes az élesben történő használatuk előtt a man-t elolvasni, bár ez nem feltétele annak, hogy az itt leírtak megérthetőek legyenek.

Hozzunk létre egy fájl az alábbi tartalommal, adatok.dat néven.

```
Sorszám;Név;Születési hely;Születési id;Anyja neve;Gyermekkori leányneve;Lakhely;Nem;Családi állapot;Gyermekek száma\n1;Kovács Gábor;Budapest;1973 Január 14;Mardin Éva;;Budapest;férfi;házas;2\n2;Bárdos Péter;Szombathely;1965 Március 24;Kele Katalin;;Pécs;férfi;független;0\n3;Szeghalminé Éva;Budapest;1957 Február 05;Péteri Eszter;Konkoly Éva;Cegléd;n;házas;1\n4;Almosné Szabó Renáta;Kecskemét;1976 December 20;Ostoros Sára;Szabó Renáta;Budapest;n;független;1\n5;Izsó Péter;Zánka;1953 November 09;Kvári Piroksa;;Budapest;férfi;elvált;3
```

Itt csak a legegyszerűbb szűrés és karakter cserélési lehetőségekkel fogunk megismerkedni.

grep, cut

A grep programot már ismerjük. Ez is egy szűrő, amely a bemenetére érkező sorok közül csak azokat küldi tovább, amelyek a megadott feltételnek megfelelnek. Listázzuk ki a "Budapest" szót tartalmazó sorokat.

```
$ cat adatok.dat | grep Budapest
1;Kovács Gábor;1973 Január 14;Mardin Éva;;Budapest;férfi;házas;2
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly Éva;Cegléd;n;házas;1
4;Almosné Szabó Renáta;1976 December 20;Ostoros Sára;Szabó Renáta;Budapest;n;független;1
5;Izsó Péter;1953
```

```
Notombelemdb='echo $tomb | tr ';' '\n' | wc -l | tr -d ' ' i="0"
until test $i -eq $tomb
do
let i=$i+1
echo $tomb | cut -d';' -f $i
done
```

A `grep -v` kapcsolójával, megfordíthatjuk a feltételt, azaz csak azok a sorok listázódnak, amelyek nem felelnek meg a feltételnek.

```
$ cat adatok.dat | grep -v Budapest
0;Név;Születési id;Anyja neve;Gyermekkori leányneve;Lakhely;Nem;Családi állapot;Gyermekek száma
2;Bárdos Péter;1965 Március 24;Kele Katalin;;Pécs;férfi;független;0
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly Éva;Cegléd;n;házas;1
```

A `grep` alaphelyzetben mint a `Linux` is, kis és nagybetű érzékeny. Ezt a `grep`-nél az `-i` kapcsolóval felfüggeszthetjük.

```
$ cat adatok.dat | grep péter
```

Így nincs találat.

```
$ cat adatok.dat | grep -i péter
2;Bárdos Péter;1965 Március 24;Kele Katalin;;Pécs;férfi;független;0
5;Izsó Péter;1953 November 09;Kövári Piroska;;Budapest;férfi;elvált;3
```

Így viszont azonosnak veszi a kis és a nagybetűket. A `-w` kapcsolóval, csak a teljes szavakra történő illeszkedést veszi megfigyelésnek. A `-x` kapcsolóval lehetőség van egész sort is megfigyelni. Tegyük fel, hogy az egy gyermekes nők nevére van szükségünk. Ekkor az eredményt tovább kell küldeni több szűrésen is.

```
$ cat adatok.dat | grep ':n;' | grep 1
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly Éva;Cegléd;n;házas;1
4;Almosné Szabó Renáta;1976 December 20;Ostoros Sára;Szabó Renáta;Budapest;n;független;1
```

Az adatokat, akármennyi szűrésen átküldhetjük. De mint említettük, nekünk csak az érintettek nevére van szükségünk. Erre a `"cut"` programot fogjuk használni. Ebben az adatfájlban, az adatmezők szeparátoraként a `;"` karakter szerepel. Ezt közölnünk kell a `cut`-tal a `-d` kapcsolóval. Ezután meg kell adnunk, hogy az adatmezők közül, hányadikra van szükségünk. Ezt a `-f` kapcsolóval tehetjük meg. Ne felejtjük levédeni a `;"` karaktert, mert különben nem tudja majd a shell értelmezni a parancssort, ugyanis a jel után következő részt, újabb parancssornak értelmezi.

```
$ cat adatok.dat | grep ';' | grep "\;1$" | cut -d\; -f2
Szeghalminé Éva
Almosné Szabó Renáta
```

Tételezzük fel, hogy csak az egyik egyén adataira lenne szükség. Hogyan lehetne egyszerűen megcsinálni, hogy választani lehessen a felsoroltak közül és annak az egynek megjeleníteni az összes adatát? Ehhez ne csak a nevet, hanem az első adatoszlopot, az index mezőt is jelenítsük meg (ami a mi egyszerű esetünkben egyben az adatsor sorszámát is jelöli). Ezután kérjünk egy választást a felhasználótól és az alapján az adott személy összes adatát jelenítsük meg. Ez nem is olyan bonyolult, mint gondolnánk.

```
$ cat adatok.dat | grep ';' | grep "\;1$" | cut -d\; -f 1-2
3;Szeghalminé Éva
4;Almosné Szabó Renáta

$ echo -n "Hányas sorszámú személy adatai kellenek ?"
$ read v
$ cat adatok.dat | grep "^$v\;"
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly Éva;Cegléd;n;házas;1
```

Amennyiben a bemenet nem egy, hanem több sorból áll, akkor minden sor adott mezője tovább adódik. Így egy adatfájlból egész mezők vihetők át egy másik fájlba. Tegyük is egy próbát, bár ne küldjük az eredményt fájlba.

```
$ cat adatok.dat | cut -d\; -f 2-4
Név;Születési id;Anyja neve
Kovács Gábor;1973 Január 14;Mardin Éva
Bárdos Péter;1965 Március 24;Kele Katalin
Szeghalminé Éva;1957 Február 05;Péteri Eszter
Almosné Szabó Renáta;1976 December 20;Ostoros Sára
Izsó Péter;1953 November 09;Kövári Piroska
```

Nem csak egy mezőt, hanem többször lekérhetünk. A "-f 1,3,4" vagy a "-f 24" módon. Az előbbi esetben az 1-es, 3-as és 4-es mezők, utóbbi esetben pedig a 2-től a 4. mezők kerülnek továbbadásra. A cut programmal nem csak adatmezőket választhatunk ki, hanem megadott pozíciójú karaktereket is. Bár ennek ebben a példában nincs értelme, de ha az eredményből, tegyük fel, csak a 6-8-dik karakterekre lenne szükségünk, ez is megoldható.

```
$ cat adatok.dat | grep ';' | grep "\;1$" | cut -d\; -f2 | cut -c 3-7
eghal
mosné
```

Mivel kétsoros volt az utolsó cut bemenete, ezért mindkét soron végrehajtotta. A több soros bemenet szétbontása is megoldható később bemutatandó technikákkal.

A "-c 2,4,7" módszerrel nem egymás melletti karakterek is megadhatók.

sort, join

Ezeket az adatfájlokat időnként érdemes karbantartani. Esetleg egy másik adatbázis-fájlt szeretnénk már egy meglévőből készíteni, de más elrendezéssel. Ezekhez alkalmas lehetőségeket fogunk most megvizsgálni.

Egy szöveges adatbázisfájl rendezéséhez a `sort` parancs használható. Tegyük fel, hogy az `adatok.dat` fájlunkat szeretnénk rendezni a nevek szerint abc sorrendbe. Ezt az alábbi módon tehetjük meg.

```
$ cat adatok.dat | sort -t ';' +1 > adatok-rendezett.dat
```

A `-t` kapcsoló után kell megadni a fájlban a mezőhatároló karaktert. a `"+"` után van megadva, hogy hányadik mező szerint rendezzen. Az első mező, a nulladik mező. Vagyis, ha a 3. mező szerint akarunk rendezni, akkor `" +2"`-t kell írni. Ezt a `-k` kapcsolóval lehet elkerülni (ez az első adatsort is, ami a mező neveket tartalmazza, belekeveri a valódi adatsorok közé. A probléma orvosolható, de most nem térünk ki rá). További lehetőségek a finomításra a következő kapcsolók:

- k szám Ha ezzel, nem pedig a `+szám` kapcsolóval hivatkozunk a mezőre, akkor az első mező sorszáma nem 0, hanem 1.

- d Rendezésnél figyelmen kívül hagyja a különleges karaktereket és írásjeleket.

- f Különbséget tesz a kis és nagybetűk között (tehát alaphelyzetben nem tesz különbséget).

- n Numerikus adatok szerint rendez. Ez a többjegyű számoknál lényeges.

- M Az első három karaktert, hónapok neveként értelmezi és rendezi.

- r Fordított rendezési sorrendet eredményez.

- u Törli a teljesen megegyező sorokat.

Láttuk, miként lehet a `cut` programmal egy adatfájlból, teljes mezőket kinyerni, vagy másik fájlba helyezni. Most nézzük, miképpen lehet két adatfájlt egyesíteni. Erre a `join` parancs alkalmas. Fontos, hogy a két adatbázisnak azonos kulcs szerint kell rendezettnnek lennie. Előtte mindkét adatbázist az egyik mező szerint rendezzük. Ha az egyik adatbázis neveket és címeket tartalmaz, a másik pedig neveket és telefonszámokat, akkor rendezzük mindkettőt a neveket tartalmazó mező szerint. Az egyesítés során, a mindkét fájl által tartalmazott neveknel, a sor már három mezőből fog állni (név, cím, telefonszám).

```
$ join -t ';' nev-cím.txt nev-tel.txt > egyesült.txt
```

A `-t` kapcsolóval a mezőhatároló karaktert adjuk meg. Ha nem az első mező alapján történt a rendezés és történik az egyesítés, azt a `-j` kapcsolóval adhatjuk meg. A `-j` kapcsoló

után közvetlenül 1-es vagy 2-es szám kell szerepeljen. Ebből a kívánt működés érdekében 1-est írunk. Ezután kell megadni, hányadik mező szerint történjen az egyesítés. Feltételezve, hogy a fenti két fájl három mezőből áll és a második a név mező akkor így kell végrehajtani.

```
$ join -t ';' -j1 2 nev-cím.txt nev-tel.txt > egyesült.txt
```

tr, sed

Most nézzünk olyan szűrőket, amikkel karakterek cserélhetők fel, vagy távolíthatók el. A legegyszerűbb a "tr" program. Előbb használati módját, majd néhány egyszerű példát mutatok.

```
$ b='echo "$a" | tr "cdk" "rht"'
```

Ez a sor, a \$a változó értékében, a "c" karaktert "r"-re, a "d"-t "h"-ra, a "k"-t pedig "t"-re cseréli, majd az új karaktersort a \$b változóba írja. A -d kapcsolóval törli a szövegből a megadott, ebben az esetben a "c" és "d" karaktereket.

```
$ b='echo "$a" | tr -d "cd"'
```

Az alábbi módon a nagybetűket lehet kisbetűkre cserélni.

```
$ b='echo $a | tr '[:upper:]' '[:lower:]'
```

Így pedig egy fájlban lehet a tab és sorvég karaktereket átalakítani. Ez az egyik fájlt a másikba másolja, de közben a TAB-okat ";" karakterre cseréli (általában a szöveges adatbázisok, az adatmezők elválasztására, e kettő közül valamelyiket szokták használni).

```
$ cat fajl | tr '\t' ';' > masikfajl
```

Így pedig a sorvégjelek távolíthatók el egy fájlból.

```
$ cat fajl | tr -d '\n' > masikfajl
```

Ez pedig a ";" mezőhatároló karaktereket cseréli sorvégjelre, így a mezőket külön-külön sorba helyezi. Később szükségünk lesz erre a megoldásra.

```
$ cat fajl | tr ';' '\n' > masikfajl
```


A -s kapcsolóval az ismétlődések távolíthatók el. Az alábbi sor a felesleges space-eket távolítja el:

```
$ echo "Kovács      Béla      Budapest." | tr -s ' '
Kovács Béla Budapest.
```

Ugyanezt az eredményt adja a tr -s '[:space:]' is.

A -s kapcsoló, az egymás után következő, az argumentumban megadott típusú karaktert von össze, egyetlen egybe. A tr 1-1 karaktert tud kezelni. Amennyiben több karakterből álló karaktersorozatot szeretnénk egy másik, akár a lecserélendővel nem is azonos hosszúságú karaktersorozatra cserélni, ehhez a "sed" program használható. A reguláris kifejezések természetesen a sed-nél is használhatók. A sed egy komoly program, rengeteg lehetőséggel. Külön könyvet is lehetne róla írni, mi most csupán néhány példát mutatunk.

A bemenetére érkező adatsorban, az össze kif1 karaktersort, a kif2 karaktersorra cseréli:

```
sed s/kif1/kif2/g
```

Csak a sor elején szereplő kifejezéseket cseréli le:

```
sed s/^kif1/kif2/g
```

Csak a sorvégi kifejezéseket cseréli ki:

```
sed s/kif1$/kif2/g
```

Minden sor elejére az adott kifejezést szúrja be:

```
sed s/"^"/"kif"/g
```

Hogyan lehet egy fájl, vagy egy bemeneten érkező adatfolyam kívánt sorszámu sorát megkapni:

```
sed -n "$sorszam p" `
```

Természetesen az adatbáziskezelés sokkal összetettebb problémákat vet fel és a megoldásukhoz is összetettebb megoldások szükségesek. Az itt bemutatott példák csak azt a célt szolgálják, hogy az alapvető lehetőségek bemutatásra kerüljenek.

Xdialog példa. Illő módon fejezzük be ezt a részt is egy Xdialog-os példával.

Válasszuk a timebox-ot, mely három spin-t, azaz olyan kis beviteli lehetőséget tartalmaz, amiben numerikus értékek állíthatók be. Kezdeti értéke az aktuális idő, óra, perc és másodperc értéke. Nyugodtan állíthatjuk, ez nem változtatja meg a gépünk időbeállítását. Arra külön kellene a kódban root jogosultsággal utasítani. A doboz visszatérési értéke a beállított idő, ":"-tal elválasztva az óra a perc és a másodperc. Ezt cut-tal külön-külön változókba tesszük és egy infobox-szal kiíratjuk, amit 5 másodpercre jelenítünk meg és letiltjuk az OK nyomógomb megjelenését, valamint az ablak bezárhatóságát is. A következő kódot helyezzük egy futási joggal rendelkező, #!/bin/bash kezdetű fájlba.

```
#!/bin/bash
# 4. fejezet 1. script

ido='Xdialog --stdout --title "Shell Programozás" --timebox \
"Ön szerint mennyi a pontos id ?" 0 0`
ora='echo $ido | cut -d: -f 1`
perc='echo $ido | cut -d: -f 2`
mp='echo $ido | cut -d: -f 3`
Xdialog --no-buttons --no-close --title "Shell Programozás " \
--infobox "Ön szerint, $ora óra,\n$perc perc, és $mp másodperc van ." 0 0 5000
```

A timebox esetében a `--stdout` biztosítja, hogy az eredmény ne a hibakimenetre, hanem a normál kimenetre menjen. Az infobox esetében pedig a `--no-buttons` kapcsoló letiltja a gombok megjelenését, a `--no-close` pedig lehetetlenné teszi az ablak bezárását. Az infobox által megjelenített szövegben a "\n" karakterek utasítják sortörésre a szöveg megjelenítésekor. A legvégén található harmadik szám pedig azt határozza meg, mennyi idő múlva záródjon be magától az ablak, ezredmásodpercben kifejezve.

18.1.9. Feltétel vizsgálat, feltételes vezérlési szerkezetek

Ebben a részben az egész scriptünk struktúráját és működését, leginkább meghatározó technikákkal, az elágazásokkal és ciklusokkal ismerkedünk meg. Ezek nagy része, különféle feltételvizsgálatokra épül.

test

Egy feltétel vizsgálatára, hogy az igaz-e, a `test` parancsot használjuk. Nem önmagában, hanem egy elágazás vagy ciklus szerves részeként használjuk. Néhány egyszerűbb vizsgálat következik.

Fájlok, könyvtárak vizsgálata..

test -d file Igaz ha a file létezik és könyvtár.

test -e file Igaz ha a file létezik.

test -f file Igaz ha a file létezik és szabályos fájl.

test -r file Igaz ha a file létezik és olvasható.

test -w file Igaz ha a file létezik és írható.

test -x file Igaz ha a file létezik és végrehajtható.

Szöveges kifejezések vizsgálata.

test -z string Igaz ha a string 0 hosszúságú.

test -n string Igaz ha a string nem 0 hosszúságú.

test string1 = string2 Igaz ha a stringek megegyeznek.

test string1 != string2 Igaz ha a stringek nem egyeznek meg.

Numerikus értékek vizsgálata.

test numkif1 OP numkif2 A következő operátorokat (OP) ismeri.

- eq egyenlő
- ne nem egyenlő
- lt kisebb mint
- le kisebb vagy egyenlő
- gt nagyobb mint
- ge nagyobb, vagy egyenlő

A numerikus kifejezés negatív is lehet, de egész számnak kell lennie.

A test ismer egy speciális numerikus kifejezést, mely a string hosszát jelenti.

-l szöveg

Egy 5 karakteres szövegrész esetén az alábbi vizsgálat igazként értékelődik ki.

```
test 5 -eq -l kif
```

A vizsgált feltétel eredményét negálja az elé írt "!" jel. Azaz, ha igaz, akkor hamissá, ha pedig hamis, akkor igazzá fordítja. Tehát hamis lesz a test ! 5 -eq 5 és igaz a test ! 2 -eq 5

Több feltétel vizsgálata is összekapcsolható a -a (and) "és" illetve a -o (or) "vagy" logikai operátorokkal.

```
test $nev = "Gábor" -a $kor -eq 30
```

Ez akkor bizonyul igaznak, ha a \$nev értéke "Gábor" és a \$kor értéke pedig 30.

```
test $kor -le 17 -o $kor -ge 61
```

Ez akkor igaz, ha a \$kor értéke, a 18 és 60 közötti intervallumon kívül esik.

If ; then ; else; fi. Az elágazásokat, feltételes vezérlési szerkezetként is szokták említeni. Ebből kettőt vizsgálunk meg. Az if elágazás szerkezete a következőképpen néz ki.

```
if test <feltétel>
then
<utasítások>
else
<utasítások>
fi
```

A then és az else közötti utasítások akkor hajtódnak végre, ha igaz a kifejezés, az else utániak pedig akkor, ha hamis.

Az if elágazások egymásba is ágyazhatók.

```
if test <feltétel1>
then
<utasítások1>
if test <feltétel2>
then
<utasítások2>
else
<utasítások3>
fi
else
```

```

<utasítások4>
if test <feltétel3>
then
<utasítások5>
else
<utasítások6>
fi
fi

```

Ez a szerkezet, ha a feltétel1 igaz, akkor végrehajtja az utasítások1-et, majd megvizsgálja a feltétel2-t és ha az is igaz, akkor végrehajtja az utasítások2-t, ha nem igaz akkor az utasítás3-at. Ha a feltétel1 nem igaz, akkor végrehajtja az else utáni részt, azaz az utasítások4-et, majd megvizsgálja a feltétel3-at, ha az igaz, akkor végrehajtja az utasítások5-öt, ha nem igaz akkor az utasítások6-ot. Az utasítássor TAB-okkal történő rendezése nem véletlen, a könnyebb átláthatóságot szolgálja. A strukturálisan egy szintbe, mélységbe tartozó szakaszok előtt, azonos számú TAB van. Vizsgáljunk meg néhány konkrét példát is.

Hozzunk létre egy `#!/bin/bash` kezdetű fájlt és adjunk neki futási jogot.

A további példákat ebbe írjuk és az eredményt a script futásakor tekinthetjük meg. Töltsük fel az alábbi tartalommal (itt már a magyarázatok egy részét kommentekben adjuk meg).

```

#!/bin/bash # 5.fej. 1.script

clear
# képernyő törlés

# Megszámoljuk, hány adatsor van az adatbázisban,
# töröljük az eredményből a számolást bezavaró space-eket,
# kivonunk belőle egyet, mert az első, a 0-dik, ami a mezők
# neveit tartalmazza.

let db=`cat adatok.dat | wc -l | tr -d ' ' \`-1
# Bekérjük, hányas sorszámú személy legyen feldolgozva.

echo -n "Adja meg a feldolgozandó személy sorszámát (1-$db) : "
read a
# Le ellenőrizzük megfelelő-e a sorszám amit a felhasználó adott.

# test akkor igaz, ha a felhasználó által adott szám, kisebb 1-nél,
# vagy nagyobb az adatbázisban lévő személyek számánál.
# A then és a fi közötti rész csak ekkor hajtódik végre.

if test $a -lt 1 -o $a -gt $db

```

```
then
# A script tájékoztatja, a felhasználót, hogy rossz számot adott meg,
# vár egy enter leütéséig, majd kilép a scriptből.
echo "Ön érvénytelen adatot adott meg ! "
echo "A script a futását befejezi. Kérem nyomjon entert."
read x
exit
# az enter leütéséig felfüggeszti a script futását
# azonnal kilép a scriptből
fi

# A megadott sorszámú személy teljes adatsorának bekérése.
# "^$a\;" jelentése: ha a sor elején található a $a értéke,
# utána pedig egy ";" jel következik. Azaz ha az első mező
# értéke azonos $a-val.

szem='cat adatok.dat | grep "^$a\;"`
# Adatmezőinek külön-külön változóba helyezése.

nev='echo $szem | cut -d\; -f2`
szulido='echo $szem | cut -d\; -f3`
anyneve='echo $szem | cut -d\; -f4`
leanyneve='echo $szem | cut -d\; -f5`
lakhely='echo $szem | cut -d\; -f6`
nem='echo $szem | cut -d\; -f7`
csalall='echo $szem | cut -d\; -f8`
gyermesz='echo $szem | cut -d\; -f9`
# a date megadja a mai dátumot és időt, a cut-tal kinyerjük
# belőle az évszámot.

datum='date | cut -d'.' -f1`

# a $szulido-ből kivesszük a születési évet.

szulev='echo $szulido | cut -d' ' -f1`
let kor=$datum-$szulido
# kiszámoljuk az alany életkorát.

# Megvizsgáljuk, az alany házas-e, ha igen akkor a then és az else,
# ha nem, akkor az else és a fi közötti rész hajtódik végre.

if test $csalall="házas"
then
# Ez mindenképpen kiíródik:
echo -n "$nev, $kor éves, $lakhely-i lakos, házasságban él, "
# Ha a gyermekek száma 0 akkor a then,
# egyébként az else utáni rész hajtódik végre.
if test $gyermesz -eq 0
then
```

```

echo "gyermektelen $nem."
else
echo "$gyermesz gyermekes $nem."
fi
else
# Ez mindenképpen kiíródik:
echo -n "$nev, $kor éves, $lakhely-i $nem."

# Ha a gyermekek száma 0 akkor a then,
# egyébként az else utáni rész hajtódik végre.
if test $gyermek -eq 0 then
echo "Családi állapota $csalall, nincs gyermeke."
else
echo "Családi állapota $csalall, $gyersz gyermeke van."
fi
fi

```

Újdonságok.

clear Törli a képernyőt.

tr -d ' ' A tr -d kapcsolója esetén elég csak egy kifejezést megadni, mert az abban megadott részeket törli. Esetünkben a szóközöket.

read x Felfüggesztjük a script futását, amíg a felhasználó kényelmesen el nem olvassa az információt, majd le nem nyomja az enter billentyűt. A read x-et használjuk, vagyis kérjük egy változó feltöltését a felhasználotól, ennek csak az a célja, hogy az enter megnyomásáig felfüggeszük a futást.

exit Hatására azonnal befejezi futását a script.

Case; esac. Most egy újabb elágazás típussal fogunk megismerkedni a case-el. Szerkezete a következő:

```

case $változó in
érték1)
<utasítások1>
;;
érték2)
<utasítások2>
;;
érték3)
<utasítások3>
;;

```

```
érték4)
<utasítások4>
;;
*)
<utasítások5>
;;
esac
```

Bár a fenti feltételes végrehajtások, if elágazásokkal is megoldhatók lennének, mint látható, ez sokkal áttekinthetőbb és elegánsabb. Fontos, hogy az utasításokat két ";" jellel, azaz ";;" módon zárni. Ez egy break-et vált ki, azaz a további vizsgálatokon nem megy végig a program, hanem kilép a case-ből. Ez gyorsítja a vizsgálatot, ha csak nem éppen az a célunk, hogy a további eset is kiértékelődjön. Az utolsó, *) meghatározás utáni <utasítások5> akkor hajtódik végre, ha egyik előtte lévő feltétel sem teljesült. Tulajdonképpen hasonlít a if else ágához. A case segítségével, könnyen létrehozhatunk menü szerű működést is a scriptben. Nézzük meg a case működését egy ilyen példán keresztül.

Hozzunk létre egy fájlt, másoljuk az alábbi tartalmat bele, majd adjunk rá futási jogot.

```
#!/bin/bash # 5.fej. 2.script

clear

# Kiírjuk a főmenüt.
echo echo "Kérem válasszon a menüpontok előtti szám bevitelével."
echo "-----"
echo "1.Fájl"
echo "2.Szerkesztés"
echo "3.Eszközök"
echo "4.Beállítások"
echo "5.Kilépés"
read v

# Bekérjük az első választást.
# Az első válasz vizsgálata:

case $v in
# a külső case eleje. (főmenü)

1)
# Amennyiben a főmenü 1. pontját választotta.
clear

# Kiírjuk, az első almenüt.
echo "Kérem válasszon a menüpontok előtti szám bevitelével."
```



```
echo "-----"
echo "1.Megnyitás"
echo "2.Mentés"
read v

# Bekérjük a második választást.
# Az első almenü válaszának vizsgálata.
case $v in
# Az első, belső case. (1.almenü)
1)
# Amennyiben a főmenü 1. pontjának, 1. almenüjét választotta.
echo "Ön egy már meglévő dokumentumot akar kinyitni."
;;
2)
# Amennyiben a főmenü 1. pontjának, 2. almenüjét választotta.
echo "Ön menteni szeretné a dokumentumot."
;;

esac
;;

2)
# Amennyiben a főmenü 2. pontját választotta.
clear

# Kiírjuk, a második almenüt.
echo "Kérem válasszon a menüpontok előtti szám bevitelével."
echo "-----"
echo "1.Másolás"
echo "2.Kivágás"
echo "3.Beillesztés"
read v

# Bekérjük a második választást.
# Az második almenü válaszának vizsgálata.
case $v in
# A második, belső case. (2.almenü)
1)
# Amennyiben a főmenü 2. pontjának, 1. almenüjét választotta.
echo "Ön kivágott egy részt."
;;
2)
# Amennyiben a főmenü 2. pontjának, 2. almenüjét választotta.
echo "Ön kimásolt egy részt."
;;
3)
# Amennyiben a főmenü 2. pontjának, 3. almenüjét választotta.
echo "Ön egy részt illesztett be."
;;
;;
```

```

esac
;;
3)
# Amennyiben a főmenü 3. pontját választotta.
echo "Ön az eszközöket választotta."
;;
4)
# Amennyiben a főmenü 4. pontját választotta.
echo "Ön a beállításokat választotta."
;;
5)
# Amennyiben a főmenü 5. pontját választotta.
echo "Vége..."
sleep 3
exit
;;
esac
# a külső case vége. (főmenü)

```

A példából hiányzik, a felhasználói válaszok helyességének vizsgálata, de most csak a case megismerése a cél. Bár a válaszok, csak egykarakteresek, azért a case elágazások jól követhetők. Az első két case elágazásnál, további menüpontokat adtunk meg és az azokra történő válasz alapján egy újabb belső case vizsgálat indult mindkét esetben. Ezzel a módszerrel, illetve ennek finomításával egyszerű menüvel láthatjuk el a scriptünket. Nézzük meg az Xdialog-gal, hogyan lehet menüt csinálni.

Ugyanezt a fenti menüt készítsük el. Az előzőekhez képest csak annyi a különbség, hogy echo és read helyett dialógus-ablakokat használunk. Az Xdialog jegyzetből nézzük át a –yesno a –fselect és a –menubox dialógus-ablakok leírását.

```

#!/bin/bash # 5.fej. 3.script

clear
until test
do

valasz=`Xdialog --stdout --title "Shell Programozás 5/3 script"
--backtitle "Menü Próba" --no-cancel --no-tags --item-help --menubox
"Válasszon a menüből." 0 0 6 "1" "Fájl" "Ez a Fájl menüpont Help-je."
"2" "Szerkesztés" "Ez a Szerkesztés menüpont Help-je." "3" "Eszközök"
"Ez az Eszközök menüpont Help-je." "4" "Beállítások" "Ez a Beállítások
menüpont Help-je." "5" "Kilépés" "Vigyázat, kilépés !" ``

case $valasz in
1) until test
do

valasz=`Xdialog --stdout --title "Shell Programozás 5/3 script"
--backtitle "Menü Próba" --no-cancel --no-tags --item-help --menubox
"Válasszon a menüből." 0 0 5 "1" "Megnyitás" "Ez a Megnyitás menüpont
Help-je." "2" "Mentés" "Ez a Mentés menüpont Help-je." "3" "Vissza..."
"Visszalépés egy menüvel fentebb." ``

```

```
case $valasz in
1)

fajl=`Xdialog --stdout --title "Shell Programozás 5/3
script" --backtitle "Ön egy már meglévő dokumentumot akar kinyitni."
--no-buttons --help "Kár bármit is megnyitni\nezz csak egy test script."
--fselect "" 0 0`

Xdialog --title "Shell Programozás 5/3 script" --msgbox
"Most a $fajl fájl megnyitása történné, ha ez egy éles program lenne." 0
0
;;
2)
fajl=`Xdialog --title "Shell Programozás 5/3 script"
--backtitle "Ön dokumentumot akar elmenteni\nAdja meg
hová és milyen néven." --help "Ez csak egy test
script\nValójában nem ment semmit." --fselect "" 0 0`

Xdialog --title "Shell Programozás 5/3 script" --msgbox
"Most a fájl, $fajl néven történ mentése következne, ha
ez egy éles program lenne." 0 0
;;

3)
break
;;
esac
done
;;
2)

until test do valasz=`Xdialog --stdout --title "Shell
Programozás 5/3 script" --backtitle "Menü Próba" --no-cancel
--no-tags --item-help --menubox "Válasszon a menüből." 0 0 5 "1"
"Kivágás" "Ez a Kivágás menüpont Help-je." "2" "Másolás" "Ez a
Másolás menüpont Help-je." "3" "Beillesztés" "Ez a Beillesztés
menüpont Helpje." "4" "Vissza..." "Visszalépés egy menüvel
fentebb."`

case $valasz in
1)
Xdialog --title "Shell Programozás 5/3 script" --msgbox
"Ön kivágott egy részt." 0 0
;;
2)
# Amennyiben a főmenü 2. pontjának, 2. almenüjét választotta.
Xdialog --title "Shell Programozás 5/3 script" \
--msgbox "Ön kimásolt egy részt." 0 0
;;
3)

# Amennyiben a főmenü 2. pontjának, 3. almenüjét választotta.

Xdialog --title "Shell Programozás 5/3 script" \
--msgbox "Ön egy részt illesztett be." 0 0
;;
4)
;;
esac
done
break
```

```
;;
3)

Xdialog --title "Shell Programozás 5/3 script" \
--msgbox "Ön az Eszközök Menüpontot választotta." 0 0
;;
4)

Xdialog --title "Shell Programozás 5/3 script" \
--msgbox "Ön az Beállítások Menüpontot választotta." 0 0
;;

5)

Xdialog --title "Shell Programozás 5/3 script" \
--yesno "Kilép ?" 0 0
v=$?
if test $v -eq 0
then
exit
fi
;;
esac
done
```

Itt, a tényleges műveleteket végző sorokat, a case-eken belül kellene elhelyezni. Amikor ezek sok sorból állnak az egész kód nehezen átláthatóvá válik. Erre, de nem csak erre használható a scripten belüli saját függvények lehetősége, melyet a 7. részben tárgyalunk.

18.1.10. Ciklusok

while/until ; test ; do ; done

Következő lehetőségünk, amivel a scriptünk struktúráját hatékonyra tehetjük, a ciklusok használata. Többször előfordul, hogy ugyanazt a műveletet, sokszor kell végrehajtani, vagy több változón, illetve fájlon. Erre a while/until és a for ciklusokkal lehet megoldást találni. Szintaxisa:

```
while test <feltétel>
do
<utasítások>
done
```

A feltétel vizsgálatra kerül, ha igaz akkor a do és a done közötti utasítások végrehajtnak, majd a shell visszatér a feltétel vizsgálatához, ha az igaz, ismét végrehajtja az utasítás sort, mindezt egészen addig, amíg a feltétel igaz.

Ha hamissá válik, akkor a done után folytatódik a script.

Másik változata az until, addig folytatja a ciklust, míg a feltétel hamis és amikor igazzá válik, akkor lép ki a ciklusból.

```
until test <feltétel>
do
<utasítások>
done
```

A két megoldás ugyanazt eredményezi.

```
while test <feltétel>
until test ! <feltétel>
```

Gyakran előfordul, hogy egy adott szakaszt, megadott alkalmanként akarjuk egymás után futtatni úgy, hogy közben nyomon követhető legyen, éppen hányadik alkalommal fut le.

Más nyelveknél, erre a for ciklust használják, de az a shell-ben, mint alább majd látjuk egészen másképpen működik. Helyette, egy while vagy until ciklust használhatunk, egy változóval, aminek az értékét minden ciklusban 1-el növeljük, így az tölti be a ciklusszámláló szerepét. A ciklusfutás megszakításának feltételében pedig meghatározhatjuk, hogy ha ez a számláló elér egy értéket, a ciklus akkor fejezze be a futását.

Erre általában az \$i változót használjuk, több egymásba ágyazott ciklus esetén pedig \$i1, \$i2 stb. Példaképpen nézzünk egy egyszerű esetet amely a begépelt szöveget megfordítja és így írja azt ki.

```
#!/bin/bash
# 6.fejezet. 1.script

clear

read szoveg
# Megfordítandó szöveg bekérése

hossz=`echo $szoveg | wc -c`
# A szöveg hosszának lekérdezése.
# Annyiszor ismétlődik a ciklus, ahány karakterből a szöveg áll.

i="0"
# A ciklusszámláló nullázása.

until test $hossz -eq $i
do
let i=$i+1
```

```
# ciklusszámláló 1-el növelése.

k='echo $szoveg | cut -c $i'
# A ciklus előre haladtával sorba kivesz 1-1 karaktert a szövegből.

fordszov="$k$fordszov"
# A fordított szöveget összeilleszti karakterenként a ciklusok során.
done

echo $fordszov
# A fordított szöveg kiírása.
```

for ; do ; done

A for ciklus a shell-ben egészen másképpen működik, mint BASIC-ben és egyéb programnyelvekben szokásos. Általában a for ciklusban megadott érték szerint hajtódik végre. A while/until ciklus esetén a megszakítás egy megadott feltételhez kötődik. A shell for ciklusa, a ciklusnak átadott értékeken egyenként végigmegy, majd kilép a ciklusból.

Szintaxisa:

```
for in <kifejezés>
do
<utasítások>
done
```

Figyeljük meg a működésbeli különbségeket:

```
$ atadott="Ez meg az és még ez is."
$ for atvett in Ez meg az és még ez is. ; do echo $atvett ; done
$ for atvett in "Ez meg az és még ez is." ; do echo $atvett ; done
$ for atvett in $atadott ; do echo $atvett ; done
$ for atvett in "$atadott" ; do echo $atvett ; done
```

A fenti utasítássor, scripten belül így nézne ki:

```
for atvett in "$atadott"
do
echo $atadott
done
```

Vagy:

```
for atvett in "$atadott"
do
echo $atadott
done
```

Érdekes, hogy scriptben a "do" után a parancs lehet új sorban, míg ha egy sorban írjuk le az egészet, akkor a "do" és a következő parancs között, nem lehet újsor karakter, vagyis egysoros parancs esetén ";" jel. Ez igaz a többi parancsra is, amikben a "do" utasítás szerepel, valamint az if ; then ; fi esetén is. (if ; then ...), (while ; do ...), stb.

Miután a fenti négy variációt végigpróbáltuk, a for ciklusra, a következőket figyelhettük meg.

Az 1. esetben, az "in" után adott értékek, szóközönként elválasztva külön-külön érték-ként vannak értelmezve és egyenként végrehajtódik velük a ciklus. A cikluson belül a "for" és az "in"-ben megadott, ebben az esetben \$atvett nevű változóban lehet az átadott értékre hivatkozni.

A 2. esetben az átadott érték, egyetlen érték-ként lett értelmezve az idézőjelek miatt.

A 3. és 4. esetben láthatjuk, hogy a fenti két egállapítás, akkor is igaznak bizonyul, ha az értékeket változóban adjuk át a ciklusnak. A for ciklus nagy előnye, hogy file neveket is át tud venni. Az alábbi példa kilistázza az aktuális könyvtár fájljait.

```
$ for fajl in * ; do echo $fajl ; done
```

Regurális kifejezések is használhatók. Az alábbi sor csak az mp3 fájlokat listázza.

```
$ for fajl in *.mp3 ; do echo $fajl ; done
```

A for működésére egy gyakorlatias példán keresztül mutatjuk be. Működése során, a már megismert utasításokat alkalmazzuk. Az alábbi script az indítási könyvtárától, rekurzívan megkeresi az mp3 fájlokat, majd az adatait egy adatbázisba menti.

Működése a következő: beolvassa a könyvtárból és az alkönyvtárakból is az mp3 fájlokat, az mp3info programmal kiveszi belőlük az idtag adatokat, majd ezeket egy szöveges adatbázisba menti, a fájl teljes elérési útjával együtt. Működéséhez viszont egy-két dolgot rendbe kell tenni.

Ugyanis az mp3info nem minden idtag adatot ad ki külön sorba. A Title és Track, az Album és Year, valamint a Comment és Genre adatokat egy sorba adja ki. Ezért ezek közé a scriptben egy újsor karaktert helyezünk, hogy minden idtag adat, külön sorban helyezkedjen el.

A script felépítése a következő:

```

until test
do
for in
do
if
then
else
fi
done
done

```

Az until egy végtelen ciklus, amiből akkor lépünk ki egy exit utasítással, amikor az if else ága végrehajtódik. Ekkor a script is befejezi futását. A for ciklusban sorban bekérjük az adott mélységben található mp3 fájlokat. A for, ha nem talál a reguláris kifejezésnek megfelelő fájlt, akkor a változóba magát a reguláris kifejezést helyezi el. Ezt használjuk arra, hogy megállapítsuk, talált-e az adott mélységben mp3 fájlt. Ha igen, akkor az if, then ága hajtódik végre, ha nem akkor az else ág. A then ágban az idtag adatokat változókba helyezi és hozzáadja az adatbázishoz őket. Az else ágban számolja, hányadik mélységben nem talált már mp3 fájlt. Ha 5 egymás utáni mélységben nem talált, akkor exit-tel befejezi a script a futását. A for cilus után, mikor az adott mélység mp3 fájljait átvizsgálta, egy mélységgel lentebb lép, a \$keres változó értékének megváltoztatásával.

Nézzük konkrétan a scriptet.

```

#!/bin/bash
# 6.fejezet. 2.script

# a változók megfelelő kezdeti értékének beállítása.
# a numerikus adatként használt változók kezdeti értékének a 0-át be kell állítani,
# csak így lehet velük az első pillanattól számolni.

keres='*.mp3'# az első mélység keresési kifejezés
melyseg="0"
sorszam="0"
nincs="0"

# az induló könyvtár kiválasztása.
dir=`Xdialog --title "Shell Programozás 6/2 script" --no-buttons
--dselect "Válassza ki a könyvtárat,\nahonnan kezdve listába akarja
gyűjteni az mp3 fájlokat." 0 0`

# Létrehozzuk az üres adatbázist, a megfelel mező-információkkal.

echo '0;Eladó;Album;Év;Számsorszám;Szám cím;Jegyzet;Stílus;Elérési út' > zenetár.dat

# mivel a feltétel megadása nélküli test parancs sosem lesz "igaz"
# ezért ez az until esetén végtelen ciklust okoz.

until test

```



```

do
for mp3fajl in $keres
do
# Megvizsgáljuk, adott mélységben talált-e mp3 fájlt.
# Ha a for, $mp3fajl változóban a keresett ($keres)
# reguláris kifejezést adja vissza, akkor nem talált.
# A test "!" opciójával megfordítjuk a jelentést,
# így ez akkor válik igazzá,
# ha talált mp3 fájlt, azaz ha a $keres és a $mp3fajl értékek nem egyenlőek.

if test ! "$keres" = "$mp3fajl" then
# a $sorszam változóban, számoljuk a talált mp3 fájlokat,
# az adatbázisban ezzel sorszámozzuk őket, ezt
# beírjuk az index mezőbe

let sorszam=$sorszam+1
echo $mp3fajl # kiírjuk a megtalált mp3 fájlt,
# hogy érzékelhessük a script futását.

# Az alábbi részben történik az mp3info által adott idtag-ok
# mindegyikének külön-külön sorba helyezése.
# Erre azért van szükség,
# hogy a különféle idtag-okat, pontosan tudjuk kinyerni.

# Az azonos sorban lévő idtag-ok esetén a második tag elé
# egy újsor karaktert szúrunk. Ne felejtsük el, az alábbi 5 sor,
# tulajdonképpen egyetlen parancssor, "\" jelekkel tördelve.

idtag=`mp3info "$mp3fajl" | \
sed s/'Track: '/'\nTrack: '/g | \
sed s/'Year: '/'\nYear: '/g | \
sed s/'Genre: '/'\nGenre: '/g | \
tr ';' ', ' | tr '\n' '\n';`
# A végén az összes újsor karaktert egy ";" mezőelválasztóra cseréljük
# így mint egy több mezőből álló egyetlen adatsort tudjuk kezelni.
# De előtte az idtag adatokban a ";" karaktereket ","-re cseréljük
# hogy ha véletlen ";" karaktert is tartalmaznak az idtag-ek,
# az ne tévessze meg a mező-elválasztást.
# Az idtag-okat tartalmazó adatsorból a különféle idtag-eket,
# külön változóba helyezzük. Az első cut az adott idtag-et veszi ki,
# de ez még tartalmazza az idtag nevét is. pl.: "Title: Ez a szám címe"
# A második cut-tal szétválasszuk az idtag nevétől az idtag tartalmat.

szamcim=`echo $idtag | cut -d';' -f 2 | cut -d: -f2`
szamsorszam=`echo $idtag | cut -d';' -f 3 | cut -d: -f2`
eloado=`echo $idtag | cut -d';' -f 4 | cut -d: -f2`
album=`echo $idtag | cut -d';' -f 5 | cut -d: -f2`
ev=`echo $idtag | cut -d';' -f 6 | cut -d: -f2`
jegyzet=`echo $idtag | cut -d';' -f 7 | cut -d: -f2`
stilus=`echo $idtag | cut -d';' -f 8 | cut -d: -f2`

# a "-f 2-" jelentése a "-" miatt :
# "a 2.mezőtől kezdve, az összes utána jövő is."
# Azért kell, mert elvileg az idtag is tartalmazhat ":" karaktert.
# és "-f 2" esetén az utána jövő részt levágná.
# Létrehozzuk a beírandó adatsort, vagy rekordot.
# A $dir/$mp3fajl a teljes elérési utat adja meg.

sor="$sorszam ; $eloado ; $album ; $ev ; $szamsorszam ; $szamcim ; \
$jegyzet ; $stilus ; $dir/$mp3fajl"
echo "$sor" >> zenetár.dat # Majd hozzáfűzzük az adatbázis végéhez.

```

```
nincs="0" # Ez nullázza az mp3 fájlokat nem tartalmazó
# mélység számolását, ha még is talált egyet.
else
let nincs=$((nincs+1) # mp3 fájlokat nem tartalmazó mélység számolása.
if test $nincs -eq 5 # Ha már egymásután 5 mélységben nem talált.
then
exit # kilépés a scriptből
fi
fi
done
# Ha az adott mélységet a for-ral átvizsgálta, a keresést eggyel mélyebbre irányítjuk.
# 0. mélység: *.mp3
# 1. mélység: */*.mp3
# 2. mélység: */*/*.mp3 stb.

keres='*/' "$keres"
done
```

18.1.11. Saját függvények készítése

Ebben a részben, a scripten belüli saját függvényekkel fogunk foglalkozni. Hosszabb scriptek megfelelő struktúráltságához, illetve a gazdaságos működéshez, elengedhetetlen ez a technika.

Szintaxisa a következő:

```
function sajátfüggvény ()
{
<utasítások>
return
}
```

A függvényeket a script elejére helyezzük. A függvény futását azon belül bármikor megszakíthatjuk a return paranccsal. A függvény pont úgy viselkedik, mintha egy külső program lenne. Azaz lehet neki értékeket átadni a szokásos bemenetére, illetve fogadni is lehet a szokásos kimenetéről. Ennek fényében a következő megoldások is alkalmazhatók:

```
echo $változo | sajátfüggvény | less
echo $változo | sajátfüggvény > fajl
változo2=$(echo $változo | sajátfüggvény)
```

Mivel ezeknél a példáknál a szokásos kimenetet a képernyőről átirányítottuk máshová, ezért a scripten belüli egyetlen echo sem jelenik meg a képernyőn, hanem továbbítódik a megjelölt kimenetre. Illetve mivel a szokásos bemenetet is átirányítottuk, ezért a függvényen belüli read parancs is a megadott bemenetről olvas. Azaz a fenti esetben sem az

echo sem a read nem használható a szokásos módon, mert az echo nem a képernyőre ír, a read pedig nem a billentyűzetről olvas.

Lehet argumentumokat is átadni neki, amikre függvényen belül \$1, \$2 \$3 stb. módon lehet hivatkozni. Ekkor ha a read parancsot használjuk, akkor a felhasználótól várja az adatbevitelt. De a függvényen belüli echo-k még mindig nem a képernyőre, hanem az átirányított kimenetre, a lenti esetben a "valtozo" nevű változóba íródnak.

```
valtozo=`sajatfugveny $arg1 $arg2 $arg3`
```

Ha a függvényen belül, szeretnénk a képernyőre írni, akkor a szokásos kimenetet ne irányítsuk át, azaz a függvényt csak a nevére hivatkozva hívjuk meg, ha pedig a felhasználótól is akarunk adatot bekérni, akkor a bemenetet se irányítsuk át, hanem a függvénynek argumentumok formájában adjunk át értékeket.

```
sajatfugveny $arg1 $arg2 $arg3
```

Ebben az esetben viszont értéket a függvénytől csak a visszatérési érték, azaz a \$? változón keresztül kaphatunk, de ezen keresztül az érték csak egy 0 és 255 közötti számérték lehet. Ennek értékét a függvényen belül a return parancs után adhatjuk meg. A return hatására a függvény befejezi a futását és a \$? visszatérési értékbe a return után megadott érték kerül.

Ezt a korlátot úgy kerülhetjük ki, hogy a scripten belül meghatározunk egy változót, amelyet arra használunk, hogy az éppen futó függvény ezen keresztül tudjon bármilyen értéket átadni. Természetesen nem csak egy, hanem több is felhasználható. Lássunk erre egy példát.

Ez a függvény a felhasználótól kérdez valamit, majd a válaszát adja vissza. Amit a \$fe (függvény eredmény) változóban kapunk vissza a függvénytől.

```
#!/bin/bash
# 7.fejezet. 1.script

function dialogus ()
{
    local valasz=""
    local visszateres=""
    clear
    echo -n "$1 "
    read valasz
    fe=$valasz
    visszateres="10"
```

```
return $visszateres
}

kerdes="Mi a neved ?"
dialogus "$kerdes"
vissza=$?
b=$fe
echo "Üdvözlöm, $b"
echo "A függvény a $vissza értékkel tért vissza."
```

Felmerülhet a kérdés, hogy a függvényen kívül miért nem eleve a \$valasz változóval jutunk hozzá a válaszhoz? Azért, mert akkor a függvényünk eleve egy komoly hibalehetőséggel rendelkezne. Mégpedig, hogy ha a függvényen belül is, és a script törzsében is azonos, globális változókat használnánk. Ez veszélyes eljárás lenne, ugyanis több függvény egymásból történő hívása esetén ezek az értékek összekeveredhetnek, illetve a függvény könnyen felülírhatja egy a script törzsében szereplő változót. Ezért érdemes a függvényen belüli változókat elkülöníteni a script többi részében szereplőtől. Ezt úgy tehetjük meg, hogy a függvény elején a local kulcsszóval deklaráljuk a változót. Meghatározás közben már értéket is kaphat.

```
local valasz=""
local a="Érték"
```

Ebben az esetben lehet a függvényen kívül is, vagy más függvényekben is egy "valasz" nevű változónk, ezek mind külön-külön értékkel rendelkezhetnek és így az értékek egy hosszabb, vagy bonyolultabb script esetén sem keverednek össze. Ezért a függvényen belül használatos változókat érdemes a függvény elején lokálisnak meghatározni. Ez alól az képez kivételt, ha tudatosan használunk globálisan, pl. a függvény és a script törzse közötti értékmozgatás céljából. A fenti példában éppen ezt használjuk fel az \$fe változó esetén, miközben viszont a \$valasz változót lokálisnak határozzuk meg benne. Láthatjuk a példában, hogy a return után megadtunk egy visszatérési értéket is a függvényben. Most csak azért, hogy láthassuk ezt a lehetőséget is. Most tegyük a fenti dialogus függvényünket kicsit látványosabbá.

Ehhez ismerkedjünk meg egy újabb paranccsal, a tput-tal. A tput, beállít egy terminált vagy lekérdezi a terminfo-t. De mi most azt a lehetőségét használjuk ki, hogy segítségével, pozicionálni lehet a kurzort a képernyőn illetve le lehet kérdezni, hogy az adott terminál hány sorból és hány oszlopból áll.

tput cols Vissza adja, hány karakter széles a terminálunk.

tput lines Vissza adja, hány karakter magas a terminálunk.

tput cup <\$x> <\$y> A kurzor pozíciót, a megadott koordinátára helyezi.

Készítünk egy függvényt, amely egy "dialógus ablakot" jelenít meg a terminálon. Ehhez több segédfüggvényt is készítünk. Lesz egy keret() nevű amely egy keretet rajzol a képernyőre. Lesz egy box-torol() amely törli a megadott négyzetben a képernyőt. Lesz egy inputbox() nevű, amellyel kérdést tehetünk fel a felhasználónak és választ kaphatunk tőle, végül pedig egy msgbox () függvény, amellyel egy üzenetet tudunk megjeleníteni megadott másodpercig. Természetesen ezeken a függvényeken rengeteget lehetne finomítani. Például a dialógus függvények csak a terminál közepén helyezkednek majd el, természetesen meg lehetne adni, hogy egyénileg lehessen őket pozicionálni. Ezen kívül csak rövid, egysoros üzeneteket kezel. Ezen is lehetne finomítani, hogy ha túl hosszú az üzenet, akkor feldarabolja és több sorban jelenítse meg azt, a dialógusbox méretét pedig ehhez igazítsa. De most a célunk csak az, hogy bemutassuk a függvények használatát. Most röviden ismertetjük a függvények működését:

keret()

Meghívása a következő módon történik:

keret <sor-pozíció> <oszlop-pozíció> <magasság> <szélesség>

Ahol a <sor-pozíció> és az <oszlop-pozíció>, a megjelenítendő keret bal felső sarkának koordinátáját határozza meg. A <magasság> a keret magasságát határozza meg, azaz, hogy hány sor magas legyen, a <szélesség> pedig a szélességét, azaz, hogy hány karakter széles legyen. A függvény először előállít két stringet, amik a keret megrajzolásához fog használni. Egyik a keret felső és alsó széle lesz, a másik a köztes részek. Például:

```
boxszel = "#####"
boxkozep= " #                # "

"#####"
" #                # "
" #                # "
" #                # "
"#####"
```

Ezeket a szélesség alapján meghatározott hosszra készíti el. A kirajzolást a sorpozíció és az oszlop-pozíció alapján kezdi el. Hogy hány darab köztes részt rajzoljon, azt pedig a magasság alapján határozza meg. A kirajzolást egyszerű echo parancs végzi, de előtte egy tput cup x y -nal pozicionálja a kurzort. Visszatérési értéke nincs.

box-torol()

Meghívása a következő módon történik:

`box-torol <sor-pozíció> <oszlop-pozíció> <magasság> <szélesség>`

Működése hasonló a `keret()` függvényhez, annyi különbséggel, hogy itt egy megfelelő hosszúságú space-eket tartalmazó string kerül kinyomtatásra, megfelelően pozicionálva és így törli az adott négyzet területét a képernyőn. Visszatérési értéke nincs.

msgbox()

Meghívása a következő módon történik:

`msgbox <szöveg> <másodperc>`

A `tput lines` és `tput cols` parancsokkal lekérdezi, az aktuális terminál méretét. A kérdés hossza alapján meghatározza a szükséges keret szélességét, majd az ablak koordinátáit, és a terminál méret alapján meghatározza, hogy az középre kerüljön. A kiszámolt értékekkel meghívja a `keret()` függvényt. Miután az kirajzolta a keretet, a megfelelő pozícióba kiírja a megadott szöveg-et, majd a `sleep` parancssal felfüggeszti meghatározott másodpercig a script futását, ezáltal időt adva az üzenet elolvasására. Utána pedig meghívja a `box-torol()` függvényt a szükséges argumentumokkal az `msgbox` törléséhez.

inputbox ()

Meghívása a következő módon történik:

`inputbox <kerdes>`

Hasonló az `msgbox()` függvényhez, annyi különbséggel, hogy a keretméret kiszámolásakor, hogy helyet a válasz beolvasásához is, azaz magasabbra formálja. Meghívja a `keret()` függvényt, kiírja a kérdést, majd egy `read`-del választ vár rá a felhasználótól. Amikor az, `enter`-rel elküldi válaszát, akkor az `$fe` scriptglobális változóba helyezi a választ, hogy azt a függvényen kívül is el lehessen érni. Majd a `box-torol()` függvénnyel törli a dialógusbox-ot.

Elég lenne a már meglévő függvényeket minden scriptünk elejére másolni és máris használhatnánk őket. Hiszen ebben a scriptben is, amit le kell programozni, ha már rendelkezünk a függvényrésszel, csak 7 sor lenne. Ennél már csak az lenne jobb, ha nem is kellene a scriptjeink elejére mindig bemásolni, hanem lenne egy külső függvénykönyvtár, amiből elérhetjük azokat. Nos, most rögtön ezt a megoldást valósítjuk meg.

Ehhez előbb készítsük el a scriptet, amit meghívva, hozzáférhetünk függvényeinkhez. Ez egy normál script, `"#!/bin/bash"` kezdettel. Mondjuk legyen a neve `"sfugv"`. Ebbe írjuk meg a függvényeinket, majd a végéhez fűzzük hozzá a függvényeket a scripten belül

lekezelő kódrészt, majd a scriptet másoljuk a /home/konyvtarunk/bin alkönyvtárba, ami a \$PATH része.

A scriptet kikommentezve:

```
#!/bin/bash
# 7.fejezet. 2.script
# =====
# ITT KEZDŐDNEK A SAJÁT FÜGGVÉNYEK
# -----
# ITT KEZDŐDIK INPUTBOX FÜGGVÉNY

function inputbox ()
{
# A függvényben használt változók, lokálisként történő meghatározása.

local boxsor=""
local boxoszlop=""
local boxmagas=""
local boxszeles=""
local kerdes=$1
local valasz=""
local kerdhosz=""

# A terminál méretének lekérdezése.
termmagas=`tput lines`
termszeles=`tput cols`
7
kerdhosz=`echo "$kerdes" | wc -c | tr -d ' '`

# A box szélességének és magasságának meghatározása.
let boxszeles=$kerdhosz+4
boxmagas="7"

# A keret bal felső sarka koordinátáinak kiszámolása,
# hogy a box középre kerüljön.
let boxsor=$termmagas/2-$boxmagas/2
let boxoszlop=$termszeles/2-$boxszeles/2

# A keret() függvény meghívása
keret $boxsor $boxoszlop $boxmagas $boxszeles

# A kérdés kiírás kezdő pozícióinak kiszámolása.
let sor=$boxsor+2
let oszlop=$boxoszlop+3

# A kurzor megfelelő helyre állítása és a kérdés kiírása.
tput cup $sor $oszlop
echo "$kerdes"
```

```

# A válasz bekérés kezdő pozícióinak kiszámolása.
let sor=$sor+2
let oszlop=$oszlop+2

# A kurzor megfelelő helyre állítása és a válasz beolvasása.
tput cup $sor $oszlop
read valasz

# A válasz $fe script-globális változóba írása.
fe=$valasz

# Az inputbox törlése.
box-torol $boxsor $boxoszlop $boxmagas $boxszeles
}
# ITT VÉGZŐDIK AZ INPUTBOX FÜGGVÉNY
#-----

#-----
# ITT KEZDŐDIK MSGBOX FÜGGVÉNY

function msgbox ()
{
# A függvényben használt változók, lokálisként történő meghatározása.
local boxsor=""
local boxoszlop=""
local boxmagas=""
local boxszeles=""
local szoveg=$1
local szoveghosz=""
local mp=$2

# A terminál méretének lekérdezése.
termmagas=`tput lines`
termszeles=`tput cols`

# A kérdés hosszának lekérdezése és az eredményből a space-ek törlése.
szoveghosz=`echo "$szoveg" | wc -c | tr -d ' '`
let boxszeles=$szoveghosz+4 boxmagas="5"

# A keret bal-felső sarka koordinátáinak kiszámolása,
# hogy a box középre kerüljön.
let boxsor=$((termmagas/2-$boxmagas/2)
let boxoszlop=$((termszeles/2-$boxszeles/2)

# A keret() függvény meghívása
keret $boxsor $boxoszlop $boxmagas $boxszeles

# A szöveg kiíratás kezdő pozícióinak kiszámolása.

```



```

let sor=$boxsor+2
let oszlop=$boxoszlop+3

# A kurzor megfelelő helyre állítása és a szöveg kiírása.
tput cup $sor $oszlop
echo "$szoveg"

# Meghatározott ideig történő várakozás,
# hogy a felhasználó elolvashassa az üzenetet.
sleep $mp

# Az msgbox törlése.
box-torol $boxsor $boxoszlop $boxmagas $boxszeles
}
# ITT VÉGZÖDIK AZ MSGBOX FÜGGVÉNY
# -----

# -----
# ITT KEZDÖDIK A KERET FÜGGVÉNY

function keret ()
{
# A függvényben használt változók, lokálisként történő meghatározása.
local boxsor=$1
local boxoszlop=$2
local boxmagas=$3
local boxszeles=$4
local k="#"
local boxsor2=""
local boxoszlop2=""
local sor="0"
local boxszel=""
local boxkozep=""
local i=""
local spacehossz=""

# A jobb alsó sarok koordinátáinak kiszámítása.
let boxsor2=$boxsor+$boxmagas-1
let boxoszlop2=boxoszlop+$boxszeles-1

# A keret alsó és felső részét alkotó string előállítás.
# A ciklus boksx-széles-szer fut le, minden ciklusban egy darab,
# a keretet kirajzolásához meghatározott karakterrel növelve a stringet.
# Az $i változóval számolva a ciklusfutást.
let i="0"
until test $i -eq $boxszeles
do
let i=$i+1
boxszel="$boxszel$k"

```

done

```
# A keret középső részeit alkotó string elállítása.
# A string első karakterének a keretet kirajzolásához
# meghatározott karaktert teszi meg,
boxkozep=$k
```

```
# majd a ciklus boxszélesség-2 alkalommal,
# egy-egy space-el növeli a stringet,
let i="0"
let spacehossz=$boxszeles-2
```

```
until test $i -eq $spacehossz
do
let i=$((i+1))
boxkozep="$boxkozep "
done
```

```
# végül a végére is helyez egy kirajzoló karaktert.
boxkozep="$boxkozep$k"
```

```
# A kurzor, a keret felső szélének kirajzolásának pozíciójába állítása,
# és a szél kirajzolása.
tput cup $boxsor $boxoszlop
echo $boxszel
```

```
# A box belseje, felső és alsó sorának meghatározása.
let sor=$boxsor
let also=$((boxsor-1))
```

```
# A box közepét alkotó string kiírása a megfelelő sorokba.
# Kezdve a $sor sorba és folytatva egyenként az alatta következőkbe,
# amíg el nem éri a szükséges alsó sort.
until test $sor -eq $also
do
let sor=$((sor+1))
tput cup $sor $boxoszlop
echo "$boxkozep"
done
```

```
# A kurzor, a keret alsó szélének kirajzolásának pozíciójába állítása,
# és a szél kirajzolása.
tput cup $boxsor2 $boxoszlop
echo $boxszel
}
```

```
# ITT VÉGZÖDIK A KERET FÜGGVÉNY
```

```
# -----
```

```
# -----
```

```

# ITT KEZDÖDIK A BOX-TOROL FÜGGVÉNY

function box-torol ()
{
# A függvényben használt változók, lokálisként történő meghatározása.
local boxsor=$1
local boxoszlop=$2
local boxmagas=$3
local boxszeles=$4
local sor="0"
local kozep=""
local i="0"

# A jobb alsó sarok koordinátáinak kiszámítása.
let boxsor2=$boxsor+$boxmagas-1
let boxoszlop2=$boxoszlop+$boxszeles-1

# A megfelelő hosszúságú, space-t tartalmazó string elállítása.
until test $i -eq $boxszeles
do
let i=$i+1
kozep="$kozep "
done

# A kirajzolt box, bal-felső sarka pozíciójától kezdve,
# a space-t tartalmazó stringgel, a box törlése, annyi soron át,
# amilyen magas a box volt.
let sor=$boxsor-1
until test $sor -eq $boxsor2
do
let sor=$sor+1
tput cup $sor $boxoszlop
echo "$kozep"
done
}
# ITT VÉGZÖDIK A BOX-TOROL FÜGGVÉNY
# -----
# ITT VÉGZÖDNEK A SAJÁT FÜGGVÉNYEK
# =====

# =====
# ITT KEZDÖDIK A SCRIPT TÖRZSE
# A VÉGREHAJTÁS ITT KEZDÖDIK EL.

# Képernyő törlés
clear

# A kérdés meghatározása.
kerdes=" Mi az Ön neve ? "

```

```

# Az inputbox meghívása a kérdéssel.
inputbox "$kerdes"

# Az inputbox() függvényben a válasszal
# feltöltött $fe script-globális változóból,
# a válasz másik változóba írása.
nev=$fe

# A visszajelző szöveg meghatározása.
szoveg=" Üdvözlöm kedves $nev ! "

# A szöveg msgbox() függvénnyel történő kiírítása,
# a meghatározott másodpercig fenntartott msgboxban.
msgbox "$szoveg" 3

# A kurzor, a képernyő bal-felső sarkába állítása.
tput cup 0 0

# ITT VÉGZÖDIK A SCRIPT TÖRZSE
# =====
# ITT KEZDŐDIK A FÜGGVÉNYEKET KEZELŐ RÉSZ

case $1 in
keret)
keret $2 $3 $4 $5
;;
box-torol)
box-torol $2 $3 $4 $5
;;
inputbox)
inputbox "$2"
;;
msgbox)
msgbox "$2" $3
;;
*)
msgbox "Ismeretlen függvényhívás" 5
esac

# Itt kerül átadásra a hívó függvénynek a visszatérési érték,
# a hibakimeneten keresztül.

echo $fe 1>&2

```

A script első pillantásra hosszú. De ezentúl elég csak a már meglévő függvényeket az új scriptünk elejére másolni és máris használhatjuk őket. Hiszen a fenti sriptben is amit le kell programozni, ha már rendelkezünk a függvény résszel, csak 7 sor !!! Ennél már csak

az lenne jobb, ha nem is kellene a scriptjeink elejére másolni, hanem lenne egy külső függvénykönyvtár, amiből elérhetjük azt. Ehhez előbb készítsük el a scriptet, amit meghívva, hozzáférhetünk függvényeinkhez. Másoljuk a fenti scriptből, a függvényeket egy másik üres script fájlba (azaz `#!/bin/bash`-al kezdődő, futási joggal rendelkező fájlba). Mondjuk legyen a neve "sfugv". Ezután fűzzük a script végére az alábbi részt, majd a scriptet másoljuk a `/home/konyvtarunk/bin/` könyvtárba, ami a `$PATH` része.

Az eredeti scriptünk helyett, pedig hozzunk létre egy másikat, aminek az alábbi legyen a tartalma:

```
#!/bin/bash
# 7.fejezet. 3.script

clear
kerdes=" Mi az Ön neve ? "
sfugv inputbox "$kerdes" 2> /tmp/$0.$$
nev='cat /tmp/$0.$$\`
szoveg=" Üdvözlöm kedves $nev ! "
svfugv msgbox "$szoveg" 3
tput cup 0 0
```

Ez a következőképpen működik:

Az sfugv scriptünket meghívjuk, első argumentumként megadva neki, hogy melyik függvényt szeretnénk használni, utána pedig a többi argumentumban az adatokat, amit a függvénynek akarunk átadni. Az sfugv script, az első argumentumban kapott érték alapján, case-zel megnézi melyik függvényt kell futtatnia, annak átadja a többi argumentumot amit kapott és lefuttatja a függvényt, majd ami értéket (ha van ilyen), vissza adott a függvény, az sfugv scripten belüli globális `$fe` változóba, azt a `stderr`, azaz a szokásos hiba kimenetre küldi. Az eredeti függvényben, amiből meghívtuk az sfugv scriptet, a függvény hibakimenetét átirányítjuk egy fájlba, majd a fájlból a `cat` segítségével helyezzük változóba. Természetesen azoknál a függvényeknél nincs erre szükség, amik nem adnak vissza értéket. Amik pedig visszaadnak, azoknál viszont azért van rá szükség, mert ha az sfugv szokásos kimenetét átirányítjuk, pl. egy változóba, akkor az sfugv script összes függvényének összes echo-ját odaírja, nem pedig a képernyőre, vagyis akkor a függvények nem tudnának kommunikálni a felhasználóval. Így viszont amit a felhasználónak üzen, azt a szokásos kimenetre írja, viszont a vissza adott értéket a hibakimenetre. Így nem keveredik a kettő össze. Innen viszont egy átmeneti fájlon keresztül tudjuk változóba tenni.

Még szót ejtünk a `$0.$$` nevű ideiglenes fájlról. Ez mindig egy egyéni nevű fájlt hoz létre, vagyis ha egyszerre több processz, használja a scriptet, akkor sem keverednek össze az eredmények, mint akkor történne, ha fix nevű fájlt használnánk. Ez a fájl elnevezés esetén a `$0` változó, a futó script nevét, a `.$$` rész pedig a futó processz számát

adja a fájl nevébe. Azaz egy myscript nevű script futása esetén, amely pl. a 6234-es pid-szám alatt fut, myscript.6234 lesz az ideiglenes fájl neve. De ezzel nem kell törődnünk, hiszen amikor kiolvassuk a fájl tartalmát, akkor is elég a \$0.\$\$ néven hivatkozni rá.

A saját inputbox() függvény írásán keresztül, jobban megérthettük az Xdialog működését is. Alap helyzetben az is a hibakimenetre ír, de nála adva van egy -stdout kapcsoló, aminek hatására a normál kimenetre adja vissza az Xdialog boxok által visszaadott értékeket. Bár ha belenézünk az Xdialog-hoz kapott samples, vagyis példa scriptekbe, (/usr/share/doc/Xdialog/samples vagy /usr/local/share/doc/Xdialog/samples,) akkor láthatjuk, hogy a legtöbb esetben ott is egy ideiglenes fájlon keresztül oldják meg az érték visszaadását. Xdialog példaként itt arra láthatunk egy lehetőséget, hogyan lehet több dialógusablakot is nyitva tartani egyszerre. Tulajdonképpen külön függvényekbe helyezzük a dialógusablakok indítását és a függvényt a háttérben futtatjuk. Ezt ugyanúgy kell tenni, mint bármilyen program esetében. "függvénynev &". Természetesen pusztán a több ablak futtatása úgy is lehetséges, ha függvény nélkül, a dialogbox-ot indító parancs után teszünk "&" jelet, de azért a függvényben való elhelyezés a jó megoldás, mert csak akkor van lehetőség, a dialogbox visszatérési értékének feldolgozására is.

```
#!/bin/bash
# 7.fejezet. 4.script

function ablak1 ()
{
Xdialog --stdout --title "Shell Programozás 7/4 script" \
--backtitle "Első ablak" -3rangesbox "Kérem állítsa be a \
kívánt értékeket" 0 0 "Biztonság" "1" "10" "5" "Gyorsaság" \
"1" "1000" "300" "Stabilitás" "1" "100" "80"
}

function ablak2 ()
{
Xdialog --stdout --title "Shell Programozás 7/4 script" --backtitle
"Második ablak" --calendar "IDŐGÉP :-)" 0 0 "24" "07" "2381"
}

function ablak3 ()
{
Xdialog --stdout --title "Shell Programozás 7/4
script" --backtitle "Harmadik ablak" --item-help --buildlist "Válassza
ki a megfelelő neveket." 0 0 "10" "1" "Kovács Béla" "on" "Kecskemét" "2"
"Kormos Katalin" "off" "Baja" "3" "Sípos Antal" "on" "Budapest" "4"
"Baranyi Andrea" "on" "Zalaegerszeg" "5" "Németh Balázs" "off" "Pécs"
```

```
}  
  
ablak1 &  
ablak2 &  
ablak3 &  
  
Xdialog --stdout --backtitle "" --title "Shell Programozás 7/4 script"  
--msgbox "Befejezze a script a futását ?" 0 0  
killall Xdialog
```

18.1.12. Tömbváltozók és többdimenziós tömbök modellezése

A bash shell ismeri a tömb típusú változókat. Nagy hátrány, hogy csak egydimenziókat ismer. Azaz a többdimenziós tömböket nem ismeri. Nézzük a tömb változók kezelését.

Meghatározás, vagy deklarálás:

```
$ declare -a valt[elemszam]
```

Dinamikus tömbökről van szó, azaz a meghatározott elemszám nem köt, hanem túl lehet rajta lépni. Deklarálni sem szükséges, hanem mint a normál változónál, amikor a tömbnek vagy annak egy elemének értéket adunk, akkor az létrejön. A tömb első eleme a 0 sorszámu.

Értékadás:

```
tomb[elemszam]="érték"
```

```
$ tomb[3]="Szabó Julianna"
```

Érték kinyerése:

```
echo ${tomb[elemsorszam]}
```

```
$ echo ${tomb[3]}
```

```
Szabó Julianna
```

```
$ b=`echo ${tomb[3]}`
```

A tömb típusú változónak az is az előnye, hogy ciklusokban könnyen kezelhetők. A ciklusszámlálóval hivatkozunk a tömb egyes elemeire.

```
i="0"
until test $i -eq 4
do
let i=$i+1
echo ${tomb[$i]}
done
```

Egy tömböt nullázni, vagyis minden elemét üressé tenni a következőképpen lehet:

```
$ tomb=( )
```

A kétdimenziós tömböket, egy excel táblához hasonlíthatjuk, aminek az egyik dimenzióját a sorok, a másik dimenzióját az oszlopok adják. Egy elemre pedig hasonlóan hivatkozhatunk, mint az excel tábla egy cellájára, azaz megadjuk a sort és az oszlopot melyek kereszteződésében a cella van, itt pedig megadjuk, az első dimenzió elemsorszámot és a második dimenzió elemsorszámot.

Kétdimenziós tömböt úgy is, modellezhetünk, ha az egydimenziós tömb elemeibe egy-egy adatsort teszünk, aminek a mezői képezik a második dimenziót. Ezeket, a már ismert módon, mint mezőkre bontott adatsorokat kezeljük. Az első dimenzió irányát, a tömb elemek alkotják amik tulajdonképpen adatsorok, a másodikát az egyes tömbelemekben tárolt további elemek, vagyis az adatsorok mezői. A nulladik elemben, tárolhatjuk a mezőneveket. Azaz a kétdimenziós tömbünk, logikailag pontosan úgy fog kinézni, mint az adatok.dat fájlunk. A 0. sorszámú tömbelem első mezőjét, ami az adatok.dat fájl "Sorszám" mezőoszlop 0. sorának felel meg, felhasználhatjuk a fájl nevének tárolására, amiből az adatokat betöltöttük. Ez a technika a kétdimenziós tömböknél még alkalmazható, de a három, vagy több dimenziósoknál már nem. Bár ezeket ritkán használjuk. Az alábbi script, az adatok.dat adatbázis fájl tartalmát listázza ki, adatsoronként, azon belül pedig mezőkre bontva azt.

```
#!/bin/bash
# 8.fejezet. 1.script

i="-1"
let sorokszama=`cat adatok.dat | wc -l | tr -d ' '-1`
until test $i -eq $sorokszama
do
let i=$i+1
a[$i]=`cat adatok.dat | sed -n "$((i+1)) p" `
mezodb=`echo "${a[$i]}" | tr ';' '\n' | wc -l | tr -d ' '`
echo "#####"
```



```

echo "##### ${i}-dik adatsor feldolgozása #####"
echo "#####"
i2="0"
until test $i2 -eq $mezodb
do
let i2=$((i2+1))
mezonev=`echo "${a[0]}" | cut -d';' -f $i2`
mezo=`echo "${a[$i]}" | cut -d';' -f $i2`
echo "$mezonev : $mezo"
done
done

```

Mivel ezzel a módszerrel, három és több dimenziójú tömböket nincs értelme modellezni, ezért nézzük meg, hogyan modellezhetünk másképpen többdimenziós tömböket. Előbb az egydimenziós, majd a többdimenziós tömb modellezését, de már elszakadva az adatbáziskezelés példájától.

Tudjuk, hogy a shell változói, alaphelyzetben string típusú változók. Ha meghatározzuk, hogy a tömb elemeit milyen karakterrel határoljuk el egymástól, akkor egy szöveges változóban is tárolható egy teljes tömb. Ebből pedig a cut paranccsal kinyerhető a szükséges tömbelem. Nézzünk egy példát, egy neveket tartalmazó, egydimenziós tömböt. Legyen a tömbelem elválasztó karakter a megszokott ";".

```

$ tomb="első elem;második elem;harmadik elem;negyedik elem;ötödik elem;hatodik elem"
$ elem1=`echo $tomb | cut -d';' -f1`
$ elem2=`echo $tomb | cut -d';' -f2`
$ elem3=`echo $tomb | cut -d';' -f3`
$ elem4=`echo $tomb | cut -d';' -f4`
$ elem5=`echo $tomb | cut -d';' -f5`
$ elem6=`echo $tomb | cut -d';' -f6`

```

Tehát a tömb adott sorszámú elemére így tudunk hivatkozni:

```

$ elemsorszam="3"
$ elem=`echo $tomb | cut -d';' -f $elemsorszam`
$ echo $elem
harmadik elem

```

Viszont ha egy ciklusban szeretnénk felhasználni a tömbelemeket, akkor tudnunk kell, hány elemből is áll a tömb. Ehhez nézzük meg, hány alkalommal fordul elő benne, a tömbelem elválasztó ";" karakter. Erre a wc programot használjuk -l kapcsolóval, ami visszaadja hány sorból is áll a bemenetére küldött adatsor. De mivel a "wc -l" a sorokat számolja meg, ezért előtte a ";" jeleket, sorvégejelekre "\n" cseréljük, majd utána a wc által adott számról levágjuk a felesleges szóközöket.

```
$ tombelemdb='echo $tomb | tr ';' '\n' | wc -l | tr -d ' '`
$ echo $tombelemdb
6
```

A tömb elemeinek feldolgozása egy ciklusban:

```
tombelemdb='echo $tomb | tr ';' '\n' | wc -l | tr -d ' '`
i="0"
until test $i -eq $tombelemdb
do
let i=$i+1
echo $tomb | cut -d';' -f $i
done
```

Mindez nem túl ismeretlen, hiszen az előző részekben már használtuk ezt a technikát, másrészt ez a bash által is ismert egydimenziós tömbbel egyszerűbben megoldható, de továbbfejlesztve ezt, akár a lényegesebb, tényleges programozási nyelvekben ismert, tömbkezelő függvények is le modellezhetőek.

Készítsünk egy függvényt, ami kiírja a tömb általunk meghatározott sorszámú elemét. A függvénynek az első argumentumban a tömböt kell átadni, a másodikban a kívánt elem sorszámát. A függvény az elemet az \$fe script-globális változóban adja vissza. Senkit ne zavarjon meg a sorok elején lévő "local" deklaráció. Csupán nem a függvény elején határozzuk meg a belső változók lokális voltát, hanem akkor, amikor értéket kapnak.

```
function tombld_kiir ()
{

local tombtorzs=$1
local elemsorszam=$2
local elem='echo $tombtorzs | cut -d';' -f $elemsorszam'
fe=$elem
}
```

Használata:

```
index=3
tombld_kiir $tomb $index
elem=$fe
```

Aki a <http://glindorf.linuxuser.hu> -ról töltötte le a saját függvények, "sfugv" scriptet és azt másolta be a /home/konyvtaram/bin könyvtárába, annak így is elérhető:

```
sfugv "tombld_kiir" "$tomb" $index 2> /tmp/$0.$$
elem='cat /tmp/$0.$$'
```

Most készítsünk egy olyan függvényt, ami az adott tömbelemet értékkel tölti fel. A függvénynek a tömb és a feltöltendő elem sorszáma mellett, a feltöltendő tartalmat is át kell adni. Ehhez a tömbből külön kell választani az értékkel feltöltendő elem előtti szakaszt és az azutáni szakaszt, majd összeilleszteni, középük szűrve az értékkel feltöltött elemet. Az új tömböt a \$fe script-globális változóban adja vissza.

```
function tombld_feltolt ()
{
    local tombtorzs=$1
    local elemsorszam=$2
    local tartalom="$3"

    # A tömb elemszámának megnézése.
    local maxelem='echo $tombtorzs | tr ';' '\n' | wc -l | tr -d ' '`

    # Ha az első elem kell, akkor nem kell
    # első levágott rész, azaz, az legyen üres.
    local ig=$((elemsorszam-1))
    if test $ig -ge 1
    then
        local tombeleje='echo $tombtorzs | cut -d';' -f 1-$ig'
    else
        local tombeleje=""
    fi

    # Ha az utolsó elem kell, akkor nem kell
    # utolsó levágott rész, azaz, az legyen üres.
    local tol=$((elemsorszam+1))
    if test $tol -le $maxelem
    then
        local tombvege='echo $tombtorzs | cut -d';' -f $tol-'
    else
        local tombvege=""
    fi

    # Ha közbülső elem, értékét kellett változtatni
    if test $elemsorszam -ne 1 -a $elemsorszam -ne $maxelem
    then
        local ujtomb="$tombeleje;$tartalom;$tombvege"
    fi

    # Ha az első elem, értékét kellett változtatni
    if test $elemsorszam -eq 1
    then
```

```

local ujtomb="$startalom;$tombvege"
fi

# Ha az utolsó elem, értékét kellett változtatni
if test $elemsorszam -eq $maxelem
then
local ujtomb="$tombeleje;$startalom"
fi

fe=$ujtomb
}

```

Használata:

```

index=3
ujtartalom="új elem-tartalom" tombld_feltolt $tomb $index "$ujtartalom"
tomb=$fe

sfugv "tombld_feltolt" "$tomb" $index "$ujtartalom" 2> /tmp/$0.$$
tomb=`cat /tmp/$0.$$`

```

Tekintsük meg az eredményt:

```
echo $tomb | tr ';' '\n'
```

Egy kis ügyeskedéssel, a tömbelemek sorba rendezése, tömbelem beszúrása, kivágása és a többi általános tömbkezelő függvény is elkészíthető.

Többdimenziós tömbök modellezése

Sajnos a bash shell nem képes többdimenziós tömbváltozók kezelésére. Holott ez egy igen hatékony eszköz a programozás során. Főként, ha adatbázis szerű alkalmazásról van szó, hiszen ott sok esetben az adattáblát, egy 2 dimenziós tömbváltozóba olvasák be, ahol az egyik dimenzió elemei az adatsorok a másik pedig az adatmezők. Most nézzük, hogyan modellezhetünk két, vagy többdimenziós tömböt, a bash egydimenziós tömbjeinek felhasználása nélkül. Ahány dimenziós a tömb, annyiféle elválasztó karakterre van szükség, amiket természetesen a tömbelemek által tartalmazott adatokban nem szerepelhetnek. Például egy kétdimenziós tömb esetén legyen a két dimenziót, vagy úgy is mondhatnám, az adatsorokat elválasztó jel a "\#". Ez alapján hozzunk létre egy kétdimenziós tömböt. Első dimenziója két indexelemmel, a második három indexelemmel rendelkezzen. Az érték amiket az elemek tartalmaznak, legyenek az adott elem indexszámai betűvel kiírva.

```
tomb2d="egy-egy;egy-kettő;egy-három#kettő-egy;kettő-kettő;kettő-három#három-egy;három-kettő;három-három"
```

Ha feltöltéskor használjuk a "\" parancssortördelés lehetőségét, mindjárt jobban átlátható mit is csinálunk.

```
tomb2d="\
egy-egy;egy-kettő;egy-három#\
kettő-egy;kettő-kettő;kettő-három#\
három-egy;három-kettő;három-három"
```

Készítsük el a fenti két függvény kétdimenziós tömböt kezelő változatait.

```
function tomb2d_kiir ()
{

local tombtorzs=$1
local dimlelemsorszam=$2
local dim2elemsorszam=$3
local elem='echo $tombtorzs | cut -d'#' -f $dimlelemsorszam | cut -d';' -f $dim2elemsor
fe=$elem
}
```

Használata:

```
index1=2
index2=3
tomb2d_kiir $tomb2d $index1 $index2
elem=$fe
```

```
sfugv "tomb2d_kiir" "$tomb2d" $index $index2 2> /tmp/$0.$$
elem='cat /tmp/$0.$$'
```

A feltöltést végző:

```
function tomb2d_feltolt ()
{

local tombtorzs=$1
local dimlelemsorszam=$2
local dim2elemsorszam=$3
local tartalom="$4"

#===== AZ ELSŐ DIMENZIÓ FELDARABOLÁSA =====
# Az első dimenzió (adatsorok) elemszámának megnézése.
local maxdim1='echo $tombtorzs | tr '#' '\n' | wc -l | tr -d ' '

# Ha az első elem (adatsor) kell, akkor nem kell
```

```

# első levágott rész, azaz, az legyen üres.
local ig=$((dimlelemsorszam-1))
if test $ig -eq 0
then
local dimltombelege=""
else
local dimltombelege='echo $tombtorzs | cut -d'#' -f 1-$ig'
fi

# Az elem, (adatsor), amely a változtatandó
# elemet (adatmezőt) tartalmazza
local adatsor='echo $tombtorzs | cut -d'#' -f $dimlelemsorszam'

# Ha az utolsó elem (adatsor) kell, akkor nem kell
# utolsó levágott rész, azaz, az legyen üres.
local tol=$((dimlelemsorszam+1))
if test $tol -le $maxdiml
then
local dimltombvege='echo $tombtorzs | cut -d'#' -f $tol-'
else
local dimltombvege=""
fi

#===== AZ MÁSODIK DIMENZIÓ FELDARABOLÁSA =====
# A második dimenzió (adatmezők) elemszámának megnézése.
local maxdim2='echo $adatsor | tr ';' '\n' | wc -l | tr -d ' '

# Ha az első elem (adatmező) kell, akkor nem kell
# első levágott rész, azaz, az legyen üres.
local ig=$((dim2elemsorszam-1))
if test $ig -ge 1
then
local adatsoreleje='echo $adatsor | cut -d';' -f 1-$ig'
else
local adatsoreleje=""
fi

# Ha az utolsó elem (adatmező) kell, akkor nem kell
# utolsó levágott rész, azaz, az legyen üres.
local tol=$((dim2elemsorszam+1))
if test $tol -le $maxdim2
then
local adatsorvege='echo $adatsor | cut -d';' -f $tol-'
else
local adatsorvege=""
fi

# ===== AZ ELSŐ DIMENZIÓ ADATSORÁNAK ÖSSZEILLESZTÉSE =====
# Ha közbülső elem, (adatmező) értékét kellett változtatni

```

```

if test $dimlelemsorszam -ne 1 -a $dim2elemsorszam -ne $maxdim2
then
local ujadatsor="$adatsoreleje;$tartalom;$adatsorvege"
fi

# Ha az első elem, (adatmező) értékét kellett változtatni
if test $dim2elemsorszam -eq 1
then
local ujadatsor="$tartalom;$adatsorvege"
fi

# Ha az utolsó elem, (adatmező) értékét kellett változtatni
if test $dim2elemsorszam -eq $maxdim2
then
local ujadatsor="$adatsoreleje;$tartalom"
fi

# ===== A TELJES KÉTDIMENZIÓS TÖMB ÖSSZEILLESZTÉSE =====
# Ha közbülső elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dimlelemsorszam -ne 1 -a $dimlelemsorszam -ne $maxdim1
then
local ujtomb2d="$dim1tombeleje#$ujadatsor#$dim1tombvege"
fi

# Ha az első elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dimlelemsorszam -eq 1
then
local ujtomb2d="$ujadatsor#$dim1tombvege"
fi

# Ha az utolsó elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dimlelemsorszam -eq $maxdim1
then
local ujtomb2d="$dim1tombeleje#$ujadatsor"
fi

# ===== AZ ÚJ TÖMB ÁTADÁSA A $fe GLOBÁLIS VÁLTOZÓBAN =====
fe=$ujtomb2d
}

```

Használata:

```

index1=2
index2=3
ujtartalom="új elem-tartalom"

```

```
tomb2d_kiir $tomb2d $index1 $index2 $ujtartalom
tomb2d=$fe
```

```
sfugv "tomb2d_feltolt" "$tomb2d" $index $index2 "$ujtartalom" 2> /tmp/$0.$$
tomb2d=`cat /tmp/$0.$$`
```

Tekintsük meg az eredményt:

```
echo $tomb2d | tr '#' '\n'
```

Természetesen ugyanígy modellezhetünk és használhatunk, háromdimenziós tömböket is. Bár felmerülhet a kérdés, azoknak mikor is láthatjuk hasznát? Például, lehetséges vele több adattáblás adatbázisok egyetlen változóba való olvasása és kezelése. Ahogy a kétdimenziós tömböt egy excel táblához, úgy a három dimenziósat, egy több munkalapos excel táblázathoz hasonlíthatjuk, ahol az első dimenziót, a munkalapok alkotják, a másodikat a munkalapok sorai, a harmadikat pedig a munkalapok oszlopai alkotják. Az adattáblák külön fájlokban is elhelyezkedhetnek és egy index fájl felsorolhatja őket. Az adattáblák első mezőoszlopa az adatsorok sorszámozását, első adatsora pedig az adatmezők neveit, illetve az első adatsor első adatmezője pedig az adattábla nevét is tartalmazhatja.

A többtáblás adatbázis, nem csak több fájlal, hanem egy táblahatároló karakter használatával, egyetlen fájlban is tárolható. Hozzunk létre egy Kecskeméti barátokat, Budapesti rokonokat és Szegedi kollégákat tároló többtáblás adatbázist egyetlen háromdimenziós tömbben. A "\" karaktert használjuk a parancssor tördelésére, az átláthatóság kedvéért. Ezt nyugodtan így ahogy van bemásolhatjuk egy terminálba:

```
$ tomb3d="\
barátok-tábla;barát-név;barát-cím#\
1;Barát Béla;Kecskemét#\
2;Barát Gábor;Kecskemét\
&\
rokonok-tábla;rokon-név;rokon-cím#\
1;Rokon
Andrea;Budapest#\
2;Rokon Gergő;Budapest\
&\
kollégák-tábla;kolléga-név;kolléga-cím#\
1;Kolléga Ferenc;Szeged#\
2;Kolléga Katalin;Szeged\
"
```


Ezután írassuk ki, az adattáblákat egy üres sorral elválasztva, az adattábla adatsorait pedig külön-külön sorokba helyezve:

```
$ echo $tomb3d | sed s/'&'/'\n\n'/g | tr '#' '\n'
barátok-tábla;barát-név;barát-cím
1;Barát Béla;Kecskemét
2;Barát Gábor;Kecskemét

rokonok-tábla;rokon-név;rokon-cím
1;Rokon Andrea;Budapest
2;Rokon Gerg;Budapest

kollégák-tábla;kolléga-név;kolléga-cím
1;Kolléga Ferenc;Szeged
2;Kolléga Katalin;Szeged
```

Lássunk egy háromdimenziós tömb meghatározott elemét kiíró függvényt:

```
function tomb3d_kiir ()
{
    local d3=$1
    local d2='echo $d3 | cut -d'ß' -f $2'
    local d1='echo $d2 | cut -d'#' -f $3'
    local d0='echo $d1 | cut -d';' -f $4'
    fe=$d0
}
```

Használata:

```
tomb3d_kiir $tomb $index1 $index2 $index3
elem=$fe

sfugv "tomb3d_kiir" "$tomb2d" $index $index2 $index3 2> /tmp/$0.$$
elem='cat /tmp/$0.$$\`
```

Jelenlegi ismereteinket alapul véve, akár egy dBASE-szerű adatbáziskezelő scriptet is írhatnánk.

Xdialog példa. Most nézzük meg, hogy milyen hatékonyan lehet a tömböket és az Xdialog lehetőségeit használni. Ezzel a scripttel egy az adatok.dat adatbázisunkhoz, vagy azzal azonos felépítésű adatbázishoz lehet további adatsorokat hozzáadni. A script nem ellenőrzi le a bevitt adatok formájának helyességét de természetesen ez megoldható. Megoldás lehetne, ha az adatbázisban a mezőnevek mellett azt is tárolnánk, milyen megszabott értékek kerülhetnek bele. Majd a script ezeket kiolvasná és az ilyen mezők esetében az adatbevitelt egy választó listából tenné elérhetővé, az Xdialog –combobox –szal. De ettől most eltekintünk.

```
#!/bin/bash
# 8.fejezet. 2.script
# Szerkesztendő adatfájl kiválasztása

datfile='Xdialog --title "Shell Programozás 8/2 script" --backtitle
"Válasszuk ki az adatbázist, amihez további adatokat szeretnél hozzáfűzni
! " --stdout --nobuttons --fselect "" 0 0' gomb=$?'

# Ha nem igen gomb lett nyomva, akkor kilép a scriptből,
# akkor is, ha az ablak be lett zárva.
if test ! $gomb -eq 0
then
exit
fi

# Az új adatsor sorszámának lekérdezése.
ujadatsorszama='cat $datfile | wc -l | tr -d ' '`

# A nulladik adatsor, azaz a fájl első sorának beolvasása
# Esetünkben ez tartalmazza az adatmezők neveit
mezosor='cat $datfile | sed -n '1 p`

# Mezők megszámlálása
mezodb='echo $mezosor | tr ';' '\n' | wc -l | tr -d ' '`

# Az egész egy nagy ciklusban van, hogy több adatsort is fel lehessen vinni.
# A ciklusból akkor lép ki, ha a felhasználó nem akar több adatsort hozzáadni.
ki="0"
until test ! $ki -eq 0
do
# A cikluson belüli tömbök tartalmának törlés
mezo=()
prev=()
# Ez a belső ciklus sorba veszi a mezőket és beolvassa
# a mezőneveket egy tömbbe
i="0"
until test $i -eq $mezodb
do
i=$((i+1))
# Aktuális mező nevének lekérdezése
mezonev[$i]='echo $mezosor | cut -d';' -f $i`
done
# Ezt a két speciális sort a két mezőbe bevihető értékekről
# tájékoztatja a felhasználót.
# Ezzel elveszti a script az univerzitását és csak az ilyen
# mezőkkel rendelkező adatbázisokhoz lesz jó.
mezonev[7]="${mezonev[7]} ( férfi / n )"
mezonev[8]="${mezonev[8]} ( független / házas / elvált / özvegy )"
```

```
# Ez a ciklus újra sorba veszi a mezőket és bekéri az új mező tartalmát a
# felhasználótól.
# Közben a $bevittadat változóból tájékoztatja az addig
# bevitt adatokról.
# Az $ujadatsor változóba gyűjtjük folyamatosan a már
# felvitt mez adatokat.
i="1"
bevittadat=""
ujadatsor="$ujadatsorszama"
until test $i -eq $mezodb
do
# Ha az első mezőnél lett "Vissza" gomb
# nyomva az $i 0 lenne.
# Ezt akadályozza meg.

if test $i -eq 0
then
i=1
fi

# A ciklus számlálása
i=$((i+1))

mezo[$i]='Xdialog --stdout --title "Shell Programozás 8/2
script" --backtitle
"Adatfelvitel\n-----\nEddig felvitt
adatok:\n$bevittadat\n" --wizard --inputbox "${mezonev[$i]}" 0
0`
gomb=$?

# A megnyomott gombok szerinti végrehajtás
case $gomb in
# Igen gomb
0)
# Az eddig bevitt mezőadatokhoz a mostani
# ciklus adatának hozzáírása.
# A $bevittadatok változónál a mező nevét is hozzáírjuk
# és egy "\n" jellel az Xdialog számára
# sortörést is teszünk bele, hogy a mezőket külön-külön sorokba
# írja ki.

bevittadat="$bevittadat\n${mezonev[$i]} : ${mezo[$i]}"
# Az adatsor esetében a ";" mezőelválasztó karaktert teszünk
# közéjük.

ujadatsor="$ujadatsor;${mezo[$i]}"
# Elmenti az aktuális ciklus adatait

bevittadatprev[$i]=$bevittadat
ujadatsorprev[$i]=$ujadatsor
;;
# Mégse gomb
1)
# Az adatsor feltöltőciklus befejezését váltja ki
# mert a feltételt teljesíti,
# de az adatbázis aktuálisnak hagyva, rákérdez az új adatsor
# felvitelére.
i=$mezodb
;;
# Mégse gomb
3)
# A ciklus számlálót kettővel csökkenti.
```

```

# Ha csak eggyel csökkentené, akkor ugyanez a ciklus futna újra,
# mert a ciklus a számláló,
# eggyel való növelésével kezdődik.
# De nekünk az ezt megelőző kell.

i=$((i-2))
# Vissza állítja az előző ciklus, kezdő adatait, amik a kettővel
# ezelőtti ciklus befejező adatai.
# Ezeket teszi aktuálissá.

bevittadat=${bevittadatprev[$i]}
ujadatsor=${ujadatsorprev[$i]}
;;
# Az ablak be lett zárva. A scriptből kilépés
255)
exit
;;
esac
done
if test ! $gomb -eq 1
then
# A bevitt adatok helyességét ellenőriztetjük a felhasználóval.

Xdialog --title "Shell Programozás 8/2 script" --backtitle
"Helyesek az adatok ?" --default-no --yesno "$bevittadat\n" 0 0
v=$?
# Ha igennel válaszoltunk, akkor hozzáírja a fájlhoz.
# Ha "nem" gombot nyomtunk, vagy bezárta az ablakot, akkor ez kimarad,
# az adatokat eldobja.

if test $v -eq 0
then
echo $ujadatsor >> $datfile
else

Xdialog --title "Shell Programozás 8/2 script" --msgbox "Az
adatokat eldobtam ! " 0 0
fi
fi

# Ha nem-et választottunk, 1-re változtatja a $ki értékét, ami hatására a
# külső ciklus abbahagyja a futását.

Xdialog --title "Shell Programozás 8/2 script"
--yesno "Kíván újabb adatsort felvenni ?" 0 0
ki=$?

# Ha az ablak be lett zárva, akkor 255, de a $ki akkor sem 0, vagyis a
# ciklus ugyanúgy befejeződik,
# mintha 1 lenne, azaz a "nem" gomb lett volna megnyomva.
done
exit

```

18.1.13. Scriptünk tesztelése, hibajavítása

Azt mondja néhány programozó, hogy a program megtervezése az összidő 1/3-ad része, a megírás a másik 1/3-ed rész, a hibakeresés és a tesztelés pedig a maradék 1/3-e rész.

Mit tehetünk, ha a scriptünk, egyszerűen nem akar működni és nem tudjuk mi a hiba. Indítsuk a scriptet a -v vagy az -x kapcsolóval

```
$ bash -v scriptünk
$ bash -x scriptünk
```

Ezek futás közben értékes információkat adnak a számunkra. Másik lehetőség, hogy a scriptben echo-kat helyezünk el, amik informálnak minket.

```
if test
then
echo "az IF then ága fut"
else
echo "az IF else ága fut"
fi
```

Vagy közben, a különféle változók értékeit kiíratjuk és ezzel is vizsgáljuk mit is csinál a scriptünk.

```
echo "\$neve változó : $neve"
```

Esetleg ha futás közben a kiírás törlődne, például clear vagy más parancs miatt, akkor felfüggesztjük a futást, egy read-dal. Fontos, hogy a read után olyan változót adjunk meg, amit egyébként nem használunk a scriptben.

```
echo "\$neve változó : $neve"; read tmp
```

Erre is használható az Xdialog:

```
Xdialog --msgbox "$neve" 0 0
```

Néhány ötlet

Néhány dolog nem igazán kívánczolt egyik részhez sem. Ezért itt felsoroljuk ezeket. Különösebb magyarázat nélkül tesszük közzé őket.

Néhány esetben szükségünk lehet véletlenszám generálásra

```
$ tol=10
$ ig=15
$ int=$((ig+$tol))
veletlenszam='echo $((RANDOM%$((intervallum+1))+$tol))'
```

Milyen hosszú egy adott string?

```
$ a="pirosalma napraforgó"
$ hossz='echo $a | wc -c | tr -d ' '`
$ echo $hossz
21
```

Hányszor fordul elő egy egysoros stringben, egy adott betű?

```
$ betu="a"
$ db='echo $((`echo $a | tr "$betu" '\n' | wc -l | tr -d ' '-1))`
$ echo $db
4
```

Hányszor fordul elő egy hosszabb kifejezés egy több soros szövegben?

```
$ kif="ter"
$ db='cat adatok.dat | tr -d '\n' | sed s/"$kif"/'\n'/g | wc -l | tr -d ' '`
$ echo $db
4
```

Hogyan lehet egy karakter ascii kódját megtudni?

```
$ char="é"
$ kod='echo $char | od -t u1 | grep 0000000 | cut -c 9,10,11 | sed s/\ //g`
$ echo $kod
233
```

Egy ascii kódot, hogyan lehet karakterré formálni?

```
$ kod=233
$ char='echo -e "\\$((printf %03o $kod))" `
$ echo $char
é
```

18.1.14. Utószó

Befejezésképpen csak annyit, hogy örömünkre szolgált, ha valakinek segíthettünk a script írás elsajátításában, vagy a Linuxhoz való közelebb kerülésében. Aki a fenti oldalakon átverekedte magát, az ha eddig nem is foglalkozott programozással, nyugodtan

nekiállhat egy valódi programozási nyelvet megismerni. Bár itt csak a programozás leg-alapvetőbb néhány fogását ismertettük, ez jó alapot nyújt ahhoz, hogy egy nagy lélegzet-vétel után, bele vessük magunkat a programozás rejtelseibe. Javasoljuk a PHP-t, vagy a java nyelvet, mivel platformfüggetlenek és mert elsősorban webes alkalmazások készítésére alkalmasak. Márpedig az internet fejlődése még messze nem ért a végére. Hazánk, pedig még csak most áll az általános felhasználók körében lezajló, "internet-robbanás" előtt, ami az EU csatlakozással nagyon rövid idő alatt be fog következni. A java már ki-lépett a webes fejlesztőeszközök köréből is. A jövő programnyelvének tartott, teljesen modern, objektum-orientált nyelvvé vált. De a megfelelő szerverek és programok telepítése után, a PHP is használható helyi gépen történő feladatmegoldásokra. A PHP4-től szintén nagyot lépett az objektumorientáltság felé, ami ma már alapvető elvárás egy modern nyelvtől. A PHP a shell után, mégis igen ismerősnek fog tűnni. Mindemellett, teljesen fel van készítve a különféle adatbáziskezelésekre is. Akár a Microsoftos vagy akár egy őskövületnek számító dBASE, Clipper, esetleg szükség esetén szöveges alapú adatbázisokat is könnyedén tud kezelni. Arra biztatunk, minden érdeklődőt, hogy senki ne hagyja, hogy a kezdeti nehézségek letörjék az érdeklődését.

Érdemes a Linuxszal foglalkozni, mert egyre többször és több helyen fogunk vele talál-kozni és egyre "felhasználóbarátabb" formában. Nem beszélve arról, hogy jelenleg az UHU-Linux nevezhető a leginkább magyar operációs rendszernek. Mint Linux disztri-búció, pedig teljesen magyar fejlesztés.

Tisztelettel, Raffai Gábor István alias Glindorf

18.1.15. Jegyzet

Xdialog 2.0.6 A leírást, Raffai Gábor István alias Glindorf készítette.

NÉV Xdialog - grafikus dialógus ablakokat jelenít meg scriptekben (a továbbiakban ablakok helyett, dialógus dobozokról beszélünk).

SZINTAXIS Xdialog [< általános opciók >] [< ideiglenes opciók >] < dialogusdoboz opciók >

...

LEÍRÁS Az Xdialog segítségével, könnyen kezelhetővé, felhasználóbaráttá és látvá-nyosabbá tehetjük scriptjeinket. Programozási ismeretek nélkül készíthetünk gra-fikus felületű segédprogramokat. Bármit, amit a shell lehetővé tesz. Az Xdia-log, grafikus dialógusdobozokban kommunikál a felhasználóval. A shellből in-formációkat közölhetünk és kérdéseket tehetünk fel vele a felhasználónak, majd a dialógus-doboz, a válaszokat át adja a scriptnek.

OPCIÓK Az általános, az ideiglenes és a dialógus opciókkal határozhatjuk meg a doboz működését és adhatunk át a shellből a dialógus ablaknak. Mint a "[]" zárójelek is mutatják, az általános és az ideiglenes opciók megadása nem kötelező, ekkor az alapértelmezés szerint történik a végrehajtás.

Általános opciók: Az < általános opciók > -al, a dialógus-dobozok, (a következőkben csak "dobozok",) általános működését, stílusát és kinézetét szabhatjuk meg. Az itt beállított tulajdonságok, öröklődnek a következő dobozokra és csak a beállítás ellenkezőjének megadásával törölhetőek.

Lehetséges általános opciók]

–wclass <név> –rc-file <gtkrc fájlnev>

A kinézetet meghatározó más gtkrc fájl adható meg.

–backtitle <másodlagos felirat>

Ez a szöveg az ablakfejléc alá kerül. Ettől egy vonallal elválasztva kezdődik a valódi szöveg.

–title <ablakfelirat>

Az ablak fejléc-feliratát határozza meg.

–allow-close | –no-close

Bezárhatóvá ill. nem-bezárhatóvá teszi az ablakot. –no-close esetén, csak szabályosan, OK, Cancel, stb. módon zárható be az ablak.

–screen-center | –under-mouse | –auto-placement

Az ablak a képernyő közepén, vagy az egérkurzor felett, illetve a automatikusan kiválasztott helyen jelenik meg.

–center | –right | –left | –fill

Az ablakban megjelenő szöveg igazítása állítható be.

–no-wrap | –wrap –cr-wrap | –no-cr-wrap –stderr | –stdout

Azt határozza meg, hogy a felhasználó választát, melyik kimenetre küldje. Alapértelmezett a hiba-kimenet. A legtöbb esetben érdemes a normál kimenetre állítani (–stdout), mert az alapértelmezett a 2-es, hibakimenet.

–separator <karakter> | –separate-output

Mivel a dialógusdoboz csak 1 soros formában adja vissza a válaszokat, azon dobozok esetén, ahonnan több felhasználói adat érkezik a scripthez, beállítja, hogy a más-más mezők, milyen karakterrel legyenek elválasztva. Alapértelmezésben "/" jel. Ha azt szeretnénk, hogy sorokba rendezve kapjuk meg a válaszokat, akkor a "\n"-t kell beállítani.

–separator ";" esetén pl. a következőképpen tehetjük külön-külön változóba a visszkapott értékeket:

```
valasz1resz='echo "$valasz" | cut -d\; -f1`
valasz2resz='echo "$valasz" | cut -d\; -f2`
valasz3resz='echo "$valasz" | cut -d\; -f3`
```

Az alapértelmezett "/" szeparátor esetén és olyan szeparátoroknál, amiknek nincs különleges jelentősége a shell részére, cut-nál nem kell "\". Azaz elég:

```
valasz1resz='echo "$valasz" | cut -d/ -f1`
```

–buttons-style default|icon|text.

A nyomógombok stílusát határozza meg határozza meg.

Ideiglenes opciók:

Ezek az opciók az Xdialog parancssorban ezután szereplő *widget*-ek megjelenését módosítják.

–fixed-font

Fixed font használata.

–tailbox, –textbox és –editbox esetén.

–password

A dialógus doboz utolsó 1 vagy 2 mezőjét, jelszómezőnek állítja be. Ekkor a begépelés során ezekben csak "*" karakter látszik. Alkalmas jelszó megadására, beállítására, megváltoztatására.

–2inputbox vagy –3inputbox esetén.

–editable

Választólista esetén engedi az egyéni adatbeírást is, nem pedig csak a felkínált választhatóságokat. Csak –combobox esetén.

–time-stamp | –date-stamp

–logbox esetén az időbélyeget határozza meg.

–reverse

Fordított időrendben naplóz a –logbox -ban.

–keep-colors

A log színezése –logbox esetén.

–interval <timeout>

Ez az opció a beviteli boxok, combo boxok, tartomány boxok, list boxok, naptár és időboxok esetében használatosak.

–no-tags

A tagok sorszámozását kapcsolja ki –menubox, –checklist és –radiolist dobozokban.

–item-help

A –menubox, –checklist, –radiolist, –buildlist és –treeview dobozok esetén használhatóak. Ekkor egy-egy tag esetében, nem csak 2 paraméter (a tag és a szöveg,) hanem egy harmadik "help" is. Ez a doboz alsó részében megjelenő szöveg lesz, amely annak megfelelően változik, hogy éppen melyik tag van kijelölve.

–default-item <tag>

Beállítja a –menübox -ban az alapértelmezett választást.

–icon <xpm filename>

A beállított képet mint ikont, kihelyezi a megadott szöveg elé.

–no-ok

Megszünteti az OK gombot. –tailbox és –logox esetén.

–no-cancel

Megszünteti a Cancel, vagy Mégsem gombot. –infobox, –gauge és –progress esetén hatástalan.

–no-buttons

Minden gombot megszüntet.

–textbox, –tailbox, –logbox, –infobox esetén a Help és a Print gombokat, –fselect és –dselect esetén pedig az "Új könyvtár", a "Fájl törlése" és a "Fájl átnevezése" gombok nem jelennek meg.

–default-no

Az alapértelmezett gomb, legyen a "Mégsem" vagy "Nem" gomb.

–wizard opció használata esetén nem lehetséges.

–wizard

"Varázsló" stílusban jelenik meg három gomb (Előző, Mégse, Következő). –msgbox, –infobox, –gauge és –progress esetén hatástalan.

–help <help>

Egy súgó gombot jelenít meg a dobozon, amit ha megnyomunk, akkor egy msgbox-ban a `-help` paraméter után megadott szöveg jelenik meg. Majd OK után visszatér az eredeti doboz.

`-infobox`, `-gauge` és `-progress` esetén hatástalan.

`-print <printer>`

`-tailbox`, `-textbox` és `-editbox` esetén határozható meg vele a nyomtató.

`-check <címke>`

Egy ellenőrző jelölőnégyzetet jelenít meg a dobozon. A doboz visszaadja, hogy kipipálta-e a felhasználó. A doboz által visszaadott értéktől egy `"\n"` azaz enterrel elválasztva kerül átadásra. De ha a válasz a scriptben egy változóba van irányítva, ott már csak egy szóköz jelenik meg köztük. Itt kétféle érték jelenhet meg, `"checked"` vagy `"unchecked"`.

`-infobox`, `-gauge` és `-progress` esetén hatástalan.

`-ok-label <címke>`

Az OK gombon az itt megadott szöveg jelenik meg.

`-cancel-label <címke>`

A Cancel/Mégsem és a No/Nem gombon az itt megadott szöveg jelenik meg. `-wizard` opció esetén hatástalan.

`-beep` Bípel egyet, a doboz megjelenésekor.

`-beep-after` Bípel egyet, a doboz becsukódásakor.

`-begin <Yorg> <Xorg>`

Ez az opció szinte az összes widget esetén alkalmazott.

`-ignore-eof`

`-infobox` és `-gauge` esetén, nem vesz tudomást a fájlvégi EOF-ról. Ugyanis az megszakítaná a működését.

`-smooth` `-tailbox` és `-logbox` esetén a teljes állományt beolvassa a memóriába, egyszerre teszi lehetővé az egész állományban a lapozást, scrollozást, de ez jelentsen is teszi.

A dialógus doboz:

Ezek a dialógus doboz típusát és pozícióját, ill. méretét határozzák meg. Itt történik a fő paraméterek átadása is. Az doboz által feldolgozandó karaktersorozat, vagy filenév. Az doboz pozíciójának, méreteinek meghatározása. `< Szélesség > < Magasság >` Így, karakter mérettel történik a meghatározása. (pl. 10 20) `< Szélesség >x< Magasság >` Így, képernyőpont mérettel történik a meghatározása. (pl. 160x300). A `<0> <0>` esetén

automatikusan, a doboz tartalma alapján történik a szükséges méret meghatározása. A `<-1>` esetén, a doboz, a teljes képernyőt kitöltve jelenik meg.

Paraméterek átadása.

Sajnos azt vettük észre, hogy nem használható a felkiáltó jel `!`. Ugyanis a `bash` azt különleges jelként értelmezi. Ez nem is lenne baj, hiszen a shell számára különleges jelentőségű karakterek ilyenén voltát meg lehet szüntetni, ha elé egy `"\"` jelet gépelünk. Sajnos a `"\!"` esetén viszont, a `"\"` jel is megjelenik a szövegben. A bevitt szövegben a sortörést a `"\n"` karakterek beírásával érhetjük el. "Ez a bevitt\nszöveg, amit\n megtördeltünk."

Ez így fog megjelenni: Ez a bevitt szöveg, amit megtördeltünk.

Háromféle kimenetre kaphat választ a script az doboztól:

1. A hibakimenet: `2>` vagy `>&2`
2. A normál kimenetre: `>` vagy `1>` vagy `>&1`
3. A shell visszatérési változójában: `$?` A választ általában, vagy egy fájlba, vagy egy változóba szeretnénk tenni.

A következőképpen tehetjük meg ezeket:

1. Fájlba:

```
Xdialog --inputbox "ez a kérdés" 0 0 > fájl
```

2. Változóba:

- a.) `valasz='Xdialog --inputbox "ez a kérdés" 0 0 2>&1'`
- b.) `valasz='Xdialog --stdout --inputbox "ez a kérdés" 0 0'`
- c.) A példascriptek legtöbbször, a következő megoldást mutatják be:
`Xdialog --inputbox "ez a kérdés" 0 0 2> /tmp/ideiglenesfile.$$ valasz='cat /tmp/ideiglenesfile.$$'`

Az arra vonatkozó adatot pedig, hogy a doboz mi módon lett bezárva, a `$?` program-visszatérési változóból tudhatjuk meg. Ennek értéke, 0, 1, 2, 3, vagy 255 lehet.

Jelentéseik:

0 OK, Yes/Igen vagy Next/Tovább gomb megnyomása történt.

1 Cancel/Mégsem vagy No/Nem gomb megnyomása történt.

2 A Help gomb megnyomása történt

3 Previous/Előző gomb megnyomása történt (csak-wizard opció esetén).

255 Hibával zárult be, nem pedig normál módon. Ha a dialógus doboz, "doboz-bezárás"-sal let bezárva, akkor is ezzel az értékkel tér vissza.

Lehetséges dialógus dobozok és a megadható paraméterek:

–yesno <karakter sor> <magasság> <szélesség>

Megjeleníti a szöveget és egy "Igen" meg egy "Nem" gombot. A válasz a \$? változóból tudható meg:

```
Xdialog --yesno "Indulhat az akció ?" 0 0
valasz=$?
if test $valasz = 0;
then
<igent nyomott>
fi
if test $valasz = 1;
then
<nemet nyomott>
fi

if test $valasz = 255;
then
<bezárta az ablakot>
fi
```

–msgbox <karakter sor> <magasság> <szélesség>

Egy üzenetet jelenít meg, egy "OK" gombbal.

```
Xdialog --title "Üzenet !!!" --msgbox "Üzenem hogy, ...?" 0 0
```

–infobox <karakter sor> <magasság> <szélesség> [<időmeghatározás>]

Szintén egy üzenetet jelenít meg, de meghatározható, hogy a doboz egy adott idő elteltével, magától bezáródjon. Az időmeghatározás egy szám, mely ezredmásodpercben határozza meg az időt. Amint a "[]" zárójel is jelzi, megadása nem kötelező. Amikor nem adunk meg semmit, akkor az alapértelmezett 1000, azaz 1 mp lép érvénybe.

–gauges <karakter sor> <magasság> <szélesség> [<százalékszám>]

Egy folyamat előrehaladását ábrázolja mértéksávon jelezve. A grafikus sáv jelzi a százalékot, ahol a folyamat tart. A folyamat előrehaladottságának százaléértékét a szabványos bemenetről kapja meg. Amennyiben újabb százaléértéket kap, frissíti a sávot, így jelzi az új százaléértéket. Ha a bemeneten az "XXX" karaktersort kapja meg, akkor az ez után a szabványos bemenetről jövő karaktereket, a sáv felett, mint szöveget jeleníti meg, felül írva az előző, itt helyet foglaló szöveget. Ez a következő "XXX" karaktersorig történik. Onnan újra mint százaléértékként értelmezi a bemeneten kapott számot. A bemeneten érkező első EOF jelre befejeződik a működése. Ezt az `--ignore-eof` kapcsolóval lehet elkerülni. Ha a folyamat túl gyors lenne és szeretnénk követhetővé tenni, használjuk közben a `sleep <mp>` parancsot, ahol a megadott másodpercre felfüggeszti a shell a futást.

```
(sleep 3 echo "25" echo "XXX"; echo "huszonöt százalék"; echo "XXX"
sleep 3 echo "50" echo "XXX"; echo "ötven százalék"; echo "XXX" sleep 3
echo "75" echo "XXX"; echo "hetvenöt százalék"; echo "XXX" sleep 3 echo
"100" echo "XXX"; echo "száz százalék"; echo "XXX" sleep 3 ) | Xdialog
--gauge "nulla százalék" 0 0
```

`--progress <karaktersor> <magasság> <szélesség> [<maxdots> [[-]<msglen>]]`

Ez szintén egy folyamat előrehaladását jelzi ki mértéksávon. Nem kell, hogy a százaléértékeket szám formájában kapja.

```
find "$HOME" *.mp3 | Xdialog --progress "mp3 keresés" 0 0

--inputbox <karaktersor> <magasság> <szélesség> [<alapérték>]

valasz='Xdialog --stdout --title "Adatbevitel" --backtitle "Név
bevitel." -inputbox "Írd be a teljes nevedet." 0 0 "Mr. "'
```

Egy egymezőös adatbeviteli dobozt jelenít meg. A `<karaktersor>` lesz a kérdés. Az `<alapérték>` jelenik meg a beviteli mezőben.

`--2inputbox <karaktersor> <magasság> <szélesség> <cimke1> <alapérték1> <cimke2> <alapérték2>`

Két darab beviteli mezőt jelenít meg. Mivel itt elvileg két értékkel tér vissza a doboz (kereszt és vezetéknev,) csak hogy a visszatérés egyetlen sorban történik, ekkor alapértelmezésben egy `"/"` jel választja el a két értéket. Pl. Kovács Gábor esetében a `$valasz` változóba ez kerül: "Kovács/Gábor"

A `"/"` helyett más szeparátor jel is meghatározható a `--separator <x>` kapcsolóval. pl

`--separator ";"` vagy a `--separator "\n"`

esetén egy enterrel választja el a két mező értékét. Felhasználható jelszó bekérésre is.

```
valasz=`Xdialog --stdout --separator ";" --title
"Adatbevitel" --backtitle "Név bevitel." --2inputsbox "Írd be a teljes
nevedet." 0 0 "Keresztnév:" "" "Vezetéknév:" "Mr. "`
```

```
--3inputsbox <karaktersor> <magasság> <szélesség> <cimke1> <alapérték1>
<cimke2> <alapérték2> <cimke3> <alapérték3>
```

Ugyanaz mint a `--2inputsbox`, de 3 mezővel. Használható pl. jelszó módosításra, vagy új jelszó bekérésére. Mint a `--2inputsbox` esetében, itt is beállítható, hogy mely mezőkbe történő adatbevitelnél, legyenek láthatók, csak "*" karakterek. Ezt a `--password` kapcsolóval lehet elérni. Egyszeri beírásakor, az utolsó mező, kétszeres beírásakor az utolsó két mező, háromszori esetén mindhárom mezőben csak "*" karakterek lesznek láthatóak. A `--password` esetén, alul látható egy jelölő négyzet, "A gépelt szöveg elrejtése" felirattal. Ha ezelől kivesszük a pipát, akkor "*" helyett, láthatóvá válnak a betűk.

```
valasz=`Xdialog --stdout --separator ";" --password --password --title
"Beállítások" --backtitle "Jelszóbeállítás" --3inputsbox "Kérem adja meg
a kívánt jelszót" 0 0 "User név:" "" "Jelszó:" "" "Jelszó megerősítése:"
""`
```

```
valasz=`Xdialog --stdout --separator ";" --password --password
--password --title "Beállítások" --backtitle "Jelszó módosítás"
--3inputsbox "Kérem módosítsa a jelszavát" 0 0 "Régi jelszó:" "" "Új
jelszó:" "" "Az új jelszó megerősítése:" ""`
```

```
--combobox <karaktersor> <magasság> <szélesség> <választás1> <választás2> ... <vá-
lasztásN>
```

Lenyíló választólista mezőt hoz létre. A `<választásN>` -ben megadott szövegek lesznek kiválaszthatóak. A kiválasztott szöveggel tér vissza a scripthez, nem pedig annak sorszámaival.

```
valasz=`Xdialog --stdout --combobox "Válasszon ki egy elemet." 0 0 "Első
tag" "Második tag" "Harmadik tag"` --rangebox <karaktersor> <magasság>
<szélesség> <minimum érték> <maximum érték> [<alapértelmezett érték>]
```

Ez egy csúszkát jelenít meg, amelyen egy numerikus érték adható meg. Megadható neki a legkisebb, a legnagyobb és az alapértelmezett érték.

```
valasz=`Xdialog --stdout --rangebox "Hány éves ön ?" 0 0 "18" "90" "30"`
```

```
--2rangesbox <karaktersor> <magasság> <szélesség> <cimke1> <minimum1> <maxi-
mum1> <alapé1> <cimke2> <minimum2> <maximum2> <alapé2>
```

Mint a `--rangebox` esetében, de két csúszkát jelenít meg egy dobozban. Mindegyikhez külön felirat, maximum és minimum, valamint alapérték rendelhető, a scripthez a szokásos `--separator` -ral elválasztva egy sorban tér vissza az érték.

`-3rangesbox <karaktersor> <magasság> <szélesség> <cimke1> <minimum1> <maximum1> <alapé1> <cimke2> <minimum2> <maximum2> <alapé2> <cimke3> <minimum3> <maximum3> <alapé3>`

Mint a `-rangebox` esetében, de három csúszkát jelenít meg egy dobozban. Mindegyikhez külön felirat, maximum és minimum, valamint alapérték rendelhető, a scripthez a szokásos `-separator` -ral elválasztva egy sorban tér vissza az érték.

`-spinbox <karaktersor> <magasság> <szélesség> <minimum> <maximum> <alapé> <címke>`

Mint a `-rangebox` -nál, de itt az egyesnél is megadható címke. Itt viszont nem egy csúszkán, hanem egy kicsi értékbeállító spin-nel adható az meg. Itt nagyobb érték is beírható mint a maximumban meghatározott, viszont a scriptnek akkor is csak a maximumban meghatározott értéket adja vissza.

`-2spinsbox <karaktersor> <magasság> <szélesség> <minimum1> <maximum1> <alapé1> <cimke1> <minimum2> <maximum2> <alapé2> <cimke2>`

Mint a `-rangebox`, de két spin dobozzal.

`-3spinsbox <karaktersor> <magasság> <szélesség> <minimum1> <maximum1> <alapé1> <cimke1> <minimum2> <maximum2> <alapé2> <cimke2> <minimum3> <maximum3> <alapé3> <cimke3>`

Mint a `-rangebox`, de három spin dobozzal.

```
valasz=`Xdialog --stdout --spinbox "Kérem módosítsa a jelszavát" 0 0
"1900" "2003" "1970" "Születési év" "1" "12" "6" "A születés hónapja"
"1" "31" "15" "A születés napja"
```

`-textbox <file> <magasság> <szélesség>`

Egy szöveges fájl tartalma jeleníthető meg vele. Egy ideiglenes fájlon keresztül a szabványos kimenet is megjeleníthető vele. A `-no-buttons` kapcsoló nem használható nála. Ha a fájlnev helyett a `"-"` adjuk meg, akkor a szabványos bemenetről olvas, az a pipeline-nal más programoktól, parancsok szabványos kimenetéről tudja átvenni a megjelenítendő szöveget (lásd.: `-editbox` példa).

```
mplayer --help > /tmp/boxtmp.$$
Xdialog --textbox "/tmp/boxtmp.$$" 0 0
```

`-editbox <file> <magasság> <szélesség>`

Hasonló mint a `textbox`, de szerkeszteni is lehet a fájlt vele. Az alábbi módon a felhasználó általi változtatások elmentésre kerülnek a fájlba.

```
cat file.txt | Xdialog --stdout --editbox "-" 0 0 > file.txt
```


`-tailbox <file> <magasság> <szélesség>`

Majdnem ugyanolyan mint a `-textbox`, de jelentősen gyorsabb scrollozást, megjelenítést tesz lehetővé. Ugyanakkor itt már használható a `-no-buttons` kapcsoló is. A "-" fájlnevként való megadásával ugyanúgy képes a szabványos bemenetről olvasni (pipeline). `-textbox` helyett inkább ez tűnik jobb választásnak.

`-logbox <file> <magasság> <szélesség>`

A `logbox` leginkább annyiban különbözik a `tailbox`-tól, hogy a `-time-stamp` kapcsolóval idő bejegyzést szűrhatunk minden sor elé, a `-date-stamp` -pal pedig a dátumot is. Illetve a `-keep-colors` kapcsolóval színezhető a megjelenítés.

`-menubox <karaktorsor> <magasság> <szélesség> <menü magasság> <azonosító1> <menüpont1> <help1>`

Egy függőleges menüt jelenít meg. A menü magasság egy szám ami azt jelenti, hogy egyszerre csak az itt megadott számú menüpont látszik, a többi görgetéssel lehet elérni. Egy menüponthoz két adat adható meg, de ha az `-item-help` kapcsoló használva van, akkor egy harmadik is. De csak akkor (ezt jelöli a zárójel). A három adat:

1. a menü azonosítója (ésszerűnek tűnik itt sorszámokat használni). Ezt az értéket fogja az ablak átadni a scriptnek. 2. A menüpont szövege. 3. A menüpont help-je, vagy többet információt róla. Ez a doboz alsó részében, de a gombok felett jelenik meg.

A `-no-tags` kapcsolóval, elrejtethetők a menüpontok azonosítói.

```
valasz=`Xdialog --no-tags --item-help --menubox "Válasszon a menüből." 0
0 3 "1" "Első pont" "első pont segítsége" "2" "Második pont" "második
pont segítsége" "3" "harmadik pont" "harmadik pont segítsége" "4"
"negyedik pont" "negyedik pont segítsége" "5" "ötödik" "ötödik pont
segítsége" `
```

`-checkboxlist <karaktorsor> <magasság> <szélesség> <lista magasság> <azonosító1> <elem szöveg1> <státusz1> <help1>`

Egy jelölőnégyzet listát ad. Itt a szöveg előtti rész pipálható ki, vagy vehető ki a pipa előle. Működése hasonló mint a `-menubox` -é, annyi különbséggel, hogy mivel itt több elem is kiválasztható, kipipálható, ezért a doboz a visszatérési értékben a `-separator` és a `-inputbox` -nál ismertetett módon adja vissza a kijelölt elemek azonosítójának, a szeparátor karakterrel elválasztott listáját. A `-menubox` -hoz képest, viszont listaelemként bővül eggyel a megadandó paraméterek száma.

1. jelölőnégyzet listaelem azonosítója. 2. Az elem szövege. 3. Az elem státusza (ez a plusz) 4. A jelölőnégyzet elem helpje (csak `-item-help` esetén).

A 3. paraméterben, az elem státuszánál, háromféle érték adható meg.

1. "off": az elem a doboz megnyílásakor nincs kijelölve. 2. "on": az elem a doboz megnyílásakor kivan jelölve. 3. "unavailable" Ez azt jelenti, hogy az elem megjelenik, de

nem-kiválasztható (–item-help esetén a help szövegeket nem a doboz alján, hanem ha az egérrel 0.5mp-re megállunk az elem felett, akkor jeleníti meg, egy kis címkén az egér mellett).

```
valasz='Xdialog --stdout --no-tags --item-help
--checklist "Pipálja ki a megfelelőket." 0 0 3 "1" "Első pont" "off"
"első pont segítsége" "2" "Második pont" "on" "második pont segítsége"
"3" "harmadik pont" "on" "harmadik pont segítsége" "4" "negyedik pont"
"unavailable" "negyedikpont segítsége" "5" "ötödik" "off" "ötödik pont
segítsége" `
```

–radiolist <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1>
<elem1> <státusz1> <help1>

A rádiógomb listát ad ki. Minden úgy működik mint a –checklist esetében, azzal a különbséggel, hogy mivel itt egyszerre csak egy elem jelölhető meg, ezért a státusz megadásakor is csak egy lehet "on" (az –item-help itt is egér melletti címke).

–buildlist <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1>
<elem1> <státusz1> <help1>

Ez egy olyan dobozt jelenít meg, amiben két lista látható és az egyik listából a másikba helyezhetünk át elemeket. Végül is egy listából lehet több elemet kiválasztani vele. Az "on" státuszúak, rögtön a kiválasztottak között vannak. A doboz a kiválasztottak listájában lévő elemek azonosítóit adja vissza, a szeparátorral elválasztva (az –item-help itt is egér melletti címke).

–treeview <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1>
<elem1> <státusz1> <elem1 mélység> <help1>

Ez fastruktúra szerűen jeleníti meg az elemeket, amik így egymásba ágyazhatóak. Ezért egy ötödik paraméter lép be, az elem mélység. Itt oda kell figyelni, hogy az elemek és a mélység értékek, helyesen kövessék egymást. Úgy kell a sorrendnek következnie, mint a minden szálát kibontott állapotban lévő fastruktúrának.

–fselect <file> <magasság> <szélesség> Ez egy jól ismert fájlkiválasztó doboz. A file paraméter, az egy szűrő. Csak az ennek a szűrőnek megfelelő fájlokat fogja mutatni. Ha azt akarjuk, hogy mindent mutasson, "*"–ot kell neki megadni. Lehetőség van könyvtár létrehozására, fájl törlésére és fájl átnevezésére is.

A –no-buttons kapcsoló, csak ezeket a lehetőségeket veszi el. A –check is használható. A fájl teljes elérési útjával és nevével tér vissza a scripthez. A –check egy "\n", azaz enterrel elválasztva a fájlnevtől, kerül a kimenetre.

```
fajl='Xdialog --help "legyél ügyes és válassz ki egy fájlt." --check
"Meggyőződteél a helyes kiválasztásról ?" --fselect "*.gz" 0 0`
```

–dselect <directory> <magasság> <szélesség>

Mint az `-fselect`, de könyvtár jelölhető ki vele.

`-calendar <karaktersor> <magasság> <szélesség> <nap> <hónap> <év>`

Egy naptárat jelenít meg. Megadható, hogy milyen dátummal induljon. Az évnek minimum 1970-nek kell lennie. "nap/hónap/év" formában adja vissza a beállított dátumot.

`-timebox <karaktersor> <magasság> <szélesség>`

Egy időpont bevitelére alkalmas doboz. Három spin elem jelenik meg rajta, az óra, a perc, és a másodperc. Alapértéke az aktuális idő. Visszatérni, "óra:perc:mp" formában tér vissza.

Példa scriptek a következő elérési úton találhatóak:

```
/usr/local/share/doc/Xdialog-2.0.6/samples/* MAN angol :  
/usr/local/man/man1/Xdialog.1* Dokumentumok angol nyelven :  
/usr/local/share/doc/Xdialog-2.0.6/*
```

18.2. Az sed áradatszerkesztő

18.3. Perl-ről röviden

Viszonylag régóta, három éve foglalkozom a Perl nyelvvel, több mint két éve pedig aktívan fejleszték webes alkalmazásokat benne. Ezen a tanfolyamon a Perl nyelv alapjaival fogunk megismerkedni. A cél nem a teljes nyelv bemutatása, hanem ízelítő nyújtása mindabból, ami rendelkezésünkre áll, feltéve ha mellette döntünk.

A Perltől sokan riadnak vissza alaptalanul, mert azt tartják róla, hogy nehezen tanulható, nagyon tömör, zavaros nyelv. Ennek a tanfolyamnak egy másik céljával választottam azt, hogy megmutassam, hogy ez korántsem így van, és valójában egy nagyon frappáns, rugalmas és barátságos nyelvről van szó.

18.3.1. Bevezető

A Perl nyelvről jelentős, azonban leginkább csak angol irodalom áll rendelkezésre. A legkorábbi magyar anyag, amit ismerek, Verhás Péter munkája (<http://www.verhas.hu/peter/perlh/main.html>). Ezen kívül még pár könyv jelent meg a témakörben, melyek hiánycikknek számítanak, ilyen például a Panem kiadó által megjelenített könyv is. Bár ez számos hibát tartalmaz, de kiindulásként használható. A legutóbbi munka a <http://www.prog.hu> oldalon megjelenő cikksorozat, ebben Barabás Péter ismertette a nyelv alapjait. További anyagokra linkeket a <http://www.perl.org.hu> címen találhatunk.

A számítógépes nyelvek jelentős múltra tekintenek vissza. Kialakulásuk során egyre finomodtak, átvettek ötleteket egymástól, beépítettek újabb és újabb nyelvi technológiákat magukba. Mára elmondható, hogy általában ezek a (modernebb) nyelvek már nem abban különböznek, hogy mire képesek, sokkal inkább abban, hogy milyen könnyen valósíthatóak meg bennük az egyes megoldások, milyen támogatottság van mögöttük. A Perl ebből a szempontból előkelő helyen szerepel, jelentős programozói bázisa van, és nagyon sok modul, mások által már megírt programkód szerezhető be, tölthető le hozzá (pl.: <http://www.cpan.org>).

A PERL betűszó jelentése: Practical Extraction and Report Language (kivonatok és jelentések készítésére használható nyelv). Ez az elnevezés onnan jött, hogy kezdetekben szövegfile-ok (Linux alatt különböző rendszerállományok, eseménynapló, stb.) gyors és egyszerű feldolgozásának megkönnyítése volt a cél. Ez azonban csak a nyelv indulásakor volt igaz, köszönhetően népszerűségének, manapság sokkal szélesebb körben lehet használni, ahogyan azt majd a későbbiekben látni fogjuk.

A Perl az egyszerű problémák egyszerűen történő megoldására készült, anélkül, hogy lehetetlenné tenné összetett programok írását. Egy adott problémát sokféleképpen oldhatunk meg benne, tudásunktól, felkészültségünkől függően. Erre a Perl-es közösség ma is büszke, nagyon gyorsan, nagyon látványos programokat lehet írni vele. A nyelv hangulatára jellemző, hogy nevének feloldására ismert egy másik kifejezés is: Pathologically Eclectic and Rubbish Lister, azaz betegesen túldíszített szemét listázó - mindenki döntse el a következőkből, hogy melyik fedi jobban a nyelv lehetőségeit!

18.3.2. A nyelv erősségei

Gyors fejlesztés A Perl félig interpreteres nyelv (ami azt jelenti, hogy a forráskódot futtatjuk, s nem először egy fordítóprogramot használunk, majd a végrehajtható, már gépkódot futtatjuk), ezért nem igényel fordítást, egyből futtatható a forráskód. Beépített lehetőségeivel egyszerű a hibakeresés. A Perl-nél ráadásul egy "csalással" is szembetaláljuk magunkat, ha egy kicsit megvizsgáljuk a futtatás folyamatát: induláskor a futtató környezet félig feldolgozza a forráskódot, ennek eredményeképpen egy gépi kódhoz közeli nyelven lesz elérhető a program, így a Perl interpreteres volta nem válik egyben hátránnyá is. Ez a működés különben nagyon hasonlít mind a Java, mind a .NET környezetekhez, azzal a különbséggel, hogy ott a fordítás nem futtatáskor, hanem előzetesen kell, hogy megtörténjen.

Lehetőségek Telepítése után reguláris kifejezései segítségével a szövegfeldolgozás mindennél egyszerűbb, továbbá objektum orientált, adatbázis-kezelő és hálózati programozási lehetőségek széles tárháza áll rendelkezésünkre, az Internetről le tölthető modulok segítségével pedig semmi sem szab határt a nyelv lehetőségeinek.

Tanulhatóság A Perl alapjai nagyon egyszerűek, fokozatosan tanulható a nyelv. Egy adott feladatra szinte mindig többféle megoldási lehetőséget ad, lehetőséget nyújtva a kezdőnek, hogy egyszerűen, a haladónak, hogy elegánsan oldja meg az adott problémát.

Hordozhatóság Minden elterjedt platformon futtatható. Elterjedt módszer, hogy a fejlesztés Windows környezetben zajlik, a program azonban végül Unix környezetben fog futni - ehhez (persze megfelelő körülményekkel, és ha nem tartalmaz operációs rendszer függő részeket a program) semmilyen módosítás nem szükséges a forrásban.

Ár Ingyenes. Segítséget is gyorsan és szintén ingyen kaphatunk, ha valakit megkérdezzünk egy levelezőlistán, s általában még jobbat is, mintha fizetnénk érte.

18.3.3. A nyelv kialakulása, fejlesztése

A nyelv fejlesztését Larry Wall kezdte meg 1987 környékén, az első kiadás 1987 decemberében látott napvilágot. Az 1987-es indulás következtében igen jelentős múltra tekint vissza, ma már több ezer önkéntes is fejleszti. Röviden összefoglalva: Larry nagyot alkotott. Ha más nagy programozóhoz szeretnénk hasonlítani a teljesítményét, akkor Ken Thompson (UNIX), vagy Dennis Ritchie (C) neve jöhet szóba. A scripting nyelvek között a Perl népszerűsége a C-vel hasonlítható össze.

Kezdetben a nyelvet feleségéről Gloria-nak nevezte el, a későbbiekben azonban mivel ez feszültségek forrása lehetett volna ("Már megint milyen béna ez a Gloria!"), más név után nézett. Először a PEARL (magyarul: gyöngy) merült fel, mint lehetőség (ahol az eredeti angol jelentésében a rövidítésnek az And szót jelenti az A betű), azonban mivel már volt ilyen néven egy grafikus programnyelv, ezért maradt a Perl rövidítés.

A második verzió magával hozta a reguláris kifejezések használatát, a hármas verziótól kezdve a program képes bináris állományok kezelésére is. A négyes verziószámnak külön története van: az első O'Reilly (egy nagyon színvonalas kiadványokat megjelentető, ma is híres) kiadó által megjelenített 'tevés' Perl könyv megjelenésére való tekintettel adták ki (mely könyvet mellesleg Larry írta).

A jelenlegi széles körben használt változat a Perl 5, mely 1994 októberében jelent meg. A 4-es verzió tulajdonságait megtartva teljesen újraírták, modularizálták, optimalizálták és kijavították a hibáit, továbbá kiegészítették objektum-orientált nyelvi lehetőségekkel. A nyelv maga ennek ellenére nem sokat változott, az újdonságok kivételével szinte kivétel nélkül megegyezik az előző verzióval. A továbbiakban ennek a változatnak a lehetőségeit kívánom majd bemutatni, hiszen ez az a verzió, amihez ma hozzá lehet jutni, ezt tartalmazzák a különböző Linux terjesztések is.

A történethez hozzátartozik még, hogy ennek a verzióknak is vannak alverziói, melyek napjainkban is folyamatosan jelennek meg. A jelenlegi, e tananyag írásakor legfrissebb

a Perl 5.8.0-ás. Ebben többek között új, stabil többszálú programozást támogató nyelvi elemek jelentek meg.

Az 5-ös széria folyamatos fejlesztése mellett, 2000 nyarán jelentette be Larry a 6-os széria fejlesztésének kezdetét. Ez nem csak a nyelv újabb dolgokkal való kiegészítését fogja magával hozni, hanem teljes újragondolás után az alapjaiból építi fel a nyelvet. A Perl 6 teljesen más lesz, és mégis nagyon hasonló a Perl korábbi verzióihoz képest. Szét lesz választva a fordítás a futtatástól, egy új, egész, jól tervezett nyelv lesz a végeredmény. A futtató résznek már megvan a neve is: Parrot-nak (magyarul: papagáj) hívják. A Parrott a Perl mellett képes lesz más nyelveken megírt programokat is futtatni, ha a megfelelő fordító elkészül hozzájuk. Jelenleg már elérhető a Parrot-nak egy kezdetleges változata (jelenleg a 0.0.7-es), és több fordító is hozzá (az egyik például Java-t fordít Parrot-ra?).

A nyelv fejlődése és népszerűsége elsősorban három dolognak köszönhető. Az első, és talán legfontosabb, hogy Larry úgy gondolta, hogy ingyenessé és szabad forrásúvá teszi. Ez neki nem jelentett külön munkát, mivel egyszer már saját magának kifejlesztette, ellenben nagyon sok embernek jó volt, hiszen egy nagyon rugalmas nyelvet kaptak, mely nagyon sokféle számítógépes környezetben használható. A másik dolog, hogy figyelembe vette nagyon sok ember kérését, ötletét a nyelv fejlesztése közben, így széles kört elégtett ki a nyelv, és szinte végtelenül rugalmassá vált. Végül a harmadik, mely az elterjedtséget okozta, hogy kiengedte a napi fejlesztést a kezéből. (Nem is olyan meglepő módon, ez nagyon hasonlít a Linux fejlődéséhez :).

A nyelv magát napjainkban a Perl Porters néven ismert csoport fejleszti, a fő irányvonalat, kérdéseket viszont továbbra is Larry határozza meg. Ők fejlesztik az új verziót is, s most is folyamatosan jelenik meg a programozási nyelv weblapján (www.perl.com) egy cikksorozata Larry-nek arról, hogy hogyan fogják módosítani a nyelvet, milyen változásokat vezetnek be, s melyeket vetnek el. A lista a felhasználók kéréslistájára épül, a fontosabbnak számító kérések mind szerepelnek benne. Amennyiben valaki érdeklődik a nyelv fejlődése iránt, mindenképpen ajánlott ezek elolvasása, egyrészt képet kaphat egy programozási nyelv fejlesztésekor felmerülő kérdésekre, továbbá megismerheti a következő verzióban megjelenő újdonságokat, változásokat is.

18.3.4. Programozzunk Perl-ben

Mire használható a Perl? A kérdés helyesebben úgy hangzik, hogy “mire nem?”, hiszen nagyon széles azoknak a lehetőségeknek a listája, amik Perl-ből elérhetőek. Akkor nem ajánlott a Perl használata, ha már van egy jól működő programunk egy másik programozási nyelven, vagy létezik jobb célnyelv az adott problémára, illetve valós idejű, vagy alacsony szintű alkalmazást (mint például egy eszközmeghajtó) szeretnénk írni, vagy ha alkalmazásunk nagyon összetett vagy többszálú, megosztott memóriájú program. Más alkalmazások fejlesztésére általában kiválóan használható a következőkben

bemutatandó tulajdonságai miatt. Ez a fejezet egy rövid áttekintést fog nyújtani a nyelv (és kiegészítéseinek) lehetőségeiről. Célja, hogy segítséges nyújtson a nyelv alapjainak megtanulásához, s kiindulási lehetőséget biztosítson a további tanuláshoz. Sok dolgot nem fogok megemlíteni benne, melyek a nyelv professzionális megismeréséhez nélkülözhetetlenek, mivel úgy gondolom, hogy egyrészt ezek ismerete csak bonyolítaná a magyarázatokat, másrészt túlnyúlik egy ilyen tananyag lehetőségein.

A nyelv lehetőségei

In medias res

Kedvcsinálóként megvizsgáljuk a tipikusnak mondható "Helló Világ!" programot - azaz megízeleljük, hogy milyen is Perl-ben programozni -, s ennek során a legalapvetőbb dolgokat is megismerhetjük. A kezdetként bemutatott programot próbálom minél jobban kivesézni (szinte amennyire csak lehet), s megpróbálok így tenni a bemutatott példák-kal kapcsolatban is folyamatosan. Ezért első olvasásra biztos vagyok benne, hogy sok rész nem lesz érthető. Ilyenkor nyugodtan lépj tovább, majd visszatérsz a későbbiekben. Tehát ez nem teljesen olyan, mint az iskola, ahol egyből mindent meg kell érteni :). Többször átolvasva az anyagot egyre érthetőbbé válik majd, addig is, a célod az legyen, hogy megértsd a nyelv alapjait, és hogy elindulj vele egy kis program megvalósítása felé. Javaslom, hogy próbáld ki mindig a bemutatott példákat, módosítsd és értsd meg, mi miért lehet úgy, ahogy van.

A Perl telepítésére UHU-Linux alatt nincs szükség, egyből rendelkezésünkre áll (hiszen például a Telepítő és a Vezérlőpult is Perl-ben van írva, ezért ott kell lennie). A szerkesztéshez talán a Midnight Commander programot a legegyszerűbb használni, ez egy a jó régi Norton Commander-hez hasonló file manager. Elindítani egy konzolon az *mc* beírásával tudjuk. Fontos még megemlíteni, hogy Linux alatt vagy futtathatóvá kell tennünk a programunkat, hogy el tudjuk indítani (*chmod 770 programneve*), és *./programneve* beírásával kell hivatkoznunk rá, vagy pedig *perl programneve*-t írjunk be. Ha futtatható a program és Midnight Commander alatt Enter-t nyomunk rajta, akkor el fog indulni. Na, de ennyi kitérő után térjünk vissza magához a Perl-hez, egyből a példaprogrammal:

```
#!/usr/bin/perl
print "Hello Világ!\n";
```

Kezdjük tehát a "Helló Világ!"-gal! A fenti program csak két sort tartalmaz, vegyük sorra ezeket, lesz mit nézni rajta! :) Az első sor, amennyiben programoztunk már UNIX rendszerek valamilyen shell-jében, akkor ismerős lehet számunkra. UNIX környezetben azt mondja meg az operációs rendszernek, hogy hol keresse azt a programot, amivel futtatni kell az állomány tartalmát. Ez a sor Windows környezet alatt való fejlesztés esetén

esetleg elhagyható (de például Windows-os Apache webservertől nem, csak megfelelő beállítások esetén), mivel itt az operációs rendszer az állományok kiterjesztését használja erre a célra. Mi történik, ha nem kell, és mégis benne hagyjuk? Egyrészt a Perl futtató figyelmen kívül hagyja: ebben a nyelvben a '#' -al kezdődő sorok a megjegyzéseket jelentik, másrészt pedig UNIX rendszerek alatt is változtatás nélkül le fogjuk tudni futtatni programunkat. Tehát szokjuk meg, hogy ott van.

A második sor a valószínűleg sokaknak ismerősnek tűnő 'print' utasítást tartalmazza. Ebben a formában a képernyőre írja ki a paraméterként átadott konstansokat, változókat (kicsit módosítva állományba írásra is alkalmas). A soron még láthatjuk, hogy egy pontosvesszővel zárul, ugyanis Perl-ben ezzel kell elválasztani az utasításokat. A Pascal nyelvhez hasonlóan előfordul, hogy nem kell egy utasítás után írni, de ha megszokjuk, hogy mindig ott van, akkor egy tipikus hibalehetőséget küszöbölünk ki.

A program tehát megjeleníti a képernyőn a "Helló Világ!" üzenetet, majd befejezi futását. Más nyelveknél esetleg megszokott 'end.', illetve hasonló záróutasításra nincs szükségünk.

Maradjunk még egy kicsit a második sornál! A sor végén található '\n' -ről még nem beszéltünk. A '\' jel azt jelenti, hogy egy különleges karakter következik, az 'n' a 'new line', azaz 'új sor' -nak a rövidítése. Emiatt a 'print' utasítás egy soremelést is ki fog írni a képernyőre (próbáljunk ki kétszer, háromszor, figyeljünk arra, hogy mindegyik elé rakjunk egy '\' -t). A '\n' -en kívül fontosabbak a '\r' (return) és a '\t' (tabulátor), de ezeken kívül még számos, másképp nem leírható jelnek létezik e módon feloldása. A következő táblázatban a '\' jel segítségével feloldható jeleket gyűjtöttem össze (az angol elnevezésüket használtam, mivel a legtöbbnek nincs jó magyar megfelelője):

```
\n - newline
\r - return
\t - tab
\f - formfeed
\b - backspace
\v - vertical tab
\a - bell
\e - escape
\007 - egy 8-as számrendszerben leírt ASCII karakter
\x7f - egy 16-os számrendszerben leírt ASCII karakter
\cC - egy "control" karakter
\\ - backslash
\" - double quote
\l - a következő karakter kisbetűs legyen
\L - a következő karakterek kisbetűsek legyenek \E-ig
\u - a következő karakter nagybetűs legyen
```


\U - a következő karakterek nagybetűsek legyenek \E-ig
\E - lezárja a \L-t és a \U-t

Mi van akkor, ha mi azt szeretnénk kiíratni mégis, hogy 'c:\nem' és nem azt, hogy 'c:<újsor>em'? Két fő választási lehetőségünk van: egyrészt használhatjuk a fenti táblázatban is szereplő '\\' jelet - ehelyett csak egy '\' jelenik meg -, vagy van egy másik lehetőségünk is, ha nem szeretnénk hogy a Perl "feloldja" a különleges karaktereinket: használjunk az idézőjel helyett az aposztrófot. A következő példát lefuttatva megtekinthetjük a különbséget:

```
#!/usr/bin/perl

print "Feloldott újsor: >\n<"; print "\n";
print 'Feloldatlan újsor: >\n<'; print "\n";
```

Ezt fogjuk látni:

```
>perl proba.pl
Feloldott újsor: >
>
Feloldatlan újsor: >\n<
```

Még mindig nem fejeztük be, de most már ígérem, az utolsó gondolat jön. :) A Perl nyelvbe ugyan be van építve a 'print' utasítás, de teljesen a később bemutatandó metódusokhoz (függvények és eljárások gyűjtőneve) hasonlít a meghívása, melyről itt ejtenék szót. A legtöbb nyelvben a metódusok paramétereit zárójelek között kell meghívni. Ez a Perl-ben sincs teljesen másképp, azonban mivel eléggé szabad nyelv, ezeket el is lehet hagyni. Azaz a következő példák teljesen működőképesek és ekvivalensek, mindenki eldöntheti, hogy melyik a szimpatikusabb számára:

```
print "Helló Világ!";
print("Helló Világ!");
```

Most hogy kaptunk egy kis ízelítőt a nyelvből, kezdjük az alapokkal! A következőkben - ahogy az az előző példában is volt - általában nem teljes, működő programokat, hanem csak az adott témához kapcsolódó részleteket fogok bemutatni.

Adattípusok

Adattípus alatt azokat a lehetőségeket értjük, amikkel egy változó értéket el tudunk tárolni. Változóknak lehet adattípusuk, és e szerint tudunk bennük különböző dolgokat eltárolni. Egy változó alatt mindig egy memóriadarabot kell értenünk. Ebben lehet eltárolni, és a későbbiekben felhasználni 'valami'-t. A Perl három fő adattípussal rendelkezik: skalár, skalárok tömbje és asszociatív skalár tömbök - ez utóbbit hash-nek is hívják (más nyelvekben esetleg szótárként találkozhattunk velük). Skalár alatt egy számot, vagy egy szöveget (sztringet) értünk (Perlben nem kell és nem lehet megmondani egy változóról, hogy konkrétan egy számot vagy egy sztringet tárol, ellenben a 'típusos' nyelvekkel, ahol meg kell - a Perl emiatt egy típus nélküli nyelv). Tömbök és hash-ek alatt skalárok egy csoportját értjük. A tömböket számmal indexeljük (számmal hivatkozunk az elemeire), az első elemük a '0' indexű, a többi rendre '1', '2', '3'?, a hash-eket pedig skalárral. Tömbök esetén az egyes elemeknek meghatározott sorrendje van (az index-ként használt számok szerint növekvő sorrendben), hash-eknél nincs meghatározott sorrend. A hash-nél azt a skalárt, amellyel hivatkozunk a hash egy elemére, kulcsnak nevezzük, így a hash egy kulcs-érték párokból felépülő párosokat fog tartalmazni.

A változókra a nevükkel tudunk hivatkozni, ennek első karaktere a hivatkozott típust adja meg. Larry nyelvész tanulmányai hatással voltak a Perl-re, ez talán itt a legszembe-tűnőbb. Az egyes változóknál ugyanis megjelenik az egyes szám és többes szám esete. Ha egy konkrét skalárra hivatkozunk, akkor egyes számot kell használnunk, ha több elemre, akkor többes számot. Az egyes szám jele a \$, a többes számé a @, vagy a %. A skalár változók mindig \$-ral kezdődnek, az egyes szám jelölésének köszönhetően, és ha egy tömb vagy egy hash egy elemére hivatkozunk, akkor is \$-t használunk. A teljes tömbre, vagy egy részére való hivatkozást @-al, a hash-eket pedig %-al jelöljük. A változók névrészeinek egy angol betűvel kell kezdődnie, ezután vegyesen (angol) betűk, egyes jelek (pl. aláhúzás) és számok következhetnek.

A változók jelölése tehát a következő:

```
\$napok # egy egyszerű skalár típus 'days'
\$napok[28] # a 29. eleme 'days' tömbnek
\$napok{'Feb'} # a 'Feb' eleme a 'days' hash-nek
\$#napok # a 'days' tömb utolsó indexe

@napok # a 'days' tömb összes eleme
@napok[3,4,5] # a 'days' tömb egy szelete, ugyanaz, mint a @days[3..5]
@napok{'a','c'} # ugyanaz, mint a (\$days{'a'},\$days{'b'})

%napok # (kulcs1, érték1, kulcs2, érték2...)
```

Minden típus saját, úgynevezett "változótérrel" rendelkezik, ami azt jelenti, hogy nyugodtan használhatunk ugyanolyan nevű tömböt, hash-t és skalár változót, a nyelv meg fogja tudni különböztetni őket, amennyiben arra mi is képesek vagyunk. :)

Értékadás

Az értékadások az egyenlőségjel segítségével történnek, a jel bal oldalán egy változó-nevet, a jobb oldalán pedig egy értéket kell megadnunk - természetesen az érték az lehet valamilyen kifejezés (összeadás, szorzás, ?) is. Mind itt az egyenlőség mindkét oldalán, mind majd később a műveleti jelek előtt és után lehetséges szóközők, tabulátorok, vagy akár újsorok használata, akár vegyesen is.

Pár egyszerű példa (láthatunk megoldásokat tömbök és hash-ek értékadására is):

```
# skalár
\ $szoveg1='ez egy szöveg!';
\ $szoveg2='ez egy másik szöveg!\n';
\ $szoveg3="ez egy harmadik szöveg!\n";
\ $szam1=5;
\ $szam2='11';
\ $kifejezes=(1+2)/3*4;

# tömb
@tomb=('első elem', 'második elem', 'harmadik elem');
\ $honapok[3]='április';
@tomb[2..4]=('harmadik elem', 'negyedik elem', 'ötödik elem');

# hash
%hash1=('1. Index', '1. elem', '2. elem', '2. index');
%hash2=('1. index'=>'1. elem', '2. elem'=>'2. index');
\ $hash3{'index1'}='elem1';
```

A skalár értékadások remélem érthetőek, talán az utolsóhoz fűznék hozzá egy rövid magyarázatot: láthatjuk, hogy a matematikában megszokott számításokat is elvégezhetjük az értékadások jobb oldalán. Hogy miért nem jobb annál, hogy egyszerűen leírjuk, hogy a 'kifejezes' változó vegye fel a 0.25 értéket? Azért, mert a későbbiekben láthatóan akár változókat is használhatunk majd a kifejezésünkben.

Tömböknél alapvetően két módszert használhatunk értékadásaink során, vagy egy listát adunk értékül a tömbnek, mint az első példánkban az látható, vagy pedig a tömb egy elemére hivatkozva, annak változtathatjuk meg tartalmát. A harmadik példában a tömb egy tartományának történő értékadásra láthatunk egy példát. A @ jelet használtuk, mivel a jobb oldalon egy lista szerepel, tehát többes számról van szó.

Végül tekintsük át a hash-hez való értékrendelés módszereit. Mint láthatjuk, nagyon hasonlít a tömbhöz, itt is adhatunk egy listát is értékül, vagy egy skalárt. Az első példát megtekintve láthatjuk, hogy amennyiben listát adunk értékül, akkor index-érték párokat kell szerepeltetni a listában. A második példában látható módon ezeket a párokat az => segítségével is megadhatjuk.

Egy változóhoz természetesen hozzárendelhetjük egy másik változó, vagy akár több változó értékét is. A következő példákban erre, illetve a tömbök, hash-ek elemeire való hivatkozásokra mutatok példákat:

```
# skalár
$valt1=$valt2;
$marcius=$honapoktomb[2]; # 0-január, 1-február, 2-március...
$juniusz=$honapokhash{'június'};
$hossz=@honapok; # a hónapok tömb elemeinek a száma
$tavasz=@honapok[2,3,4]; # $tavasz='márciusáprilismájus';

# tömb
@honapok=(@tavasz, @nyar, @osz, 'december', $januar, 'február');
@tomb=(\hash{'első'}, \hash{'második'});
@fordit[0, 1]=@fordit[1, 0];

# hash
%hash=('január'=>$honapok[0], 'február'=>$honapok[1]);
```

A fenti példák szinte teljesen megegyeznek az előző példákban tárgyaltakkal. Nézzük meg azonban a negyedik skalár példát. Itt azt láthatjuk, hogy egy skalárnak egyszerűen egy tömböt adunk értékül. Itt megint, egy az emberi nyelvekben is megtalálható jelenséget látunk: egyes dolgok más és más környezetben másképp viselkednek. Itt, ha úgy hivatkozunk egy tömbre, mint egy skalár változóra, akkor skalár változóként is viselkedik, az értéke pedig a tömb elemeinek száma lesz. Nézzük meg a tömböknél szereplő utolsó példát. Itt azt láthatjuk, hogy hogyan tudjuk egy tömb két tetszőleges elemét külső változó segítségével megcserélni. Ez a példa is a nyelv rugalmasságát mutatja, kevés nyelv van, amelyben ez megvalósítható, annak ellenére, hogy sok esetben hasznos funkció.

Az értékadások végéhez közeledünk, de még nem beszéltünk az értékadások ellentett műveletéről, mikor fel akarunk szabadítani egy változót, mert szükségünk van az általa foglalt memóriára, vagy egyéb okokból. Általában, ha rövid, kevés memóriát igénylő programot írunk, akkor erre a lépésre nincs is szükség, hiszen a Perl értelmező a program futásának végén felszabadítja az összes változót. Ellenkező esetekben azonban két 'utasítás' áll rendelkezésünkre, ebből tekintsük meg az alapeseteket az elsőre, az undef-re:

```
undef $skalar;
undef @tomb;
undef %hash;
```

A tömbök és a hash-ek egyes elemeit is felszabadíthatjuk a fenti módszerrel:

```
undef \%tomb[5];  
undef \%hash{'hatodik'};
```

Első ránézése tisztának tűnik a dolog? Vizsgáljuk meg közelebbről, hogy mi is történt a második esetben a hash-nél! Valójában ekkor csak a 'hatodik' indexű elemhez hozzárendelt változó értékét szabadítjuk fel, a hozzárendelés továbbra is megmarad, csak éppen a \$hash{'hatodik'} értéke ezentúl nem definiált. Ez a más programnyelvekben használatos NULL értékhez hasonlít. Mit tehetünk, amennyiben ezt a hozzárendelést is megszeretnénk szüntetni? Természetesen erre is van megoldás, erre szolgál a delete.

```
delete \%hash{'hatodik'};
```

Műveletek skalár változókkal

Egy programozási nyelvnek természetesen tudnia kell műveleteket végeznie is az egyes változókkal, ahogy az előzőekben is hivatkoztam már rá. Erre első nekifutásra a matematikában megszokott műveleteket használhatjuk: összeadás (+), kivonás (-), szorzás (*), osztás (/), maradékképzés-modulus (%), hatványozás (**) és a zárójelek. A művelet kiértékelésénél a sorrendet az operátorok precedencia (elsőbbrendűségi) sorrendje határozza meg. A zárójelezés a legmagasabb szintű, ezután a hatványozás, majd az osztás és a szorzás egy szinten, továbbá az összeadás és kivonás következik. Ez megegyezik azzal, ahogy mi is elvégeznénk a műveletet, ahogy remélhetőleg matematika órán megtanultuk. :)

Amennyiben egy olyan változóra használjuk ezeket az operátorokat, melyek sztring típusúak, akkor azok előzőleg automatikusan számokká alakulnak. Lássunk ezekre pár példát (a számok helyett használhatunk természetesen változókat is):

```
\$szam=(10+2-6)/3; # \%szam=2  
\$szam=10%3; # \%szam=1  
\$szam=(1+2)*(3+4); # \%szam=21  
\$szam=-3**2; # \%szam=-9  
  
\$szam='10'+2; # \%szam=12  
\$szam='tiz'+2; # \%szam=2  
\$szam='6alom'+1 # \%szam=7
```

A fenti példáknál az egyetlen meglepetés talán a negyedik, hatványozást tartalmazó példa, ahol az eredmény nem a várt 9, hanem -9. Miért ezt az értéket kaptuk? A válasz egyszerű: mert a hatványozás magasabb rendű művelet, mint a kivonás és ezért először az hajtottódott végre, s utána a negálás. Az utolsó példák működésénél annyit kell megértenünk, hogy egy szöveg számmá alakításakor a felesleges (azaz betű) részeket elhagyja

a Perl, és ami megmarad, azzal dolgozik. A 'tiz'-ből nem marad semmi, így az értéke, ha éppen számot adunk hozzá, akkor 0, a '6alom'-ból pedig a '6' marad, ezért kapunk 7-et végeredményként.

A Perl nyelvben lehetőség van (a például a C-ben is használható) autoinkrementálásra (++), autodekrementálásra (–) is. Ez egy változó eggyel való növelésére, vagy csökkentésére szolgál. Az operátort a változtatandó változó elé és mögé is írhatjuk. Ennek akkor van jelentősége, mikor az értékét előbb fel szeretnénk használni egy kifejezés részeként. Ha a változó elé írjuk, akkor az értékadás a kifejezés kiértékelése előtt, amennyiben mögé, akkor a kifejezés kiértékelése után hajtódik csak végre. A következőkben erre láthatunk példákat:

```
\$szam1=3; \$szam1++; # \$szam1=4;
\$szam1=3; \$szam1--; # \$szam1=2;
\$szam1=3; \$szam2=\$szam1++; # \$szam1=4; \$szam2=3
\$szam1=3; \$szam2=\$szam1--; # \$szam1=2; \$szam2=3
\$szam1=3; \$szam2=++\$szam1; # \$szam1=4; \$szam2=4
\$szam1=3; \$szam2=--\$szam1; # \$szam1=2; \$szam2=2
```

A logikai műveletek is a nyelv részei. Ezekből kétfajta is van, a 'sima' logikai operátorok, melyeknél a művelet eredménye igaz, vagy hamis lehet, illetve a bitenkénti logikai operátorok, ahol a logikai művelet az egyes bitekre hajtódik végre. Ezek a következők:

```
\$true=1; \$false=0;

\$ertek=\$true && \$false;
\$ertek=\$true || \$false;
\$ertek=!\$false;

\$szam=3 & 1; # \$szam=1
\$szam=5 | 2; # \$szam=7
\$szam=5 ^ 3; # \$szam=6
\$szam=5 << 2; # \$szam=20
\$szam=5 >> 2; # \$szam=1
```

Az első sorban csak a logikai értékeket definiáltuk. Teljesen helyes akkor lenne, amennyiben a \$false értékének üres sztringet (") adnánk, mivel a Perl ezt adja vissza hamis érték esetén, de példánk így is megállja helyét, mivel egy üres sztring értéke 0, ha automatikus konverzió hajtódik végre és számmá alakítjuk.

A második blokk első sora a logikai VAGYot, a második a logikai ÉSt a harmadik pedig a logikai TAGADÁSt mutatja be. A harmadik blokkban a bitenkénti műveletekre látunk

példát, az első sora az ÉS, a második a VAGY, a harmadik pedig a kizáró VAGYra (XOR) példa. A negyedik és az ötödik a bitenkénti görgetésre, értelemszerűen az első balra, a második jobbra görget.

Szükségünk lehet sztring típusú változók összefűzésére (angolul konkatenálásnak hívják, ne ijedjünk meg, ha valahol ezzel a kifejezéssel találkozunk) is. Mit tehetünk ilyenkor? Természetesen erre is van megoldás. A sztringekre kétfajta művelet van definiálva a Perl-ben: a konkatenálás-összefűzés (.) és az ismétlés (x, azaz kis iksz). Tekintsünk meg erre is példákat:

```
\$sztring1='első';
\$sztring2='program';
\$sztring3=\$sztring1.' '.\$sztring2;
# \$sztring3='első program'
\$sztring4='-'+'x3.'-' ; # \$sztring4='-+-+--'
```

Ha egy változón saját magán szeretnénk csak végrehajtani egy műveletet, erre is lehetőség van, az egyenlőség elé kell írni a műveleti jelet:

```
\$szam+=5; # \$szam=\$szam+5
\$szam-=5; # \$szam=\$szam-5
\$szam*=5; # \$szam=\$szam*5
\$szam/=5; # \$szam=\$szam/5
\$szam%=5; # \$szam=\$szam%5
\$szam**=5; # \$szam=\$szam**5
\$szam.=5; # \$szam=\$szam.5
\$szam x=5; # \$szam=\$szam x 5
\$szam&&=1; # \$szam=\$szam&&5
\$szam|=1; # \$szam=\$szam|5
\$szam&=5; # \$szam=\$szam&5
\$szam|=5; # \$szam=\$szam|5
\$szam^=5; # \$szam=\$szam^5
\$szam<<=3; # \$szam=\$szam<<5
\$szam>>=3; # \$szam=\$szam>>5
```

A Perl nyelvben lehetőség van úgynevezett többszörös értékadásra. Ekkor az értékadás jobb oldalára egy másik értékadást írunk, vagy másképpen és talán egy kicsit pontosabban megfogalmazva: egy értékadás bal oldalára több változót is írhatunk. Megint másképp fogalmazva, egy értékadást kifejezésként is használhatunk, az ilyen kifejezések értéke a jobb oldalon levő érték lesz. Lássunk erre két példát:

```
\$szam2=\$szam1=1;
\$szam2=(\$szam1+=5)-3;
```

Az első talán a legtipikusabb, sorozatos értékadás, melyet általában programok elején, inicializáláskor használunk, ahol mind a \$szam1, mind a \$szam2 változó értéke 1 lesz. A második kifejezés először a belső értékadást fogja végrehajtani, melynek eredményeképpen a \$szam1 változó értéke 5-el nő, azaz 6 lesz, majd pedig a 6-ból vonunk le 3-at, így a \$szam2 változó értéke 3 lesz.

Az alfejezet végére az összehasonlítások maradtak. Ezek hasonlítanak a logikai műveletekhez, olyan kifejezések, melyeknek igaz, vagy hamis lehet az értéke. Általában egy változó értékét vizsgáljuk meg segítségével (kisebb, mint 4?, üres sztring?, stb.). A sztringekre és a számokra külön összehasonlító operátorok vannak, az alábbi táblázat ezeket foglalja össze (az első oszlop a számokhoz, a második a sztringekhez tartozik):

>	- gt	- nagyobb, mint
>=	- ge	- nagyobb, vagy egyenlő
<	- lt	- kisebb, mint
<=	- le	- kisebb, vagy egyenlő
==	- eq	- egyenlő
!=	- ne	- nem egyenlő
<=>	- cmp	- összehasonlítás

A következő példák a számokra való használatukat mutatják be (sztringekre ugyanez, egy sztring akkor kisebb egy másiknál, ha az (angol) abc-ben előrébb van.

```
print 8>7; # =1, azaz igaz
print 7>8; # = "", azaz hamis
print 8>=7; # =1, azaz igaz
print 7>=8; # = "", azaz hamis
print 8<7; # = "", azaz hamis
print 7<8; # =1, azaz igaz
print 8<=7; # = "", azaz hamis
print 7<=8; # =1, azaz igaz
print 7==7; # =1, azaz igaz
print 7==8; # = "", azaz hamis
print 7!=7; # = "", azaz hamis
print 7!=8; # =1, azaz igaz
print 7<=>7; # =0, azaz egyenlő
print 8<=>7; # =1, azaz nagyobb
print 7<=>8; # =-1, azaz kisebb
```

Az alfejezetből kimaradt az illesztő operátor működése, mivel annak megértéséhez a reguláris kifejezések ismerete szükséges. A reguláris kifejezések alfejezetben mind a magyarázata, mind pedig példák a működésére megtalálhatóak.

Műveletek tömbökkel, hash-ekkel

A tömbök, hash-ek használatához is találunk beépített parancsokat, s bár személy szerint én hiányolok pár lehetőséget (unió, metszet, komplementer), azonban így is jóval több lehetőséget tartalmaz a nyelv, mint például a C. Nézzük végig ezeket a parancsokat!

Kivágás, csere (splice). Már láthattuk, hogy hogyan tudunk hozzáadni egy tömbhöz értékeket, de még nem tudjuk, hogy hogyan lehetne egyszerűen kicserélni, vagy levágni elemeket. Erre szolgál a splice parancs. Segítségével egy tömbből a megadott pozíciótól a megadott hosszig kivágja, vagy ha meg van adva egy lista, akkor kicseréli rá a tömb elemeit:

```
@tomb1=splice(@tomb, 4);  
@tomb2=splice(@tomb, 4, 2);  
@tomb3=splice(@tomb, 4, 2, 'ötödik', 'hatodik');
```

Az első példa levágja a tömb elemeit az ötödik elemtől kezdve. A második példa két elemet vág le az ötödik elemtől, a harmadik példa pedig kiveszi az ötödik és a hatodik elemet, és a helyükre beírja, hogy 'ötödik' és 'hatodik'. Mind a három tömb azokat az elemeket fogja tartalmazni, amelyek el lettek távolítva a tömbből.

Verem (push, pop). A verem egy nagyon széleskörűen alkalmazott adatszerkezet, általában a megvalósítása tömb segítségével történik. Röviden leírva a működését: hozzá lehet adni és ki lehet venni belőle elemeket. Amit utoljára betettünk, azt tudjuk kivenni elsőre (Last In First Out - LIFO), másképpen fogalmazva, fordított sorrendben kapjuk vissza az elemeket, mint ahogy azokat a szerkezetben elhelyeztük.

Az adatszerkezet használatához a push és a pop parancsokat vehetjük igénybe. A push egy tömbhöz hozzá ad egy skalárt, a pop pedig egy tömbből kivesz egyet (véglegesen eltávolítva onnan):

```
@verem=(1..10);  
push(@verem, 11);  
push(@verem, 12, 13, 14, 15);  
print pop(@verem);
```

Sor (shift, unshift). A verem mellett a másik gyakran használt adatstruktúra a sor. Ezt is tömb segítségével valósíthatjuk meg, lényege, hogy amit elsőnek betettünk, azt vesszük ki elsőnek (First In First Out - FIFO). Használatához a shift és az unshift eljárások állnak rendelkezésünkre. Az unshift beszúr elemeket, a shift pedig kivesz:

```
@sor=(5..10);
unshift(@sor, 4);
unshift(@sor, 1, 2, 3);
print shift(@sor);
```

Sorrend megváltoztatása (sort, reverse). Gyakran lehet szükségünk egy tömb elemeinek rendezésére. Erre szolgál a sort parancs, mely egy rendezett tömböt ad vissza (tehát nem helyben, magát a tömböt rendezi). A C nyelv qsort algoritmusával rendez, tehát viszonylag gyors. Alapesetben ASCII sorrend szerint hajtja végre a feladatát (tehát nem a számok értéke szerint!, ezért a 13 kisebb lesz, mint a 2, mert az 1 az kisebb, mint a 2), de az összehasonlító feltétel megváltoztatható, akár egy külön programrész, szubrutin írható rá - így elég összetett, több szempont szerinti rendezések valósíthatóak meg egy gyors algoritmusú rendezés segítségével.

```
@rendezett = sort(@rendezetlen);
@rendezett = sort({lc(\$a) cmp lc(\$b)}, @rendezetlen);
@rendezett = sort({\$a <=> \$b}, @rendezetlen);
```

Az első sor az alapeset, legtöbbször így használjuk az eljárást. A második sor mutatja be, hogy hogyan lehet megváltoztatni a feltételt, a harmadikban pedig egy példát látnunk arra, hogy hogyan lehet számokat rendezni. A \$a és a \$b változók tartalmazzák a két összehasonlítandó értéket, ezeknek nem kell értéket adnunk. Értelemszerűen, ha felcseréljük a \$a-t és a \$b-t, akkor csökkenő sorrendben rendezett tömböt kapunk eredményül. A második és a harmadik példában valójában egy szubrutin foglal helyet az első paraméter helyén, ehelyett egy szubrutin nevét is lehet használni.

Adott tömb esetén, amennyiben szükségünk van a sorrend megfordítására, akkor a reverse eljárást használhatjuk. Ez a paraméteréül megadott tömböt fordított sorrendben adja vissza. Amennyiben rendezünk és utána lenne szükségünk a fordított sorrendre, akkor memóriahasználati szempontból inkább ne ezt használjuk, rendezzünk eleve fordított sorrendben (a két változó cseréjével: 'sort (\$b cmp \$a, @rendezetlen)' ugyanaz lesz, mint a reverse sort (\$a cmp \$b, @rendezetlen), csak az utóbbi tovább tart).

```
@vissza = reverse(@elore);
%hash1 = reverse(%hash2);
```

Az első példa egyértelmű, megfordítja a tömb elemeinek sorrendjét. Kérdés viszont, hogy mit csinál a második, hiszen nincs igazán értelme sorrendről beszélni a hash-ek esetén (a fordító logikája szerinti sorrendben tárolódnak a memóriában)? Az értékadás-kor már bemutattam, hogy egy hash-nek egy tömböt is lehet értékül adni, ekkor páronként kerülnek be a hash-be a tömb elemei, az első (páratlan sorszámúak) a hash kulcsai,

a másodikak (párosak) pedig ezek értékei lesznek. Ha megfordítjuk egy hash sorrendjét, akkor úgy viselkedik, mintha egy ilyen tömbnek a sorrendjét fordítanánk meg, azaz az értékek kulccsá válnak és fordítva. Nagyon hasznos trükk.

Műveletek a tömb összes elemén (chomp, chop, map, grep). A tömbök minden elemén végrehajtandó feladat elvégzése szintén gyakori feladat különböző programrészek, algoritmusok megvalósításánál. Természetesen számos módszer létezik, az egyik legkézenfekvőbb, hogy valamilyen ciklusszervezési módszerrel (lásd később) végighaladunk az egyes elemeken. Erre azonban vannak a Perl-nek elegánsabb megoldásai is. Például, mikor egy szöveges állományból betöltjük a sorokat egy tömbbe, akkor a sorvég jelek ott maradnak a sorok - azaz most már a tömb elemeinek végén. Ezt a problémát lehet megoldani a chop eljárás segítségével, melynek egyetlen paramétere van: a fel dolgozandó tömb. Az utasítás helyben dolgozik, azaz az eredménye a megadott tömbön jelentkezik - levágja a sorok utolsó karakterét. Ennek az eljárásnak a "biztonságosabb" változata a chomp, ez csak akkor vágja le az utolsó elemet, amennyiben az újsor karakter, illetve Windows alatt ha újsor karakterpáros. Ezt a nyelv fejlődése során vezették be a felhasználók tapasztalataira épülő egyik javaslat hatására - hiszen többek között például akkor nem működik a chop, ha a file utolsó sorának végén már nincsen újsor karakter. A következő példák e függvények használatát mutatják be (megjegyzendő, hogy tömb helyett skalár is átadható ezeknek a rutinoknak, ilyenkor az adott skaláron végzik tevékenységüket):

```
chop(@sorok); # a "sorok" elemeinek utolsó karakterét levágja
chomp(@sorok); # ez pedig a "biztonságosabb" változat
```

Igen, ezek a függvények jól használhatóak, de mit tegyünk, ha mi pont a fordítottját szeretnénk tenni a dolognak, például hozzáadni egy újsort vagy pedig egy teljesen más feladatunk akad? Erre van a map parancs. Két paramétert kell neki megadni, az első egy parancs - amit tenni szeretnénk az egyes elemekkel -, a parancs eredménye lesz az adott elem új tartalma, a második a tömb. Az aktuális elem \$_ változóba kerül bele. Íme pár példa:

```
@ujtomb=map(''.\$_.' ', @egytomb);
@ujtomb=map(\$_**2, @egytomb);
@tomb=map(substr(\$_, -1) eq "\n":substr(\$_, 0, -1)?\$_, @tomb);
```

Mint látható ez a parancs már nem a tömbön magán dolgozik, hanem az eredménye egy új tömbbe kerül bele. Az első példa idézőjelbe teszi minden elemét a tömbnek, a második pedig a tömb minden elemét négyzetre emeli. Az utolsó példa gyakorlatilag a chomp megvalósítása, persze lehetett volna hatékonyabban is írni, de törekedtem az érthetőségre, hiszen a feltételes utasításokat még nem mutattam be. Röviden annyit csinálok, hogyha az utolsó karaktere az adott elemnek újsor, akkor a kérdőjel előtti részt adja

vissza, azaz az utolsó karakter levágásra kerül, különben pedig a kérdőjel utánit, azaz az eredeti értéket.

Ebben a blokkban még a grep utasítás maradt hátra. Segítségével kiválogatást végezhetünk, azaz egy általunk megadott feltétel segítségével megadhatjuk, hogy a visszaadott tömbben szerepeljen-e az adott elem, vagy nem.

```
@haromjegyű=grep(\\$_>99 and \\$_<1000, @szamok);
@nemmegjegyű=grep(substr(\\$_, 0, 1) ne '#', @sorok);
```

Az első példa a 'szamok' tömbből visszaadja a háromjegyű számokat (feltéve, hogy egész számokról beszélünk). A második segítségével, amennyiben a 'sorok' tömbbe egy állomány sorai vannak betöltve, akkor kiszűri belőle azokat, melyek nem '#' karakterrel kezdődnek.

Kulcs meglétének vizsgálata hash-ben (exists). A végére maradtak a hash-ekkel végezhető műveletek. Hash-eket használva, szükségünk lehet arra, hogy megvizsgáljuk, definiálva van-e egy adott kulcs a hash-ben vagy nem? Erre szolgál az exists operátor, ami igaz értéket ad vissza, ha létezik, s hamisat, ha nem:

```
exists( \\$hash{'kulcs'} ); =='', ha nincs, =1, ha van
```

Hashek kulcsai, értékei (keys, values). Tipikus feladat lehet, hogy egy hash minden elemére, vagy minden értékével szeretnénk tenni valamit, erre szolgálnak a keys és values operátorok. A keys a hash kulcsait adja vissza egy tömbben, a values pedig az egyes értékeket. A keys-re egy példa (a values ugyanígy működik):

```
print map( \\$_."="."\\$hash{\\$_}."\\n", keys %hash );
```

A példa kiírja a hash elemeit kulcs=érték formában, külön sorokban.

Kulcs-érték párok (each). Amennyiben szeretnénk egy hash elemein végigfutni és utána használni a továbbiakban valamire, akkor az each parancsot használhatjuk. Ez egy kételemű listát ad vissza, melyben a soron következő kulcs és érték értékek vannak benne. Ha a hash elemeinek végére értünk, akkor mindkét elem undefined lesz, majd kezdődik előről.

```
(\\$kulcs1, \\$ertek1)=each(%hash);
(\\$kulcs2, \\$ertek2)=each(%hash);
(\\$kulcs3, \\$ertek3)=each(%hash);
```

Elágazások

Egy programban legyen az bármilyen egyszerű is, nagyon valószínű, hogy lehetővé kell tennünk, hogy bizonyos feltételek teljesülése és nem teljesülése esetén más és más történjen. Erre szolgálnak a feltételes utasítások. A feltételes utasítások legalább egy feltételtől, s legalább egy utasításblokkból állnak. A feltétel egy kifejezés lehet, melynek értéke vagy igaz, vagy hamis, az utasításblokk pedig egy olyan szerkezet mely (általában) több utasítást fog egybe. Az utasításblokkokat kapcsos-zárójelek közé írjuk, s a kapcsos-zárójelek akkor sem hagyhatóak el, ha egyetlen utasítást fognak közre (ellenben a Pascal begin-end-jével, vagy a C kapcsos-zárójeleivel).

Perl-ben egyetlen lehetőség áll rendelkezésünkre elágazások használatára, ez az if-es szerkezet, azonban ezt olyan sokféleképpen és rugalmasan használhatjuk, hogy nincs is más lehetőségre szükségünk. A legáltalánosabb formája a következő:

```
if (feltétel1)
{
    # ha feltétel1 igaz
    utasítások;
}
elsif (feltétel2)
{
    # ha feltétel1 nem igaz és feltétel2 igaz
    utasítások;
}
else
{
    # ha egyik feltétel sem igaz
    utasítások;
}
```

Mind az elsif és az utána következő blokk, mind az else és az utána következő blokk tetszés szerint elhagyható. Elsif blokkból többet is használhatunk. A fenti program működése tehát a következő: ha feltétel1 igaz, akkor a hozzátartozó blokk hajtódik végre, különben ha a feltétel2 igaz, akkor a második blokk fut le, egyébként pedig az else-hez tartozó blokk. Lássunk a használatára példákat:

```
if (\$szam<10) { print "0".\$szam } else { print \$szam }
if (\$parancs eq 'előre')
{
    print "\$ertek1 \$ertek2 \$ertek3\n";
}
```

```
elseif (\$parancs eq 'hatra')
{
    print "\$ertek3 \$ertek2 \$ertek1\n";
}
elseif (\$parancs eq 'első')
{
    print "\$ertek1\n";
}
elseif (\$parancs eq 'utolsó')
{
    print "\$ertek3\n";
}
elseif (\$parancs eq 'középső')
{
    print "\$ertek2\n";
}
else
{
    print "Ismeretlen parancs!\n";
}
```

A fenti példák remélhetőleg egyértelműek. A feltételeknél a műveleteknél megismert összehasonlításokat alkalmaztam. Az if szó helyett további lehetőség az unless szó használata, amely gyakorlatilag az ellenkezőjét fogja tenni az if-nek, akkor hajtódik végre az if-hez tartozó utasításblokk, ha a kifejezés nem igaz. Elsunless utasítás nem létezik. A használata ezenkívül teljesen megegyezik az if-ével:

```
unless (feltétel1)
{
    # ha feltétel1 nem igaz
    utasítások;
}
elseif (feltétel2)
{
    # ha feltétel1 igaz és feltétel2 igaz
    utasítások;
}
else
{
    # ha feltétel1 igaz volt, a többi feltétel nem
    utasítások;
```

```
}
```

Amennyiben az utasításblokkunk csak egy utasításból áll, rendelkezésünkre áll egy további lehetőség is:

```
utasítás if (feltétel);  
utasítás unless (feltétel);
```

Ekkor az utasítás csak akkor fog végrehajtódni, ha a feltétel igaz (if), illetve hamis (unless). Erre a lehetőségre pár példa:

```
print "Túl nagy szám" if (\$szam>9999);  
print "Hiba!!!" unless (\$hiba==0);
```

Az első üzenet akkor jelenik meg, hogyha a `szam` változó értéke nagyobb, mint 9999, a második pedig akkor, ha a `hiba` nevű változó értéke nem 0. Természetesen ezeknek az utasításoknak az “egymásba ágyazása”, azaz feltételes utasítás használata egy feltételes utasítás utasításblokkjában minden további nélkül lehetséges.

Ciklusszervezési lehetőségek

Gyakran lehet szükségünk arra is, hogy a programunk egy részét többször végrehajtsuk, például amíg egy feltétel igaz, vagy amíg egy feltétel hamis. Erre a Perl nyelv számos lehetőséget biztosít. Képzeljük el, hogy mi tennénk, ha ki kellene írunk az összes négyjegyű egész számot! Biztosan nem írnánk le mindet. A példákban ennek megoldási lehetőségeit fogom bemutatni.

Az egyik legegyszerűbb ciklusszervezési lehetőség a `while` ciklus. A szintaktikája a következő:

```
while (feltétel)  
{  
    utasítások;  
}
```

Az utasításblokk addig ismétlődik, amíg a feltétel igaz (más oldalól megközelítve, amíg hamissá nem válik). A példa (ami a négyjegyű egészeket írja ki):

```
\$szam=1000;  
while (\$szam<10000)  
{  
    print \$szam++."\n";  
}
```

A while-hoz hasonló ciklusszervezési lehetőségünk az until, ami ugyanúgy működik, mint a while, kivéve, hogy a “ciklusmagot”, azaz az utasításblokkban szereplő utasításokat addig ismétli, amíg feltétel igazzá nem válik, azaz ameddig hamis. Ezzel megoldva a példánk a következőképpen alakul:

```
\$szam=1000;
until (\$szam>9999)
{
    print \$szam++."\n";
}
```

Az if és az unless szerkezetekhez hasonlóan mind a while, mind az until szerkezet lehetőséget ad arra, hogy a feltételt egy utasítás után írjuk:

```
\$szam=1000; print \$szam++."\n" while (\$szam<10000);
\$szam=1000; print \$szam++."\n" until (\$szam>9999);
```

A do függvény segítségével utasításblokkot is használhatunk az egyetlen utasítás helyett. Így egyetlen különbség lesz a “sima” while és until ciklusokhoz képest: a ciklusmag akkor is végre fog hajtódni egyszer, ha a feltétel nem igaz (while), illetve hamis (until) (ez az előző példára is igaz!). Ilyenkor a ciklusmag elé a do szót kell írunk. A példa do-while-al:

```
\$szam=1000;
do {
    print \$szam++."\n";
} while (\$szam<10000);
```

Illetve do-until segítségével:

```
\$szam=1000;
do {
    print \$szam++."\n";
} until (\$szam>9999);
```

A következő ciklusszervezési lehetőségünk a for ciklus. Nagyon rugalmas szerkezet, rendkívül széleskörűen felhasználható. A következőképpen lehet használni:

```
for (utasítás1; feltétel; utasítás2)
{
    ciklusmag;
}
```


Az utasítás1 utasítás az egész ciklus elején hajtódik végre, majd végrehajtódnak az úgynevezett ciklusmag helyén szereplő utasítások, s utána az utasítás2 utasítás, majd megint a ciklusmag és megint az utasítás2, és így tovább, amíg a feltétel igaz, és hamissá nem válik. Ez így elsőre biztosan bonyolultnak hangzik, lássuk, hogy hogyan oldható meg a feladatunk ezzel a ciklusszervezési módszerrel:

```
for (\$szam=1000; \$szam<10000; \$szam++)
{
    print \$szam."\n";
}
```

Első alkalommal értéket adunk a szam változónak, majd amíg nem érjük el a 10000-et, addig kiírjuk a szam változó értékét és növeljük eggyel. A zárójelben lévő részek tetszés szerint elhagyhatóak, vagy akár (a feltétel kivételével) több is írható belőlük, vesszővel elválasztva:

```
for (;;) {print "Végtelen ciklus!";}
for (\$x=0, \$y=1; \$x<10; \$x++, \$y+=2) { print \$x.' '.\$y."\n";}
```

A ciklusok használatakor felmerülhet az az igény, hogy a ciklusmag közepén valamely feltétel esetén ki szeretnénk lépni a ciklusmagból. Erre háromfajta lehetőségünk van, a next, a last és a redo utasítás. A next segítségével az utasításblokk végére ugorhatunk, a redo segítségével az elejére (így ugyanazokkal az értékekkel még egyszer végrehajtódik a ciklusmag), a last pedig befejezi a ciklus végrehajtását és kilép belőle. A következő példák mutatják be ezeknek az utasításoknak a használatát:

```
while (\$line=EgyUjSor)
{
    if (\$line eq 'kilép') { last }
    if (\$line eq 'üdv') { print "Hello!\n"; next }
    if (\$line eq 'hello') { \$line='üdv'; redo }
    print "Nem értem.\n";
}
```

A ciklus addig ismétlődik, míg tud új sort beolvasni. Ha beolvasott egy sort, akkor a tartalma szerint a következők történhetnek: 'kilép' esetén kilép a ciklusból, 'üdv' és 'hello' esetén visszaír egy 'Hello!' -t, a többi esetben pedig azt írja ki, hogy 'Nem értem.'. A 'Hello!' kiírása után nem íródik ki a 'Nem értem.', mivel a next utasítást használtuk, az 'üdv' pedig megváltoztatja a line változó értékét és a redo-val újra feldolgoztatja azt, ezért kapjuk a 'Hello!' üzenetet.

E rész végére a rendkívül jól használható foreach utasítás maradt. Ez veszi egy tömb összes elemét és mindegyikhez végrehajtja a hozzá tartozó utasításblokkot. Egy példa a használatára:

```
@tomb=('első','második','satöbbi','utolsó');
\$_i=1;
foreach (@tomb)
{
    print "A \$_i. elem: \$__\n"; \$_i++;
}
```

Mint látható, az aktuális elem az '_' változóban áll rendelkezésünkre. Ezt már használtuk korábban (grep-nél, map-nél) és rejtett változónak hívjuk. A fenti ciklusszervezési módszerekkel rendkívül változatos és elegánsan használható ciklusszervezési eszközök-höz juttat minket a Perl, melyek kevés nyelvben használhatóak ilyen kényelmesen és változatosan.

Metódusok (Eljárások és függvények)

Mint minden más fejlett nyelv, a Perl is biztosít lehetőséget a gyakran használt programrészek állandó ismételtetése helyett metódusok használatára, azonban nem tesz különbséget az eljárások, illetve függvények között (eljárás, amit meghívunk, de nem tér vissza értékkel - ilyen a print -, függvény, amit meghívunk, és visszatér egy értékkel - ilyen a szinusz-t megvalósító sin). Mindkét metódustípus deklarációja megegyezik, s az, hogy egy metódust mire használunk, csakis tőlünk függ. Egy példán keresztül megpróbálom bemutatni, hogy mit nyerünk akkor, ha metódusokat használunk:

```
\$szam=1;
print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
\$szam=5;
print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
\$szam=8;
print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
\$szam=9;
print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
```

Ezt a következőkre tudjuk leegyszerűsíteni:

```
\$szam=1; kiir();
\$szam=5; kiir();
\$szam=8; kiir();
\$szam=9; kiir();

sub kiir
{
    print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
}
```

Ez azon az előnyön kívül, hogy nem kell 4-szer beírni a kiíró sort, azzal az előnnyel is jár, hogy ha a későbbiekben módosítani kell a programot, hogy az eggyel nagyobb számokat írja ki, akkor nem négy helyen kell azt megtennünk. Amennyiben elképzeljük, hogy 20-30 soros az ismétlendő rész (ennél sokkal-sokkal hosszabb megoldások is vannak), még érthetőbb lesz a probléma.

De nézzük végig, hogy mit is csinál a fenti program, mit is jelentenek az egyes sorai. A kiir() utasítások hatására a program átugrik a 'sub' részhez, végrehajtja az ott leírtakat, majd annak végén visszaugrik oda, ahol volt. A 'sub' szó azt jelenti, hogy egy metódust (szubrutint) szeretnénk most leírni, majd kapcsos zárójelek között a tartalma jön.

Mint látható volt, az eljárásunk a 'szam' értékét használja. Felmerülhet bennünk a kérdés, hogy nem lehetne-e még egyszerűbben megoldani a problémánkat? Természetesen lehet, paraméterátadással. Ilyen paraméterátadást használunk akkor is, mikor leírjuk, hogy sin(30). Ekkor felsoroljuk a metódus neve után, hogy milyen változókat, konstansokat szeretnénk átadni a számára, majd ezeket a metóduson belül használni fogjuk tudni. Nézzük, ezt hogyan lehet megoldani:

```
kiir(1);
kiir(5);
kiir(8);
kiir(9);

sub kiir
{
    (\$szam)=@_;
    print "A szám értéke: \$szam. Eggyel kisebb: ".$szam-1."\n";
}
```

A változás annyi, hogy eljárásunk kibővült egy sorral, ahol is a 'szam' változónak értéket adunk. Az eljárásnak átadott változók az '_' nevű tömbbe (@_) kerülnek bele (már megint egy rejtett változó), s abból vesszük ki őket ebben a sorban.

Bővítsük tovább tudásunkat a függvényekkel, azaz az olyan metódusokkal, ahol egy értéket kapunk vissza. Íme egy példa ennek megvalósítására:

```
print kob(4);
print kob(3);

sub kob
{
    return $_[0]**3;
}
```

A példában a köbre emelő függvényt valósítottuk meg. A függvény visszatérési értékét a 'return' utasítás segítségével határozhatjuk meg, az eljárás egyszerűen veszi az első

paramétert (az '_' tömb első eleme: \$_[0]), és visszaadja annak a köbét. A függvény visszatérési értéke ha nem adunk meg 'return'-nel semmit, akkor az utolsó kifejezés értékét veszi fel, azaz írhattuk volna a következőket is a függvényünkben:

```
sub kob
{
    $_[0]**3;
}
```

Hogyan oldhatjuk meg, ha több paramétert szeretnénk visszaadni? Kétféle módszerrel. Az egyik, amely talán magától értetődik: a visszatérési értékünk legyen egy tömb, melynek elemei az általunk visszaadandó értékek:

```
@tomb=paratlanszamok(5,9);
```

```
sub paratlanszamok
{
    ($szam1,$szam2)=@_;
    for($szam1++; if ($szam1%2==0); $szam1<$szam2; $szam1+=2)
    {
        push(@ret,$szam1);
    }
    return @ret;
}
```

A függvény a páratlan számokat adja vissza a két paraméter között (paraméterként egész számokat kell megadni, hogy helyesen működjön a függvény). A ret tömbbe kerülnek bele a számok, s ahogy az előbb is, itt is a return paranccsal térünk vissza a függvény végén.

A másik megoldás a cím szerint átadott paraméterek használatával történik. Perlben is kétféleképpen használhatjuk a paramétereket, az egyik, mikor létrehozunk egy új változót és azt használjuk, módosítjuk (ahogy a köbös példa kivételével eddig tettük). Ez az érték szerinti átadás megfelelője. A másik módszer, mikor az '_' tömb elemeire közvetlenül hivatkozunk, ha megváltoztatjuk ezek értékét, akkor a "külső" változó értéke is módosul (tehát Perlben valójában csak cím szerinti átadás történik). Lássunk erre is egy példát:

```
@tomb=(4,7,2,8,2,4,7,9);
rendez(@tomb);
print "A tömb elemei: ".join(', ',@tomb)."\n";
```

```
sub rendez
{
    @_ = sort({lc($a) cmp lc($b)}, @_);
}
```

Reguláris kifejezések

A reguláris kifejezések a Perl egyik erőssége (de nem sajátja, már több nyelv kiegészítéseként is megjelent reguláris kifejezések használatát lehetővé tevő modul), már a nyelv második verziójától a nyelv részét képezik (viszont az, hogy bele van építve a nyelvbe, az ismertebb nyelveknél egyikénél sem fordul elő). Segítségükkel többsoros kódrészleteket helyettesíthetünk egyetlen kifejezéssel, és sok esetben jóval elegánsabban oldhatjuk meg problémáinkat, mint használatuk nélkül. Ebben a részben csak egy rövid ízelítőt próbálok nyújtani arról, hogy mire használhatóak, s milyen lehetőségek rejlenek bennük, a reguláris kifejezésekkel kapcsolatos témakör oly nagy, hogy már több könyv is született, mely csak ezzel foglalkozik.

Mi is valójában egy reguláris kifejezés? Első ránézésre jelek összeviasszása (második ránézésre is), azonban ha jobban megismerjük, akkor egy nagyon jól használható eszköz. Röviden arról szól, hogy egy sztringben szeretnénk egy részletet keresni. A gyökerei valahol a joker karaktereknél vannak, a legismertebb ilyen joker karakterek a `*` és a `?`, mikor DOS (vagy akár Windows, UNIX) alatt ezeket a jeleket használjuk, akkor `*` esetén bármennyi és bármilyen, `?` esetén pedig egy bármilyen karaktert értünk alatta. Példával illusztrálva a `*.txt` alatt az összes `.txt` kiterjesztésű állományt értjük, hell?`.txt` alatt pedig például a `hello.txt`-t, a `hella.txt`-t, stb. - a `?` helyén bármely karakter szerepelhet.

A legegyszerűbben talán egy példával lehet rávilágítani a reguláris kifejezések működésére. Kezdjük az illesztő operátorral, melyet a skalár műveleteknél kihagytunk! A feladat nagyon egyszerű egy tömbből kiválogatjuk azokat az elemeket, melyek tartalmazzák az 'egy' szórészletet, azaz itt a reguláris kifejezésünk az 'egy' részletre fog rákeresni:

```
@tomb=('teljes', 'egyetem', 'begy', 'lehet', 'stb');
foreach(@tomb)
{
    if (/egy/)
    {
        print "A(z) \$_ szó tartalmazza az 'egy' részletet\n";
    }
}
```

A hangsúly az `if` feltételén van. A `/`-ek közé egy reguláris kifejezés van írva, ez kerül illesztésre, s amennyiben benne van az adott változóban, akkor igaz az értéke, ha nincs, hamis. Az aktuális változó, ha nem adjuk meg, akkor a `$_` rejtett változó. Megadni a következőképpen tudtuk volna:

```
@tomb=('teljes', 'egyetem', 'begy', 'lehet', 'beetet', 'stb');
```

```
foreach(@tomb)
{
    if (\$_=~ /egy/)
    {
        print "A(z) \$_ szó tartalmazza az 'egy' részletet\n";
    }
}
```

Ez így nagyon egyszerűnek tűnik, ezért kezdjük el bonyolítani. A feladat legyen az, hogy válasszuk ki azokat a szavakat, amelyekben legalább kettő 'e' betű van. Ezt a következőképpen tudjuk megtenni:

```
@tomb=('teljes', 'egyetem', 'begy', 'lehet', 'beetet', 'stb');
foreach(@tomb)
{
    if (/e.*e/)
    {
        print "A(z) \$_ szó legalább két e betűt tartalmaz\n";
    }
}
```

Az általunk alkalmazott trükk az volt, hogy egy 'e', bármilyen karakterek, 'e' részletre kerestünk rá. Ha volt ilyen, akkor két 'e' betű van a szóban. A kifejezésben (a '/' jelek között) a '.' egy bármilyen (kivéve a soresmelés) karaktert jelöl, a '*' pedig azt, hogy az előző karakterből bármennyi (0 vagy több) lehet, azaz pontosan azt jelenti, ami nekünk is kellett. Ha azokra a szavakra szeretnénk rákeresni, melyek egy 'e'-t, egy bármilyen karaktert és egy másik 'e'-t tartalmaznak, akkor egyszerűen el kell hagynunk a '*'-ot.

Megismertünk tehát két "joker" karaktert a reguláris kifejezések világából, most nézzünk egy táblázatot a továbbiakról:

```
. - bármely karakter
* - 0, vagy több az előző karakterből
+ - 1, vagy több az előző karakterből
? - 0, vagy 1 az előző karakterből
^ - sor (string) eleje
$ - sor (string) vége
```

Ha a fenti karakterekre szeretnénk rákeresni, akkor használhatjuk a '~t', írjuk a karakter elé! Vegyünk egy bonyolultabb példát, a fentiek szemléltetésére:

```
foreach (@emailcimek)
{
    if (/^.+@\@.+ \....?\$/ )
    {
        print "A(z) \$_ cím szabályos e-mail címnek tűnik\n";
    }
}
```

A fenti kódrészletben egy e-mail címet ellenőriztünk le, azzal a megkötéssel, hogy azt veszünk szabályos e-mail címnek, amiben van egy @, utána valahol egy pont majd két vagy három karakter a végén. A fentiek jelentése rendre: a sztring eleje (^), egy vagy több bármilyen karakter (.) egy @ (@), egy vagy több bármilyen karakter (@_), egy pont (\.), kettő vagy három bármilyen karakter (??), sztring vége. Hány sor kellene a reguláris kifejezések használata nélkül, hogy megoldjuk ezt a problémát?

Vezessünk be két újdonságot, melyek egy kicsit jobbá, illetve egyszerűbbé teszik a kifejezésünket:

```
foreach (@emailcimek)
{
    if (/^[a-z]+\@[a-z.]+\.{2,3}\$/ )
    {
        print "A(z) \$_ cím szabályos e-mail címnek tűnik\n";
    }
}
```

Az újdonság a szögletes és a kapcsos zárójelek használata. A példa kedvéért most csak az angol kisbetűket tartalmazó e-mail címeket fogadjuk el. A kapcsos zárójelek között karaktereket kell megadnunk és ezen karakterek lehetőségét jelölik. Lehet rövidíteni is, a '-' jellel elválasztva egy tartományt adhatunk meg. A '[a-z]' tehát az jelenti, hogy egy kis angol betű. A '[a-z.]' pedig értelemszerűen, hogy egy kis angol betű, vagy egy pont. A kapcsos zárójel az előző karakterre hivatkozik, egy tartományt (mint a fenti példában), vagy egy számot adhatunk meg benne, mely azt mondja meg, hogy az előző karakterből mennyinek kell lennie.

Végezetül, mielőtt rátérnénk a műveletekre, még a következő táblázattal a rendelkezésünkre álló "makrókat" szeretném bemutatni, melyek segítségével több esetben rövidíthetünk:

```
\d - számjegy
\D - nem-számjegy
\w - szó
```

`\W` - nem-szó
`\s` - elválasztó karakter (szóköz, újsor, tabulátor)
`\S` - nem-elválasztó karakter

Eddig tehát az illesztő operátorral ismerkedtünk meg, azonban a Perl-ben nem csak ez az eszköz van, ahol reguláris kifejezéseket használhatunk. Következőnek ismerkedjünk meg a helyettesítő operátorral, melynek segítségével a sztringünk egyes részeit tudjuk lecserélni az általunk megadottakra.

Vegyünk egy egyszerű példát, valósítsunk meg egy a `chomp` parancshoz hasonló helyettesítő operátort:

```
\$line="Ez egy sor\n\r";  
\$line=~s/[\r\n]//g;
```

Mint látjuk a szintakszisa hasonlít az illesztő operátorhoz, azzal a különbséggel, hogy még egy `'/'` kerül a végére. A `'g'` betű egy paraméter, azt adja meg, hogy globálisan kell végrehajtani a cserét, azaz ha több előfordulást talál, akkor mindet cserélje le. Enélkül csak az első előfordulást cserélnénk le. Nézzük meg végül, hogy mit csinál a kódrészlet: az összes `return` és `newline` karaktert kitörli (lecseréli a két `'/'` közötti részre, azaz esetünkben egy üres sztringre). Ezzel az utasítással tehát a reguláris kifejezésünkre illő sztringrészleteket tudjuk lecserélni az általunk megadottakra.

Talán ez a két parancs az, melyeket a legtöbbet használhatunk a reguláris kifejezéseknél. Ezeken kívül a Perl még tartalmaz párat, továbbá rengeteg trükköt vihetünk végbe velük. További ismerkedéshez a minden Perl disztribúcióban benne található reguláris kifejezéseket bemutató tutorial-t javaslom.

Szöveges állománykezelés, input/output

Szöveges állományok kezelése Perl nyelvben is a megszokott `'open'` és `'close'` eljárásokkal van megvalósítva. Az egyik különbség, hogy a megnyitott állományokhoz rendelt állományazonosítókra a Perl egy külön változótípust használ (melyet nem említettem meg a korábbiakban az egyszerűbb érthetőség miatt). A következő programrészlet megnyit egy szöveges állományt, és kiírja a sorait a képernyőre:

```
open(FILEID, '<proba.txt');  
print <FILEID>;  
close(FILEID);
```

Az első sorban a `'FILEID'` nevű állományazonosítóhoz rendeljük hozzá az állomány megnyitásával magát az állományt (egy állomány több azonosítóhoz is hozzárendelhető). A későbbiekben ezzel az azonosítóval tudunk a `'megnyitott'` állományra hivatkozni. Az `'open'` második paramétere az állománynév, melyben a UNIX környezetben

megszokott kisebb-nagyobb jeleket használhatjuk a megnyitási mód meghatározására. Példánkban olvasásra nyitjuk meg az állományt, ha a '>' jelet használjuk, akkor írásra nyitódik meg, s a tartalma egyből elvész, ha pedig a "»" jeleket használnánk, akkor hozzáfűzésre nyílna meg az állomány. Az utolsó sorban megszüntetjük a hozzárendelést (nem szükséges, de illik megszüntetni, kilépéskor a Perl bezárja a nyitva maradt állományokat).

A második sorban kerül beolvasásra és kiíratásra az állomány tartalma. A kisebb-nagyobb jelek közé tett állomány-azonosító szolgál egy sor beolvasására. Ha tömbként van kezelve, akkor az állomány összes sorát tartalmazza, ha skalárként, akkor mindig csak a következőt. Példánkban azért kerül kiírása a teljes állomány, mert a 'print' utasításnak meg lehet adni egy tömb változót, s ilyenkor összefűzve megjeleníti az elemeit. Egy '\$sor=<FILEID>' tehát egy sort, míg egy '@sorok=<FILEID>' a teljes állományt, soronként a tömb egyes elemeihez rendelve tölti be.

Az állományba való írást a következőképpen valósíthatjuk meg Perl-ben (az állomány azonosítóját - vessző nélkül! - kell a print után írni):

```
print FILEID "Hello World!";
```

A Perl nyelvben léteznek a standard kimenetre, bemenetre és hibakimenetre beépített állományazonosítók. Ezek rendre 'STDOUT', 'STDIN' és 'STDERR'. Ezek használata teljesen megegyezik a többi állományazonosítóéval:

```
print STDERR "Hello World!";  
\$sor=<STDIN>;
```

Az utóbbi példa tehát egy sort fog beolvasni a standard bemenetről, ami alapértelmezésként a billentyűzet, de ha 'perl programnév <egyállomány>' módon hívjuk meg a programunkat, akkor az egyállomány tartalma lesz.

Összefoglalás

Mint láthattuk, a Perl nyelv nagyon sok olyan egyedi beépített tulajdonsággal rendelkezik, melyet széleskörűen lehet felhasználni, s mellyel kevés nyelv büszkélkedhet. Az előzőek azonban csak egyszerű ismerkedést biztosítottak a nyelv lehetőségeivel, korántsem lett bemutatva minden, amit a magával a nyelvvel, illetve a kiegészítéseivel megvalósíthatunk.

További ismerkedéshez a <http://www.perl.org.hu/> címen található linkeket ajánlom, melyek közül számos színvonalas mű segíthet a további ismerkedésben.

18.4. Az awk

18.5. A grep

18.6. Az Xdialog

18.7. Alapok

18.8. Itt egy c forráskód, fordítsuk le

Egyszer csak hopp találkozunk egy kis `c` forráskóddal. Valamit tenni kellene vele, hogy lássuk a működését. A `gcc` az a program amire szükségünk van, hogy a programunk futtatható legyen.

```
1          /* talaltunk egy forraskodot, hello.c */
2
3  #include <stdio.h>      /* header fajl beepitese programunkba */
4
5  int main(void)          /* foprogramunk */
6
7  {                        /* kezdo kapcsoszarojel */
8
9      printf("Helló!\n");  /* kiiratom i-t decimalisan */
10
11     return(0)            /* visszateresi ertek, hogy a shell */
12                          /* tudja minden rendben volt a */
13                          /* futtataskor */
14
15 }
```

A `gcc` számára több kapcsolót is megadhatunk. Lehetőség van az elő-feldolgozó¹, ill. a többi lépés kimenetének megtekintésére. Az előzőleg előkerült `hello.c` programunk segítségével nézzük hogyan is tudjuk lefordítani, kipróbálni.

A `gcc hello.c` parancsot kiadva az alábbi történik:

```
covek@linux:~/c/programok$ gcc hello.c
covek@linux:~/c/programok$ ls
a.out  hello.c
covek@linux:~/C/prg/elagazas$
```

Az `ls` parancs kilistázza az aktuális alkönyvtárat. A keletkezett `a.out` egy alapértelmezett fájlnev ami a fordításkor jött létre. Megnyitva egy szerkesztővel:

¹Később erről még lesz szó!

```

00000000 7F 45 4C 46 x 01 01 01 00 x 00 00 00 00 x 00 00 00 00 .ELF.....
00000010 02 00 03 00 x 01 00 00 00 x 00 83 04 08 x 34 00 00 00 .....4...
00000020 A0 07 00 00 x 00 00 00 00 x 34 00 20 00 x 06 00 28 00 .....4. ....
00000030 1B 00 18 00 x 06 00 00 00 x 34 00 00 00 x 34 80 04 08 .....4...4...

```

Ebből ami lényeges: `.ELF`. Jelentése: futtatható és összeépíthető formátum (Executable and Linkable Format). Az a szabványos Linux futtatható formátum. Adjuk ki a `./a.out` parancsot. Lefuttatva ezt kapjuk:

```

covek@linux:~/C/prg/elagazas$ ./a.out
Helló!
covek@linux:~/C/prg/elagazas$

```

A `gcc`-vel részletesebben később fogunk foglalkozni. Amit most láttunk arra volt jó, hogy azt lássuk nem is olyan vészes egy `c` program használatbavétele! A következő fejezetek sajnos eléggé száraz anyagot tárnak fel, de mindenképpen át kell tekinteni az alapokat ahhoz, hogy a bonyolultabb részeket megértsük.

18.9. Szintaktikai elemek

- változók, mutatók,
- operátorok,
- utasítások,
- függvények,
- összetett adatszerkezetek.

18.9.1. Változótípusok

Nézzük meg milyen változó típusok lehetségesek:

egész típusú	<code>int</code>	egész–
karakter	<code>char</code>	8 bit jellegű
lebegőpontos	<code>float</code>	lebegőpontos–
dupla lebegőpontos	<code>double</code>	jellegű
felsorolás	<code>enum</code>	

A változó deklarálásával tulajdonképpen tudatjuk a számítógéppel, hogy az adott változó mekkora helyet fog elfoglalni a memóriában és lefoglaljuk számára a helyet. A programon belül több helyen is deklarálhatunk változókat. Később részletesebben megnézzük melyiknek mi az előnye. A definíció alatt a változó értékének definiálását értjük.

```

1   int i;           /* az i változó számára lefoglalunk 2 byte-ot. */
2   int j, k, l=5; /* ebből a példából látszik, hogy több
3                   változót is fel lehet sorolni és meg
4                   kezdőértéket is adhatunk. */
5   char c;          /* ez itt egy 1 byte helyet foglaló karakter
6                   típusú változó */

```

Integer esetében a kikötés annyi, hogy az integer nem lehet kisebb mint a shortint és nem lehet nagyobb mint a longint. Ha a float 2 byte akkor a double 4, azaz dupla pontosságú lebegőpontos szám.

Módosítók

```

int:      unsigned int    u;  vagy  unsigned    u;  (előjeltelenített)
          long           li;      long           li;  4 bájt
          short          si;      short          si;  16 bit

```

```

char:     signed   char    sc;
          unsigned char    uc;

```

```

float:    —

```

```

double:   long    double    ld;                                10 bájt

```

Itt arról van szó, hogy egy `int i;` változó értéke előjeles, tehát -128 és 127 közötti értékeket vehet fel 8 bit-en. `unsigned i;` viszont 0 -tól 255 -ig.

Az `extern`, `static` és `inline` módosítók:

Tegyük fel, hogy a programunkat több részre bontjuk a könnyebb kezelhetőség érdekében. Van a.c, b.c forrásfájlunk, modulunk. Az a.c -ben deklarálunk egy változót amit a b.c -ből is el szeretnénk érni: `extern int x;`. Ekkor az `x` elérhető mindkét forrásból. Viszont `static int y;` esetén csak abban a forrásállományból érhető el az `y` ahol az deklarálva lett. Ott viszont globálisan használható, értéke modulon belül állandó, kívülről nem elérhető. Ugyan így lehet függvényben is `static` egy, több változó:

```

int fuggveny()
{
    static first = 0;

    if (first == 0)
    {
        first = 1;
    }
}

```

Itt azt jelenti hogy a függvényvégrehajtás után is megőrződik a `first` változó értéke, de lokális marad a függvényen belül. Bizonyos okoknál fogva ez nem mindig szerencsés megoldás.

A `static`-ot függvényekre is lehet használni, mondjuk a.c-ben

```
static int fuggveny ()
{
    blabala;
}
```

Azt jelenti, hogy csak az a.c-ben levő függvények látják a `fuggvenyt()`, más modulokból nem látható.

Az `inline` pedig arra kézteti a fordítót, hogy a függvény tartalmát annak hívásának helyére behelyettesítse, így megspórolja a függvényhívást. Hasonlóan az előfordító makroihoz (`#define`) (Megjegyzés: `inline` nincs minden c fordítóban)

Típus definiálása

`typedef régi újtipus` Készítünk magunknak egy új típust.

`typedef char BOOL;` Ez a típus egy karakter típus amit úgy hívunk, hogy `BOOL`.

```
BOOL a; /* a valojaban karakter tipusu. */
int i;          int i;
double d;       double d;

i = 5;          d = 3.14;
d = i;          i értéke double . i = d;          itt is double
                típusúvá alakul   i = d + 0.5;      lesz i értéke!
```

Ne felejtsük el, hogy a műveletek mindig a magasabb osztályban hajtódnak végre! Automatikus típuskonverzió jön létre.

18.9.2. Operátorok

- elválasztó
- aritmetikai
- relációs

- logikai
- bit
- speciális

Elválasztó operátorok

{ }, (), [], , , ;

{ } operátorpár az úgynevezett blokk operátorok. A logikailag összetartozó utasításcsoport összefogására használhatjuk. A ciklustörzs, az igaz, hamis ág összetartozó parancs-sorainak blokkba foglalására².

() argumentum zárójelezésre, műveletvégrehelyezési sorrend meghatározására használható operátorpár.

[] páros a tömböknél használatos, ahol a szögletes zárójelek közé a tömb elemszámának mennyiségét jelölő szám, vagy az éppen hivatkozott tömbelem sorszáma kerül.

, vessző, makrónál, paraméterek felsorolásánál használhatjuk.

; pontosvessző, kifejezés lezárására kötelezően használatos. Ha elmarad bosszúságot okozhat a hiba feltárásánál.

Aritmetikai operátorok

= (értékadás), + (összeadás), - (kivonás), * (szorzás), / (osztás), % (modulo képzés), - (egy operandusú vagy unáris mínusz).

Fontos, hogy odafigyeljünk a balértékszabályra: $a = 5;$, rossz példa a $5 = a;$, mert a változó mindig baloldalt kell, hogy legyen!

% operátor a modulóképzés operátora. Ezzel a művelettel egy egész osztás maradékát kaphatjuk meg, mint az alábbi példánál!

```
a = 29;
b = 3;

c = a%b;
```

- operátor az egy operandusú vagy unáris mínusz. Egy számot tartalmazó változó előjelét változtatja meg.

```
a = 5;
b = -a;
```

Az a értéke nem változik, viszont b változó értéke -5 lesz.

²mint a PASCAL begin és end

Relációs op.

< (kisebb), <= (kisebb egyenlő), == (egyenlő), != (nem egyenlő), >= (nagyobb egyenlő), > (nagyobb).

Logikai operátorok

&& (és), || (vagy), ! (nem)

Ha a logikai kiértékelés az első vizsgálat után igaz a vizsgálat nem folytatódik tovább hanem igaz lesz. A többi feltétel nem kerül kivizsgálásra.

```
if ( a == 0 && 27 == getchar()) break;

if ( a == 0 )    /* vagy    if ( !a ) */
```

Bit operátorok

Bitenkénti műveletvégrehajtás operátorai:

& (bitenkénti és), | (vagy), ^ (kizáró vagy), ~ (unáris, egyes komplement képzés), << (szorzás 2-vel), >> (osztás 2-vel).

```
1      char a,b,c;
2
3      a = 0xf0;          /*      a 11110000      */
4      b = 0xaa;          /*      b 10101010      */
5
6      c = a & b;          /*      c 10100000      a bitenkenti es eredménye */
7
8      c = a | b;          /*      c 11111010      bitenkenti vagy */
9
10     c = a ^ b;          /*      c 01011010      kizaro vagy */
```

Speciális operátorok

rekurzív csoport, ++ (inkrementáció), -- (dekrementáció), & (címképző), * (indirekciós), . (pont), -> (nyíl), ?: (feltételes értékadás), sizeof(), (cast) (típuskonverzió kikényszerítése).

A sizeof() speciális operátor egy változó, vagy egy változótípus méretét adja vissza byte-ban.

A . (pont) és a -> (nyíl) operátor a struktúráknál használatos operátor.

```
1      a = a + b;          a += b;
2
3      a == a << 1;        a <<= 1;
```

```

4
5   int i = 5, j;
6
7   i = i++;          i = ++i; /* vigyázat, nem mindig adja ugyanazt
8                       az erteket */
9   j = ++i          j = i++ /* elsonel novele i erteket 1-el
10                          es az lesz j erteke, masodiknal
11                          j megkapja i erteket es novele! */

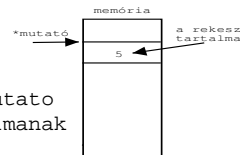
```

Pointer és címképző operátor használata példája:

```

1   int i, *ip; /* *ip egy mutató, ami a memória
2               egy adott területére mutat */
3   i = 5;
4   ip = &i;    /* cím hozzárendelése ip i értéket
5               tartalmazó memória kezdőcímére mutat */
6
7   int i, *ip, j;
8   i = 5;
9   ip = &i;
10
11  j = *ip;    /* indirekció, amikor j értéke az *ip mutató
12               által meghatározott memóriacím tartalmának
13               értékeivel lesz egyenlő

```



j értéke legyen egyenlő az ip által megcímezett memóriarekesz tartalmával.

Feltételes utasítás:

```

int a = 5, b = 6, c;

c = ( a > b ) ? a : b;

```

ha a (a > b) kifejezés igaz akkor a értékét kapja c , ha hamis akkor pedig b értékét.

```

int i, j;

j = sizeof(i); /* i méretét adja vissza */

j = sizeof(int); /* int típus méretét kapjuk */

int a = 5, b = 6;
double d;

d = a/b;

d = (double)a/b; /* kényszerítés, cast-olás */

```


18.9.3. Utasítások

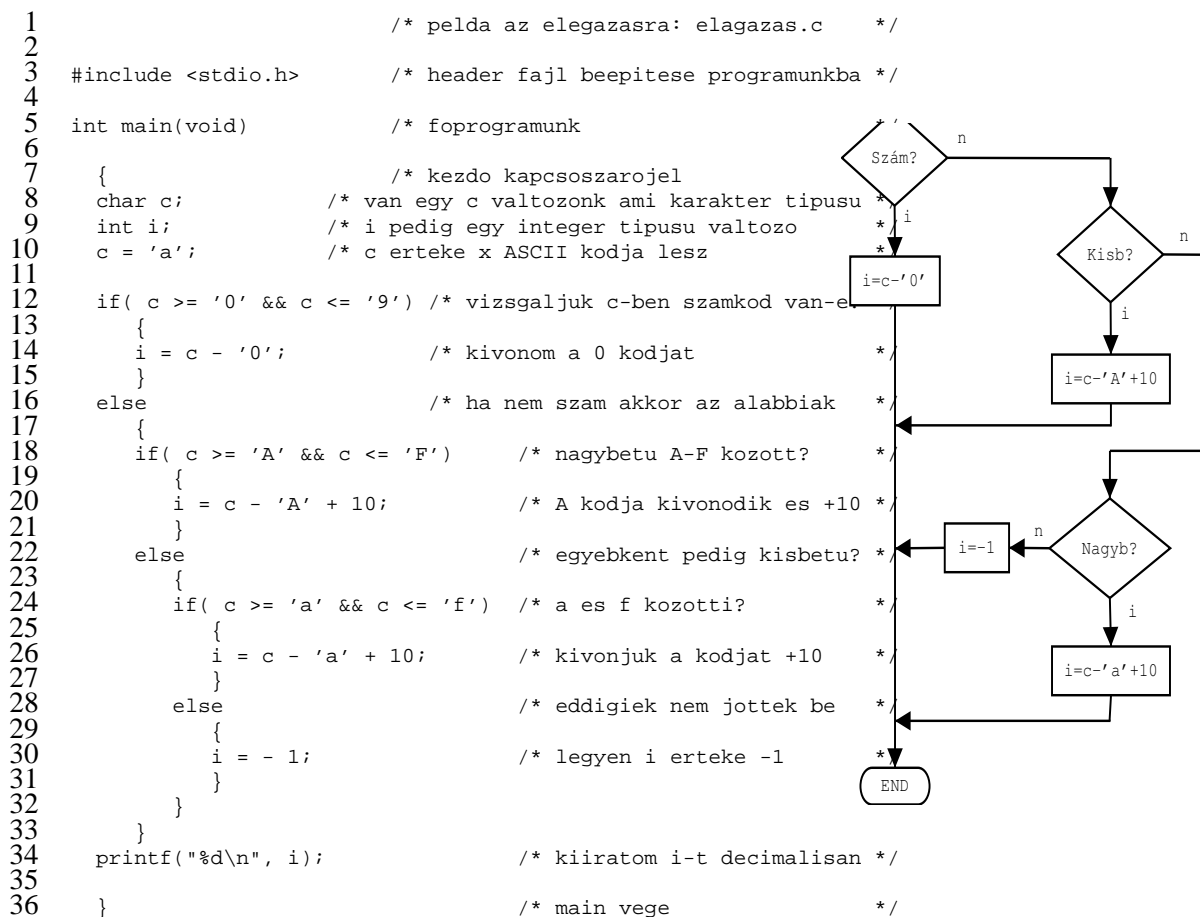
Feltételes elágazás

if(argumnetumok) kifejezés;

Ha a zárójelben található feltétel igaz akkor a kifejezés végrehajtható, egyébként pedig nem.

```
if( igaz ez a feltétel? )
{
    utasítás(ok); /* feltétel igaz */
}
else
{
    utasítás(ok); /* ha a feltétel hamis volt */
}
```

Az else ág elhagyható, nem feltétlenül szükséges. Gépeljük be az alábbi példaprogramot. Ügyeljünk a tagolásra, hogy a nyitó és a záró kapcsos zárójelek pontosan követhetőek legyenek. Ezzel áttekinthetőbb lesz a forrása programodnak.



Az előző példa kicsit tömörebben, de sokkal olvashatalanabban:

```
if(c>='0' && c<='9') i=c-'0';
else if(c>='a' && c<='f') i=c-'a'+10;
    else if(c>='A' && c<='F') i=c-'A'+10;
    else i=-1;
```

Tehát a megfelelő tagolás és magyarázószöveg hozzáfűzése nagyban segít abban, hogy egy hónap múlva is tudjuk mit is csinál egy bonyolultabb program.

18.9.4. Konstansra ugrás elágazás

Ennél az elágazásnál a változó értékét vizsgáljuk és ha valamelyik értékkel megegyezik a felsorolt értékek egyike akkor arra a feladatsorra ugrik. Ha a `break` utasítás elmarad akkor sorban egymás után végrehelytődnek a az egyes `case` értékek: utáni utasítások. Tehát erre érdemes odafigyelni, különben kis programunk furcsaságokat művelhet.

```
switch( változo )
{
    case ertekek1: utasitasok; break; /* opcionalis, ha nincs
                                     itt akkor a kovetkezo
                                     is vegrehejtodik */
    case ertekek2: utasitasok; break; /* opcionalis */
    default: utasitasok;
}

1      /* switch - case szerkezetre ime egy pelda */
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char c;
7      int i;
8      c = 'x';
9
10     switch(c)
11     {
12         case '0':
13         case '1':
14         case '2': /* vegig azt vizsgaljuk, hogy szam-e */
15         case '3': /* a c-nek atadott ertekek */
16         case '4':
17         case '5':
18         case '6':
19         case '7':
20         case '8':
21         case '9': i = c - '0'; break;
22         default: i = -1;
23     }
24     printf("%d\n", i); /* kiiratom i-t decimalisan */
25 }                      /* main vege */
```

Van még egy utasítás ez pedig a `goto CIMKE` utasítás. Nem fogjuk használni, ha csak nincs más megoldás.

18.9.5. Ciklusszervező utasítások

for ciklus

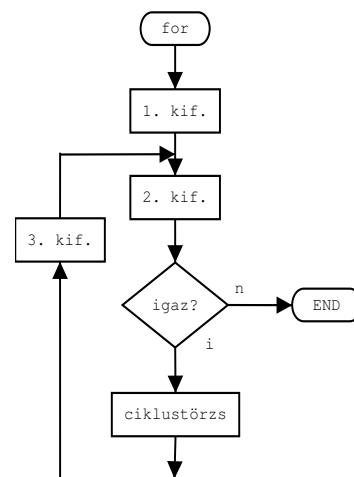
```
for(1.kif; 2.kif; 3.kif)
{
    utasítások;
}
```

Az 1. kifejezés egy értékadó utasítás. A 2. kifejezés a vizsgálat tárgya és ha igaz vagyis nem nulla akkor a ciklustörzs végrehaltásra kerül. A 3. kifejezés a ciklustörzs végrehaltása után kerül kiértékelésre, végrehaltásra.

```
int main(void)
{
    int i;
    long l=1;

    for(i=1; i<=10; i++)
    {
        l = l * i;
    }

    /* vagy: for(i=1,l=1;i<=10;i++) */
    /*      l*=i;                      */
}
```



Legnagyobb közös osztó kiszámítása:

411 =	923	mod	512
101 =	512	mod	411
7 =	411	mod	101
3 =	101	mod	7
Inko 1 =	7	mod	3
0 =	3	mod	1

```
int main(void)
{
    int a,b,c;

    a=123;
    b=201;
    for(; c=a%b;)
    {
        a=b; b=c;
    }

    /* egyik szam */
    /* masik szam */
    /* for(;c=a%b;a=b,b=c); */
}
```

Elöltesztelő ciklus

```
while(kifejezés)
{
    utasítások;
}
```

Ha a kifejezés igaznak bizonyul az utasítások végrehelytődnek, ha nem a program az utasításokat lezáró kapcsos zárójel után folytatódik.

```
1 int main(void)
2 {
3     int a,b,c;
4
5     a= ;
6     b= ;
7     while(c)
8     {
9         c=a%b;
10        a=b;
11        b=c;
12    }
13 }
```

Végtelen ciklust így a legegyszerűbb kreálni: `while(1)` vagy `for(;;)`.

Hátultesztelő ciklus

Ennek a ciklusnak a törzse egyszer mindenképpen kiértékelődik!

```
do
{
    utasítások;
}
while(kif.);
```

```
1 /* legnagyobb kozos osztó kiszamitasara
2    egy másik megoldas, hetultesztelos
3    ciklus felhasznalasaval. */
4
5 int main(void)
6 {
7     int a,b,c;
8
9     a=123;
10    b=201;
11    do
12    {
13        c=a%b;
14        a=b;
15        b=c;
16    }
17    while(c);
18 }
```

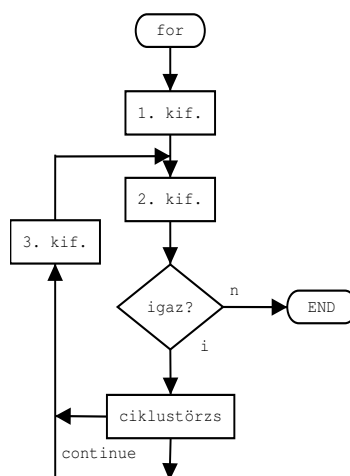
A break, feltétlen kiugrás, utasítás

Egy végtelen ciklusból a `break;` hatására tudunk kiugrani. A program futása a ciklustörzs utáni utasítással folytatódik.

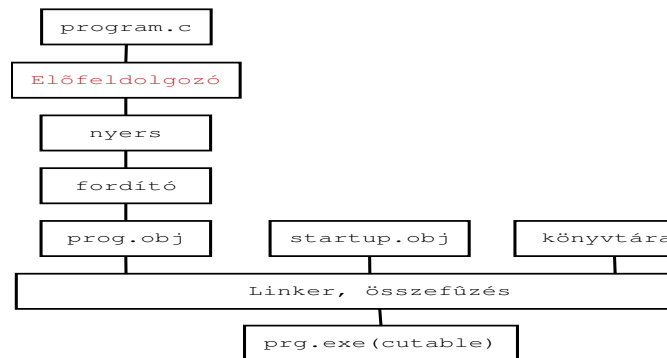
```
while(1)
{
    utasítások;
    if( ) break;
}
```

A continue, folytatás, utasítás

```
while(1)
{
    utasítások;
    if( ) continue;
    /* mindenkeppen szeretnem, hogy a ciklus */
    /* ismetelten lefusson */
}
```



18.9.6. Előfeldolgozó utasítások



18.1. ábra. A fordítás menete

Az előfeldolgozó utasítások a fordítás során hajtódnak végre és speciális szerepük van. Ez látszik a 18.1. ábrán. Minden előfeldolgozó utasítás `#` -al kezdődik.

```

1 {1}
2
3 #include <stdio.h>           /* header */
4 #include "enyem.h"          /* saját header file */
5 #define NULLA 0             /* fordítás idejű szoveghelyettesítés */
6 #define MAX(a,b) (a>b)?a:b /* makró, figyelni a szokozokra */
7 #define ERR(a,b) printf(a),exit(b) /* makró hibakezelesre */
8
9 #undef X                     /* szoveghelyettesítés visszavonása */

1 {4}
2 #define X 5
3
4 #undef X
5 #define X 15
  
```

`#include "enyem.h"` sorban az `enyem.h`, egy saját készítésű könyvtárállomány behívását teszi lehetővé. Ilyen könyvtarakat magunk is készíthetünk, ennek módjáról még lesz szó a későbbiekben.

Függvények

- Minden függvénynek van visszatérési értéke (kivéve, ha mi nem akarjuk). Alapértelmezésben integer típusú.

– Deklaráció

```
típus függvénynév(prgszignatúra_típus(ok));
```

```
int max3(int, int, int);
```

A deklaráció történhet külön header fájlban is. Ennek a moduláris programozásban van fontos szerepe. Mire is gondolhatunk? Megírunk egy célra alkalmas függvényt, ez olyan jól sikerült, hogy más programunkban is felhasználhatjuk. Készítünk külön könyvtárat ami tartalmazza a függvényt és elkészítjük a hozzá tartozó header fájlt is. Ezáltal univerzális függvényünk több programban is felhasználható.

– Definíció

```
1  típus függvénynév(változók felsorolása)
2  átadott változók deklarációja
3  {
4      blokkra lokális változók létrehozása
5      műveletek;
6      return változó;
7  }
```

A függvény definíciója valójában leírja, hogy mit is csinál az adott függvény. Készíthetünk például egy speciális számításhoz külön függvényt amit később több programban is használhatunk.

```
1  int max3(a,b,c)      függvény(int a, int b, double c)
2  int a,b,c;          {
3      {                // függvénytorzs
4      int i;           }
5      i=(a>b)?a:b;
6      i=(i>c)?i:c;      // függvények definíciójára
7      return i;        // két pelda
8  }
```

Egy példa a függvény() használatára:

```
1  /* három számból melyik a nagyobb? */
2  #include <stdio.h>
3  int max3(int,int,int); // deklaralom a függvenyt, //
4  // amit max3 -nak hívtam //
5  int main(void)        // ez a main //
6  {                     // kezdete //
7      int x=5,y=4,z=9;   // deklaralom es erteket adok //
8      int r;            // deklaralom //
9      r=max3(x,y,z);     // függvenyt meghivom //
10     printf("maxérték: %i\n", r);
11     // visszateresi erteket kiiratom //
12     /* ide return nem kell!!! */
```

```

13     }
14     int max3(a,b,c)           // definialom fuggvenyem           //
15     {
16         int a,b,c;           // bejovo valtozok deklaracioja    //
17         {                   // belseje fuggvenynek             //
18             int i;           // lokalis valtozo deklaracioja    //
19             i=(a>b)?a:b;      // melyik nagyobb ha a i=a amugy i=b //
20             i=(i>c)?i:c;      // c nagyobb?           //
21             return i;        // ami nagyobb az adodik vissza //
22         }                   // itt a vega fuggvenyenemnek        //
23     }

```

Másik példa amikor több visszatérési értéke van egy függvénynek:

```

1      /* másodfokú függvény megoldása          */
2      /*      ax^2+bx+c=0                      */
3      #include <stdio.h>                       // standard input output -ot használjuk //
4      #include <math.h>                       // math.h -bol is kell nekem az sqrt() //
5
6      int masod(double,double,double,double*,double*);
7      // fuggvenydeklaracio, ahol a           //
8      // double* lesz a pointer ahova        //
9      // az eredmeny kerul.                   //
10     int masod(a,b,c,x1,x2)                   // fuggvenydefinicio //
11     {
12         double a,b,c;                       // deklaralom a bejovoket //
13         double *x1,*x2;                     // deklaralom a kimenoket! //
14         {                                   // itt kezdjuk a fuggvenyt //
15             double d;                       // deklaralom localis d-t //
16             d=b*b-4*a*c;                   // szamlalo gyok alatti resze //
17             if(d>0)                         // gyok alatt kisebb mint nulla? //
18             {                               // egy if kezdete //
19                 d=sqrt(d);                 // ha nagyobb a negyzetgyok //
20                 // alatti akkor szamolunk //
21                 *x1=(-b+d)/(2*a);          // egyik eredmeny kiszamitasa, //
22                 *x2=(-b-d)/(2*a);          // masik eredmeny kisz, //
23                 return 1;                  // case-nek vezzerles //
24             }                               // if1 bezar //
25             if(d<0)                         // if2 feltetel //
26             {                               // if2 kezdete //
27                 *x1=(-b)/(2*a);            // valos resz visszaadasa //
28                 *x2=sqrt(-d)/(2*a);        // komplex resz visszaadasa //
29                 return 2;                  // case-nek erteke a kiirataashoz //
30             }                               // if2 vege //
31             *x1=(-b)/(2*a);                // 3. eset vizsgalata //
32             return 3;                      // case-nek 3-at a kiirataashoz //
33         }                                   // fuggvenyunk vege //
34
35     int main(void)
36     {
37         // ez itt a FO fuggvenyunk, //
38         // kezdete //
39         double A=1 ,B=4 ,C=1 ;             // deklaralunk, ertekeket adva //
40         double x1,x2;                       // visszateresi ertekek deklaralasa //
41         int r;                              // deklaralom fuggveny visszat //
42         r=masod(A,B,C,&x1,&x2);              // meghiv fugveny //
43         switch(r)                           // vizsgalom mi jon vissza //
44         {                                   // switch case kezdete //
45             case 1: printf("x1= %lf x2= %lf\n",x1,x2);return; //
46                     // kiirjuk ha van ket eredmeny //
47             case 2: printf("re= %lf im= %lf\n",x1,x2);return; //
48                     // kiirjuk valos es kepzetes reszt ha gyok //
49                     // alatt nullanal kisebb erteke van //
49             case 3: printf("x1=x2= %lf\n",x1);

```

```

50                                     // kiirjuk ha gyok alatt nolla van           //
51     }                               // switch vege                               //
52 }                                   // main vege                               //

```

Figyelem!

A fordítást és futtatást az alábbiak szerint tudjuk elvégezni:

```

$ gcc -lm -o masod masod.c
$ ./masod
x1= -0.267949    x2= -3.732051
$
$

```

A `gcc` az `-lm` kapcsoló hatására a `math.h` headerben található függvényeket is megtalálja. Az `-o` kapcsoló lehetővé teszi, hogy megadjuk mi legyen az indítható állomány neve. A példa szerint `masod` az a fájl ami `./masod` formában indítható.

Ez pedig egy példa a rekurzív függvényhívásra. Akkor amikor egy függvény meghívja önmagát:

```

1  /* 10 faktorialisa rekurziv fuggvenyhivassal, 10-tol visszafele 1-ig */
2
3  #include <stdio.h>           // standard input output -ot használjuk //
4
5  long fakt(int);             // deklaralom fuggvenyem                //
6
7  long fakt(n)                // definialom fuggveny                 //
8  {
9      int n;                  // bejovo változo tipusa             //
10     {
11         long r;              // lokalis r tipusa          //
12         if(n>1)              // vizsgalat bejovo nagyobb mint n? //
13         {
14             r=fakt(n-1)*n;    // meghivja onmagat         //
15         }
16         else r=1;             // ha n = 1 kilep           //
17         return r;            // visszaadjuk r erteket    //
18     }
19
20     int main(void)
21     {
22         long f;
23         f=fakt(10);
24         printf("%ld\n",f);    // kiiratom a visszkapottat //
25     }                         // longdec formaban         //

```

18.10. Összetett adatszerkezetek, TÖMB

Tömb definíciója: olyan összetett adattípus, amely azonos típusú elemeket tartalmaz.

Tömb deklarációja:

típus tömb_neve[tömb_elemek_száma_azaz_mérete];

A tömb nem lehet void típusú!

```
int it[8]                /* -> 0, 1, 2, ... , 7                */
    it[0]=2; it[1]=3; ... /* feltöltjük a tömböt értékekkel */
    it[7]=14;

int it[8] = {11,12, ... , 17}; /* rögtön értéket is adunk neki *
    it[2] = it[7];             * a 7. elem értéke bekerül a 2.*
                                * elem helyére, értékátadás    */
```

18.10.1. Karakter tömb, string

```
char c[5] = {'a','c','m','z',0}

/* mindig nulla az utolso char eseten jelzi a string veget */

'a' -> mindig cim
"a" -> mindig kod

char string[]={"szoveg"}; /* a kapcsoloszarokeket el lehet hagyni */
```

18.10.2. Konstansok

Konstansok = nem látható karakterek, állandók.

\a	csengő karakter, BELL, LF
\r	kocsi vissza, CR
\t	tabulátor
\b	backspace
\	\, backlash
\'	', aposztrof
\"	", idézőjel
\xFF	x hexa kódban
\ooo	oktális
\n	új sor karakter

18.10.3. Buborék rendezés

A tömb elemeit összehasonlítja és ha kell cserél.

1. 2 csere 9 3 5
2 9 csere
2 3 9 csere

2. 2 csere 9 3 5
2 3 csere
2 3 3 csere
2 3 5 9



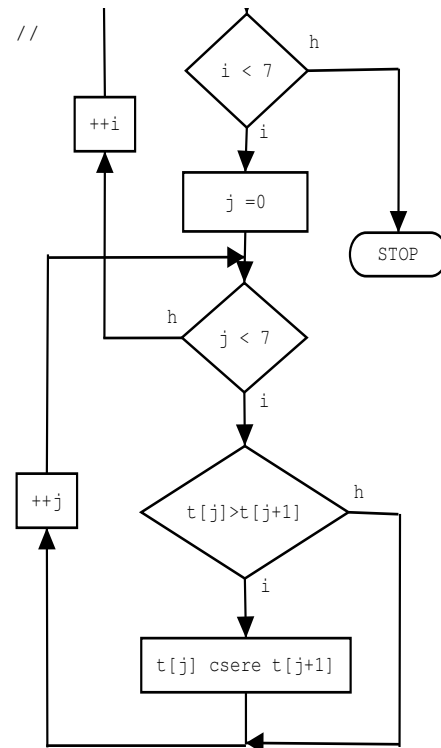
Buborék I.

Viszi tovább a nagyobbbat.

```

1  # include <stdio.h>
2
3
4  int t[8]={1,6,2,8,13,4,45,9}; // tomb feltoltesei //
5
6  void main (void)           // foprogram //
7  {
8      int s;                  // s segedvaltozo, egesz //
9      int i, j;               // ciklusvaltozok, egesz //
10     for (i=0;i<7;i++)
11     {
12         for (j=0;j<7;j++)
13         {
14             if (t[j]>t[j+1]) // ha j edik elem nagyobb //
15             {
16                 // mint a j+1 edik elem //
17                 s=t[j];t[j]=t[j+1];t[j+1]=s; // cserelgessunk //
18             }
19         } // belso for vege //
20     } // kulso for vege //
21 } // main vege

```

**Buborék II.**

```

1  # include <stdio.h>
2
3
4  int t[8]={9,2,16,3,47,13,22,1};
5
6  void main (void)
7  {
8      int s;
9      int i,j,k;
10     for (i=0;i<7;i++)
11     {
12         k=0;
13         for (j=0;j<7-i;j++)
14         {
15             if (t[j]>t[j+1])
16             {s=t[j];t[j]=t[j+1];t[j+1]=s;k=1;}
17         }
18         if (k==0) break;
19     }
20 }

```

Buborék III.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      int a,n,i,j,cs,x;          /*          idáig ok          */
7      int tomb[15];
8      char string[255];
9
10     printf("Mennyi elemet rendezzek? (max 15)\n"); /* elemek szama? */
11     do
12     {
13         gets(string);          /* bekerunk egy erteket          */
14         n=atoi(string);      /* ascii bol csinalunk integert */
15     }
16     while(n>15);              /* a ciklus addig varakozik meg a
17                             /* beirt szam nem kissebb vagy
18                             /* egyenlo mint 15
19     for(a=0;a<n;++a)          /* ciklus az adatok beolvasasara */
20     {
21         printf("Kérem a(z) %d. elemet!\n",a+1); /* elemek bekerese */
22         gets(string);          /* string-be kerül          */
23         tomb[a]=atoi(string); /* tombot feltöltjük, stringbol int */
24     }
25
26     for(a=0;a<n;++a)          /* tomb elemeinek kiiratasa */
27     {
28         printf("A beírt %d. elem: %d\n",a+1,tomb[a]);
29     }
30     /* a 0-dik elem, de kiírva jobb így */
31     i=n-1;
32     while(i>=1)              /* buborek2 tartalmat felhasznaljuk */
33     {
34         cs=0;
35         for(j=0;j<i;++j)
36             if (tomb[j]>tomb[j+1])
37             {
38                 cs=j;
39                 x=tomb[j+1];
40                 //printf("%d\n ",x); /* ha kiváncsiak vagyunk kiirathatjuk */
41                 tomb[j+1]=tomb[j];
42                 //printf("%d\n ",tomb[j+1]);
43                 /* ha kiváncsiak vagyunk kiirathatjuk */
44                 tomb[j]=x;
45                 //printf("%d\n ",tomb[j]);
46                 /* ha kiváncsiak vagyunk kiirathatjuk */
47             }
48             /* if zárása */
49             i=cs;
50         }
51         /* while zárása */
52     for(a=0;a<n;++a)          /* rendezett tomb kiiratasa */
53     {
54         printf("A(z) %d. elem: %d\n",a+1,tomb[a]);
55     }
56     /* main vege */

```

```

1. 3 2 1 0
2.      3
3.      2 3
4.    1 2 3
5. 0 1 2 3

```

Többsdimenziós tömbök

```

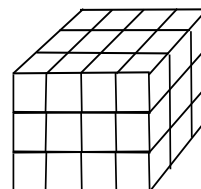
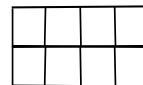
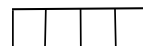
int[8]           1 dimenziós
int[3][3]        2 dimenziós
int[3][3][3]     3 dimenziós

```

```

1  int t[8]={1,2,3, ... ,8};
2  int t[2][2]={1,2,3,4};          /* vagy: = {{1,2},{3,4}} */1/dimenzió
3  int t[3][3][3]={4,9,18,27,33, ... }; /* 27db összesen */
4                                     /* vagy */
5  int t[3][3][3]={
6      {
7          {1,2,3}, /* t[3][3][3] elso [3] tombok szama */
8          {1,2,3}, /* masodik [3] sorok szama */
9          {1,2,3}, /* harmadik [3] elemek szama 1 sorban */
10         },
11         {
12             {1,2,3},
13             {1,2,3}, /* erteekadas: t[2][1][3]=12 */
14             {1,2,3}, /* melyik tomb melyik sor melyik elem */3/dimenzió
15         },
16         {
17             {1,2,3}, /* ha nem adom meg az erteeket */
18             {1,2,3}, /* egy elemnek akkor random */
19             {1,2,3}, /* erteeket vesz fel */
20         },
21     };

```



Rendezés

Szélső érték kiválasztása:

```

1  # include <stdio.h>
2
3
4  int t[12]={12,11,10,9,8,7,6,5,4,3,2,1};
5
6  int main (void)
7  {
8      int i,j;
9      int s;
10     for (i=0;i<11;i++)
11     {
12         for (j=i+1;j<12;j++)
13         {
14             if (t[i]>t[j])
15             {
16                 s=t[i];t[i]=t[j];t[j]=s;
17             }
18         }
19     } /* for(i=0;i<8;i++) */
20     /* printf("%4d", t[i]); */
    /* printf("\n"); */

```

```

1  9 -2 4 5
-2 9 1 4 5
    1 9 4 5
        4 9 5
            5 9
-2 1 4 5 9

```

Pointerek, mutatók

```

int *ip, jp, i;          /* ip -> integer típusu pointer */
int t[5];

ip = t;

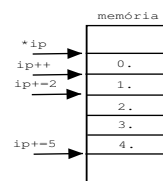
```

Ha egy típussal inkrementálok akkor annyival mozdul amekkora a mérete.

`ip = &t[3]; i = jp - ip;` Ha egész számot adok a pointernek amekkora a típus mérete: * szám .

Pointer aritmetika:

- inkrementálás, növelés
- dekrementálás, csökkentés
- egész hozzáadása
- egész kivonása
- 2 azonos típusú pointer kivonása



```

1  int t[200];
2  int *ip;
3  int j;
4
5  ip=t;
6  j=ip[99]; /* j= *(ip+99); */
7             /*      ^      */
8             /* indirektcio */
9  ip=&i;     /* ertekadas   */

1 # include <stdio.h>
2 # include <string.h>
3 char a[]="lakvfaoskvraln";
4 void qs(int,int,char *);
5
6 int main (void)
7 {
8     int l;
9     l=strlen(a);
10    printf("%s\n",a);
11    qs(0,l-1,a);
12    printf("%s\n",a);
13 }
14
15 void qs(int l, int r, char *x)
16 {
17     int i,j;
18     char s,c;
19     i=l; j=r;
20     c=x[(r+l)/2];
21     do
22     {
23         while (x[i]<c) i++;
24         while (c<x[j]) j--;
25         if (i<=j)
26         {
27             s=x[i];x[i]=x[j];x[j]=s;
28             i++;j--;
29         }
30     }
31     while (i<=j);
32     if (l<j) qs(l,j,x);
33     if (i<r) qs(i,r,x);
34 }

```

Shell rendezés

A Shell féle rendezés a távol lévő elemeket hasonlítja össze, utána közeledik egymáshoz a két érték összehasonlításában.

Rendezetlenség: csökkentjük a halmaz entrópiáját.

```

1  # include <stdio.h>
2  # include <string.h>
3
4  unsigned char a[]="lksadfjf";
5
6  void sh(unsigned char *,int); /* egy karakternyi pointert */

```

```

7                                     /* es int tipust atadunk */
8  int main (void)
9  {
10  int l;
11  l=strlen(a); /* char tomb hosszat adja vissza a \0 nelkul */
12  printf("%s\n",a); /* string kiiratas */
13  sh(a,l); /* fuggvenyhivas */
14  printf("%s\n",a);
15  }
16
17  void sh(unsigned char *x, int n)
18  {
19  int i,j,k; /* k tavolsag jellegu valtozo, ha elfogy vege */
20  unsigned char s;
21
22  for (k=n/2;k>0;k/=2) /* k=k/2 */
23  {
24  for (i=k;i<n;i++) /* lepkes */
25  {
26  for (j=i-k;j>=0&&x[j]>x[j+k];j-=k)
27  {
28  s=x[j];x[j]=x[j+k];x[j+k]=s; /* cserelegessunk */
29  }
30  }
31  }
32  }

```

18.10.4. Struktúra

Definíciója

Def.: többtípusú adatokat fűzünk össze segítségével.

Deklarációja

```

strukct color_chr /* color_chr a struktura neve */
{
  int x; /* így nez ki a mi strukturank */
  int y;
  int color;
  int bkcolor;
  char code;
}

```

Valahol a program során:

```

1  struct color_chr a; /* valtozokent hasznalom */
2
3  a.x = 5; /* a nevu struktura x mezo erteke legyen 5 */
4

```



```

5
6 char c;
7 struct color_chr a;
8
9 a.code = 'A';
10
11 c = a.code;

```

A konstansokat a függvényekhez soroljuk mert értéket szolgáltatnak!

```

1 typedef struct
2 {
3     int x;
4     int y;
5     int color;
6     int bkcolor;
7     char code;
8 } color_chr;
9
10
11 color_chr a;
12
13 a.code = 'B';
14
15 sizeof()          /* operator, megmondja a pontos *
16                   * meratet a struct -nak          */
17
18
19 typedef struct    /* bit field                      */
20 {
21     unsigned x:    7;
22     unsigned y:    5;
23     unsigned color: 4;
24     unsigned bkcolor:3;
25     unsigned code:  7;
26 } color_chr;
27
28
29 int s: 1;          /* nulla vagy -1                      */
30
31 struct color_chr
32 {
33     int x;
34     int y;
35     int color;
36     int bkcolor;
37     char code;
38 } a;

```

Nyíl operátor

```

struct color_chr a;

struct color_chr *ap;

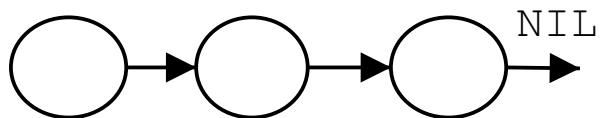
ap = &a;
ap ->x = 5;

```

```
/* az ap nevu struct pointer által mutatott  *  
* struct x mezoje legyen egyenlo 5          */
```

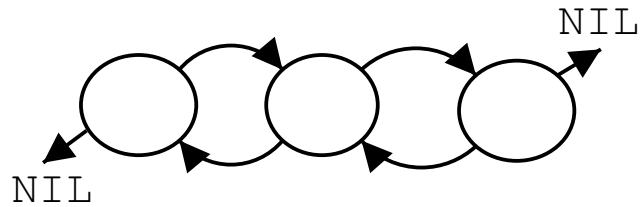
Rekurzív struktúra

A struktúra tartalmazhat mindent kivéve az olyan struktúrát ami ugyan olyan mint ön-maga!



18.2. ábra. Lista adatszerkezet

```
1 struct LIST
```



18.3. ábra. Kétszeresen láncolt adatszerkezet

```

2      {
3      int v;
4      struct LIST *next; /* C -ben ez nem lehetséges */
5      };
6
7
8  typedef struct
9  {
10     int v;
11     LIST *next;          /* nem megvalósítható      */
12     } LIST;

```

18.10.5. UNION

Különböző típusú adatokat fűzünk össze és kétféleképpen láthatjuk őket.

```
union FOO
{
    float f;
    long l;
};

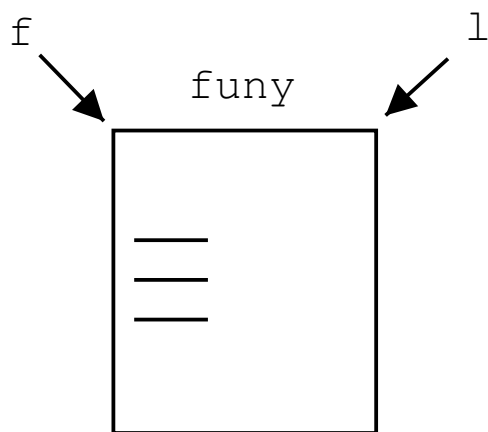
union FOO funy;

funy.l = 10000;
funy.f = 3.1415;
```

```
1 union IFC          /* 1/byte = nibble */
2 {
3     float f;
4     char bytes[4];
5 };
6
7
8 typedef struct
9 {
10    unsigned data:    12;
11    unsigned busy:    1;
12    unsigned ready:   1;
13    unsigned unknown1:1;
14    unsigned unknown2:1;
15 } DATA;
16
17
18 union IFC
19 {
20     DATA d;
21     char bytes[2];
22 };
23
24
25 union IFC ifc;
26
27
28 ifc.d.unknown1 = 1;
29
30 _do();
31
32 void _do(void)
33 {
34     outportb(cim.ifc.bytes[0]);
35     outportb(cim.ifc.bytes[1]);
```

18.10.6. REGS union

```
typedef BYTE_REGS /* byte regiszter definialas */
{
    unsigned char al,ah,bl,bh,cl,ch,dl,dh;
```



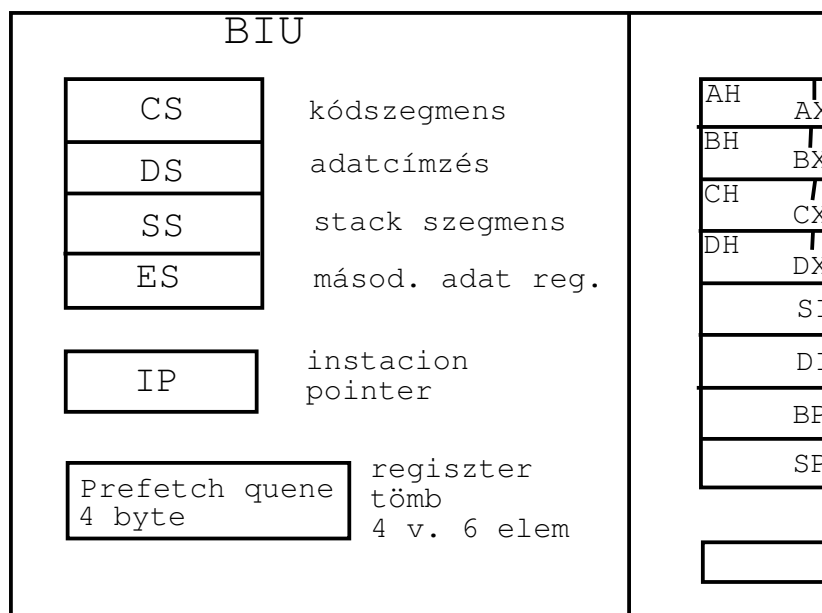
18.4. ábra. union

```
};
```

```
typedef WORDREGS
{
    unsigned ax,bx,cx,dx,si,di,sp,bp;
};
```

4 szegmens plusz az instaction pointer. Logikai címből fizikai, SEG : OFFS

```
union REGS
{
```



18.5. ábra. Intel CPU regiszterei

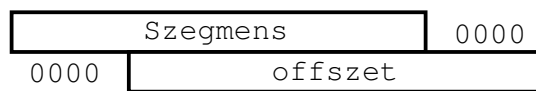
```
WORDREGS x;
BYTEREGS h;
};

union REGS r;

    r.x.ax = 1; /* r utani pont union,
                  x utani pedig struct-ra vonatkozó */

    r.h.ch = 5;
```

Perifériakezelés:



$$P = (16 * S + O) \bmod 2^{20}$$

18.6. ábra. Szegmens és offset

- polling, lekérdezés
- megszakítás (hardver)
- DMA (Direct Memori Access)

18.10.7. A printf() és scanf() függvények

```
int printf(const char *format, ...);
```

*format formátumstring

- formátum specifikátor
- általános szöveget kiírja a standard outputra, azaz a képernyőre.

A formátum specifikátorokat értelmezi és annak megfelelően írja ki az adatokat.

```
printf("szöveg %i szöveg %i szöveg %f",i,i,f);
```

Formátum specifikátor

%-al kezdődik

%[flags][width][.prec][FhLl]type

type.

type	i, d	- int
	u	- unsigned
	x, X	- unsigned hexadecimális
	o	- unsigned oktális
	f	- float 123.456
	e, E	- float 1.23456e2 ~ 1.23456.10 ²
	g, G	- rövidebb -"-
	c	- char
	s	- string
	p	- pointer

	F	- far, távoli pointer (segmens.ofszet)
	N	- near, közeli pointer (ofszet)
[FhLl].	h	- short, rövid integer
	l	- duplapontosság
	L	- longdouble

[width]. width: megadja a minimális kiírás méretét, jobbra rendez.

[flags].

- + számjegyek esetén, pozitív előjelnél tegyük ki a + jelet
- ne jobbra hanem balra rendezzen
- # ha oktális akkor írja ki az 1db nullát a szám elé hexadecimális szám esetén pedig 0x –et.

[.prec].

- tizedes jegyek száma lebegőpontos számnál
- egész szám esetén a minimálisan kiírandó tizedesek számok
- string maximális kiírható karakterek száma

scanf() függvény

A standard inputról olvas.

```
int scanf(const char *format, ...);
```

Argumentum listán adja vissza a beolvasott paramétereket. nem a változót hanem a változó címét adjuk meg.

```
scanf("%i", &i);
```

%[flags][width][FhLl]type

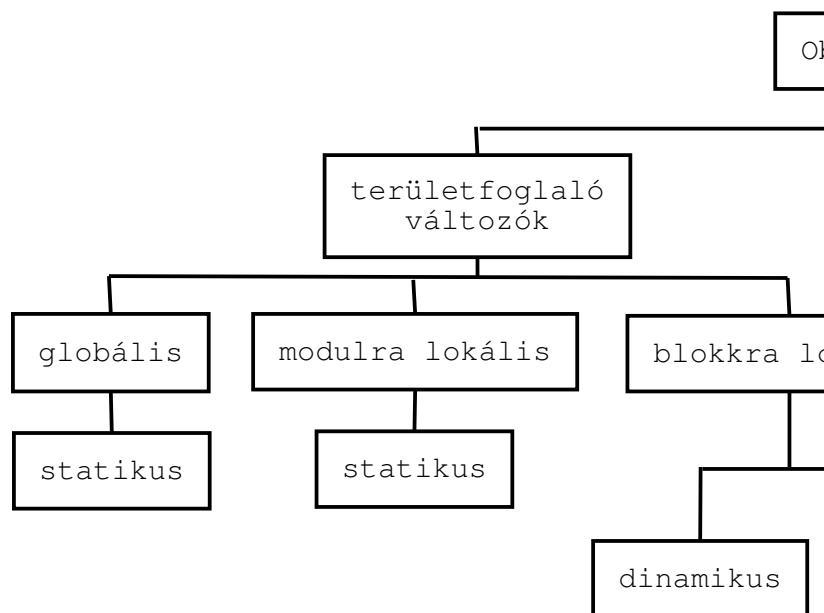
type.	type	d, D, i, I	– int
		x, X	– hexadecimális
		u	– unsigned
		o	– unsigned oktális
		c	– char
		s	– string
		p	– pointer
		f, e, E, g, G	– lebegopontos

```
scanf("%i %i %i", &i, &j, &k);
```

```
scanf("%s", nav); /* szokoz megszakitja a beolvasast! */
```

[flags]. – ne legyen benne * !

18.10.8. Objektumok

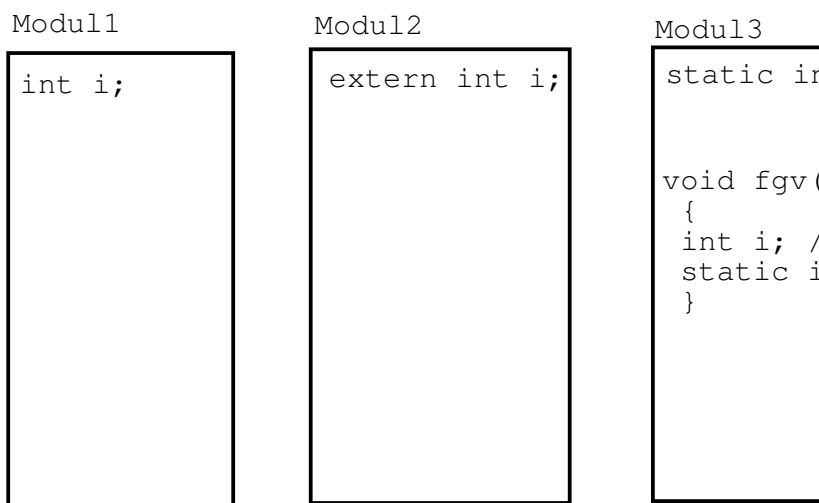


18.7. ábra. Objektumok

- modul az a forráskódkészlet ami önállóan fordul le. (header nem)
- globális az az objektum amely minden modulból látható lehet.
- modulra lokális az az objektum amely a modul össze függvényéből látható lehet.
- blokkra lokális az a változó amely csak az adott függvényben érvényes.

- statikus az a változó amely fordítási időben jön létre.
- dinamikus az a változó amely futási időben jön létre.

A statikus változók inicializálása, kezdőérték adás egyszer érvényes.



18.8. ábra. Modulok

A moduláris programozás lényege, hogy a programunkat több kisebb részre szedjük szét. Így a program átláthatóbb és könnyebben fejleszthetővé válik. A fejlesztés során egy csoport a különálló részeket egyszerűbben módosíthatja. A csoport tagjai a különálló részeket külön-külön tesztelhetik, fejleszthetik.

Példa a moduláris programozásra, 1. modul:

```
1  #include <stdio.h>
2
3  char glbtxt[]="G text\n";
4
5  void fgv1(void);
6  void fgv2(void);
7
8  void main(void)
9  {
10     fgv1();
11     fgv2();
12 }
13
14 void fgv1(void)
15 {
16     printf("modul_1.c\n%s", glbtxt);
17 };
```

2. modul:

```
1  #include <stdio.h>
2
3  extern char glbtxt[];
4
5  void fgv1(void);
6  void fgv2(void);
7
8  void fgv2(void)
9  {
10     printf("modul_2.c\n%s", glbtxt);
11     fgv3();
12 }
```

3. modul:

```
1  #include <stdio.h>
2
3  static char glbtxt[]="ML text\n";
4
5  static void fgv2(void);
6
7  void fgv3(void);
8
9  void fgv3(void)
10 {
11     printf("modul_3.c\n%s", glbtxt);
12     fgv2();
13 }
14 void fgv2(void)
15 {
16     char glbtxt[]="BL text\n";
17     printf("modul_3.c\n%s", glbtxt);
18 }
```

18.11. Fájlkézelés

Fájl

- A fájl UNIX/Linux alatt olyan mint egy karaktertömb!
- Tartalma lehet bináris vagy karakter, text típusú.

Fájlpozícionáló mutató

- Megmutatja, hogy a következő „akció” hol történik meg.
- Automatikusan növekszik (karakter, byte)

A magasszintű fájlkezelés formázott fájlkezelést takar.

```
* FILE *fp;
```

Szöveges állomány esetén kiíratáskor a soremelést és a kocsivisszát lenyeli!

18.11.1. Magasszintű fájlkezelés

A magasszintű fájlkezelést a `<stdio.h>` könyvtár teszi lehetővé. Struktúraként van definiálva a FILE környezet: `FILE *fp;`

– Megnyitás:

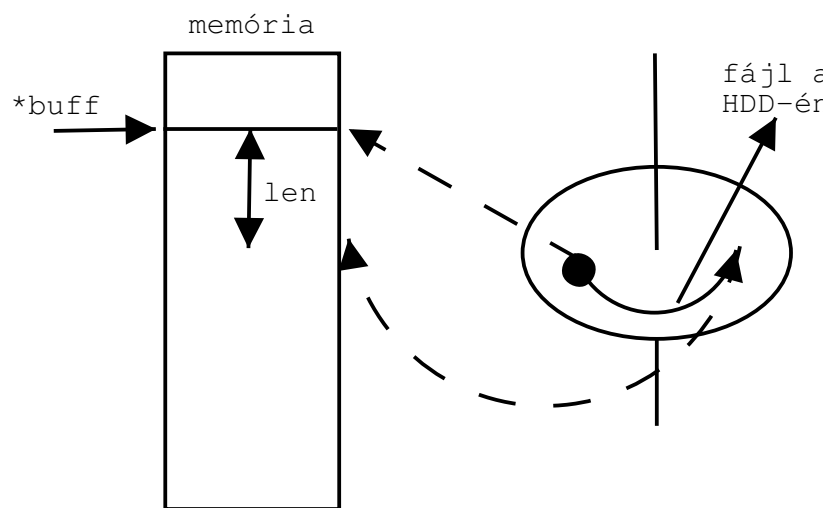
```
FILE *fopen(char *path, char *mode);
```

1. `FILE` a környezet
2. `*fopen` ez egy mutató
3. `char *path` az elérési útvonal
4. `char *mode` a megnyitás módja:
 - * `r` olvasás
 - * `w` írás, de ha létezik nullázza a fájlt, valamint létre is hozza ha még nem létezett ilyen nevű
 - * `a` append, hozzáfűzés
 - * `r+` írásra és olvasásra
 - * `w+` létrehozásra, írásra, olvasásra, nulláz ha már létezett
 - * `a+` hozzáfűzés, olvasás
 - * `t` text mód `rt` megnyitás olvasásra text módban
 - * `b` bináris `rb` megnyitás olvasásra binárisan

Sikertelen megnyitás esetén `NULL` pointer a visszatérési értéke, ha pedig sikeres akkor pedig a mutató

– Lezárás:

```
int fclose(FILE *fp);
```



18.9. ábra. *buff, fájl a memóriában

Ha sikeres akkor nulla, ha sikertelen `-1` lesz a visszatérési érték.

- Olvasás a fájlból karakterenként:

```
int getc(FILE *fp);
```

`int` típusú a visszatérési értéke, mert transzparens (átlátszó) az át-
vitele

- Írás fájlba karakterenként:

```
int putc(int chr, FILE *fp);
```

-1 -el tér vissza hiba esetén, siker esetén pedig maga a karakter a visszatérési érték

Példa a magasszintű fájlkezelésre, csak olvashatóan megnyitjuk /etc/passwd-ot és kiíratjuk tartalmát:

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      FILE *fp; /* deklaralom a pointert ami a fajlra fog mutatni */
6      int ci;   /* ci-be fog agetc() karaktereket pakolni */
7      fp = fopen("/etc/passwd","r"); /* nyitas */
8      while(1) /* jo kis vegtelen ciklus */
9      {
10         /* kezdete */
11         ci = getc(fp); /* ci-be minden ciklas utan */
12         /* a kovetkezo karakter kerul */
13         if(ci == EOF) break; /* ha ci erteke -1 azaz EOF, */
14         /* fajlvege lenne kilep */
15         printf("%c",ci); /* ciklasonkent kiiradjuk a karaktereket */
16     } /* cikluszaro kapocs */
17     fclose(fp); /* fajlzaras */
18 }

```

Egy másik példa, ami kicsit tömörebb, egyben olvashatatlanabb is:

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      FILE *fp;
6      int ci;
7      for(fp=fopen("/etc/passwd","r"); /* ha sikeres a megnyitas */
8          /* akkor nem nulla, igaz */
9          (EOF!=(ci=getc(fp)))?printf("%c",ci):fclose(fp);
10         /* nem fajlvege? akkor kiiratom, ha igen akkor lezarom */
11         );
12 }

```

– Olvasás fájlból:

```
int fread(void *buffer, size_t size, size_t n, FILE *fp);
```

1. size_t size mekkora egy elem mérete
2. size_t n hány legyen az előzőből

– Írás fájlba:

```
int fwrite(void *buff, size_t size, size_t n, FILE *fp);
```

– Fájlpozíciómutató helyét visszaadó függvény:

```
long ftell(FILE *fp);
```

– A fájlpozíciómutató mozgatása:

```
long fseek(FILE *fp, long offset, int whence);
```

- Fájlhender kikeresése, hogy megtudjuk állapítani egy másik függvénnyel a fájl méretet:

```
fileno(FILE *fp);
```

Számoljuk ki a fájl méretet:

```
l = filelength(fileno(fp));
```

- Írjunk fájlba, formázottan:

```
int fprintf(FILE *fp, const char *format, ...);
```

Minden pontban megegyezik használata a `printf()` függvénnyel.

Nézzünk egy példát ami a nyomtatóra fog írni:

```
fprintf(stdprn, " Helló\n");  
/* stdprn DOS nyomtató, lpt1-en */
```

18.12. NCURSES

Ez a rész az angol NCURSES-Programing-HOWTO alapján készült: <http://www.linuxdoc.org/HOWTO/NCURSES-Programing-HOWTO>. Ugyan itt található az a programgyűjtemény is ami alapján az alábbi példaprogramokat átírtuk.

18.12.1. Mi az az ncurses?

Az ncurses lehetővé teszi számunkra, hogy konzolon, text módban is tudjunk ablakokat, menüket, stb. létrehozni. Tehát tudunk segítségével jó kinézetű felhasználói felületet létrehozni.

18.12.2. Hogy fordítsuk le ncurses programunkat?

```
covek@linux:~/c/programok$ gcc hello_world.c -lncurses
```

18.12.3. Lássuk az első példát

```
1  /* #include <stdio.h>  standard könyvtár beépítet ncurses.h  */  
2  * megteszi helyettünk.  */  
3  #include <ncurses.h>          /* ncurses könyvtár beépítése  */  
4  
5  int main()                    /* fő függvényünk  */  
6  {                             /* kezdet  */  
7      initscr();                /* curses mod kezdete  */  
8      printw("Hello Világ!!!"); /* Hello Világ kiírása  */  
9      refresh();                /* Kiírás a valódi képernyőre  */
```



```
10     getch();                /* Billentyű leutesre varunk */
11     endwin();               /* curses modból kilepünk */
12     return 0;               /* visszateresi érték nulla */
13 }
```

Példaprogramunk kiírja a „Helló Világ!!!” feliratot majd egy billentyűzet leütése után kilép. Programunk bemutatja, hogy kell inicializálni, manipulálni majd lezárni a curses módot. Nézzük lépésről lépésre, soronként.

initscr()

Inicializálja a terminált curses módba. Néhány megvalósításban a képernyőt is törli. Ez a funkció inicializálja a curses rendszert és lefoglalja a szükséges memóriaterületet ablakunk számára és más adatstruktúráknak.

Az a misztikus refresh()

A következő sorban kiíratjuk a `printw` segítségével a „Helló Világ!!!” stringet a képernyőre. Ez a függvény a használatát tekintve megegyezik a normál `printf` használatával. A kiíratás a standard képernyőre történik az alap kordinátáktól, azaz 0, 0-tól, a bal felső sarokba.

Amikor a standard képernyőre írunk a `printw`-vel valójában az még csak egy előkészített kép lesz a bufferben, amit a `refresh()` hatására tudunk megjeleníteni.

A kép manipulálása ezzel a háttérben megtörténhet és egy mozdulattal azt kitehetjük a szemünk elé a képernyőre. Kezdők számára ez kicsit furcsának tűnhet és hajlamosak vagyunk a `refresh()` meghívását elfelejteni.

endwin()

Végül nem szabad elfelejtkezni a curses mód lezárásáról a `endwin()` meghívásával. Ezáltal kilépünk a curses módból normál terminál módba. Nem szabad elfelejtkezni rólla!

18.12.4. A kezdetek után ...

Az előbbi példánk megértése után belecsaphatunk a részletekbe. Kezdjük neki.

18.12.5. Inicializálás

A curses rendszer előkészítéséhez inicializálni kell, erre szolgál a `initscr()` függvényhívás. Minden további curses funkció csak ez után vehető használatba. Többek között ezután inicializálható a szín, egérkezelés, stb.

Nézzük milyen függvények szolgálnak inicializálásra ...

raw() és cbreak()

Ezzel a két funkcióval tudjuk megakadályozni a sorok bufferelését. A különbség annyi a kettő között, hogy más a CTRL-Z és a CTRL-C kontrollkarakterek kezelése. `cbreak()` ugyan úgy kezeli ezeket mint minden más karaktert. Talán jobb ha a `raw()` használatát részesítjük előnyben.

echo() és noecho()

Használatukkal a viszhang tiltható, engedélyezhető. Így lehetőség van arra, hogy olyan koordinátára kerüljön az echo-zott `getch()` bekért adat ahova mi akarjuk.

keypad()

Segítségével lehetőség van a funkcióbillentyűk, nyilbillentyűk, stb. lekezelésére. `keypad(stdscr, TRUE)`-val engedélyezhetjük. Erről a lehetőségről a későbbiek során lesz még bővebben szó.

halfdelay()

Nem használjuk gyakran, half-delay mód engedélyezésére szolgál. Kissé hasonlít a `cbreak()`-hoz. 10 másodpercek elteltével hibával (ERR) tér vissza, ha nem áll rendelkezésre bemenő adat. Használatára egy lehetséges példa a jelszóbekérés, amikor egy idő után hibával térhet vissza.

Egyéb inicializáló függvények

Van még pár inicializáló függvény a curses mód előkészítéséhez, de nem igazán használatosak, így most eltekintünk ismertetésüktől.

Egy példa

Lássuk az előbb látottakra második példánkat.

```

1  #include <ncurses.h>
2
3  int main()
4  {   int ch;
5
6      initscr();                /* curses mod kezdete      */
7      raw();                    /* sorbuffer tilása       */
8      keypad(stdscr, TRUE);    /* F1, F2, stb. ...      */
9      noecho();                /* echo tiltasa getch()-nál */
10
11     printf("Nyomd le az F1 funkcióbillentyűt!\n");
12     ch = getch();              /* Ha raw()-ot használunk
13                                * egy enterrel adjuk at
14                                * a programnak          */
15     if(ch == KEY_F(1))         /* keypad() engedelyezes nelkul */
16                                /* ezt nem tudnank hasznalni    */
17     printf("F1 billentyűt lenyomtad.");
18                                /* noecho() nelkul escape
19                                * karakterek latszodnanak a
20                                * kepernyon          */
21     else
22     {   printf("Roszz billantyűt nyomtál le :))");
23         attron(A_BOLD);
24         printf("%c", ch);
25         attroff(A_BOLD);
26     }
27     refresh();                /* Kiiratjuk a valos kepernyore */
28     getch();                  /* Varunk egy billentyure      */
29     endwin();                  /* curses modbol kilepunk      */
30     return 0;
31 }
```

A program magától érthetődő. Bár használtunk olyan függvényt is amiről eddig még nem volt szó. `getch()` bekér egy karaktert a felhasználótól. Ez a függvény úgy működik mint a `getchar()` azzal a különbséggel, hogy nem kell ENTER-t leütni a bevitelhez. Az attribútumok ki és bekapcsolását az `attroff` és `attron` függvényekkel tudjuk megoldani. A példában ennek használatát láthatjuk. A `getch()`-ról a billentyűzetkezelő részben lesz szó bővebben.

18.12.6. Néhány szó az ablakokról

Mielőtt beleugranánk a többi ncurses funkció ismertetésébe tisztázzunk néhány dolgot az ablakokkal kapcsolatosan.

Az ncurses segítségével egy ablakot tudunk definiálni egy képzeletbeli képernyőre. Az így definiált ablak nincs keretezve. A curses segítségével inicializált alapértelmezett (`stdscr`) ablak 80×25^3 méretű.

³illetve a mérete függ az éppen aktuális futtatási környezettől is!

Például, ha ilyen hívást használsz

```
printw("Szervusz barátom!");
refresh();
```

Ez kiírja a stringet a `stdscr`-re az aktuális kurzorpozícióra, a `refresh()` segítségével pedig kiíratható.

Ha egy `w`-t adunk az előbbi funkciókhoz

```
wprintw(win, "Szervusz barátom!");
wrefresh(win);
```

További három funkcióhoz juthatunk.

```
printw(string);           /* kiír az aktualis kurzorpozicióra */
mvprintw(y, x, string);  /* mozgatjuk y,x koordinatara es      *
                          * kiiratjuk                          */
wprintw(win, string);    /* kiiratas az ablakon beluli      *
                          * kurzorpozicióra                  */
mvwprintw(win, y, x, string); /* y,x pontra mozgatunk az ablakban *
                              * majd kiiratunk                      */
```

Általában a `w` nélküli funkciók makrózhatók és kiterjeszthetők `w` funkciókkal.

18.12.7. printw()–hez kapcsolódó függvények

1. `addch()` class: egyszerű karakterek kiírása attribútumokkal
2. `printw()` class: formázott kiírás hasonlóan mint `printf()`
3. `addstr()` class: stringek kiírása

Ezeket a funkciókat használva látványosabbá teheted programodat.

addch() class

Ez a funkció elhelyez egy karaktert az aktuális kurzor pozícióba. Attribútumokat adhatunk a kiíratáshoz: kövér, inverz stb., részletesebben később nézzük át a lehetőségeket.

Kombinálni is lehet az attribútumok használatát:

- `addch(ch | A_BOLD | AUDERLINE);` kövér és aláhúzott. Ezek az `ncurses.h` header fájlban vannak definiálva.

- `attrset()`, `attron()`, `attroff()`. Segítségükkel attribútumokat lehet beállítani, ki illetve bekapcsolni.

Ezekon kívül van még néhány speciális karakter is a karakteres rajzoláshoz. Így tudunk rajzolni táblázatokat, függőleges, vízszintes vonalakat, stb. `ncurses.h`-ban megnézhetjük a lehetőségeinket, keressünk rá a `ACS_` kezdetű makrókra.

`mvaddch()`, `waddch()` és `mvwaddch()`

`mvaddch()` használatával mozgatjuk a kurzorpozíciót és kiíratunk:

```
move(row,col); /* kurzor mozgatasa a megadott sorra, oszlopra */
addch(ch);
```

használható a fenti helyett a `mvaddch(row,col,ch)`;

`waddch()` hasonló az `addch()`-hoz. Nos, ismerős az alap funkciót nyújtó `addch()`, de ha egy stringet akarunk kiírni, bosszantó azt karakterről karakterre megtenni. Szerencsére az `ncurses` nyújt `printf` vagy `puts` szerű funkciókat.

`printw()` class

Ez a funkció hasonlóan használható mint a `printf()`, hozzáadva a kurzorpozícionálási lehetőségeket.

`printw()` és `mvprintw()`. `printf()` működéséhez hasonlóan történik a kiírás, de kurzorpozíciót is mozgathatunk. `mvprintw()` segítségével egyszerre mozgathatunk és írhatunk. Ha először pozicionálni akarunk akkor pedig használhatjuk mozgásra a `move()` kiírásra pedig a `printw()` függvényeket.

`wprintw()` és `mvwprintw()`. Ez a két függvény az argumentumok használatának lehetőségével többet nyújt számunkra.

`vwprintw()`. Ez pedig a `vprintf()` függvénnyel hasonlatos, argumentumok variálhatóságát nyújtva.

Nézzünk egy egyszerű példát, `printw()`.

```

1  #include <ncurses.h>           /* ncurses.h beépíti a stdio.h-t is */
2  #include <string.h>
3
4  int main()
5  {
6      char msg[]="Középen a sting!"; /* uzenet ami megjelenik a kepernyon */
7      int row,col;                  /* sor es oszlop valtozok deklaralasa */
8      initscr();                   /* curses mod inditasa */
9      getmaxyx(stdscr,row,col);    /* lekerdezzuk a maximalis szelesseget */
10     /* es magassagot */
11     mvprintw(row/2,(col-strlen(msg))/2,"%s",msg);
12     /* kiiratjuk a kepernyo kozepere */
13     mvprintw(row-2,0,"Ez a terminál %d sor és %d oszlop\n",row,col);
14     printf("Próbáld átméretezni az ablakot (ha lehetséges) és futtasd újra a programot!");
15     refresh();
16     getch();
17     endwin();
18
19     return 0;
20 }

```

A program azt szemlélteti, hogy milyen könnyen használható a `printw`. Csak megadjuk a koordinátákat a megjelenítendő szöveghez és kész is.

Itt találkozhattunk egy új, `ncurses.h` header-ben definiált makróval a `getmaxyx()`-al. Segítségével lekérdezhető az aktuális ablakméret, két integer típusú változóba kerülnek az értékek.

addstr() class funkció

`addstr()` segítségével egy karaktert tudunk kitenni az adott ablakba. Ez a függvény az `addch()`-val megegyezően működik. Van `w` változata is `waddstr()`, valamint `mvaddstr()`, és `mvwaddstr()` (`mvaddstr()` egyesíti a `move()` és a `addstr()` funkciókat). Másik hasonló függvény a `addnstr()`, amely egy integer paramétert (`n`) ad. Ez a függvény `n` számú karaktert helyez el a képernyőn. Ha értéke negatív teljes string lesz hozzáadva.

Fontos emlékeztető

Minden esetben a függőleges koordináta adat az első, majd azt követi a vízszintes. Kezdeknek okozhat kellemetlen meglepetést, ha ezt elfelejtik. Emlékezzünk erre!

18.12.8. Adat bekérés, scanw()

Az adatokat be is kell kérni, erre a feladatra is van több lehetőségünk.

1. `getch()` class: bekér egy karaktert
2. `scanw()` class: formázott beolvasás
3. `getstr()` class: String beolvasása

getch()

Egy karakter beolvasására használható függvény. De nem használható a `cbreak()`, `curses` módban nem olvas a bemenetről folyamatosan karaktert csak egy új sor karakter, illetve EOF esetén. Így ajánlatos kerülni használatát, hacsak nem kell. Másik sokat használt függvény a `noecho()`. Hatására a bevitt karakter nem látszik a képernyőn. Ez a két függvény tipikus példa a billentyűzetkezelésre, amiről később fogunk bővebben olvasni.

scanw() osztályú függvények

`scanf()`–nél megszokott módon használható, hozzáadva azt, hogy bárhol tudunk beolvasni az ablakban.

scanw() és mvscanw. Használata az `sscanf()` függvényével megegyező, ahol a `wgetstr()`⁴-vel egy sor olvasható be.

wscanw() és mvwscanw(). Olvasás az ablakból, argumentumokkal kiegészítve.

vwscanw(). Megegyezik a `vscanf()` használatával, variálható argumentum használatával.

getstr() osztályú függvények

String beolvasására használhatóak. Sorban beolvassa a karaktereket addig még új sor, sosemelés vagy fájlvége karakterek valamelyikét nem kapja. Az `str` egy karakter típusú tömbre mutató pointer amit a felhasználónak kell létrehoznia.

⁴lentebb lesz még róla szó

Példa

```

1  #include <ncurses.h>                /* ncurses.h beépíti a stdio.h-t is */
2  #include <string.h>
3
4  int main()
5  {
6      char msg[]="Várok egy stringre: "; /* uzenet ami megjelenik a kepernyon */
7      char str[80];
8      int row,col;                      /* sor es oszlop valtozok deklaralasa */
9      initscr();                        /* curses mod inditasa */
10     getmaxyx(stdscr,row,col);          /* lekerdezzuk a maximalis szelesseget */
11                                         /* es magassagot */
12     mvprintw(row/2,(col-strlen(msg))/2,"%s",msg);
13                                         /* kiiratjuk a kepernyo kozepere */
14     getstr(str);
15     mvprintw(LINES - 2, 0, "Ezt írtad: %s", str);
16     getch();
17     endwin();
18
19     return 0;
20 }

```

18.12.9. Attribútumok

Nézzük egy példán keresztül milyen speciális effektekhez juthatunk általuk. A bemenetre adott fájlt megjeleníti úgy, hogy a kommenteket kövéren szedi.

```

1  #include <ncurses.h>
2
3  int main(int argc, char *argv[])
4  {
5      int ch, prev;
6      FILE *fp;
7      int goto_prev = FALSE, y, x;
8
9      if(argc != 2)                      /* volt ket filenav? ha nem */
10     { printf("Használat: %s <egy c fájl>\n", argv[0]);
11       exit(1);
12     }
13     fp = fopen(argv[1], "r");            /* letezik a benemeti file? */
14     if(fp == NULL)
15     { perror("Hiba! Nem tudtam megnyitni a fájlt!");
16       exit(1);
17     }
18
19     initscr();                          /* curses mod kezdete */
20
21     prev = EOF;
22     while((ch = fgetc(fp)) != EOF)
23     { if(prev == '/' && ch == '*')      /* Ha / es * van csak egymas utan */
24       {                                /* bekapcsol a kover kiiratas */
25         attron(A_BOLD);
26         goto_prev = TRUE;              /* Vissza az elozi / karakterre */
27                                         /* es kiiratni koveren */
28       }
29       if(goto_prev == TRUE)
30       { getyx(stdscr, y, x);

```



```

31         move(y, x - 1);
32         printw("%c%c", '/', ch); /* kiíratni az aktualis karaktert */
33
34         ch = 'a';                  /* 'a' egy karakter a          *
35                                     * megallashoz                */
36                                     // "/" "/" kommentek.
37         goto_prev = FALSE;         /* FALSE-ra allitjuk a visszat */
38     }
39     else
40         printw("%c", ch);          /* ami nincs kommentezve normal */
41                                     * kiíratással megy kepernyore */
42     refresh();
43     if(prev == '*' && ch == '/')
44         attroff(A_BOLD);          /* visszakapcsolas kover modbol */
45                                     * ha * utan egy / kovetkezik */
46     prev = ch;
47 }
48 getch();
49 endwin();                          /* curses modbol kilepes */
50 return 0;
51 }
```

Ne aggódjunk az ismeretlen részek miatt. Koncentráljuk a `while` ciklusra. Ha `/*` karaktereket talál átvált kövér, majd `*/` után normál kiíratásra. Ebhez a `attron()` és `attroff()` függvényeket használjuk.

A program bemutat két további függvényt is, a `getxy()` és a `move()` függvényekét. Az elsővel lekérdezhető a kurzor aktuális pozíciója, a második pedig mozgatja azt.

Ez a program tényleg egy egyszerű program és nem csinál ennél többet. Lehetne adott esetben kibővíteni úgy, hogy más más színnel legyenek a szintaktikai elemek megjelölve.

A részletek

Nézzük, hogy milyen lehetőségeink vannak. A `attron()`, `attroff()`, `attrset()`, és az `attr_get()`, stb. függvények használata során ki/be tudjuk kapcsolni az attribútumkezelést, attribútumot tudunk bekérni illetve színes képernyőt tudunk elővarázsolni.

Ezek a videó attribútumok a `< curses.h >`-ban vannak definiálva.

A_NORMAL	Normál megjelenítés (nincs kiemelés, highlight)
A_STANDOUT	A legjobb kiemelési mód a terminálon.
A_UNDERLINE	Aláhúzás
A_REVERSE	Inverz mód
A_BLINK	Villogtatás
A_DIM	Félfényes mód
A_BOLD	Extra fény vagy kövér
A_PROTECT	Védett mód
A_INVIS	Láthatatlan vagy üres mód
A_ALTCHARSET	Alternatív karakter beállítás
A_CHARTEXT	Bitmaszk kiterjesztése egy karakternek
COLOR_PAIR(n)	Szín pár szám n

Az utolsó a színkezelés, amiről a következő alfejezetben lesz szó.

Lehetőségünk van az attribútumok összekapcsolására az [OR()] használatával. Például ha inverz és villogó módot is szeretnénk egyszerre:

```
attron(A_REVERSE | A_BLINK);
```

attron() vagy attrset()

Mi a különbség a két függvény között? Az `attrset()` az ablakra mint egészre, még az `attron` csak az éppen aktuális részre van hatással. A `standend()` segítségével kikapcsolható minden attribútumkezelés normál módba. A `attrsetr(A_NORMAL)` ugyan úgy használható.

attr_get()

Az aktuálisan használt attribútumokat és színpárokat adja vissza.

attr_függvények

Van még egy sorozat függvény, úgy mint `attr_set()`, `attr_on`, stb.. Ezek hasonlóak a fentiekhez, kivéve az `attr_t` függvényt.

chgat() függvény

Kurzor mozgatása nélkül kapunk attribútum változtatási lehetőséget egy csoport karakterre az aktuális pozíciótól kezdődően.

A -1 kapcsoló segítségével egy egész sor attribútumát változtathatjuk meg a sor végéig segítségével.

```
chgat(-1, A_REVERSE, 0, NULL);
```

Egyéb függvények, `wchgat()`, `mvchgat()`.

```

1  #include <ncurses.h>
2
3  int main(int argc, char *argv[])
4  {
5      initscr();                /* curses modot kezdunk */
6      start_color();           /* szimkezelő funkciók kezdete */
7
8      init_pair(1, COLOR_RED, COLOR_YELLOW);
9      printw("Egy hosszú string amit nem tudtam teljesen begépelni ");
10     mvchgat(0, 0, -1, A_BLINK, 1, NULL);
11     /*
12      * Az első két paraméter a kezdeti pozíciókat adja meg
13      * Harmadik (-1) a sor végéig aktualizál
14      * A negyedik paraméter normal attribútum amit akarunk használni
15      * Otodik szín index az init_pair() sorban megadottak szerint
16      * nulla (0) ha nem akarunk szintkezelni
17      * A hatodik pedig mindig NULL
18      */
19     refresh();
20     getch();
21     endwin();                 /* curses mod vége */
22     return 0;
23 }
```

Ez egy egyszerű példa az attribútumok kezelésére. Ha nem akarunk szín kezelést használnunk 0-t.

18.12.10. Az ablakkezelő függvényekről

Az ablakok kezelése a legfontosabb az ncurses koncepciójában. Láthatjuk a standard ablak (`stdscr`) fölött az összes függvényt működni. Nos készíthetünk egyszerű megjelenésű GUI⁵-t. Az ablakokat tudjuk manipulálni a képernyő külön részenként. De mielőtt nagy programot kezdenénk írni tanulmányozzuk a lehetőségeinket kisebb részenként.

Az alap

Egy ablakot a `newwin()` függvénnyel hozhatunk létre. Ez nem jelenik meg közvetlenül az aktuális képernyőn. A memóriában jön létre az ablak struktúrája, amit tudunk manipulálni, átméretezni, meghatározni tulajdonságait, stb.. Tehát az ncursesben egy ablak csak a háttérben jön létre amit a képernyőtől függetlenül tudunk kezelni. A `wprintw()` stb. segítségével megjeleníthetjük, a `delwin()` segítségével pedig megsemmisíthetjük azt, és felszabadítjuk az előzőleg számára lefoglalt memóriát.

⁵Általános Felhasználói Interfészt


```

54         wrefresh(local_win);                /* megmutatja az ablakot          */
55
56         return local_win;
57     }
58
59 void destroy_win(WINDOW *local_win)
60 {
61     /* box(local_win, ' ', ' '); : Ez a függvény
62      * nem törli a sarkokat, ami ronda nyomot hagy
63      * az ablak uan.
64      */
65     wborder(local_win, ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ');
66     /* A paraméterek felsorolása
67      * 1. win: az ablak amit megdolgozunk
68      * 2. ls: karakter amit a bal oldal rajzolásához használtunk
69      * 3. rs: karakter amit a jobb oldal rajzolásához használtunk
70      * 4. ts: karakter amit a felső oldal rajzolásához használtunk
71      * 5. bs: karakter amit a alsó oldal rajzolásához használtunk
72      * 6. tl: karakter ami a bal felső sarkot rajzolja meg
73      * 7. tr: karakter ami a jobb felső sarkot rajzolja meg
74      * 8. bl: karakter ami a bal alsó sarkot rajzolja meg
75      * 9. br: karakter ami a jobb alsó sarkot rajzolja meg
76      */
77     wrefresh(local_win);
78     delwin(local_win);
79 }

```

Magyarázat

Nem kell megrémülni a példa nagysága miatt, de így volt lehetőség néhány fontos dolgot bemutatni. A program létrehoz egy bekeretezett ablakot, amit a nyílbillentyűkkel lehet fel, le, jobbra, balra mozgatni. A mozgatás úgy jön létre, hogy eltüntetjük és újrarajzoljuk az ablakot. Ne menjünk az ablakon kívülre. Az ellenőrzést elhagytuk a programból. Nézzük sorról sorra.

A `create_newwin()` függvény létrehoz egy ablakot és bekeretezve jeleníti meg azt. A `destroy_win()` pedig törli az ablakot, a keretet egy üres karakter segítségével és a `delwin()` függvény pedig felszabadítja a memóriát is. Attól függően, hogy melyik irányba mozgatjuk az ablakot törli, majd létrehozza azt.

A `destroy_win`-ben láthatjuk, hogy `wborder`-t használtunk. Használatával kapcsolatos információk a forráskódban lévő kommentek-ben található. Segítségével keret rajzolható az ablak köré, négy sarok és négy oldal külön-külön. Egy példa látható alább:

```
wborder(win, ' | ', ' | ', ' - ', ' - ', ' + ', ' + ', ' + ', ' + ' );
```

Ez a sor létrehoz egy ablakot kerettel:

```

+-----+
|       |
|       |
|       |
+-----+

```

Ami még a példában volt

A példában szerepelt még változóként a COLS, LINES amelyek segítségével inicializáltuk a képernyőméretet az `initscr()` után. Könnyedén lekérdezhető a képernyő mérete, amiből kiszámíthatóak a középponti koordináták. A `getch()` függvény mint általában beolvassa a billentyűzetről mint ebben az esetben is. A switch-case szerkezet nagyon gyakori bármelyik GUI alapú programban.

Egyéb keretező függvények

Az előző programban egy rendkívül rosszul működő billentyűzet lenyomásra ablak el-tüntetését és újrarajzolást használtunk. Így írtunk egy másik jobban működő programot ami más módon készíti el a keretezést.

A következő program `mvhline()` és `mvvline()` függvényeket használva valósítja meg az előző effektust. Használatuk egyszerű. Vízszintes vagy függőleges vonalakat rajzolnak meghatározott méretben a megadott pozícióban.

```

1  #include <ncurses.h>
2
3  typedef struct _win_border_struct {
4      chtype  ls, rs, ts, bs,
5              tl, tr, bl, br;
6  }WIN_BORDER;
7
8  typedef struct _WIN_struct {
9
10     int startx, starty;
11     int height, width;
12     WIN_BORDER border;
13 }WIN;
14
15 void init_win_params(WIN *p_win);
16 void print_win_params(WIN *p_win);
17 void create_box(WIN *win, int bool);
18
19 int main(int argc, char *argv[])
20 {
21     WIN win;
22     int ch;
23
24     initscr();                /* curses mod kezdete          */
25     start_color();            /* szinkezeles kezdete        */
26     cbreak();                /* sor buffereles tiltasa     */
27     keypad(stdscr, TRUE);    /* speci billentyuk engedelye */
28     noecho();
29     init_pair(1, COLOR_CYAN, COLOR_BLACK);
30
31     /* inicializaljuk a ablak parametereket */
32     init_win_params(&win);
33     print_win_params(&win);
34
35     attron(COLOR_PAIR(1));
36     printf("F1 lenyomásával tudsz kilépni!");
37     refresh();

```

```

37         attroff(COLOR_PAIR(1));
38
39         create_box(&win, TRUE);
40         while((ch = getch()) != KEY_F(1))
41         {
42             switch(ch)
43             {
44                 case KEY_LEFT:
45                     create_box(&win, FALSE);
46                     --win.startx;
47                     create_box(&win, TRUE);
48                     break;
49                 case KEY_RIGHT:
50                     create_box(&win, FALSE);
51                     ++win.startx;
52                     create_box(&win, TRUE);
53                     break;
54                 case KEY_UP:
55                     create_box(&win, FALSE);
56                     --win.starty;
57                     create_box(&win, TRUE);
58                     break;
59                 case KEY_DOWN:
60                     create_box(&win, FALSE);
61                     ++win.starty;
62                     create_box(&win, TRUE);
63                     break;
64             }
65         }
66         endwin();
67         return 0;
68     }
69     /* curses mod vege */
70 }
71 void init_win_params(WIN *p_win)
72 {
73     p_win->height = 3;
74     p_win->width = 10;
75     p_win->starty = (LINES - p_win->height)/2;
76     p_win->startx = (COLS - p_win->width)/2;
77
78     p_win->border.ls = '|';
79     p_win->border.rs = '|';
80     p_win->border.ts = '-';
81     p_win->border.bs = '-';
82     p_win->border.tl = '+';
83     p_win->border.tr = '+';
84     p_win->border.bl = '+';
85     p_win->border.br = '+';
86 }
87 void print_win_params(WIN *p_win)
88 {
89     #ifdef _DEBUG
90     mvprintw(25, 0, "%d %d %d %d", p_win->startx, p_win->starty,
91             p_win->width, p_win->height);
92     refresh();
93     #endif
94 }
95 void create_box(WIN *p_win, int bool)
96 {
97     int i, j;
98     int x, y, w, h;
99
100     x = p_win->startx;
101     y = p_win->starty;
102     w = p_win->width;

```

```

99     h = p_win->height;
100
101     if(bool == TRUE)
102     {   mvaddch(y, x, p_win->border.tl);
103         mvaddch(y, x + w, p_win->border.tr);
104         mvaddch(y + h, x, p_win->border.bl);
105         mvaddch(y + h, x + w, p_win->border.br);
106         mvhline(y, x + 1, p_win->border.ts, w - 1);
107         mvhline(y + h, x + 1, p_win->border.bs, w - 1);
108         mvvline(y + 1, x, p_win->border.ls, h - 1);
109         mvvline(y + 1, x + w, p_win->border.rs, h - 1);
110     }
111     else
112         for(j = y; j <= y + h; ++j)
113             for(i = x; i <= x + w; ++i)
114                 mvaddch(j, i, ' ');
115
116     refresh();
117 }

```

18.12.11. Színkezelés

Alapok

Az élet színek nélkül nem ér semmit. A curses hatékony színkezeléssel rendelkezik. Kezdjük rögtön egy kis példaprogrammal.

```

1  #include <ncurses.h>
2
3  void print_in_middle(WINDOW *win, int starty, int startx, int width, char *string);
4
5  int main(int argc, char *argv[])
6  {   initscr();                               /* curses mod kezdete          */
7      if(has_colors() == FALSE)
8      {   endwin();
9          printf("A termináloed nem támogatja a színek kezelését!\n");
10         exit(1);
11     }
12     start_color();                             /* szinkezeles kezdete          */
13     init_pair(1, COLOR_RED, COLOR_BLACK);
14
15     attron(COLOR_PAIR(1));
16     print_in_middle(stdscr, LINES / 2, 0, 0, "Hurrá, színesben jobb ...");
17     attroff(COLOR_PAIR(1));
18     getch();
19     endwin();
20 }
21
22 void print_in_middle(WINDOW *win, int starty, int startx, int width, char *string)
23 {   int length, x, y;
24     float temp;
25
26     if(win == NULL)
27         win = stdscr;
28     getyx(win, y, x);
29     if(startx != 0)
30         x = startx;

```



```

31     if(starty != 0)
32         y = starty;
33     if(width == 0)
34         width = 80;
35
36     length = strlen(string);
37     temp = (width - length) / 2;
38     x = startx + (int)temp;
39     mvwprintw(win, y, x, "%s", string);
40     refresh();
41 }

```

Mint láthatjuk a színkezelés előtt meg kell hívni a `start_color()` függvényt. Csak ez után van lehetőség a színek kezelésére. A terminálunk szín kezelési képességét a `has_colors()` függvény kérdezi le. Ha a visszatérési érték hamis (FOLSE) a terminál nem támogatja a színkezelést.

Curses inicializálja az össze színtámogatást a `start_color()` hívásával. Ez lehetővé teszi a hozzáférést az előre definiált konstansokhoz, mint pl.: `COLOR_BLACK`, stb.. Kezddhetjük is a színek kezelését, de párokat kell definiálni. A színek mindig párokban használhatóak. Az előtér és háttérszíneket a `init_pair()` függvénnyel lehet definiálni. Ezután a színpár száma használható normál attributunként a `COLOR_PAIR()` függvényben. Először úgy látszik, hogy nehézkes a használata, de ez egy elegáns, könnyű megoldás a színek menedzselésére. Csak bele kell nézni a „dialog” forráskódjába ami shell scripteket jelenít meg ablakban. A fejlesztőknek kell definiálni a előtér és háttér színpár kombinációkat mielőtt azokat használni szeretnék. Ez könnyen elkészíthető majd attributunként hozzáférhetünk a színpárokhoz.

A következő színek előre definiálva vannak a `curses.h`-ban. Ezeket közvetlenül használhatjuk többféle szín függvényben.

<code>COLOR_BLACK</code>	0
<code>COLOR_RED</code>	1
<code>COLOR_GREEN</code>	2
<code>COLOR_YELLOW</code>	3
<code>COLOR_BLUE</code>	4
<code>COLOR_MAGENTA</code>	5
<code>COLOR_CYAN</code>	6
<code>COLOR_WHITE</code>	7

Szín definíciók megváltoztatása

Az `init_color()` függvény ad erre lehetőséget az rgb színek arányainak megváltoztatásával. Például a vörös szín tehető lágyabbá az alábbiak szerint:

```

init_color(COLOR_RED, 700, 0, 0);
/* paraméter 1      : szín neve
 * paraméter 2, 3, 4 : rgb, min.: 0, max.: 1000 */

```

Ha a terminál ezt nem támogatja a függvény hibával (ERR) tér vissza. A `can_change_color()` függvénnyel leellenőrizhető ennek a lehetősége. Az rgb szín skálája 0-tól 1000-ig tart. A vörös szín 1000(r), 0(g), 0(b)-vel van definiálva.

Színbeállítás

A `color_content()` és `pair_content()` függvények használatával megállapítható a színbeállítás és a háttér, előtér kombináció színe.

18.12.12. Hogy tudom lekérdezni a funkcióbillentyűket, nyíl billentyűket, stb..

Alapok

Felhasználói interfész (GUI) nélkül nincs megfelelő felhasználói felület, interaktivitás. Egy curses program lehetővé teszi a megfelelő billentyűzet és egérkezelést. Először nézzük a billentyűzetet.

Az eddigi ncurses példákból látszik, hogy milyen egyszerű beolvasni felhasználói billentyű leütést. Erre szolgál a `getch()` függvény. A `cbreak` módban egész sort, keypad engedélyezi a funkció és nyíl billentyűk, stb. lekérdezését. Az inicializálást láthatjuk a példákból.

`getch()` visszatérési értéke egy integer, melynek értéke normál karakter esetén a karakterkódot tartalmazza. Más lehetséges visszatérési érték a `curses.h`-ban van definiálva. Például ha F1-et nyomunk le a visszatérési érték 265 lesz. Ezt a `KEY_F()` makró írja le a `curses.h`-ban. Ez egyszerű és hordozható billentyűkezelést tesz lehetővé.

Például ha a `getch()`-t használjuk:

```
int ch;
```

```
ch = getch();
```

`getch()` időkorlátozás nélkül vár arra, hogy lenyomjunk egy billentyűt, amit integerben ad vissza. Ami ezután ellenőrizhető, hogy definiálva van-e a `curses.h`-ban, majd ha akarunk ismételten jöhet a billentyű lekérdezés, már ha akarjuk.

Az alábbi kódrészlet mutatja a működését:

```
if(ch == KEY_LEFT)
    printf("Bal nyilat nyomtad le.\n");
```

Nézzük a következő példát, egy menüben fel le nyilak segítségével navigálhatunk.

Egy egyszerű billentyűhasználat példája

```

1  #include <ncurses.h>
2
3  #define WIDTH 30
4  #define HEIGHT 10
5
6  int startx = 0;
7  int starty = 0;
8
9  char *choices[] = {
10     "Lehetőség 1",
11     "Lehetőség 2",
12     "Lehetőség 3",
13     "Lehetőség 4",
14     "Kilépés",
15     };
16  int n_choices = sizeof(choices) / sizeof(char *);
17  void print_menu(WINDOW *menu_win, int highlight);
18
19  int main()
20  {
21      WINDOW *menu_win;
22      int highlight = 1;
23      int choice = 0;
24      int c;
25
26      initscr();
27      clear();
28      noecho();
29      cbreak();
30      startx = (80 - WIDTH) / 2;
31      starty = (24 - HEIGHT) / 2;
32
33      menu_win = newwin(HEIGHT, WIDTH, starty, startx);
34      keypad(menu_win, TRUE);
35      mvprintw(0, 0, "Választás fel és le nyíl, Enter kiválaszt.");
36      refresh();
37      print_menu(menu_win, highlight);
38      while(1)
39      {
40          c = wgetch(menu_win);
41          switch(c)
42          {
43              case KEY_UP:
44                  if(highlight == 1)
45                      highlight = n_choices;
46                  else
47                      --highlight;
48                  break;
49              case KEY_DOWN:
50                  if(highlight == n_choices)
51                      highlight = 1;
52                  else
53                      ++highlight;
54                  break;
55              case 10:
56                  choice = highlight;
57                  break;
58              default:
59                  mvprintw(24, 0, "Bevitt karakter = %3d,
60                      remélhetőleg ezt tudom kiírni
61                      min '%c'", c, c);
62                  refresh();

```

```

60             break;
61         }
62         print_menu(menu_win, highlight);
63         if(choice != 0)
64             /* A felhasználó választása után végtelen ciklusból kilep */
65             break;
66     }
67     mvprintw(23, 0, "A %d választottad a %s -val\n",
68             choice, choices[choice - 1]);
69     getch(); /* az eredeti programból ez a sor hiányzott! */
70     clrtoeol();
71     refresh();
72     endwin();
73     return 0;
74 }
75
76
77 void print_menu(WINDOW *menu_win, int highlight)
78 {
79     int x, y, i;
80
81     x = 2;
82     y = 2;
83     box(menu_win, 0, 0);
84     for(i = 0; i < n_choices; ++i)
85     {
86         if(highlight == i + 1) /* Legyen látható a választás */
87         {
88             watttron(menu_win, A_REVERSE);
89             mvwprintw(menu_win, y, x, "%s", choices[i]);
90             wattroff(menu_win, A_REVERSE);
91         }
92         else
93             mvwprintw(menu_win, y, x, "%s", choices[i]);
94         ++y;
95     }
96     wrefresh(menu_win);
97 }

```

18.12.13. Egér kezelő interfész

Előzőleg láthattuk a bilentyűkezelést, nézzük mit tudunk kezdeni az egerünkkel. Általában A GUI lehetővé teszi az eger és billentyűzetkezelést is.

Az egérkezelés alapjai

Mielőtt bármit is csinálnánk, az egérkezelést lehetővé kell tenni a `mousemask()` függvénnyel.

```

mousemask( mmask_t newmask, /* a funkció amit hasznalunk */
           mmask_t *oldmask) /* előzőleg használt funkció */

```

Az első paraméter a következő függvényben egy bitmaszk a figyelni kívánt egérfunkciókhoz. Alapértelmezésként minden egérfunkció ki van kapcsolva. Minden egérfunkciót az `ALL_MOUSE_EVENTS` maszk segítségével tudunk aktiválni.

Az alábbi felsorolás tartalmazza az összes használható bitmaszkot:

Név	Leírás
BUTTON1_PRESSED	egér 1 gombja lenyomva
BUTTON1_RELEASED	egér 1 gombja felengedve
BUTTON1_CLICKED	egér 1 gombjával klikkelés
BUTTON1_DOUBLE_CLICKED	egér 1 gombjával dupla klikkelés
BUTTON1_TRIPLE_CLICKED	egér 1 gombjával hármas klikkelés
BUTTON2_PRESSED	egér 2 gombja lenyomva
BUTTON2_RELEASED	egér 2 gombja felengedve
BUTTON2_CLICKED	egér 2 gombjával klikkelés
BUTTON2_DOUBLE_CLICKED	egér 2 gombjával dupla klikkelés
BUTTON2_TRIPLE_CLICKED	egér 2 gombjával hármas klikkelés
BUTTON3_PRESSED	egér 3 gombja lenyomva
BUTTON3_RELEASED	egér 3 gombja felengedve
BUTTON3_CLICKED	egér 3 gombjával klikkelés
BUTTON3_DOUBLE_CLICKED	egér 3 gombjával dupla klikkelés
BUTTON3_TRIPLE_CLICKED	egér 3 gombjával hármas klikkelés
BUTTON4_PRESSED	egér 4 gombja lenyomva
BUTTON4_RELEASED	egér 4 gombja felengedve
BUTTON4_CLICKED	egér 4 gombjával klikkelés
BUTTON4_DOUBLE_CLICKED	egér 4 gombjával dupla klikkelés
BUTTON4_TRIPLE_CLICKED	egér 4 gombjával hármas klikkelés
BUTTON_SHIFT	SHIFT le volt nyomva, funkció váltás
BUTTON_CTRL	CTRL le volt nyomva, funkció váltás
BUTTON_ALT	ALT le volt nyomva, funkció váltás
ALL_MOUSE_EVENTS	minden egérfunkció figyelése
REPORT_MOUSE_POSITION	egérkurzor mozgásának figyelése

Egérfunkció bekérése

Ha az egérfunkciók figyelése már engedélyezve lett, a `getch` osztályú függvény a `KEY_MOUSE` visszaadja az egér adatait. Majd az egérfunkciókat a `getmouse()` függvénnyel lekérdezhetjük.

Az alábbi kód illusztrálja a működést.

```
MEVENT event;

ch = getch();
if(ch == KEY_MOUSE)
    if(getmouse(&event) == OK)
```

```

    . /* amit az egerrel csinalunk */
    .
    .

```

getmouse() visszaadja az egéreseményt egy pointer-ben. A struktúra ami tartalmazza.

```

typedef struct
{
    short id;          /* ID megkülönböztetni a tobbfunkcios eszkozokat */
    int x, y, z;       /* egér koordináták */
    mmask_t bstate;    /* nyomogomb status bitek */
}

```

A bstate a számunkra legérdekesebb, megmutatja az egérgombok állapotát.

Majd az alábbi kódtörredéssel mi lekérdezhethetjük az egér állapotváltozását.

```

if(event.bstate & BUTTON1_PRESSED)
    printf("Bal egérgomb lenyomva");

```

Tegyük az előbbieket össze

Készítsünk néhány menüpontot és engedélyezzük az egérkezelést. Elkészíteni ezt egyszerű, billentyű kezelést eltávolítottuk.

```

1  #include <ncurses.h>
2
3  #define WIDTH 30
4  #define HEIGHT 10
5
6  int startx = 0;
7  int starty = 0;
8
9  char *choices[] = { "Lehetőség 1",
10                     "Lehetőség 2",
11                     "Lehetőség 3",
12                     "Lehetőség 4",
13                     "Kilépés",
14                     };
15
16  int n_choices = sizeof(choices) / sizeof(char *);
17
18  void print_menu(WINDOW *menu_win, int highlight);
19  void report_choice(int mouse_x, int mouse_y, int *p_choice);
20
21  int main()
22  {
23      int c, choice = 0;
24      WINDOW *menu_win;
25      MEVENT event;

```

```

26      /* curses modba kapcsolunk */
27      initscr();
28      clear();
29      noecho();
30      cbreak(); // sor buffereles kikapcsolva
31
32      /* Probaljuk kozepre helyezni az ablakot */
33      startx = (80 - WIDTH) / 2;
34      starty = (24 - HEIGHT) / 2;
35
36      attron(A_REVERSE);
37      mvprintw(23, 1, "Kilépéshez klikkelj a kilépés menüpontra. \
38      (Jobban működik virtuális konzolon)");
39      refresh();
40      attroff(A_REVERSE);
41
42      /* Eloszor a menut kell kiiratnunk */
43      menu_win = newwin(HEIGHT, WIDTH, starty, startx);
44      print_menu(menu_win, 1);
45      /* minden egerfunccio lekerdezesenek engedelyezese */
46      mousemask(ALL_MOUSE_EVENTS, NULL);
47
48      while(1)
49      {
50          c = wgetch(menu_win);
51          switch(c)
52          {
53              case KEY_MOUSE:
54                  if(getmouse(&event) == OK)
55                  { /* Amikor a bel egergombbal klikkelt */
56                      if(event.bstate & BUTTON1_PRESSED)
57                      {
58                          report_choice(event.x + 1, event.y + 1, &choice);
59                          if(choice == -1) //Kilepest valasztva
60                              goto end;
61                          mvprintw(22, 1, "Menükezelés módja: %d \
62                          Választott menü \"%10s\"",
63                          refresh();
64                      }
65                      print_menu(menu_win, choice);
66                      break;
67                  }
68              }
69          }
70      }
71
72      end:
73      endwin();
74      return 0;
75  }
76
77  void print_menu(WINDOW *menu_win, int highlight)
78  {
79      int x, y, i;
80
81      x = 2;
82      y = 2;
83      box(menu_win, 0, 0);
84      for(i = 0; i < n_choices; ++i)
85      {
86          if(highlight == i + 1)
87          {
88              watttrn(menu_win, A_REVERSE);
89              mvwprintw(menu_win, y, x, "%s", choices[i]);
90              wattroff(menu_win, A_REVERSE);
91          }
92          else
93              mvwprintw(menu_win, y, x, "%s", choices[i]);
94          ++y;
95      }

```

```

88     }
89     wrefresh(menu_win);
90 }
91
92 /* Viszajelzes a választás szerinti pozíció alapján */
93 void report_choice(int mouse_x, int mouse_y, int *p_choice)
94 {
95     int i, j, choice;
96
97     i = startx + 2;
98     j = starty + 3;
99
100    for(choice = 0; choice < n_choices; ++choice)
101        if(mouse_y == j + choice && mouse_x >= i && \
102           mouse_x <= i + strlen(choices[choice]))
103        {
104            if(choice == n_choices - 1)
105                *p_choice = -1;
106            else
107                *p_choice = choice + 1;
108            break;
109        }
110 }

```

Egyéb függvények

A `mouse_trafo()` és a `wmouse_trafo()` függvények segítségével lehet konvertálni az egér koordinátákat a képernyő relatív koordinátaivá.

A `mouseinterval` függvény beállítja a maximális (ezredmásodpercben) azt hogy klikkelés-kor mekkora idő telhet el a kattintás és az elengedés között. Az alapérték 50 milisesundum.

18.12.14. Képernyő manipuláció

Ebben a részben megismerkedünk néhány függvénnyel amelyek segítségével néhány programot írunk. Ezek fontosak lehetnek játékprogramok írásánál.

`getyx()`

A `getyx()` függvény segítségével a kurzorpozíciók kérdezhetők le.

```

getyx(win, y, x);
/* win: ablakmutató
 * y,x: y és x koordinátákat adja vissza
 */

```

A `getparyx()` függvény segítségével egy fő ablakban lévő almenü ablakának kezdeti koordinátái adhatóak vissza. Segítségével könnyebben kezelhetők az így használt ablakok.

A `getbegyx()` és a `getmaxyx()` függvényekkel a kezdeti és a maximális koordináták kérdezhetőek le.

Képernyő állapotmentés

Előfordulhat, hogy játékot írunk. Ekkor szükség lehet egy képernyőállapot mentésére. A `scr_dump()` függvény valósítja ezt meg és a `scr_restore()` függvény pedig lehetővé teszi a visszaállítást.

Ablak állapotmentés

Egy ablak aktuális állapotát mentést, visszatöltést a `getwin()` és a `putwin()` függvények teszik lehetővé.

A `copywin()` segítségével ablak másolása lehetséges.

18.12.15. Egyéb

Nézzünk néhány egyéb függvényt.

`curs_set()`

Legyen a kurzorunk láthatatlan:

- 0: láthatatlan
- 1: normál
- 2: nagyon látható

Curses mód háttérben

Van amikor a curses módot szeretnénk a háttérbe küldeni, majd ismételten visszatérni ebbe a módba. A `def_prog_mode()` függvénnyel el kell menteni a tty módot, majd hívható az `endwin()` a curses mód elhagyásához. A `reset_prog_mode()` függvény a curses módba való visszatérést teszi lehetővé. Ezt mutatja be az alábbi példa:

```

1  #include <ncurses.h>                /* ncurses könyvtár beépítése */
2
3  int main()                          /* fő függvényünk */
4  {                                  /* kezdet */
5      initscr();                      /* curses mód kezdete */
6      printw("Helló Világ!!!\n");    /* Hello Vileg kiírása */
7      refresh();                     /* Kiírás a valós képernyőre */
8      def_prog_mode();               /* tty mód mentése */
9      endwin();                       /* curses módból kilépünk */
10     getch();

```

[illegible]