Széchenyi István Egyetem

Távközlési Tanszék

# Hálózati operációs rendszerek

(Linux)

Szerzők ABC sorrendben:

Kallai Péter

Kovács Ákos

Lencse Gábor

Molnár Zoltán Vilmos

Sinkó Gergely

2013. 11. 18.

végig átnézve


# Tartalomjegyzék

# 1 Bevezetés

Ez a jegyzet a Széchenyi István Egyetem Távközlési Tanszékének *infokommunikáció* szakirányain oktatott *Hálózati operációs rendszerek* tantárgy első félévéhez tartozik. Felhívjuk a figyelmet, hogy a jegyzet magában nem elegendő, a tárgyhoz tartozik még minden olyan segédanyag, ami a tárgy oldalára a félév során felkerül. A tantárgy anyagának elsajátításához és a tárgykövetelmények teljesítéséhez erősen ajánlott a gyakorlatokon és az előadásokon való részvétel!

A tárgy anyagába e jegyzeten kívül még a más „Unix rendszerek témakör” tartozik, ezekről külön segédanyagok szólnak.

Jelölések:

Az apró betűvel írt részek érdekességek, a tárgy hallgatóinak olvasásra javasoljuk őket, mivel a témakörben való általános műveltséghez tartoznak, de nem kell őket megtanulni.

*A dőlt betűvel írt feladatokat érdemes megpróbálni önállóan megoldani. Ha nem sikerül, nem baj, de a jegyzet második olvasásakor akkor is kíséreljék meg újra! (A közölt megoldások nem feltétlenül jobbak más megoldásnál, nem bemagolásra szánjuk őket!)*

## *1.1 A UNIX története*

A UNIX első változatát 1969-ben készítette el Ken Thompson és Dennis M. Ritchie az AT&T Bell Laboratóriumában egy DEC PDP-7 típusú számítógépre. 1973-ban a UNIX rendszermagját átírták C nyelvre, és ingyenesen hozzáférhetővé tették az egyetemek számára. A 80-as évek elején már százezernél is több számítógépen futott UNIX. A gondot az jelentette, hogy az egységesség ellenőrzése hiányában mindenhol átszerkesztették, így sok változat alakult ki. Ezekből két jelentősebb, a Berkeley egyetemen fejlesztett BSD UNIX, illetve az AT&T hivatalos változata a System V (System Five, Relase 4-nél tart SVR4), amit az USL (Unix System Laboratories) fejleszt tovább. A két szabványt próbálták valamelyest egyesíteni, így született meg az IEEE, az ANSI és az ISO együttműködésével a POSIX (Portable Operating System Interface for UNIX) ajánlás. A lényege, hogy bármilyen programot ír a fejlesztő, az a POSIX szabványos UNIX-okon gyakorlatilag változtatás nélkül futtatható.

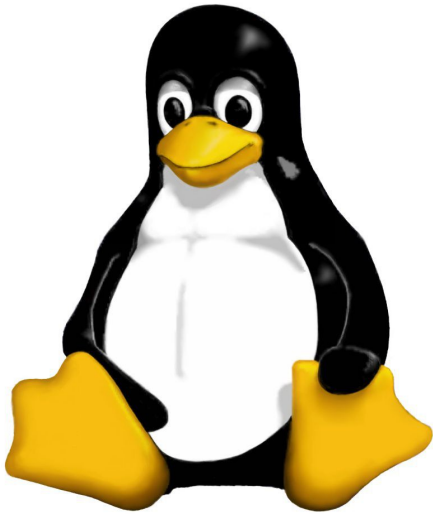
Fontos megkülönböztetni a UNIX és a „Unix” használatát, míg a UNIX az USL licenccel rendelkező USL forráskódból származó rendszereket jelöli, a „Unix” az összes „Unix típusú” rendszert.

## *1.2 A UNIX és a LINUX kapcsolata, a Linux története*

A Unix alá kiadott felhasználói programokat, amiket forráskódban (C nyelvben megírva) adtak ki, bárki fordítgathatta és átírhatta kedve szerint. Ezért Richard Stallman létrehozta az

FSF (Free Software Foundation) alapítványt, melynek célja egy szabadon, (forráskódban is) ingyen hozzáférhető szoftverkörnyezet biztosítása bárki számára. Az FSF szponzorálja a GNU projektet (GNU's Not UNIX), melynek célja egy minél teljesebb Unix rendszer létrehozása, ami teljesen szabadon hozzáférhető. Ennek a jogi megfogalmazása a GNU GPL (GNU General Public License).

A GNU környezet segítségével megnyílt az ajtó a Unix típusú rendszer IBM PC-re való adoptálására. Linus Torvalds egyedül nekiállt a PC alapú Unixos rendszermag megírásának, hogy kipróbálja az i386 processzor védett módú lehetőségeit. Először assembly nyelven (0.01 verzió, 1991. augusztus vége) írta. Ez egy nagyon kezdetleges rendszer volt, igazából a Minix alatt lehetett fordítani és még lemezmeghajtót sem tudott kezelni. Linus ezután C nyelvben kezdte el fejleszteni a rendszerét, ami meggyorsította a fejlesztést. 1991. okt. 5-én hirdette meg Linus az első „hivatalos", 0.02 verziót. Ezen már futott a GCC (GNU C Compiler) és a bash. A terjesztés célja nem az volt, hogy felhasználókat szerezzen, hanem az, hogy segítséget kapjon a rendszermag (KERNEL) fejlesztésében.



1. ábra: A Linux 1.0.0 hivatalos emblémája (TUX)

A Linux 1.0.0 1994 márciusában jelent meg. Ez már teljesen megfelelt a POSIX szabványnak. A verziót ezentúl három, ponttal elválasztott számmal jelölték. Az első az úgynevezett fő verziószámot jelöli. Akkor változtatják nagyobbra, ha valami, a rendszermagot érintő alapvető változtatás történik. Ilyen volt a 2.0.0 megjelenésénél a betölthető rendszermodulok megjelenése.

A második szám az egyes fejlődési szakaszokat jelöli. A 2.4.x verziószámig élt az a konvenció, hogy ha a második szám páros (pl.: 2.4.20), akkor az egy stabil, fejlesztők által garantált működésű rendszermag, ha viszont páratlan (pl.: 2.5.20) akkor ez még csak teszt változat, nem stabil. A harmadik szám a kisebb változtatásokkal növekszik. A legfrissebb verzió mindig megtalálható a http://www.kernel.org webcímen. A 2.6.x szériában valamelyest megváltozott a kernel fejlesztése. A 2.6.x a stabil kernelfa 2.7.x nincs. A stabil kernelfa jelenlegi fejlődése a következők szerint zajlik: a fejlesztő (kernel hacker) patch-et (foltot) készít a stabil forráshoz, amit elküld a stable@kernel.org címre, a küldő kap egy ack-et, ha a patch a várakozási sorba került, és egy nak-et ha elutasították. Az áttekintési ciklusban az „áttekintő bizottság" eldönti, hogy a patch ack-et vagy nak-et kapjon, ha itt

sem utasították el akkor a patch bekerülhet a következő stabil kernelbe. Meg kell jegyezni, hogy miután 2005. áprilisától visszavonták az addig ingyen használható BitKeepert, a 2.6.x-es kernelfát a szintén Linus által írt git patch menedzsment rendszerrel fejlesztik. Azóta a kernel fejlesztési üteme a 2.6.x-es kernelfánál hihetetlenül felgyorsult.

2011 júliusában Linus úgy döntött szakít az eddig használt verziószámozással és egycsapásra 3.x kernelfára ugrott. Linus azt nyilatkozta nagy változást nem eszközöl. Gyakorlatilag a 2.6.40-es kernelt az új verziószámozás miatt 3.0-ként adta ki. A 2.6.x kernelfát olyan sokáig fejlesztették, hogy szinte újraírták az egészet.

A jegyzet aktuális javításának pillanatában (2013. 08. 18.) a 3.10.7 kernel az utolsó stabil verzió (www.kernel.org).

## *1.3 Linux disztribúciók*

A Linux kernel önmagában még csak egy működő rendszermag, amivel igazából semmit sem tudnánk kezdeni, így szükség van kezelő- és felhasználói szoftverekre, hogy teljes rendszert alkossunk. A különböző Linux disztribúciók a meglévő rendszermag köré építették fel saját rendszereiket, tartalmazzák a felhasználói programokat, és könnyedén tudjuk ezeket gépünkre telepíteni. Mi a tanulmányaink során a Debian GNU/Linuxszal (a 7.1 verzióval) fogunk foglalkozni. A Debian GNU/Linuxról bővebben a http://www.debian.org címen olvashatunk. Magyarországi tükörszervert pedig az ftp://ftp.hu.debian.org címen találhatunk.

Más disztribúciók:

- Ubuntu: Debian alapú disztribúció, a fejlesztők a legfrissebb csomagokat teszik bele. Honlapja: www.ubuntu.com.
- Slackware: Az első disztribúciók egyike. Más csomagkezelőt használ, mint a Debian, frissebbek a stabil kiadások csomagjai. Honlap: www.slackware.org.
- Fedora, CentOS, Red Hat: A Red Hat az egyik legelterjedtebb pénzes Linux disztribúció az üzleti szférában. A szupportja miatt az Oracle is támogatja. A Fedora és a CentOS a Red Hat által szponzorált, de nem támogatott nyílt forrású desktop, valamint szerver operációs rendszer. Honlap: www.redhat.com
- Mandrake, Mandriva: A Mandriva a Mandrake utódja. Az első Windows-szerű Linux. Honlap: www.mandriva.org
- Suse, Novell: A Suse volt a másik nagy elterjedt disztribúció. Csomagkezelője a Yast2 ami grafikus, könnyen kezelhető. A Suse Linuxot a Novell felvásárolta, rengeteg pénzt fektet a fejlesztésekbe. Gyakorlatilag elmondható, hogy ma ez a legjobb támogatással rendelkező Linux disztribúció. Természetesen sem a támogatás, sem a disztribúció nem ingyenes.

A Linuxok egy másik „családja” a live disztribúcióké. Ezek telepítés nélkül képesek elindulni CD-ről, a beállításainkat meglévő winchesteren vagy más adathordozón tárolhatjuk, hogy később visszatölthessük. Általában grafikus felülettel rendelkeznek. Klasszikus példája a Knoppix. Fontos megjegyezni, hogy bármely Linux disztribúcióból készíthetünk live rendszert.

### 1.3.1 Debian GNU/Linux

A Debian disztribúció a Linux kernelre épülő operációs rendszer, mely elsősorban a GNU/GPL licencnek megfelelő csomagokat tartalmaz, innen a név GNU/Linux. Nem csak Linux kernelre történtek fejlesztések a Debiannál. Létezik FreeBSD kernelre épülő Debian terjesztés, a Debian GNU/kFreeBSD és létezett HURD-re épülő Debian disztribúció, a Debian GNU/HURD. A Debian csapat egyszerre több verziót tart karban, a stable, testing, sid verziókat. Egy verzió az életét mindig sid-ként kezdi. A sid a *still in development* kifejezésből ered (és persze egy Toystory figura neve is). Az egyes csomagokat a sid-ből a testing verziókba teszik, majd itt hosszú tesztelési fázis után, a testing verzió stable lesz. A stable verziók, több release-t élnek meg. Ennek oka általában a biztonsági hibákat javító csomagok száma. Hiszen képzeljük el, ha mindig az eredeti release-t kell feltenni, akkor egy stable az élete végén már rengeteg csomagfrissítést igényelne. A jegyzet utolsó javításakor a stable verzió a *wheezy* (ez a 7.1 verziót jelenti, ami 2013. június 15-én jelent meg, a 7.0 verzió megjelenési dátuma: 2013. május 4.), a testing a *jessie,* és jelen sorok írásakor még karbantartott oldstable ág (ami a korábban a stabil kiadás volt) most a *squeezee* lett. (Érdeklődőknek bővebben: http://www.debian.org/releases/) Magyarországon több Debian tükörszerver is létezik, az ftp.hu.debian.org-ot célszerű használni, ez jelenleg a leggyorsabb Debian mirror. Létezik a Széchenyi István Egyetemnek is bejegyzett tükörszervere ez a debian.sth.sze.hu melyet az egyetem kollégiuma tart karban. És a laborban is üzemeltetünk egy nemhivatalos tükröt a debian.tilb.sze.hu címen.

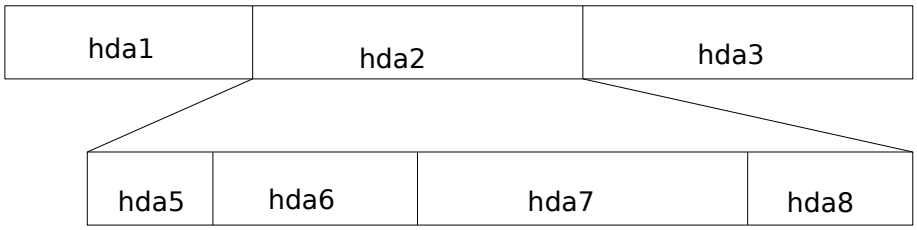
A 6.0-ás verzió egyik újdonsága a teljesen szabad kernel, mely nem tartalmaz olyan firmware-eket, melyek nem felelnek meg a Debian Free Software Guidelines – DFSG irányelveinek. Azaz a programok forráskódjának is hozzáférhetőnek és szabadon terjeszthetőnek kell lennie. Ha ez nem teljesül, akkor a Debian fejlesztői nem tudják biztosítani a támogatást az „idegen” kódra, mint ahogy azt az általuk karbantartott kódok esetében teszik. Ezért hardvergyártók által kiadott ingyenes, de a DFSG-nek meg nem felelő firmware-ek a main archívumából átkerültek a non-free tárolóba, melyeket valamely csomagkezelővel tudunk telepíteni ha valamely hardverünk ezt igényli a működéséhez.

Továbbá, a Debian 6.0 bemutatja a függőség alapú betöltő rendszert, amivel a rendszerbetöltés gyorsabbá válik a folyamatok párhuzamos futtatása és a függőségek helyes követése által.

### 1.3.2 Debian GNU/Linux telepítésének előkészítése

Linuxot telepíteni teljesen üres winchesterre a legegyszerűbb. A disztribúciók nagy része automatikusan felajánlja számunkra a partíciók elkészítését. Mi az a partíció és miért van rá szükségünk?

Egy PC merevlemezén 4 db elsődleges partíció lehet, és az egyiken (extended partíció) belül több darab másodlagos partíció helyezkedhet el. Ezeken Linux vagy más operációs rendszer alatt fájlrendszereket lehet létrehozni. A fájlrendszereket a Linux képes kezelni, és könyvtárakba felcsatolni különböző paraméterekkel: csak olvasható, írható olvasható, stb. Lehetőségünk van bootloaderek segítségével különböző partíción lévő más-más típusú operációs rendszerek betöltésére.



2. ábra: partíciók

Fontos megjegyezni, hogy a squeeze-től elkezdve minden egyes merevlemezt – még az IDE csatolófelületűeket is – sd{xy} eszközként tartja számon a rendszer, így csak régebbi operációs rendszereknél találkozhatunk hd{xy} eszközökkel. Ez a 2.6.20 kernelben módosított libata modul miatt van így. Ubuntunál 2007 óta használják. A Debian kernelében csak a 2.6.30 óta vezették be a UUID-val egyetemben.

## *1.4 A Linux rendszer elindulása*

### 1.4.1 POST

A számítógép bekapcsolása után a POST indul el (Power On Self Test). A POST a számítógép hardvereit ellenőrzi le, hogy megfelelően működnek-e. A POST lefutása után a rendszer egy boot betöltőt keres egy boot szektorban, a BIOS-ban megadott boot sorrendnek megfelelően. Ez lemezek esetében egy 446 byte-os rész, mely lehet az MBR-ben (Master Boot Record) vagy ha itt nincs, akkor a partíciós tábla szerinti (az MBR felső 64 byte-ja) aktív partíción (a nagy bootloaderek kicsit máshogy működnek). A boot szektor memóriába történő töltődése után az betölti a kernelt a memóriába, és átadja a vezérlést az operációs rendszernek.

A régi 2.4.x-es Linux kernel rendelkezik egy ilyen 512 byte-os rendszerbetöltővel, ami képes arra, hogy saját magát kicsomagolja és betöltse. Erről meggyőződhetünk, ha egy ilyen kernelt dd parancs segítségével egy flopira másolunk és arról bootolunk. A kernel el fog indulni és csak a root „/” partíció felcsatolásakor fog hibaüzenettel megállni, ha nem talál ilyet. A kernel különböző paramétereit, így a root partíciót az rdev paranccsal módosíthatjuk.

### 1.4.2 PXE (Preboot Execution Environment)

A PXE az Intel fejlesztése, melynek célja, hogy a rendszer hálózatról DHCP és TFTP segítségével bootoljon fel. A számítógép bekapcsolása után, egy DHCPDISCOVER-t küld, amire a DHCP szerver DHCPOFFER-rel válaszol, melyben megad egy TFTP szervert és egy rajta található NBP-t (Network Bootstrap Program). Linux alatt ilyent a syslinux (és egyes újabb GRUB verziók) csomag tartalmaz. Az NBP egy futtatható állomány mely minimális funkciókkal rendelkezik: UDP kezelés, hogy a kernelt a távoli gépről le tudjuk

szedni. Innen minden „ugyanúgy” zajlik, mintha lemezről bootolnánk. (FONTOS!!! A terjedelem miatt a fenti leírás közel sem teljes!)

Irodalom:

http://en.wikipedia.org/wiki/Preboot_Execution_Environment
http://www.pix.net/software/pxeboot/archive/pxespec.pdf

### 1.4.3 LILO

A LILO (LInux LOader) az első elterjedtebb összetett bootloader. A rendszer a POST után meghívja az első fokozatot (first-stage), amely majd a második fokozatot (second-stage bootloadernek is nevezik az ilyen típusú betöltőket) indítja el, ami kommunikál a felhasználóval, választási lehetőséget biztosít számára, hogy több operációs rendszer közül válasszon. A lilo-t a `/etc/lilo.conf` állomány beállításával tudjuk konfigurálni, mely a lilo parancs kiadása után lép érvénybe. Az „új” lilo képes a korábbi 1024 cilinder felett lévő kerneleket betölteni, van menüje mind karakteres mind grafikus felületre. Hátránya, hogy a kernelek és initrd-k fix helyen kell, hogy legyenek a partíción, különben a LILO hibával megáll.

### 1.4.4 GRUB

A GRUB a GNU projekt loadere. Hasonló elven működik, mint a LILO, csak egy fokozattal több van benne (1.5 stage 30kbyte közvetlenül az MBR után). Ami újdonság benne, hogy ismeri és kezeli a fájlrendszereket (az XFS-t csak az 1.x verzió), melyekről a Linux képes bootolni. A kernelnek nem kell fix helyen lennie a partíción, mert a loader second-stage-je képes a partíciót végignézni, és ha létezik a kernel, akkor betölti azt. A konfigurációs állománya a `/boot/grub/menu.lst` Debian alatt. Ha ez az állomány valamilyen okból kifolyólag nem érhető el, a stage 2 egy CLI-t (Command Line Interface) ad a felhasználónak, ahol a felhasználó betöltheti a kernelt a megfelelő paraméterekkel, és mivel a bootloader kezeli a fájlrendszereket, megkeresi a kernelt és betölti. Debian alatt létezik update-grub parancs, mely a grubot feltelepíti a rendszerünkre úgy, hogy a már meglévő operációs rendszereket, kerneleket is hozzáadja. A GRUB a legtöbb operációs rendszert felismeri, ám néhány rendszert (Pl.:Windows7, OS/2) csak chain-loading funkcióval tud elindítani mely annyit tesz, hogy nem a tényleges rendszert hanem annak boot loader-ét tölti be.

#### 1.4.5 GRUB2

A GRUB2 a Debian Squeeze rendszerrel lépett be a Debain release-ek közé (a legelső disztribúció mely használta az Ubuntu 9.10-es - Karmic Koala -). Konfigurációs állománya a `/boot/grub/grub.cfg`. A legfőbb változások a következők:

- Kivették a 1.5 stage-et, mert ennek feladatát már a stage2 látja el.
- Amennyiben a GRUB2 utolsó frissítésekor nem érzékelt 1-nél több operációs

rendszert, úgy a menü nem jelenik meg. Ezt a bootolási folyamat közben a SHIFT-et nyomva tartva felülírhatjuk; ezzel úgynevezett rescue módba lehet lépni.

- A különböző fájlrendszerek, illetve szolgáltatások modulként vannak kezelve, és csak azokat tölti be magának, melyek a bootolási folyamathoz kellenek.
- Lehetőség van még úgynevezett LIVE disztribúciók image fájljainak betöltésére közvetlenül merevlemezről.
- UUID-k kezelése. Az Universally Unique Identifier segítségével az adott partíciót azonosítani lehet a csatolási pont (lásd később) és az eszköz neve nélkül. Az egyediséget a szám nagysága biztosítja. A Wikipédia szerint 1 trillió (milliárd[2]) UUID kellene minden 10 nanoszekundumban létrehozni 10 milliárd évig, hogy elfogyjanak ezek az egyedi azonosítók. A UUID kinyerhető a következőképpen:

```
root@teacherb:~# ls -l /dev/disk/by-uuid/
összesen 0
lrwxrwxrwx 1 root root 10 aug   22 09.03 42bb6882-397e-4b32-b8e9-76662329dba6
-> ../../sdb1
lrwxrwxrwx 1 root root 10 aug   22 09.03 a7cfa56b-7c88-4070-aa99-457943d0a410
-> ../../sda1
```

Vagy a blkid paranccsal:

```
root@aeryn:~# blkid
/dev/sda1: UUID="89a32e40-7dab-4210-b5b0-329d2a28b165" TYPE="ext3"
/dev/sdb1: UUID="ad650b26-7c19-46af-9002-5fad23688d6f" TYPE="ext4"
/dev/sdb5: UUID="8f288b87-346d-48b6-86a2-579eb88ab79f" TYPE="swap"
/dev/sdd1: UUID="984286ac-5260-4200-8c18-9b30c348fa7b" TYPE="ext3"
/dev/sde1: UUID="089ccd12-79ba-4ab9-b25c-f87d7c7f4d30" TYPE="ext3"
/dev/sdc1: UUID="1192db68-a2d6-4191-9f76-1b6abd6f366b" TYPE="ext3"
```

- Már nem csak x86-os platformokat támogat (Pl.: PowerPC)
- Egyedi szkriptelhetőség, melyeket a `/etc/grub.d` könyvtárban kell elhelyezni
- A változtatások csak az update-grub paranccsal lépnek érvénybe.

## 1.4.6 Initrd

Régen a kernel miután a memóriába töltődött és elindult, az init parancsot hívta meg. Ez napjainkra megváltozott, a kernel mérete drasztikusan nőtt és nő. Ezért a Linux kernel manapság több fázisban indul el, van egy alap kernel, melynek viszonylag szerény a mérete és tudása. Ez pont annyira elegendő, hogy egy RO (Read Only) fájlrendszert a memóriába ramdiskbe töltsön. Ezt a fájlrendszert nevezik initrd-nek. Az initrd feladata, hogy felismerje a gépben lévő hardvereket és betöltse a kezelésükhöz szükséges modulokat. Menet közben kiváltották ezzel a megoldással a régi inode-okat pocsékoló `/dev` könyvtárat, és a szükséges eszközfájlokat az `udev` segítségével hozzák létre. Ezzel átláthatóbb lett a `/dev` könyvtár, hiszen az eddig feleslegesen meglévő állományok eltűntek. A korai initrd a cramfs-re épült (sarge) és az `mkinitrd` paranccsal hozhattuk létre. Etch alatt már nem ezt a megoldást alkalmazzák, hanem `cpio`-val tömörített könyvtárat használnak, melyet pl. a `yaird`, `mkinitrd` hozhatunk létre. Mind a cramfs mind a `cpio`-s megoldás esetében

`gzip`-pel tömörítve van az initrd, melyet a kernel csomagol majd ki induláskor. Mindkét esetben, miután az initrd elvégezte feladatát, a vezérlést a root partícióra teszi át a `pivot_root` parancs segítségével. Ezek után a Linux az `init` meghívásával folytatja a bootolást.

A cramfs-es initrd-t `gunzip`-pel történő kicsomagolás után `mount -o loop kicsomagoltfajlneve csatolasipont` paranccsal tudjuk felcsatolni. A `cpio`-st pedig kicsomagolhatjuk a következő paranccsal: `gzip -dc < [file] | cpio -i -d [célkönyvtár]`.

### 1.4.7 Init

Ha eddig a pontig eljutott a számítógép, akkor először a `/linuxrc`-t keresi, ami általában egy szkript, ha talál ilyet, akkor az a 0-s PID[1]-el fut le. Ha ez sikeres volt vagy nem létezik, a következő fájlokat keresi a rendszer: `/sbin/init`, `/etc/init`, `/bin/init`. Ha létezik közülük bármelyik is, akkor elindul az 1-es PID-el. Az init (init alatt, ha csak nem hangsúlyozzuk külön, ezek után a sysvinit csomagot fogjuk érteni) a `/etc/inittab`-ban lévő bejegyzések alapján folytatja a rendszer betöltését.

### 1.4.8 Az /etc/inittab

```
# Az alapértelmezett futási szint.
id:2:initdefault:

# ez a szkript fut le legelsőnek
si::sysinit:/etc/init.d/rcS

# single módra történő váltáskor root jelszót kér.
~~:S:wait:/sbin/sulogin

# futási szintekhez tartozó parancsok

l0:0:wait:/etc/init.d/rc 0 # halt
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2 # Debian alapértelmezett futási szintje
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6 # reboot
# Ha bármilyen ok miatt megszakad az init, akkor root jelszót kérjen
z6:6:respawn:/sbin/sulogin

# CTRL-ALT-DEL hatására mi történjen.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Áramszünetre vonatkozó utasítások
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop
```

1 PID: Process ID, processz azonosító. A UNIX rendszerek alatt a futó alkalmazások egy számmal vannak megjelölve ezeket nevezzük PID-eknek

```
# Format:
#  <id>:<futási szint>:<action>:<parancs>
# ezekket a paracsokat hajtja végre az init a megadott futási szinteken (ez most
# gyakorlatilag a login prompt.

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

# soros porti login
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100

# modemes login
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3
```

A respawn után lévő parancsokat az init figyeli, ha „meghalnak”, újraindítja. Ide célszerű lehet például a syslogot tenni.

Vannak tendenciák a régi, jól bevált init lecserélésére. Ennek oka, hogy a mostani nagy disztribúciók sokáig töltődnek, ha desktop gépként funkcionálnak (szerver esetén mindegy, hiszen jó eséllyel sose állítjuk le, legalábbis nem szeretnénk). Ezek az init típusok párhuzamosan indítanák a különböző processzeket. Ilyen init típus az upstart, mely az init leváltását visszafele kompatibilis módon oldja meg.

### 1.4.9 Függőség alapú bootolás

A Debian rendszer eddigi bootolási folyamata már egy ideje elavultnak számított, sok olyan feltételezést tartalmazott, melyek a bootolás adott pillanatában nem teljesültek. Pl.: Az NFS mountolása előtt nem indult el a hálózatkezelő, vagy néhány merevlemez még nem volt felcsatolva, de a fsck már használta volna. Ezért a squeeze-től úgynevezett függőség alapú vagy eseményvezérelt lett a bootolási folyamat. Tehát itt már az upstart végzi a boot folyamatot. Mondhatnánk, hogy akkor miért is kell az init? Azért, kellett megtartani, mert rengeteg már meglévő rendszer használja, és valószínűleg még 5-10 év múlva is lesz olyan működő konfiguráció mely ezt használja. Az upstart lényege, hogy a különböző szolgáltatások párhuzamosan indulnak el, figyelve az egymásra való függőségre. Vagyis a fenti példákból kiindulva először a hálózatkezelő lesz elindítva, és csak azután kezdődik meg az NFS partíciók felcsatolása, de ezek mellett természetesen más szolgáltatások párhuzamosan elindulhatnak.

Az ilyen rendszerben található indítószkripteket LSBinit szkripteknek nevezzük. (LSB: Linux Standard Base) Fő különbségük az eddigi init szkriptekhez képest, hogy kiegészülnek egy függőségeket leíró bejegyzéssel. Alant példaként az ssh daemon LSBinit szkript plusz bejegyzése magyarázattal:

```
### BEGIN INIT INFO
# Provides:             sshd               ( milyen szolgáltatást nyújt az adott
                                            szkript melyre más LSBinit szkriptek
                                            hivatkozhatnak '# Required-Start' sorban)

# Required-Start:       $remote_fs $syslog ( mi kell, hogy el tudjon
                                           indulni, mindenképpen kötelező
                                           sor, még ha üres is)

# Required-Stop:        $remote_fs $syslog ( mi kell, hogy le tudjon
                                           kapcsolni, ezeknek még futnia
                                           kell)

# Default-Start:        2 3 4 5            ( milyen runlevelnél induljon el )
# Default-Stop:                            ( milyen runlevelnél ne induljon el)
# Short-Description:  Secure Shell server  ( a daemon rövid leírása )
### END INIT INFO
```

Vannak úgynevezett előre definiált szolgáltatások melyekre egy ilyen LSBinit szkriptben hivatkozni lehet függőségként. Ilyenek például:

- $local_fs – az összes helyi fájlrendszer fel van csatolva
- $network – hálózati szolgáltatás elindítva
- $named – névfeloldásért felelős daemon elindítva
- $remote_fs – minden fájlrendszer felcsatolva
- $syslog – rendszernaplózás elindítva
- $time – rendszeridő beállítva NTP vagy hardveres óra segítségével
- $all – ha már minden más szolgáltatás elindult

Saját magunk által írt programunkhoz is készíthetünk LSBinit szkriptet, ehhez nagy segítséget nyújthat a `/etc/init.d/skeleton` nevű fájl, amely a főbb funkciókat már tartalmazza, csak testre kell szabni.

## *1.5 Futási szintek (runlevelek)*

A Linux lévén SystemV típusú rendszer, ún. futási szintekkel rendelkezik. Összesen 7 futási szint van, melyek a következők:

- 0 halt
- 1 single user
- 2-5 multiuser
- 6 reboot

A Debian nem tesz különbséget a 2-5 futási szintek között. Más disztribúciók megkülönböztetnek hálózati runlevel-t, grafikus runlevel-t. A nagy UNIX disztribúciók

közül az AIX, HP-UX rendelkeznek még futási szintekkel. A BSD típusú rendszerek és néhány Linux terjesztés nem a SystemV init-et (sysvinit) használja.

## *1.6 A Debian GNU/Linux csomagkezelői*

Az **dpkg** a Debian GNU/Linux jellegzetessége. Rendszerünkön nyilvántartja a feltelepített csomagokat. Ezeket bármikor kedvünk szerint módosíthatjuk. Munkánk megkönnyítése érdekében használhatunk valamilyen *frontend*et. A legelterjedtebb az APT (*Advanced Package Tool*) Ebben a csomagban a legfontosabb program az **apt-get**. Szintaxis: `apt-get [kapcsolók] <parancs> [csomagnév]`

Ennek több, mint 10 parancsa van, ezek közül csak néhány legfontosabbat nézünk meg.

- **update**: csomaglista frissítése. A rendszerünk installálásakor az install szkript megkérdezi, hogy milyen médiákról telepítjük rendszerünket. Ezeket más néven apt source-oknak (apt forrásoknak) nevezzük. Az apt források lehetnek a telepítő cd-k, URI-k (http://, ftp://, stb.), vagy akár helyi könyvtárak is. Ha már a meglévő rendszerünkön szeretnénk apt forrást változtatni, vagy újjal bővíteni, akkor a `/etc/apt/sources.list` fájlt kell módosítanunk, de használhatjuk az `apt-setup` parancsot is Az `apt-get update` helyett javasolt a `dselect update` használata, ami részletesebb, jobb adatbázis fájlt hoz létre, ami a telepíthető csomagokat tartalmazza.
- **install**: csomag telepítése a rendszerünkre. Mellette használhatjuk, és néha hasznos is a `--reinstall` kapcsoló, ami a már előzőleg feltelepített csomag újratelepítését írja elő. Ha a telepített csomag `.deb` fájlja megtalálható a gépen a csomagok cache könyvtárában, akkor nem tölti le újra a rendszer. Egy ilyen eset az `ssh` újratelepítése, amikor egészen biztosak akarunk lenni, hogy az általunk használt `ssh` nincs megfoltozva (csak előtte bizonyosodjunk meg róla, hogy a csomagok számára fenntartott cache könyvtárban, nincs az `ssh .deb` csomagja, vagy adjuk ki az `apt-get clean` parancsot)
- **remove**: csomag eltávolítása a rendszerünkről, ennek szintén van egy hasznos kapcsolója a `--purge`. Ezzel azt érjük el, hogy az eltávolításkor a telepítő megpróbálja leszedni a konfigurációs fájlokat is (rendszerint sikerül neki).
- **clean**: az apt a letöltött csomagokat, a `/var/cache/apt/archives` könyvtárban tárolja. Ezzel a paranccsal takaríthatjuk ki ezt a könyvtárat.

Egy másik nagyon fontos program az **apt-cache**. Szintaktikája: `apt-cache [kapcsolók] <parancs> [csomagnév]`. Opciók:

- **search**: csomagkeresés valamilyen jellegzetesség (pl.: név vagy funkció alapján)
- **show**, **showpkg**: csomag információ (függőségek, verziószám stb.)

Az **aptitude** is az apt frontend-je. Az `aptitude` programmal könnyedén tudunk böngészni a már telepített és a még nem telepített programok között.

**dpkg-reconfigure**: szintakszis: `dpkg-reconfigure [-a] <csomagnév>`. A Debian csomagot, `-a` esetén pedig az összes csomagot állítja be újra a telepítő.

**dpkg --get-selections > getsel.txt**: a gépünkön lévő csomagok státuszáról ad információt, amit a `getsel.txt` fájlba tesz (installed - telepített, deinstalled - eltávolított, purge - eltávolított a konfigfájljaival együtt, hold – visszatartott, az újabb verzió nem kerül telepítésre)

**dpkg --set-selections <getsel.txt**: a `getsel.txt` alapján, beállítja a csomagok státuszát, amit egy `apt-get dselect-upgrade` paranccsal érvényesíthetünk.

Fentebb már megemlítettük a `dselect` parancsot. A `dselect` a `dpkg` olyan frontendje, ami némi „grafikával” rendelkezik. Használata bonyolultabb, mint az `aptitude`-é, de sok Debian fan csak ezt az alternatívát tudja elképzelni. Mi csak a fent említett update kapcsolóját használjuk.

Az **apt-build** program a gépünkre optimalizálva teszi fel a csomagot forrásból. Ehhez fel kell telepítenünk az `apt-build` csomagot a következő paranccsal: `apt-get install apt-build`, valamint egy deb-src forrást kell adnunk a `sources.list` fájlhoz. A Debian forrás sort másoljuk és a deb szócskát a sor elején deb-src-re egészítjük ki.

Szintén hasznos program a `deborphan`, ami a gépünkön lévő *obsolated* (már nem használt) library csomagokat listázza ki. Ezt egy kis shell szkript segítségével és a fenti utasítások használatával törölhetjük a rendszerünkről. Az `orphaner` egy keretrendszeres GUI a `deborphan`-hoz.

Valamint utolsó sorban megemlítjük a `tasksel` nevű programot, ami arra hivatott, hogy egy általunk kiválasztott funkciót/funkciókat ellátó gép dependenciáját (függőségét) beállítja, majd telepíti a csomagokat.

## 1.7 A Debian GNU/Linux processzek (újra)indítása, leállítása

A Debian alatt a későbbiekben oktatott szerver démonokra általánosan jellemzőek lesznek, hogy a /etc/szervernév.conf vagy /etc/szervernév könyvtárak alatt lehet beállítani őket. Elindítani a következő paranccsal lehet az egyes démonokat:

```
/etc/init.d/szervernév start
```

leállítani:

```
/etc/init.d/szervernév stop
```

újraindítani:

```
/etc/init.d/szervernév restart
```

## 1.8 A deb csomag

A Debian disztribúció csomagjai `ar` paranccsal vannak betömörítve. Egy `.deb` állomány kibontása:

```
laptop:/tmp/deb# ar -t kernel.deb
debian-binary
control.tar.gz
data.tar.gz
laptop:/tmp/deb# ar -x kernel.deb
laptop:/tmp/deb# ls
control.tar.gz  data.tar.gz  debian-binary  kernel.deb
laptop:/tmp/deb#
```

A `-t` kapcsolóval listázzuk ki mi az, ami az ar archívumon belül van. A `-x` pedig kicsomagolja az archívum fájljait. Mint láthatjuk egy `control.tar.gz` egy `data.tar.gz` és egy `debian-binary` (szövegfájl) van, ez utóbbi az archívum verzióját adja meg. A `data.tar.gz` azokat a fájlokat tartalmazza, amelyek a csomagból telepítésre kerülnek, számunkra ez most nem annyira érdekes. A `control.tar.gz` annál inkább. Ez tartalmazza a csomag függőségeire vonatkozó információt, a csomag telepítése előtt vagy után végrehajtandó parancsokat. Nézzük meg, mit tartalmaz a `control.tar.gz` állomány:

```
laptop:/tmp/deb# tar -tzf control.tar.gz
./
```

```
./postinst
./config
./postrm
./preinst
./prerm
./templates
./control
```

A control fájl a függőségeket és a csomag információkat tartalmazza. A preinst, postinst, config esetünkben egy-egy perl szkript. Preinst esetén a csomag telepítése előtt, a postinst esetén pedig, a telepítés után fut le (pl.: ilyen szkriptek kérdezgetnek bennünket postfix telepítésnél). A prerm, postrm parancsok a telepítés előtt illetve után eltávolítandó fájlokról gondoskodnak. Ezek szintén perl szkriptek.

## *1.9 A vi, vim kezelése*

A `vi` egy alapvető szövegszerkesztő program UNIX rendszerekhez. A `vim` a `vi` utódja, lényegesen többet tud nála. Szintaxis kiemeléssel rendelkezik (színezi a beírt forrásokat). Néhány hasznos segédprogram van hozzá például `vimdiff`. (A `view` a `vi` futtatása read-only módban csak megjelenítésre. Ezek Debian esetén gyakran szimbolikus linkek más programokra.) A `vi` 2 üzemmódban dolgozik: megjelenítő (visual) és szerkesztő módban.

Visual módban, a „:” gomb megnyomása után például a következő funkciók érhetők el:

a = beírás (az aktuális karakter után),

i = beszúrás (az aktuális karakter elé),

r = felülírás mód 1 karakter erejéig,

R = felülírás mód,

o = új sor, x betű törlés,

v: kijelölést kezd,

y: kijelölést befejez másolásra,

p: beilleszt,

d = sortörlés,

:q = kilép,

:w = írás (lemezre),

:q! mindenképpen lépjen ki mentés nélkül (így nem rontjuk el az eredeti fájlt).

Példa: kilépés, mentéssel = :wq

Kilépés, mentés nélkül = :q!

# 2 Alapvető parancsok a UNIX-ban

Mielőtt nekiállunk részletesen elemezni a Unix rendszereket, szükséges néhány alap parancs ismerete. A Unixban minden felhasználó egy saját környezetben dolgozik. Ezeket SHELL-eknek hívjuk. Alapesetben ez a `bash`. A `bash` kezelése egyszerű, és felhasználóbarát. A kiadott parancsokat visszahívhatjuk a fel és le gombot nyomogatva. A parancsok, fájl nevek és könyvtár nevek megadásánál nem szükséges a teljes nevet kiírni, hanem a TAB gomb lenyomásával kiegészíti azt. A `bash`-ról a későbbiekben lesz még szó.

Most nézzük meg (ismételjük át) a legfontosabb parancsokat! Amit a parancsok leírásában szögletes zárójelek közé („`[`” és „`]`”) teszünk, az opcionális, azaz használható is, de el is hagyható. A csúcsos zárójelpár („<” és „>”) egy paraméter magyarázatát tartalmazza. Mindezek ún. metanyelvi zárójelek, a parancsok kiadásakor nem szabad használni őket!

Az alapvető Unix parancsok és aktuális dokumentációjuk elérhető:

http://www.gnu.org/software/coreutils/manual/

**`ls`**: list, fájlok és könyvtárak listázása. Szintaxis: `ls [kapcsolók] <milyen könyvtár>`. Legtöbbször használt kapcsolók: `-l` long, tehát hosszú listázás: fájlok és könyvtárak jogai, tulajdonos és csoport kiírása. `-a`, `--all` kapcsoló minden fájlt kilistáz, azaz a „**`.`**”-al kezdődő nevű *rejtett fájl*okat is megjeleníti. Itt jegyezzük meg, hogy létezik két speciális fájl a könyvtárakban. Az első a „**`.`**”, ami magát az adott könyvtárat jelenti, a második a „**`..`**”, ami a könyvtár szülő könyvtárát jelenti. Természetesen a „**`..`**” a „/” (gyökér könyvtárt) esetében szintén önmagát jelenti. A kapcsolókat lehet egymás után megadni pl.: `ls -la /home`

**`cd`**: change directory = könyvtárváltás. A „**`.`**” az aktuális könyvtárat jelöli a „**`..`**” eggyel feljebb lép a könyvtárfában, ha nem a „/” („root”) könyvtárban voltunk. A `cd ~[felhasználónév]`, ha nem adunk meg felhasználónevet a saját home könyvtárunkba lép, ha van megadva a tilde (~) után felhasználónév, akkor a megadott felhasználó könyvtárába léphetünk be megfelelő jogosultságok esetén. A „-” eggyel ezelőtti könyvtárba lép vissza, tehát ha a `/home` -ból `cd /` paranccsal a „/” gyökérkönyvtárba léptünk a `cd -` visszaléptet minket a `/home` -ba.

Pl.: `cd /home/lencse` teljes elérést megadva, vagy lépésenként:

```
lencse@tilb:/# cd home
lencse@tilb:/home# cd lencse
lencse@tilb:~/#
```

esetleg:

```
lencse@tilb:/# cd ~lencse
```

**pwd**: print working directory = aktuális könyvtár kiíratása, ahol éppen tartózkodunk

**cp**: azaz copy = másolás. `cp [kapcsolók] <mit> <hova/milyen_néven>`

A `-r` kapcsoló rekurzív másolást jelent, azaz az adott könyvtárat a tartalmával együtt másolja.

Példa:

```
root@tilb:# cp /home/lencse/kiskutya.gif /home/drmomo/bloki.gif
```

**mv**: azaz move = mozgatás: `mv <mit> <hova>`, azonos köteten belül átnevezés vagy más könyvtárba való belinkelés, kötetek közötti mozgatás esetén a fájl/könyvtár létrehozása és törlése!

```
root@tilb:# mv /home/lencse/kismacska.jpg /home/drmomo/cica.jpeg
```

**rm**: azaz remove = törlés: `rm <mit>`. Leggyakrabban használt kapcsolók: `-r` rekurzív törlés, `-f` force mindenképpen töröljön.

**mkdir**: könyvtár létrehozása, `-p` hatására az egész könyvtár struktúrát létrehozza, ha az nem létezett.

**rmdir**: azaz remove directory = könyvtár törlése. Csak üres könyvtárat lehet letörölni.

```
# rmdir /home/buksi
```

**mount**: fájlrendszer becsatolására szolgál (bővebben később)

**cat**, **tac**: szöveges állományok megjelenítése az alapértelmezett kimenetre, a `tac` visszafelé jeleníti meg.

```
# cat /home/lencse/kiskutya.txt
```

**sort**: sorba rendezi a STDIN-re érkező adatokat (egy sor számít egy adatnak), ha nincs kapcsoló, akkor lexikografikusan, `-r` (reverse) fordítva, `-n` numerikusan rendezve.

**df**: a felcsatolt (mounted) fájlrendszerek foglaltságát jeleníti meg. Leggyakrabban használt kapcsolója a `-h` (human-readable format), ami annyit jelent, hogy nem kilobyte-okban, hanem mega- vagy gigabyte-okban adja meg a foglaltságot. A `-i` kapcsoló a mountolt köteteken lévő inode-okról (inode-okról később) ad információt.

**du**: a fájlok, és könyvtárak helyfoglalását adja meg. Kapcsolók: `-a` vagy `--all`, `-k`: kilobyte, `-m`: megabyte, `-h` human-readable.

**touch**: fájllétrehozás (0 byte méretű), vagy ha már létezik a fájl, akkor a módosítás ideje változik meg.

```
# touch /home/lencse/macska.txt
```

**ps**: processz lista. A parancsnak kétféle szintaxisa van! Egy adott UNIX rendszerben ezek közül lehet, hogy csak az egyik működik helyesen, a másik lehet, hogy hibásan, de lehet, hogy figyelmeztetést ad, esetleg akár mindkettő is működhet! A legelterjedtebb a BSD szintaxis, ahol a kapcsolók előtt *nincs kötőjel („-”)!* Kapcsolók: `a` minden (*all*) processz, beleértve azokat is, amelyeket más felhasználók futtatnak. `u` user-oriented output, `x` minden egyéb a felhasználó által elindított, de tty-hez nem köthető processzek kiíratása, `w` széles forma, nem vágja le, ha „kilóg” a képernyő szélén, hanem sortörést csinál (ha túl hosszú lenne a sor, akkor több „`w`” megadásával több sorba töri a processz lista sorait). Pl.:

```
# ps aux
```

Létezik a parancsnak POSIX kompatibilis szintaxisa is, itt a kapcsolók előtt *van kötőjel („-”)!* Kapcsolók: `-e` minden (*every*) processz, beleértve azokat is, amelyeket más felhasználók futtatnak. További opciók `-f` (full) és `-l` (long) kimeneti formátum. Pl.:

```
# ps -efl
```

**kill**, **killall**: processzek vezérlésére szolgál. Ilyenek lehetnek a `-9`, vagy más néven a `-KILL`, a `-CONT`, ami a `-STOP` szignállal megállított processz végrehajtását folytatja, esetleg a `-HUP` (HangUP), ami egy processzt újraindít. A `kill` után a PID-et kell megadni, a `killall` a processz nevét kéri és lehetősége (jogosultság) szerint az utasítást az összes olyan nevű processzen végrehajtja. Elég veszélyes távoli bejelentkezésnél rootként kiadni egy `killall -9 sshd` parancsot, hiszen ez az összes ilyen nevű processzt leállítja, beleértve azt is, amin mi épp kiadtuk a parancsot.

**echo "szoveg"**: a parancs segítségével sztringet írathatunk ki, `-n` kapcsolója nem tesz sortörést, miután kiírta a szöveget.

**man**: azaz manual = majdnem minden parancshoz és feltelepített futtatható programhoz segítséget, leírást kapunk. Ezt a `man` nevű paranccsal lehet megtekinteni.

**more**: kiírja a megadott szöveges állományt az alapértelmezett kimenetre oldalanként tördelve visszafele lapozásra nincs mód. A szóköz billentyű hatására egy képernyőt halad előre, az Enter hatására pedig egy sort. A per jel („/”) után megadott (reguláris) kifejezésre lehet vele keresni, és még számos funkciója van. Pl.: `more hosszuszoveg.txt`

**less**: tudja azt, amit a `more`, csak lehet visszafele is lépkedni. Hosszú fájloknál gyorsabb, mert nem olvassa be végig a fájlt. (És még rengeteg funkciót nyújt, többek között számos `vi` parancsot is.)

**head**: szöveges állomány kiíratása (megadott `-n` vagy tíz sor).

**tail**: ugyanúgy szöveges állomány kiíratása (megadott `-n` vagy utolsó 10 sor), viszont `-f` kapcsolóval lehet követni a szöveges állomány változását. Kiválóan alkalmas a log fájlok követésére.

```
tail -f /var/log/syslog
```

**tar**: fájlok és könyvtárak fájlba csomagolása. Szintaxis: `tar [kapcsolók] [hova] [mit]`. Kapcsolók: `-c` = create, `-a` = add: becsomagol, `-x` = extract: kicsomagol, `-v` = verbose: képernyőn megjelenít, `-f` = fájl, `-z` = `gzip`-el akarjuk (ki/be) tömöríteni, `-j` = `bzip2`-vel akarjuk (ki/be) tömöríteni.

Az `ize.tar.gz` fájlba becsomagolom az `ize` könyvtár tartalmát:

```
# tar -cvfz ize.tar.gz ize/
```

Kicsomagolom az `ize.tar.gz fájlból`

```
# tar -xvfz ize.tar.gz
```

**ln**: link létrehozása. Későbbiekben magyarázzuk el a link fogalmát. Alap esetben ún. hard linket hoz létre, `-s` kapcsolóval pedig szimbolikus linket lehet létrehozni. Szintaktika: `ln [kapcsolók] <fájl> <link neve>`

**sync**: a parancs hatására a memóriában lévő adatokat kiíratjuk az adathordozóra.

**lsmod**, **modprobe**, **insmod**, **rmmod**, **depmod**, **modinfo**: a Linux kernel moduljait tudjuk listázni, betölteni, eltávolítani, függőségi viszonyt beállítani, (későbbiekben lesz még róluk szó).

**su**: felhasználók közötti váltás, `su <felhasználónév>`, `su` felhasználónév nélkül a root-ra vált. A „-” -t, ha használjuk (`su - user`), akkor a `su` parancs alapértelmezett környezeti változó beállításokkal vált át a másik felhasználóra.

**sudo**: rendszergazdai (root) jogokkal futtathatunk parancsokat, ha van rá engedélyünk. A jogokat a `/etc/sudoers` fájlban állíthatjuk be.

**fuser**: processzeket azonosít fájlok vagy socketek használatával. Leggyakrabban olyan processzek keresésére alkalmazzuk, amelyek portokat nyitnak, vagy eszközöket használnak. Segítségével a processzeket ki is „lőhetjük”. Pl.: `fuser -vn tcp 80`; `fuser -km /dev/hda1`; `fuser -m /dev/sda1`. Az első példánk kiírja, hogy mely processzek használják a 80-as TCP portot, a második kill-eli a `/dev/hda1`-et használó összes processzt (umount előtt nagyon barátságos használni), a harmadik csak megjeleníti a `/dev/sda1` -et használó processzek ID-jét.

**date**: a rendszeridőt kérdezhetjük le, állíthatjuk (`-s`) be ezzel a paranccsal.

**mc**, **mcedit**: Norton Commander szerű fájlkezelő, és editor.

**sed**, **awk**, **grep**: különböző sztring műveleteket hajthatunk végre velük. A későbbiekben tárgyaljuk szerepüket, fontosságukat.

**find**: fájlok keresésére szolgál. A kapcsolóival kereshetünk módosítási időre, jogokra, reguláris kifejezéssel vagy anélkül fájlnévre; a találatokon pedig parancsokat hajthatunk végre.

# 3 Shell szkriptek

Azt a programot, amely a rendszer használata során parancsainkat várja és végrehajtja, *shell*nek (parancsértelmezőnek, héjnak) nevezzük. A parancsértelmező típusa és viselkedése rendszerenként változó, illetve egy adott operációs rendszeren belül akár több fajta shellt is találhatunk, attól függően, hogy melyiket szoktuk meg, vagy melyiket szeretjük. A továbbiakban röviden bemutatunk néhány shelltípust, és a különbségek érzékeltetésére megmutatjuk, hogy az egyes típusoknál hogyan kell egy környezeti változó értékét beállítani.

Néhány shell típus a UNIX világából:

- bash (Pl.: GNU/Linux GNU/Hurd) [Mérete ≈ 600Kbyte]
- csh (Pl.: IBM AIX, IRIX) [Mérete ≈ 250Kbyte]
- ksh (Pl.: AT&T UNIX, Solaris, HP-UX) [Mérete ≈ 1100Kbyte]
- ash, dash (Pl.: Minimum shell, boot lemez esetén) [Mérete ≈ 21 – 90Kbyte]

A bejelentkezéskor alapértelmezetten elinduló parancsértelmezőt a `chsh` paranccsal tudjuk beállítani.

## *3.1 Shell-ek fajtái, kezelésük*

### 3.1.1 A bash

A Linux különböző kiadásai általában a bash nevű shellt használják alapértelmezettként. A bash különböző kényelmi szolgáltatásokat nyújt a parancsok begépelésének megkönnyítésére:

- A ↑ és ↓ billentyűkkel böngészhetjük a régebben kiadott parancsokat (history)
- A [TAB↔] megnyomásával egy parancs, fájl vagy könyvtár nevét egészíti ki (és minden mást, amit mi beállítunk neki pl. felhasználónév)
- A [CTRL-R] billentyűkombináció megnyomása után kereshetünk a régebben beírt parancsok között úgy, hogy elkezdjük újra begépelni a parancsban szereplő karakterláncot
- A "!" segítségével az utána írt karakterekkel, mint prefix-szel kezdődő utoljára beírt parancssort hajtja ismét végre – óvatosan használandó!

Például a TERM környezeti változó beállítását a következőképpen végezhetjük el:

```
bash-2.05> export TERM=vt100
bash-2.05> set |grep TERM
COLORTERM=
TERM=vt100
```

A bash támogatja az aliasok használatát. Ennek segítségével egy karakterlánchoz parancsokat és kapcsolóit rendelhetjük. A parancs kiadásakor az alias kerül elsőnek kiértékelésre. Ahogy a lenti példában is láthatjuk ily módon egy létező parancs nevéhez teljesen más funkciót rendelhetünk (3 példa):

```
bash-2.05> alias ls='/bin/ls --color'
bash-2.05> alias dir='ls -la'
bash-2.05> alias cp='logout'  # nem ajanlott!!!
```

Az alias parancs kiadásával a meglévő aliasokat listázhatjuk, `unalias` paranccsal megszüntetjük azokat az alias nevével, hivatkozva az alias-ra:

```
bash-2.05> alias
alias cp='logout'
alias dir='/ls -la'
alias ls='/bin/ls --color'
bash-2.05> unalias cp
alias dir='ls -la'
alias ls='/bin/ls --color'
```

### 3.1.2 A ksh

A ksh egy másik elterjedt shelltípus, amelyet főleg a programozók kedvelnek, mivel szkriptelési lehetőségei bővebbek, mint pl. a bash vagy a csh esetén. A környezeti változók beállítása, illetve az alias-ok létrehozása itt is hasonlóképp történik:

```
$ export TERM=vt100
$ alias dir='ls --color'
$ unalias dir
```

A ksh-nak számos kiterjesztése létezik, pl. a dtksh (desktop ksh) olyan modulokat is tartalmaz, amely a Motif grafikus programkönyvtárral együttműködve közvetlen grafikus megjelenítésre is alkalmas.

### 3.1.3 A csh

A csh szinte minden Unix-szal együtt született shell, a legrégebb óta alkalmazott, szintaktikája nagyon közel áll a C nyelvhez. A ↑ és ↓ billentyűk itt is rendelkezésre állnak, parancs-visszakeresés céljára, illetve a [TAB« ] kiegészítő funkció is működik. Környezeti változók beállítása:

```
% setenv TERM vt100
```

## 3.2 A bash shell szkriptek elemei

A Unix típusú rendszerek parancsértelmezői tudnak szöveges fájlokat feldolgozni, amelyek a parancsértelmező számára érthető, végrehajtható parancsokat tartalmaznak. A shellek számára a magic number (a fájl első 2 bájtja) és a fájl `x` (futtatható) joga jelöli ezt. Ez a fejezet a UNIX shell-ek, konkrétan a bash szkriptelési lehetőségeit mutatja be közel sem teljesen, hiszen erről külön könyvek vannak, melyek egyenkénti terjedelme is ezen jegyzet sokszorosa. Javasoljuk a `man bash` használatát, illetve kényelmi megfontolásból ajánljuk a következő linket: http://www.gnu.org/software/bash/manual/bashref.html

Mielőtt bármibe is belekezdenénk, nézzünk meg néhány szakkifejezést, melyeket a magyar és az angol szakirodalomban használnak.

| Jel | magyar név | angol név |
|---|---|---|
| {} | kapcsos zárójel | (curly) brace |
| ( ) | (íves) zárójel | parenthesis |
| ~ | tilde | tilde |
| [ ] | szögletes zárójel | (square) bracket |
| " | idézőjel, macskaköröm | double quote |
| ' | aposztróf | (single) quote |
| ` | “visszafele” aposztróf | back quote |
| / | per | slash |
| \ | vissza per | backslash |
| # | zenei kettőskereszt | hash mark |
| ^ | kalap | caret / circumflex accent |

Egy shell szkript alapvetően sorokból épül fel. Első közelítésként tekintsünk el attól, hogy egy parancs több sorba is átnyúlhat, és attól, hogy egy sorba pontosvesszővel elválasztva több parancs is írható. Most azt fogjuk megvizsgálni, hogy a parancsértelmező hogyan dolgoz fel egy parancssort. Ez az alábbi lépéseket foglalja magában az itt következő sorrendben:

- *kapcsos zárójel kifejtése* (brace expansion)
- *tilde kifejtése* (tilde expansion)
- *paraméter*, *változó* és *aritmetikai kiértékelés* és *parancshelyettesítés* – balról jobbra haladva (parameter, variable and arithmetic expansion and command substitution – done in a left-to-right fashion),

- *szavakra bontás* (word splitting)
- *elérési utak és fájlnevek kifejtése* (pathname expansion)
- *idézőjelszerű karakterek eltávolítása* (quote removal).

Mindezek után történnek meg az *átirányítások* (redirections), végül végrehajtódik a parancs.

A továbbiakban ezeket fogjuk megvizsgálni.

### 3.2.1 A kapcsos zárójel kifejtése (Brace expansion)

A kapcsos zárójel szövegrészek automatikus behelyettesítésével történő *írásrövidítés*re használható. Mindig szavakon belül működik, a szóhatárokat szóközök jelzik. Példák:

```
bash-2.05> echo E-mail: {Smith,Taylor}@ieee.org
E-mail: Smith@ieee.org Taylor@ieee.org
bash-2.05> echo E-mail:lencse@rs1.{szif,sze}.hu
E-mail:lencse@rs1.szif.hu E-mail:lencse@rs1.sze.hu
```

A kifejezések egymásba is ágyazhatók, és egymás után több is szerepelhet belőlük:

```
bash-2.05> echo lencse@{{rs1,mail}.,""}{sze,szif}.hu
lencse@rs1.sze.hu lencse@rs1.szif.hu lencse@mail.sze.hu lencse@mail.szif.hu
lencse@sze.hu lencse@szif.hu
```

Egy másik példa:

```
mkdir -p ./{bin,boot,dev,etc,home,lib,mnt,opt,proc,root,sbin,sys,tmp,usr/{bin,
include,lib,local,sbin,share,src},var/{cache,lib,lock,log,run,spool,tmp}}
```

FIGYELEM: A tördelés kedvéért egy szóközt kellett elhelyezni a parancsban. Nagyon fontos, hogy a brace-en belül a felsorolás elemei vesszővel vannak elválasztva, a vessző előtt és után szóköz nem állhat!!!

A fenti parancs gyakorlatilag egy root („/”) könyvtárstruktúrát hoz létre, ott ahol épp kiadjuk a parancsot (pl.: `/tmp`). Ahogy a jegyzet elején említettük, az mkdir parancs `-p` kapcsolóval a teljes elérési utat létrehozza, amennyiben az nem létezik.

### 3.2.2 A tilde kifejtése (Tilde expansion)

A ~ (tilde) jel speciális jelentéssel bír a UNIX shellek többségénél: ha egy felhasználónevet írunk utána, akkor az egész string egyenértékű lesz a felhasználó home könyvtárának elérési útjával. Ha a „~” mögé közvetlenül / jelet, illetve további elérési utat írunk, saját home könyvtárunkhoz képest értelmezett relatív elérési utat adunk meg:

```
bash-2.05> echo ~
/root
bash-2.05> echo ~lencse
```

```
/home/lencse
bash-2.05> echo ~/jegyzet
/root/jegyzet
bash-2.05> echo ~lencse/public_html
/home/lencse/public_html
```

További szempontok:

- egy szóban legfeljebb 1 darab szerepelhet
- az előtte álló szövegrész változatlan marad
- egymagában alkalmazva a saját home könyvtárunkat kapjuk

### 3.2.3 Paraméterek és változók kifejtése (Parameter expansion)

A bash-ban a változók ugyanúgy tetszőleges értéket tartalmazhatnak, mint az egyéb nyelvekben. Ha az értéküket szeretnénk megadni, dollárjel segítségével kell a tartalmukra hivatkozni:

```
bash-2.05> alma=apple
bash-2.05> echo alma
alma
bash-2.05> echo $alma
apple
bash-2.05>echo ”Az alma angolul: $alma”
Az alma angolul: apple
```

Mindig célszerű a változók neveit nagybetűvel írni! Ez egy olyan konvenció, ami segíti az olvasót, betartása nem kötelező (akkor is működik, ha nem tartjuk be), de a saját érdekünkben ajánlott.

### 3.2.4 Eredmények helyettesítése (Command substitution)

Néha szükséges, hogy egy parancs kimenetét felhasználjuk valamilyen célra, ilyenkor a parancs eredményét a következőképp tudjuk behelyettesíteni egy másik parancs parancssorába:

```
bash-2.05> mkdir gyak
bash-2.05> cd gyak
bash-2.05> echo egy ketto harom negy ot hat het > file_list.txt
bash-2.05> cat file_list.txt
egy ketto harom negy ot hat het
bash-2.05> touch `cat file_list.txt`
bash-2.05> ls
egy file_list.txt harom hat het ketto negy ot
```

Egy másik lehetséges megoldás (ez általában minden shell esetében működik):

```
bash-2.05> touch $(cat file_list.txt)
```

A végrehajtott parancs kimenetét akkor is visszahelyettesíti a parancssorba (egyetlen sorba írva), ha a kimenet többsoros:

```
bash-2.05> echo "1. sor" > probafile
bash-2.05> echo "2. sor" >> probafile
bash-2.05> cat probafile
1. sor
2. sor
bash-2.05> echo `cat probafile`
1. sor 2. sor
```

### 3.2.5 Matematikai kifejezések kiértékelése (Arithmetic expansion)

Ahhoz, hogy a bash a matematikai kifejezéseket kiértékelje, speciális zárójelezést kell alkalmazni: `$((kiértékelendő kifejezés))`.

```
bash-2.05> echo 2*3
2*3
bash-2.05> echo 2**3
2**3
bash-2.05> echo $((12*33)) # szorzás
396
bash-2.05> echo $((2**(2,3))) # 2^3 mivel (2,3) közül mindig az utolsó érték érvényes
8
bash-2.05> echo $((2!=(a=3)))
1
lencse@dev:~/bash$ echo $((12/10))
1
lencse@dev:~/bash$ echo $((12%10))
2
lencse@dev:~/bash$ echo $((12/10*10))
10
lencse@dev:~/bash$ echo $((12*10/10))
12
lencse@dev:~/bash$ echo $((12*10/2**3))
15
```

A bash a kiértékelt kifejezések értékét long integer típusú változókban tárolja el. A kiértékelés prioritása, szintaktikája és módja a C nyelvével azonos.

### 3.2.6 Szavakra bontás (Word splitting)

A bash fontos tulajdonsága, hogy egy bemenetre érkező több szóból álló stringet vagy adatsort szavakra bont. A szavak határát az IFS (Internal Field Separator) nevű változóban megadott karakterek alapján állapítja meg. Ezek alapesetben a szóköz, TAB és az „új sor” (\n) karakterek ( `IFS=' \t\n'` ). A szavakra bontás miatt azokat a fájlneveket, melyekben szóköz található, védő idézőjelekkel vagy backslash-sel kell ellátni.

```
bash-2.05> touch ”trukkos nev”
bash-2.05> ls –l
total 1
-rw-r--r--       1 lencse     users        7 Sep  17  10:31 trukkos nev
bash-2.05> rm trukkos nev
rm: cannot remove ’trukkos’: No such file or directory
rm: cannot remove ’nev’: No such file or directory
bash-2.05> rm trukkos\ nev
```

### 3.2.7 Elérési utak és fájlnevek kifejtése (Path name expansion)

Ha egy parancsban fájlok vagy könyvtárak egy meghatározott csoportjára szeretnénk hivatkozni, akkor az úgynevezett „joker” karaktereket kell használnunk. A bash esetén ezek a következők:

- * – helyén bármennyi (akár nulla, akár több) tetszőleges karakter állhat
- ? – helyén egy darab tetszőleges karakter állhat
- [XYZ] – helyén a felsorolt karakterek bármelyikéből egy darab állhat
- [a-g] – helyén a megadott karakter-tartományból való karakterek bármelyike, de pontosan egy darab állhat
- [^XYZ] – helyén a megadott karakterek kivételével pontosan egy karakter állhat

Példák:

```
bash-2.05> mkdir gyak
bash-2.05> cd gyak
bash-2.05> touch 11 111 121 131 141
bash-2.05> ls 1*1
11 111 121 131 141
bash-2.05> ls 1?1
111 121 131 141
bash-2.05> ls 1[12]1
111 121
bash-2.05> ls 1[1-3]1
111 121 131
bash-2.05> ls 1[^2]1
111 131 141
```

A joker karakterek esetén, a fájlnév elején álló „.” karaktert speciálisan kell kezelni, mert erre nem érvényesek a joker szabályok. Explicit illeszkedésre van szükség:

```
bash-2.05> mkdir gyak
bash-2.05> cd gyak
bash-2.05> touch .1 .11 1 11
bash-2.05> ls *1*
1 11
bash-2.05> ls ?1*
11
bash-2.05> ls .1*
.1 .11
```

Ugyanígy van az útvonalakban szereplő "/" esetén is:

```
bash-2.05> echo "three ones" >111
bash-2.05> mkdir 1
bash-2.05> echo "file 'one' in the directory 'one'">1/1
bash-2.05> ls 1?1
111
bash-2.05> ls 1/1
1/1
```

### 3.2.8 Idézőjelek kifejtése (Quote removal)

Munkáink során az egybefüggő szöveges változókat (string-eket) általában *idézőjelszerű karakterek*kel védjük. Ilyenek az aposztróf (single quote) és az idézőjel (double qoute). Az idézőjelek között álló változók kiértékelésre kerülnek, míg az aposztrófok között állóak nem!

```
bash-2.05> echo ”Ez itt a string”
Ez itt a string
```

Az idézőjelek is megvédhetők:

```
bash-2.05> echo \”Ez itt a string\”
”Ez itt a string”
```

Különbség az ' és az " között:

```
bash-2.05> export valtozo="ertek"
bash-2.05> echo "$valtozo"
ertek
bash-2.05> echo '$valtozo'
$valtozo
```

A kifejtés során a backslash, az idézőjel és az aposztróf karakterek eltávolításra kerülnek, kivétel ha valami megvédi őket, illetve természetesen ha valamilyen kiértékelés eredményeként jönnek létre. Példák:

```
bash-2.05> echo test\1 'test2' "test3"
test1 test2 test3
bash-2.05> echo test\\1 test\'2 test\"3
test\1 test'2 test"3
bash-2.05> echo "'"
'
bash-2.05> echo '"'
"
```

### 3.2.9 A standard ki és bemenetek átirányítása (Redirection)

A UNIX-ban három standard I/O csatorna létezik:

| I/O | File descriptor |
|---|---|
| Standard input (STDIN) | 0 |
| Standard output (STDOUT) | 1 |
| Standard error (STDERR) | 2 |

A standard bemenetről (STDIN) fogadják a programok a bemenő adatokat, és a standard kimenetre (STDOUT) küldik a kimenő üzeneteiket. A standard error (STDERR) a hiba kimenet, ide írják a programok a hibaüzeneteket. Alapértelmezésben a standard output és a standard error a konzolra, vagy a terminálra vannak irányítva, azonban a felhasználóknak lehetősége van átirányítani ezeket a ki és bemeneteket.

A standard kimenet átirányítása:

```
# ls –l > lista #ls -l parancs kimenetét a lista fájlba teszem
# echo elso sor > file #az „elso sor” stringet a fájlba teszem
# echo masodik sor >> file #a „masodik sor” -t hozzáfűzöm
# cat file
elso sor
masodik sor
```

A standard bemenet irányítása pl. kernel patch-elés esetén:

```
bash-2.05> patch –p1 < /usr/src/patch-2.4.22rc3.patch
```

A következő példában a `sort`, illetve a `grep` program a standard bemenetről veszi az adatot, ami viszont egy pipe (cső) kimenetéhez van csatlakoztatva:

```
bash-2.05> touch alma korte dinnye uborka
bash-2.05> ls | sort –r
uborka
korte
dinnye
alma
bash-2.05> ls | grep alma
alma
```

A standard ki- és bemeneteket, egymásba fűzhetjük, vagy fájlba tehetjük. Erre látunk

példákat a következőkben:

```
1.  bash-2.05> touch proba1 proba2 proba3
2.  bash-2.05> ls proba{1,2,3,4}
3.  ls: proba4: Nincs ilyen fájl vagy könyvtár
4.  proba1  proba2  proba3
5.  bash-2.05> ls proba{1,2,3,4} 2> /dev/null
6.  proba1  proba2  proba3
7.  bash-2.05> ls proba{1,2,3,4} > /tmp/lsp
8.  ls: proba4: Nincs ilyen fájl vagy könyvtár
9.  bash-2.05> cat /tmp/lsp
10. proba1
11. proba2
12. proba3
13. bash-2.05> ls proba{1,2,3,4} > /tmp/lsp 2>&1
14. bash-2.05> cat /tmp/lsp
15. ls: proba4: Nincs ilyen fájl vagy könyvtár
16. proba1
17. proba2
18. proba3
19. bash-2.05>
```

Az 1. sorban létrehozott fájlokkal fogunk kísérletezni. A 2. sorban kilistázzuk a `proba1`, `proba2`, `proba3`, `proba4` fájlokat, melyekből a `proba4` nem létezik, ezt egy hibaüzenetként kapjuk vissza (3. sor). Az 5. sorban a parancsot úgy adtuk ki, hogy a STDERR-t a `/dev/null`-ba irányítottuk át: `2> /dev/null`. A 7. sorban a STDOUT-ot irányítjuk a `/tmp/lsp` fájlba, és mint a 8. sorban láthatjuk, csak a hibaüzenetet kapjuk meg, hogy a `proba4` fájl nem létezik. A `/tmp/lsp` fájl pedig a listázott fájlok neveit tartalmazza. Néha szükséges, hogy a kiadott parancsunk hibaüzeneteit és a STDOUT-ot egy fájlba irányítsuk. (Ilyen eset például, amikor `strace`-el egy program futását vizsgáljuk. A `strace` ilyen esetekben minden olyan információt, amit nem a vizsgált fájl közöl, a STDERR-ba küld. Ez, ha megnézzük nagyon sok „hibát” eredményez, amit nem tudnánk nyomon követni, mert „kifut” a képernyőről.) Erre mutat példát a 13. sor. Ha a `/tmp/lsp` helyére `/dev/null`-t írunk, akkor az összes lehetséges üzenetet megsemmisítjük. (Ez például `crontab`ok használatakor fontos, ha nem akarjuk, hogy minden kis warning-ra e-mailt küldjön a rendszer.)

```
bash-2.05> cat > irok <<EOF
> Ez arra szolgál,
> hogy több sorban lehessen dolgozni.
> Így egy művelettel tudok szöveget
> tárolni, jelen esetben az irok fájlba.
> Addig van másodlagos promptom,
> amíg egy új sorba EOF -et nem írok.
> A másodlagos promptnál sorszerkesztőt használunk,
> így nincs lehetőségünk a már beírt sorokat javítani.
> EOF
bash-2.05>
```

A fenti egyszerű parancs arra szolgál, hogy az EOF kifejezésig mindent, amit írunk, bele cat-oljuk egy irok nevű állományba.

Ez a megoldás különösen hasznos szkriptek esetén, ha egy sornál hosszabb szöveget szeretnénk kiírni (például a tájékoztató üzenetet a felhasználó részére). Még arra is van

lehetőség, hogy a szöveg sorai elől a tabulátor karaktereket töröljük a "-" használatával. (Ennek értelme, hogy a szkriptben a szöveget áttekinthetőség érdekében beljebb kezdjük, de a sor elejétől szeretnénk megjeleníteni.) Példa:

```
bash-2.05> cat test
cat <<- VEGE
non indented line
    indented line
VEGE
bash-2.05> ./test
non indented line
indented line
bash-2.05>
```

Az így elhelyezett szöveget angolul „Here Document”-nek nevezik.

### 3.2.10 Parancsok végrehajtása (Command Execution)

A `bash` parancsvégrehajtásának mindenre kiterjedő leírása meglehetősen bonyolult lenne, ezért ezt most egyszerűsítjük, ám tudni kell, hogy ez így nem pontos, szükség esetén meg kell nézni a hivatalos leírást!

Tehát a helyzet igen erősen leegyszerűsítve a következő.

Amennyiben a parancssorban környezeti változónak történő értékadás van, akkor a shell az értékadást jelentő "=" jeltől jobbra levő részen végrehajtja a fent felsorolt kifejtéseket, kiértékeléseket, szavakra bontást, átirányítást, majd az eredményt értékül adja a környezeti változónak. Példa:

```
bash-2.05> almak=$(echo alma{le,bor,reszelo})
bash-2.05> echo $almak
almale almabor almareszelo
```

Amennyiben a kifejtések és a szavakra bontás után marad vissza egy vagy több szó, akkor a shell az elsőt parancsnak, a többit a parancs argumentumának tekinti. Amennyiben a parancs nem tartalmaz per ("/") jelet, akkor sorrendben először megnézi, hogy van-e ilyen nevű függvénye, ha nincs, akkor beépített parancsa, ha ez sincs, akkor pedig a PATH nevű környezeti változóban megadott könyvtárakban sorban keresi az első ilyen nevű programot. Ha nem talált ilyet, akkor hibajelzést ad. Ha megtalálta, akkor végrehajtja, átadva neki a többi szót argumentumként. Ha volt a parancsban per jel, akkor természetesen egyből az adott útvonalon elérhető programot kísérli meg végrehajtani. Példa:

```
bash-2.05> rm $(echo kerge{marha,birka})
rm: "kergemarha" nem törölhető: Nincs ilyen fájl vagy könyvtár
rm: "kergebirka" nem törölhető: Nincs ilyen fájl vagy könyvtár
bash-2.05> $(echo kerge{marha,birka})
-bash: kergemarha: command not found
```

Amennyiben a felhasználó másként nem rendelkezett, a shell megvárja a parancs befejeződését, és csak utána dolgoz fel következő parancsot. Ha nem ezt szeretnénk, akkor a

parancs után írt "&" jellel kérhetünk *aszinkron végrehajtás*t, ami interaktív módban (lásd később) azt jelenti, hogy rögtön visszakapjuk a promptot és a parancs végrehajtása a háttérben fut, szkript (fájlba írt parancsok) végrehajtása esetén pedig a parancs végrehajtásával konkurrens módon elkezdődik a következő sor feldolgozása (feltéve természetesen, hogy az adott sorban más parancs nincsen).

Fontos még tudni, hogy a programok végrehajtáskor visszatérési értéket adnak vissza. Ez a C nyelvben megszokott módon egy egész szám. Használható például logikai értékként is úgy, hogy a 0 hamis, az 1 (és esetleg minden más) igaz.

### 3.2.11 Feltételes kifejezések (Conditional Expressions)

A programozási nyelvekben megszokott feltételes és ciklusszervező utasításoknak szükségük van valamilyen logikai feltételeket kiértékelő megoldásra. Itt erre a `test` parancsot tudjuk használni. Hozzá teljesen hasonlóan működik a `[` parancs, annyi különbséggel, hogy ezt „esztétikai okokból” a neki megfelelő `]`-lel le kell zárni.

Mindkét esetben egy program futtatásáról van szó, aminek kapcsolókat (options) adhatunk meg, amelyekkel kifejezzük, hogy mit kell vizsgálnia. A vizsgálat eredményét a visszatérési érték mutatja.

A vezérlési szerkezetek megismerése nélkül, pusztán a feltételes kifejezések illusztrációjára nézzünk egy példát:

```
bash-2.05> if [ 2 -gt 1 ]; then echo nagyobb; else echo kisebb; fi
nagyobb
```

A konkrét feltételvizsgálat lehet fájlokra vonatkozó (létezik-e, olvasható-e, stb.), egész számot adó kifejezések közötti viszonyt vizsgáló (pl. kisebb, nagyobb, stb.), egyetlen karakterláncra (string) vonatkozó vagy karakterláncok közötti viszonyt vizsgáló (pl. egyenlő-e a hosszuk). Lehet továbbá logikai értékek között is műveletet végezni.

**A test parancs fájlokra alkalmazható kapcsolói:**

-r Értéke igaz, ha a fájl létezik, és olvasható
-w Értéke igaz, ha a fájl létezik, és írható
-x Értéke igaz, ha a fájl létezik, és végrehajtható
-f Értéke igaz, ha a fájl létezik, és közönséges fájl
-d Értéke igaz, ha a bejegyzés létezik és könyvtár
-c Értéke igaz, ha a bejegyzés létezik és karaktereszköz-meghajtó
-b Értéke igaz, ha a bejegyzés létezik és blokkeszköz-meghajtó
-p Értéke igaz, ha a bejegyzés létezik és nevesített csővezeték (named pipe)
-L Értéke igaz, ha a bejegyzés létezik és közvetett hivatkozás (symbolic link)
-S Értéke igaz, ha a bejegyzés létezik és egy socket
-u Értéke igaz, ha a fájl létezik, és setuid bitje be van állítva
-g Értéke igaz, ha a fájl létezik, és setgid bitje be van állítva
-k Értéke igaz, ha a fájl létezik, és sticky bitje be van állítva
-s Értéke igaz, ha a fájl létezik, és hossza nem nulla

**A test parancs egész számokra alkalmazható operátorai:**

| | | |
|---|---|---|
| n1 -eq n2 | Értéke igaz, ha n1 és n2 egyenlők | = = |
| n1 -ne n2 | Értéke igaz, ha n1 és n2 nem egyenlők | != |
| n1 -gt n2 | Értéke igaz, ha n1 nagyobb, mint n2 | > |
| n1 -ge n2 | Értéke igaz, ha n1 nagyobb vagy egyenlő, mint n2 | >= |
| n1 -lt n2 | Értéke igaz, ha n1 kisebb, mint n2 | < |
| n1 -le n2 | Értéke igaz, ha n1 kisebb vagy egyenlő, mint n2 | <= |

**A test parancs karakterláncokra alkalmazható kapcsolói:**

| | |
|---|---|
| -z C1 | Értéke igaz, ha C1 karakterlánc és hossza 0 |
| -n C1 | Értéke igaz, ha C1 karakterlánc és hossza nem 0 |
| C1 = C2 | Értéke igaz, ha C1 és C2 karakterlánc azonos |
| C1 = = C2 | Értéke igaz, ha C1 és C2 karakterlánc azonos |
| C1 != C2 | Értéke igaz, ha C1 és C2 karakterlánc nem azonos |
| C1 | Igaz értéket ad vissza, ha C1 karakterlánc hossza nem nulla |

**A test parancs logikai operátorai:**

| | |
|---|---|
| ! | Tagadás (NOT) (egytényezős) |
| -a | Logikai ÉS (AND) (kéttényezős) |
| -o | Logikai VAGY (OR) (kéttényezős) |
| (...) | Kiértékelési sorrend, a zárójelben együtt értékelődik ki az eredmény |

Használatukra példákat a vezérlési szerkezetek megismerése után mutatunk.

### 3.2.12 Vezérlési szerkezetek

A vezérlési szerkezetek közül ebben a jegyzetben részletesen kettőt mutatunk be. A `for` ciklusnak azt a fajtáját, ahol a ciklusváltozó az értékét egy halmazból veszi fel, valamint az `if` feltételes elágazást. Ezenkívül használható még a `for` ciklusnak a C nyelvben megszokott fajtája (ahol ciklus fejrészének megadására a C nyelvben használt zárójelpár helyett dupla zárójeleket kell írni, a ciklus magját pedig a `bash`-nél szokásos módon `do done` pár közé kell zárni), megismerjük még a `while` és az `until` ciklusokat is. Van még `case` és `select` is, de ezekkel már nem foglalkozunk.

A `for` ciklusnál a halmaz megadása többféle módon is történhet. A legegyszerűbb esetben akár fel is sorolhatjuk az elemeit, például:

```
bash-2.05> for i in alma korte szilva
> do
>   echo $i
> done
alma
korte
szilva
```

Létezik olyan program, amivel egyszerű módon tudunk számsort előállítani, ez a `seq`.

Példa:

```
bash-2.05> for szam in $(seq 1 5)
> do
>   echo $((szam*szam))
> done
1
4
9
16
25
```

Megjegyezzük, hogy ha a `seq` parancsnak 3 számot adunk meg, akkor a középsőt lépésköznek értelmezi.

Egy másik megoldás ciklusszervezésre:

```
lencse@dev:~$ for i in {1..3}
> do
>   echo $i
> done
1
2
3
```

Itt is van lehetőség lépésköz megadására, de ez az utolsó szám:

```
lencse@dev:~$ for i in {1..6..2}
> do
>   echo $i
> done
1
3
5
```

Megadhatjuk a halmazt továbbá fájlnévhelyettesítő joker karakterekkel is. Példa:

```
bash-2.05> for fajl in /tmp/*.jpg
> do
>   rm $fajl
> done
```

Az `if` feltételes elágazásnál a döntés alapjául felhasználhatjuk a megismert feltételes kifejezéseket. Példa: Írjuk ki az 1-10 egész számok közül azokat, amelyek 4-gyel oszthatóak:

```
bash-2.05> for szam in $(seq 1 10)
> do
>   if [ $((szam%4)) -eq 0 ]
>     then echo $szam
>   fi
> done
```

Amint láttuk, az `else` ág elhagyható.

A `for` ciklusnak a C nyelvben megszokott fajtáját az alábbi példával mutatjuk be:

```
lencse@dev:~$ for (( i=0; i<5; i++))
> do
>   echo $((i*i))
> done
0
1
4
9
16
```

Ugyanezt a feladatot a `while` ciklus segítségével a következőképpen oldhatjuk meg:

```
i=0;
while [ $i -lt 5 ]
> do
>   echo $((i*i))
>   i=$((i+1))
> done
0
1
4
9
16
```

Az elöl tesztelő `while` ciklusnak más programozási nyelvekben (pl. Pascal) létezik a hátul tesztelő párja, a `repeat until` ciklus. A `bash` esetén azonban az `until` ciklus is elöl tesztelő, pusztán annyiban tér a `while` ciklustól, hogy a megadott feltétel nem a ciklusban maradásra, hanem a kilépésre vonatkozik:

```
i=0;
lencse@dev:~$ i=0
lencse@dev:~$ until [ $i -eq 5 ]
> do
>   echo $((i*i))
>   i=$((i+1))
> done
0
1
4
9
16
```

A fenti ciklusszervezési megoldások közül válasszuk azt, ami az adott feladatra valamilyen szempont szerint a legalkalmasabb. Erre vannak nyilvánvaló szempontok, például ha halmazos megadásra van szükség/lehetőség, akkor a `for <változó> in` szerkezetet választjuk, ha számsorokat kezelünk, akkor a `for <változó> in` szerkezetet kiegészíthetjük a `seq` vagy a kapcsos zárójeles megoldással, illetve használható a `for` ciklusnak a C nyelvre hasonlító fajtája. Egyéb esetben célszerű lehet a `while` vagy az `until` ciklus. A kiválasztási kritérium lehet a program érhetősége, rövidsége is.

### 3.2.13 Egyszerű shell script példák

Mint már említettük, hogy szkripteket egy futtatható fájlba írva is megoldhatók a feladatok. Nézzünk először egy nagyon egyszerű példát, amivel egy szöveget íratunk ki.

```
bash-2.05> ./hw.sh
Hello World!
bash-2.05> cat hw.sh
#!/bin/bash
echo "Hello World!"
laptop:~/bash#
```

Az első sor a „#!” -el kezdődik, ami a Unix magic number-t állítja be: az értelmező tudni fogja, hogy a futtatható állomány egy olyan szkript (program), amit a „#!” után szereplő, teljes elérési úttal megadott értelmezővel kell végrehajtatni. Az `echo` parancsot pedig már mindenki ismeri.

A shell képes a program argumentumait kezelni, és változókba helyezni őket: `$#` jelenti az argumentumok számát, `$0` magát a program nevét, `$1` az első argumentumot, `$2` a másodikat, stb. A következő példa az argumentumok számának kiírása és ellenőrzése. Itt a könnyebb hivatkozás kedvéért a sorokat számoztuk, de ez nem része a programnak!

```
1 #!/bin/bash
2 if [ $# -lt 1 ] || [ $# -gt 3 ]; then
3     echo "vagy kevés(<1) vagy túl sok (>3) argumentumot adott meg!";
4     exit 1;
5 else
6     echo "ennyi argumentumunk van: $#";
7 fi
```

A 2. sorban az argumentumok számának vizsgálata történik. A korábbiakban már megtanultuk, hogy hogyan lehet feltételes kifejezést írni, ennek ismeretében nyilván való, hogy ha az argumentumok száma kevesebb, mint 1 (nem adunk meg egyáltalán argumentumot) vagy több, mint 3, akkor a 3-4. sor kerüljön végrehajtásra, egyébként pedig a 6. sor.

Most kitűzünk egy feladatot, aminek a megoldását is közöljük, de bátorítjuk az Olvasót, hogy először próbálja meg önállóan megoldani a feladatokat. (Nem baj, ha nem sikerül, sokszor a hibáinkból tanulunk. A megoldás puszta elolvasása sokkal kevesebbet ér, mintha valaki előbb önállóan próbálkozik!)

**Feladat:**

*Írjon bash shell scriptet, amely megszámolja, hogy a /dev könyvtárban hány blokkeszköz-meghajtó (block special device) van!*

**Egy lehetséges megoldás:**

```
#!/bin/bash
BDC=0; # BlockDeviceCount
```

```
for i in /dev/*;
do
  if [ -b $i ]; then
    BDC=$((BDC+1));
  fi
done
echo $BDC
```

Fontos, hogy ez csak minta, Unix alatt a feladatokat tipikusan sokféleképpen meg lehet oldani!

### 3.2.14 Bash specifikus fájlok

A bash-nek több üzemmódja létezik: interaktív login shell, interaktív, de nem login shell, valamint nem interaktív shell. Ha a bash egy interaktív login shell (pl. mikor belépek egy gépre), akkor végrehajtja `/etc/profile` -ban lévő parancsokat, ha létezik a fájl. Utána ebben a sorrendben próbálva az elsőt (ami létezik) `~/.bash_profile`; `~/.bash_login`; `~/.profile`. A felhasználó által kiadott parancsok bekerülnek `~/.bash_history` fájlba ezt a HISTFILE bash változó beállításával változtathatjuk meg (érdekes lenne egy ilyen parancs: `export HISTFILE=/dev/null`). Ha a bash login shell, mikor kilép, végrehajtja `~/.bash_logout` szkriptet. Interaktív, de nem login shell indítása esetén (pl.: bash-ban kiadjuk a `bash` parancsot, akkor interaktív de nem login shellt kapunk) a `/etc/bash.bashrc`, majd a `~/.bashrc` kerül kiértékelésre. Nem interaktív shell (szkriptek) a BASH_ENV-et kifejti, és azt használja fájlnévként, de nem használja a PATH-t.

### 3.2.15 A bash néhány fontosabb környezeti változója

IFS # Internal Field Separator, white space karakterek beállítása: alapértelmezetten a szóköz, a tabulátor és az újsor karaktereket tartalmazza

PATH # elérési utak beállítása: amennyiben egy parancs kiadásakor (program indításakor) nem adunk meg útvonalat, a benne szereplő könyvtárakban keresi

HOME # a felhasználó home könyvtára

MAIL/MAILCHECK # mail: a felhasználó postafiókja, mailcheck: ennyi másodpercenként ellenőrzi, van-e új levél

PS1, 2, (3, 4) # elsődleges, másodlagos, stb. promptok

HISTSIZE # hány parancsot tároljon a bash

HISTFILE, HISTFILESIZE # ebben a fájlban, ilyen méretig (ennyi sorig) tárolja a korábban kiadott parancsokat

A következő környezeti változókat a bash saját maga állítja be:

PWD, OLDPWD # az aktuális és az azt megelőző könyvtár ("`cd -`" emlékeznek?)

UID, EUID # felhasználó user ID-je

HOSTNAME # gép hostneve ( `hostname -F hostnévfájl` paranccsal módosítható)

### 3.2.16 Folyamatok kezelése (Process and Job Control)

A futó programokat *folyamat*oknak (process) nevezzük.

Egy hasonlattal élve: a program és a folyamat közötti kapcsolat olyan, mint a Kék Duna keringő kottája és annak egy előadása közötti kapcsolat. Ugyanazon kotta alapján eljátszhatják a Bécsi filharmonikusok is, meg a Mucsaj pusztai tűzoltózenekar is. Hasonlóan, a GNU C fordítóval (gcc – GNU C Compiler) lefordíthatjuk a Linux kernelt vagy egy kezdő programozó 5 soros, 3 hibát tartalmazó alkotását. A gcc program kódja azonos, a végrehajtás jellemzői (futási idő, erőforrásigény) eltérőek. Természetesen egy programot azonos paraméterekkel indítva is a program két külön példánya fog futni: az is két külön folyamat.

Az egyes folyamatok lehetnek egymástól függetlenek, de lehet közöttük többféle kapcsolat is. Itt most két fajta kapcsolattal foglalkozunk:

- Ha egy folyamat más folyamatokat indít el, akkor az elsőt *szülő* (parent) az általa elindítottakat *gyerek* (child) folyamtoknak nevezzük. A gyerekek gyerekei is a szülő *leszármazott*ai.
- Az egyazon pipeline elemeit alkotó folyamatok (és azok leszármazottai) együttesen egy *job*nak számítanak. Például: `find / -name core | sort -r`

A valós életbeli analógiának megfelelően a gyerekek <u>örökölhetnek</u> a szülőtől. A környezeti változók öröklése bash shell esetén NEM automatikus, kizárólag a szülő folyamat által *exportált* (`export <változó név>`) változók értéket öröklik a gyerek folyamatok. Vannak olyan korlátozások, amelyeket a gyerekek automatikusan örökölnek a szülőtől; például ha beállítjuk a maximálisan írható fájlméretet (lásd később: `ulimit -f`). A szülő folyamat automatikusan megkapja és felhasználhatja az általa indított gyerek folyamatok visszatérési értékét (erre már láttunk példát az `if` vezérlési szerkezetben felhasznált feltételes kifejezéseknél).

Az egy jobot alkotó folyamatok a pipe miatt bizonyos értelemben „sorsközösségben” vannak. Az természetes, hogy a „`|`” jel jobb oldalán levő „fogyasztó” folyamat csak olyan inputot tud felhasználni, amit a „`|`” jel bal oldalán levő „termelő” folyamat már kimenetként létrehozott. Ám a kettő közötti puffer méretét sem célszerű határtalanul növelni (megengedett maximális értéke beállítható: `ulimit -p`), ezért szükség esetén a termelő folyamat futását a rendszer felfüggeszti. Amennyiben az egy jobot alkotó programok közül valamelyiknek a futása hiba miatt megszakad, akkor „broken pipe” hibaüzenetet kapunk, és a többi program is befejeződik.

A folyamatok kezelhetők szignálok küldésével (a már korábban megismert `kill` parancs segítségével), de a bash shell is nyújt egy eszközkészletet a jobok kezeléséhez. Ha egy folyamatot aszinkron módon indítunk, akkor az alábbihoz hasonlóan a shell egy sort ír ki:

```
bash-2.05> ls &
[1] 26629
```

Ennek az értelmezése a következő: a szögletes zárjelben levő szám a job sorszáma (job number) a másik szám pedig a jobot alkotó pipeline utolsó (és jelen esetben egyetlen) folyamatának process ID-je. A továbbiakban a jobra a %1 *job specifikáció*val (jobspec) tudunk hivatkozni. A job specifikációval történő hivatkozást (az általunk használt Linux disztribúcióban) a `kill` parancs is elfogadja, de léteznek olyan job kezelő parancsok, amelyek kifejezetten ezt igénylik.

Hogyan tudja a kill parancs, hogy egy szám job number vagy process ID? A válasz nagyon egyszerű: ha előtte % jel van, akkor job number, ha nincs, akkor process ID. A % jel és a job sorszám együtt alkotja a job specifikációt.

Tisztáznunk kell még az előtérben és a háttérben futó programok közötti különbséget. Ha egy programot parancssorból az & jel használta nélkül elindítunk, akkor az *előtérben* fut, a terminálról bemenetet fogad, és oda kimenetet küld. Amennyiben az & jelet használjuk, akkor a program ún. aszikron módban fut, a terminálról nem fogad inputot; amit gépelünk, azt ismét a shell fogja feldolgozni. Ekkor azt mondjuk, hogy a program a *háttérben* fut. Az előtérben futó programokat a billentyűzetről nagyon egyszerűen vezérelhetjük a Ctrl és még valamelyik billentyű együttes lenyomásával. Néhány példa:

- Ctrl-Z: felfüggesztés (suspend) – a program futása felfüggesztődik, tipikus folytatás a `bg` vagy az `fg` paranccsal (az elsővel előtérben, a másodikkal háttérben fog futni).
- Ctrl-C: a program futásának megszakítása – Ez a program futásának befejezését jelenti, általában a SIGINT szignál küldésével ekvivalens.
- Ctrl-S: a terminál befagyasztása – Megállítja a terminálon a program kimenetének megjelenítését. Lásd még Ctrl-Q
- Ctrl-Q: befagyasztott terminál felélesztése – A Ctrl-S-sel megállított kimenet megjelenítésének folytatása.

További parancsok elérhetők például:

http://web.cecs.pdx.edu/~rootd/catdoc/guide/TheGuide_38.html

Már említettük a `bg` és az `fg` parancsokat. Ezek argumentum nélkül kiadva az aktuális jobra (current job) vonatkoznak (amit utoljára indítottunk el `&` jellel háttérben vagy utoljára függesztettünk fel Ctrl-Z-vel). Megadható nekik job specifikáció a már megismert módon %*n* alakban, ahol *n* a job sorszáma, illetve a következő rövidítések is használhatók: `%%`, `%+`, sőt a `%` jel önmagában az aktuális jobot jelöli, `%-` pedig az előzőt. (Ha csak 1 job van, akkor `%-` is az aktuálisat jelenti.)

A `jobs` paranccsal tudjuk megjeleníteni a futó jobokat. Lássunk egy példát:

```
bash-2.05> dd if=/dev/urandom of=/dev/null &
[1] 29625
bash-2.05> find / -name nevesincs 2>/dev/null &
[2] 29626
bash-2.05> jobs
[1]-  Running                 dd if=/dev/urandom of=/dev/null &
[2]+  Exit 1                  find / -name nevesincs 2>/dev/null
bash-2.05> kill %1
bash-2.05>
[1]+  Félbeszakítva           dd if=/dev/urandom of=/dev/null
```

Az első parancs a véletlenszám-generátorból másol a végtelen kapacitású nyelőbe, a második pedig az egész fájlrendszerben keres egy fájlt. (Az elsőt feltétlenül szükséges kilőni, ha nem szeretnénk a gépünk erőforrásait a következő újraindításig pazarolni.)

A `jobs` parancs kimenetében a 2-es sorszámú job az aktuális (+) és az egyes sorszámú az előző (-). Amint látjuk, a 2-es sorszámú már be is fejeződött, csak késleltetve került kiírásra, ahogyan az 1-es sorszámú kilövése után is csak plusz egy Enter hatására jelent meg a „Félbeszakítva” üzenet. Ennek oka is a háttérben való futás.

### 3.2.17 A prompt vezérlése

A parancsértelmező a prompt megjelenítésével jelzi, hogy készen áll a parancsaink fogadására. A jegyzet példáiban egységesség céljából egy nagyon egyszerű promptot szoktunk használni. Ennél sokkal informatívabb a Debian rendszerben alapértelmezésben használt elsődleges prompt. A `$` jel előtt a `user@host:dir` formátumot használja, ami könnyen érthető és megfelel az `rcp`, `scp` parancsok formátumának is:

```
lencse@dev:~$
```

Ennek előállításához a PS1 változó értékét így állíthatjuk be:

```
PS1='\u@\h:\w\$'
```

Nem nehéz felismerni, hogy \u, \h és \w rendre a felhasználói nevet (username), a gép nevét (host) és az aktuális könyvtárat (working directory) jelölik. Egyszerű felhasználó esetén szokásos prompt jel a „$”, rendszergazda esetén pedig a „#”.

A másodlagos prompt beállítására egy példa:

```
lencse@dev:~$ PS2='folytasd:'
lencse@dev:~$ echo "uj sorban
folytasd:irom tovabb"
uj sorban
irom tovabb
lencse@dev:~$
```

## *3.3 Reguláris kifejezések*

A r*eguláris kifejezés*eket más néven *szabályos kifejezés*eket (regular expression, rövidítve regexp vagy csak regex) sok Unix segédprogramban használják. Sajnálatos módon az egyes programok között van némi eltérés, ezért mi most a POSIX szabványban megadott két fajtát fogjuk megtanulni. A Basic Regular Expression (BRE) szabvány a régi programokkal

(lehetőség szerint) kompatibilis, de egységes szabvány, az Extended Regular Expression (ERE) pedig az új lehetőségeket is nyújtja (bár egyetlen lehetőségben szűkebb, lásd később). A BRE és ERE szabványok meglehetősen hasonlóak, a fő különbség az, hogy mikor kell backslash-t használnunk, illetve az ERE lehetőségeinek bővebb volta.

A reguláris kifejezésekkel tulajdonképpen karakterláncokat (string) helyettesíthetünk egy, csak a keresett csoportra jellemző „mintával". Azt mondjuk, hogy ez a minta (a regex) *illeszkedik* bizonyos karakterláncokra és nem illeszkedik más karakterláncokra.

A reguláris kifejezéseknél vannak ún. *metakarakter*ek (metacharacter), azaz *speciális jelentéssel bíró karakterek* amelyek itt sajnálatos módon egészen mást jelentenek, mint például a fájlnévhelyettesítésnél! Ügyeljünk arra, hogy a kettőt ne keverjük össze!!!

A metakarakterek között is különleges a backslash („\"), amely általában arra használatos, hogy a metakaraktereket „megvédje" a speciális jelentéstől, és így magát az illető karaktert jelentse.

Ez így is van az ERE szintaxis esetén. Sajnos a BRE-nél néhány esetben éppen fordítva van, azaz a speciális jelentés eléréséhez van szükség a backslashre. Éppen ezért, e sorok írója (Lencse Gábor), oktatásra sokkal alkalmasabbnak tartja az ERE szintaxist. Ezért a BRE csak némely régi programmal való kompatibilitás fenntartására szolgáló eltérő lehetőségként szerepel. Azonban sajnos a POSIX szabvány elég régi, nem tartalmazza a regex témában azóta elért fejlesztéseket. A tárgyba most ennyi fér bele, de fontos tudni, hogy az egyes nyelvekben (pl. perl) használt reguláris kifejezések ezenkívül még sok lehetőséggel rendelkeznek. Érdeklődőknek javasoljuk a következő oldalt:

http://www.regular-expressions.info/posix.html

Először megismerünk néhány metakaraktert és a jelentésüket (más néven azt, hogy „mire/mikor illeszkedik"):

| metakarakter | jelentése |
|---|---|
| . | bármelyik karakter (de csupán 1 darab) |
| [...] | A bracketben felsorolt karakterek közül bármelyik (de csak 1db). Lehetőség van intervallum megadására is, például [a-c] azt jelenti, hogy az „a", „b", „c" karakterek közül pontosan egy, sőt vegyesen is használható. Például [a-dfhl-n] ekvivalens azzal, hogy [abcdfhlmn] Ha magát a „-" karaktert szeretnénk megadni, akkor vagy az elejére vagy a végére kell tennünk, ha a „]" karaktert, akkor az elejére, ha pedig a „^" karaktert, akkor nem az elejére (lásd lent, hogy miért). |
| [^...] | Tetszőleges karakter a bracketben felsoroltak kivételével. Itt is használhatunk intervallumot is. |
| ^ | „sor eleje" – A mögötte álló kifejezés akkor illeszkedik, ha az közvetlenül a sor elején áll. |
| $ | „sor vége" – Az előtte álló kifejezés akkor illeszkedik, ha az közvetlenül a sor végén áll. |

A fentiekből építkezhetünk egymás után írással, illetve megadhatunk különféle számosságokat is. Az alábbi számosságot jelentő metakarakterek mindig az őket közvetlenül

megelőző karakterre vonatkoznak (kivéve, ha zárójelet használunk – lásd később), azaz a számosság megadásának nagyobb a prioritása, mint az egymás után írásnak.

| metakarakter/jelölés | jelentése |
|---|---|
| * | bármennyi (0 vagy több) az előtte álló kifejezésből (mohó, azaz: mindig a lehető leghosszabbra illeszkedik) |
| + | 1 vagy több az előtte álló kifejezésből (szintén mohó) |
| ? | 0 vagy 1 az előtte álló kifejezésből |
| {n} | pontosan n (egész szám) darab az előtte álló kifejezésből |
| {n,} | legalább n (egész szám) darab az előtte álló kifejezésből |
| {n,m} | legalább n (egész szám), maximum m (egész szám) darab az előtte álló kifejezésből |

Van még két további speciális karakter:

| metakarakter/jelölés | jelentése |
|---|---|
| \| | választás (alternation) – Az előtte és az utána álló kifejezés bármelyike. Ennek a legkisebb a prioritása az egymásután íráshoz, illetve a számosság megadásához képest. |
| ( ) | Zárójellel (a prioritásból adódóhoz képest máshogyan) csoportosíthatjuk a kifejezés elemeit. |

A fentiekhez képest a BRE esetén teljesen hiányoznak a hozzájuk tartozó funkciókkal együtt: „|”, “+” és „?”. A „^” és „$” karakterek közönséges karakternek számítanak, ha nem a speciális jelentésüknek megfelelő pozícióban vannak. Sőt, még a „*” is közönséges karakter, ha olyan helyen van, ahol nem alkalmas számosság kifejezésére. A ( ) és { } zárójeleket metakarakter funkcióban kell backslash-sel védeni, anélkül közönséges karakternek számítanak!

És a BRE-knél van egy további funkció: a visszahivatkozás (back reference): egy *backslash*t követő decimális számjegy pontosan arra illeszkedik, amire a kifejezésben található a számjegynek megfelelő sorszámú (a kezdő zárójelek szerint számozva) íves zárójelpárban található kifejezésrész illeszkedett. Ennek a használata erősen ellenjavallt több okból is: egyrészt, mivel az implementációjával algoritmikus hatékonysági problémák vannak, másrészt ezzel a BRE-k kifejező ereje eltér a formális nyelvekben használt *reguláris nyelvosztály* kifejezőerejétől (aminek az ERE kifejezőereje megfelel).

Vannak még előre definiált karakterosztályok, amelyeket kényelmi szempontból érdemes lehet használni. Ezeket a „[” és „]” jelek közé kell beírni, és akár a „kézzel” történő megadással vegyesen is használhatjuk, például a 20-as számrendszer jegyei megadhatók így is: [[:xdigit:]g-jG-J]. A POSIX szabvány a következőket tartalmazza:

**[:alnum:], [:alpha:], [:blank:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], [:xdigit:]**

Most nézzünk néhány példát reguláris kifejezések használatára! A továbbiakban, ha másképpen nem jelezzük, akkor reguláris kifejezés (RE) alatt mindig ERE-t értünk!

*1. példa: Írjunk olyan RE-t, ami hatásfokot kifejező számértékre illeszkedik!*

Megoldás: Diszkutáljuk először a feladatot! A hatásfokot kifejező számérték alatt értsük a [0,1] intervallum elemeit. Engedjük meg tehát a 0-t, az 1-et, valamint – első közelítésként – a „0,” kezdetű tetszőleges hosszúságú számjegysorozatot:

```
0|1|0,[0-9]*
```

A figyelmes olvasó hamar észreveheti, hogy például a „**0,**” nem túl jó megoldás. Első ötletként követeljünk meg legalább egy tizedesjegyet, vagyis legyen:

```
0|1|0,[0-9]+
```

Ez már viszonylag jó, de ha nem szeretnénk megengedni, hogy az utolsó tizedesjegye 0 legyen, akkor még alakíthatunk rajta:

```
0|1|0,[0-9]*[1-9]
```

Ha tesztelni szeretnénk a munkánkat, akkor először is készítsünk egy tesztfájlt, amiben vannak az általunk jónak és az általunk rossznak tartott szám típusokból is példák. Legyen például a tesztfájl a következő:

```
lencse@dev:~$ cat hatasfok
0
1
5
0,
0,0
0,1
0,010
,012
0,32
0,012
0,123000kutya
ez1rossz
```

A tesztelésnél az ERE szintaxis érdekében használjuk a **grep -E** vagy az **egrep** parancsot, és mindenképpen védjük meg a regexp-et a shell kiértékelésétől! Első látásra jónak tűnik a következő parancs:

```
egrep '0|1|0,[0-9]*[1-9]' hatasfok
```

Az eredmény mégsem az, amit szeretnénk! Amint ugyanis nemsokára megtanuljuk, a **grep** minden olyan sort kiír, amiben van a mintára illeszkedő rész! Tehát adjuk meg azt, hogy a sorban más ne is lehessen! Ezt – kellő körültekintés hiányában – egyszerűen (de hibásan) megvalósíthatjuk a következőképpen:

```
egrep '^0|1|0,[0-9]*[1-9]$' hatasfok
```

Mi a hiba a fenti parancsban? Az, hogy a választást jelölő "|" jel prioritása kisebb, mint az egymás után írásé, így ha például egy 1-es van a sorban, akkor azzal szemben semmiféle követelményt nem támasztunk arra nézve, hogy előtte vagy utána mi lehet vagy nem lehet még! Javítsuk ki zárójelezéssel és futtassuk le a parancsot!

```
lencse@dev:~$ egrep '^(0|1|0,[0-9]*[1-9])$' hatasfok
0
1
0,1
0,32
0,012
```

Ennyi gyakorlás után javasoljuk, hogy az Olvasó önállóan diszkutálja és oldja meg a következő példát és csak utána vesse össze a saját eredményeit az itt közölttel!

*2. példa: Írjunk reguláris kifejezést, ami lehetséges feszültség értékekre illeszkedik!*

Diszkusszió: A feszültség érték egy valós szám, a feszültség mértékegysége a Volt, jele a V. A feszültség értékét jelző valós szám lehet előjel nélküli, de mindenképpen lehet negatív is. Döntés: fogadjuk el azt is, ha a pozitív voltát jelzik! A számérték előtt (vele közvetlenül egybeírva) tehát opcionálisan szerepelhet egy „+" vagy egy „–" jel. A szám kezdődjön egy vagy több számjeggyel, opcionálisan folytatódjon tizedesvesszővel, de amennyiben tizedesvessző van, akkor azt mindenképpen kövesse legalább egy nem 0 értékű számjegy. Végül az egész záruljon egy „V"-vel. (Nyomdailag igényes könyvekben a számérték és a mértékegysége között van egy a normál szóköznél keskenyebb térköz, de most ne legyen!)

A RE tehát:

```
[+-]?[0-9]+(,[0-9]*[1-9])?V
```

A tesztelés pedig:

```
lencse@dev:~$ cat feszultseg
1
V
-10V
0,31V
+3V
0,0V
lencse@dev:~$ egrep '^[+-]?[0-9]+(,[0-9]*[1-9])?V$' feszultseg
-10V
0,31V
+3V
```

Kérdés: most miért nem volt szükséges zárójelbe tenni a „^” és a „$” jel közötti részt? Válasz: mert nem használtunk „|” jelet.

*3. példa: Írjunk reguláris kifejezést, ami egy szövegnek azokra a soraira illeszkedik, amelyek pontosan egy mondatból állnak!*

Diszkusszió: A feladat nem említ nyelvet, így tetszőleges nyelvbeli lehet a mondat! :-) Egy ilyen feladat esetén nyilván valóan nem várható el komoly szintaktikai még inkább nem szemantikai elemzés, tehát keressük meg egy mondat nyilvánvaló jellemzőit! Egy mondat nagy betűvel kezdődik, mondatzáró írásjellel végződik és benne nem található mondatzáró írásjel. (Nagybetű, szám, egyéb írásjel lehet.) Az egyszerűség kedvéért a mondat belső részéből csak a mondatzáró írásjeleket zárjuk ki, minden mást engedjünk meg!

A diszkussziónak megfelelő RE:

```
^[A-Z][^\.\?!]*[\.\?!]$
```

Megjegyzés: a „!”-et nem kell védeni, mert nem metakarakter.

Egy egyszerű példa a tesztelésre:

```
lencse@dev:~$ cat mondat
A..
B.
C.?
Dani!
ember.
F!!
lencse@dev:~$ egrep '^[A-Z][^\.\?!]*[\.\?!]$' mondat
B.
Dani!
```

Az ebben a fejezetben szereplő parancsok, és a reguláris kifejezések a Unix világában nélkülözhetetlenek lettek. Az ember hamar rájön, hogy rengeteg munkát spórolhat meg egy jól megírt szkript segítségével. Ezért kérünk mindenkit, hogy ne a fenti példákat „magolják” be! Keressenek feladatokat, kihívásokat, amelyek segítségével sokkal mélyebben elmerülhetnek a reguláris kifejezések világában! Néhány ötlet: készítünk reguláris kifejezést, amelyik telefonszámokra, e-mail címekre, MAC címekre, egyszerűsítés nélküli, illetve egyszerűsített IPv6 címekre, IPv4 címekre, stb. illeszkedik!

A reguláris kifejezések nélkülözhetetlenségét bizonyítja az is, hogy nem csak a shell szkriptek, hanem a honlapokon lévő űrlapok feldolgozásának is elengedhetetlen kellékei. Gondoljunk csak bele, hogyan tudnánk ellenőrizni egy e-mail cím vagy telefonszám helyességét? Vagy az űrlapba beírt HTML és egyéb források kiszűrését (amivel pl. kártékony kódokat futtathatnának a gépeinken) mivel valósítanánk meg, ha nem lenne kezünkben ez az eszköz?

## *3.4 Gyakran használt segédprogramok*

A Unix rendszerek alapvető tulajdonsága, hogy az egyes feladatok megoldását igen gyakran több egyszerű program együttműködésével érjük el. A bash shell szkriptek fent megismert alapelemein túl van néhány olyan program, amely igen hasznos lehet a mindennapi rendszergazdai feladatok ellátásában. Ezek is olyan építőelemek, amelyek igen gyakran szerepelnek szkriptekben.

### 3.4.1 SED

A sed (stream editor) egy nagyon jól paraméterezhető szövegszerkesztő. A STDIN-ről a STDOUT-ra dolgozik, ezzel kissé elüt a szerkesztő programok nagy részétől (mcedit, vim). (Az ilyen programokat *szűrő*nek is nevezik.) Tökéletes választás, ha automatizálni akarunk különféle szerkesztéseket (pl.: logfájl kimenetének formázása). Néhány példán mutatjuk be a működését.

1. példa: `sed 3,6d` 3-ik sortól a hatodik sorig töröljük a STDIN-ről érkező adatokat.

```
laptop:~/sed# cat sorok.txt
1. sor
2. sor
3. sor
4. sor
5. sor
6. sor
7. sor
laptop:~/sed# cat sorok.txt | sed 3,6d
1. sor
2. sor
7. sor
laptop:~/sed#
```

2. példa: `sed 3d` a 3. sort törli csak. A feltétel negálható is: a `sed '3!d'` parancs a 3. sor kivételével a többit törli ki.

```
laptop:~/sed# cat sorok.txt | sed '3!d'
3. sor
laptop:~/sed#
```

3. példa: `sed 's/mit/mire/'` szöveg helyettesítése

```
laptop:~/sed# cat sorok.txt | sed 's/4. sor/ez volt a 4. sor/'
1. sor
2. sor
3. sor
ez volt a 4. sor
5. sor
6. sor
7. sor
laptop:~/sed#
```

A mit helyén nemcsak szöveg állhat, hanem reguláris kifejezés is, erre később mutatunk példát!

4. példa: a sorban található összes illeszkedés cseréje a „g” flag hatására (anélkül csak az elsőt cseréli)

```
laptop:~/sed# echo "kutya kutya kutya kutya kutya"
kutya kutya kutya kutya kutya
laptop:~/sed# echo "kutya kutya kutya kutya kutya" | sed 's/kutya/macska/'
macska kutya kutya kutya kutya
laptop:~/sed# echo "kutya kutya kutya kutya kutya" | sed 's/kutya/macska/g'
macska macska macska macska macska
laptop:~/sed#
```

Alapértelmezés szerint a sed BRE-t használ, a -r opcióval vehetjük rá ERE használatára!

5. példa: reguláris kifejezés használatával hajtunk végre cserét

```
lencse@dev:~/sed$ cat allatok
kutyakutyakutyakutyakutya
tigris
kutya
lencse@dev:~/sed$ cat allatok | sed -r 's/(kutya)+/macska/'
macska
tigris
macska
```

Mint láthatjuk, a sed egy igen hasznos program. Ez a néhány példa közel sem teljesen mutatja be képességeit.

Megjegyzés: a fent bemutatott sortörlés, szöveghelyettesítés teljesen hasonlóan működik például a vi (és változatai, pl.: vim, elvis) parancs módjában is.

### 3.4.2 Awk

Az awk szintén egy szövegfeldolgozó, ami soronként dolgozza fel a STDIN-ről érkezett adatokat és ezt a STDOUT-ra küldi.

Használható egy szkript értelmezőjeként a szkript első sorában megadva, illetve parancssorból úgy, hogy feladatát meghatározó szkriptet a parancssorba írjuk be (aposztrófokkal védve a bash kiértékelésétől). Megismerését az előbbi módon kezdjük.

Egy **awk** szkript a következő módon épülhet fel:

```
#!/usr/bin/awk -f
BEGIN{
utasítás1;
utasítás2;
utasításN;
```

```
}
/regexp amire illeszkedik a sor/ {
utasítás1;
utasítás2;
utasításN;
}
/regexp amire illeszkedik a sor/ {
utasítás1;
utasítás2;
utasításN;
}
END{
utasítás1;
utasítás2;
utasításN;
}'
```

Itt az első sorban a „**#!**” magic number azt jelenti, hogy az utána következő programmal kell a szkript további részét végrehajtatni. Természetesen az **awk** elérési útja ettől eltérő is lehet. (GNU rendszerekben a program neve **gawk**, de jó esetben van rá softlink **awk** néven.) A "**-f**" már az **awk**nak szól, azt jelenti, hogy a programot nem a standard inputon kapja, hanem a fájlból kell vennie.

A fenti példában láthatjuk, hogy van egy **BEGIN** és egy **END** blokk. A **BEGIN** blokk a bemenet feldolgozása előtt végrehajtódik, szükség esetén ide célszerű a változók kezdőérték-adását tenni – a változókat deklarálni nem kell: első használatkor automatikusan létrejönnek: a numerikus változók kezdőértéke 0, a szöveges változóké az üres string. Az **END** blokkban pedig olyan műveleteket hajthatunk végre, melyek a bemenet feldolgozása után adnak eredményt.

Mint említettük, az **awk** sorfeldolgozó. Az egész sorra, amire a per jelek közötti regexp illeszkedik (megjegyzés: ha NINCS megadva ilyen feltétel, az minden sorra illeszkedik) végrehajtja a feltétel utáni megadott blokk utasításait. Az utasításokban az egész sorra a **$0** változóval, a sor első mezőjére a **$1**, a másodikra **$2** stb. módon hivatkozhatunk. A mezőkre bontást az **FS** (field separator) előre definiált változó értéke alapján végzi. Alapértelmezésben a szóköz és a tabulátor karakterek mentén tördel, de ezt értékasással át is állíthatjuk (tipikusan a **BEGIN** blokkban). Az adott sorban levő mezők számát a **$NF** adja meg. A program szintaktikája a C nyelvhez erősen hasonló (de lazább, például a sor végi pontosvesszők elhagyhatók), használhatjuk annak vezérlési szerkezeteit, kifejezéseit is.

Bevezetésként tekintsük **/etc/passwd** fájl sorait feldolgozó alábbi egyszerű programot:

```
#!/usr/bin/awk -f
BEGIN {
  FS=":";
}
{
  if ($3 >= 1000 && $3 <= 65000)
    print $1
}
```

A **BEGIN** blokkban beállítottuk a kettőspont mezőelválasztó karaktert. A főprogram

egyetlen blokkból áll, amihez nem tartozik reguláris kifejezés, tehát az input fájl minden egyes során le fog futni. A benne használt **if ( )** szintaktikája megegyezik a C szintaktikájával (használható benne: **==**, **<**, **<=**, **>=**, **>**, **&&**, **||**, stb). A **print** parancs pedig – akárcsak sok programozási nyelvben – a mögötte álló kifejezéseket írja ki. A **print** parancs szintaktikája a következő stílusú: **print $1 "szöveg1" $2 "szöveg2"** stb., A kapott sztringeket és változókat közvetlenül egymás mögé írja ki (tehát két vagy több változó kiíratása esetén nekünk kell gondoskodni az elválasztó karakterekről pl.: tabulátor: "**\t**"; szóköz: " " stb.), a kiírás végén automatikusan soremelést használ. (Használható egyébként a C nyelv **printf** függvényének megfelelő szintaxis is, az természetesen az első argumentumaként megadható formátum stringgel vezérelhető, és ott nincs automatikus soremelés.)

Ezek után nézzük, mit csinál a fenti kis rövid szkript! Ha feltételben a **/etc/passwd** fájl aktuális sorában a UID-t (felhasználói azonosító) tartalmazó harmadik mező, azaz **$3** értéke nagyobb vagy egyenlő, mint 1000 és **$3** értéke kisebb vagy egyenlő, mint 65000, tehát közönséges felhasználóról van szó, akkor kiírja a **$1**-et, vagyis a **$3** az UID-hez tartalmazó felhasználó nevét.

Ha a fenti programot **users** néven mentettük el az aktuális könyvtárban, akkor a megfelelő jogosultság beállítása után a következő paranccsal futtathatjuk le a **/etc/passwd** fájl soraira:

```
cat /etc/passwd | ./users
```

Egészítsük kis a fenti programot úgy, hogy írja ki a **min** és **max** változókkal megadott UID értékek közé eső felhasználók számát (a határokat is beleértve) a program futása végén:

```
#!/usr/bin/awk -f
BEGIN {
  FS=":";
  min=1000;
  max=65000;
  user_count=0; # Ez elhagyhato!
}
{
  if ($3 >= 1000 && $3 <= 65000) {
    print $1
    user_count++;
  }
}
END{
  print user_count
}
```

Vegyük észre, hogy az **if ( )** szerkezetben blokk használatára volt szükség! Mit írna ki a program, ha erről elfeledkeztünk volna? (Súgás: hány utasítást hajtana végre feltételesen? Mit tartalmazna tehát a **user_count** változó a program lefutásának végén?)

Az **awk** programoknál tipikus, hogy a standard inputról a standard outputra dolgoznak. De

ez nem szükségszerű. Most mutatunk egy példát arra, hogyan lehet az inputot egy fájlból venni. Feladat: írjuk ki azon felhasználók számát, akiknek az userID-je legalább 1000.

```
#!/usr/bin/awk -f
BEGIN {
  FS=":";
  while ( getline < "/etc/passwd" )
    if ( $3 >= 1000 )
      user_count++;
  print user_count;
}
```

A fenti program az inputot a **/etc/passwd** fájlból veszi a **getline** függvény segítségével. A függvény visszatérési értéke akkor lesz 0, ha már nem volt több sor a fájlban. Mivel az inputot nem a standard inputról vettük, ezért a **BEGIN** blokkban dolgoztunk.

Írjunk olyan programot, ami a Unix **wc** parancsának funkcióját látja el, azaz a standard outputra kiírja a standard inputon kapott fájlban levő sorok, szavak és karakterek számát!

```
#!/usr/bin/awk -f
{
  sor++;
  szo+=NF;
  kar+=length($0)+1;
}
END {
print sor " " szo " " kar;
}
```

A program törzse minden sorra lefut, a sorokat így nagyon könnyű megszámolni. Az **NF** (Number of Fields) változó tartalmazza a sorban levő mezők számát – az **FS** alapértelmezett értéke (whitespace) szerint szavakra tördelünk, az adott sorban levő karakterek száma pedig azért 1-el nagyobb, mint a sort tartalmazó $0 hossza, mert a fájlban a sorvég jelet is tároljuk viszont a **$0**-ban már nincs benne. Figyeljük meg, hogy a kiírás az **END** blokkban van, ami az összes sor feldolgozása után fut le, és a számértékek közé szóközt teszünk!

Egyszerű feladatok megoldására parancssorból is kiválóan használható az **awk**. Ennek előnye, hogy nem kell szövegszerkesztővel a programot megírni, futtathatóvá tenni, hanem egyből használhatjuk. Ha például arra vagyunk kíváncsiak, hogy mely könyvtárak foglalnak el egy bizonyos méretnél nagyobb helyet, akkor erre kiváló az alábbi parancssor:

```
du | awk '{ if ( $1 > 1024 ) print $2}'
```

A legelső példaprogramunk is elég rövid ahhoz, hogy parancssorból kényelmesen használhassuk. Egyben megmutatunk egy másik lehetőséget is a mező elválasztó

beállítására.

```
cat /etc/passwd | awk -F \: '{if ($3 >= 1000 && $3 <= 65000) print $1}'
```

Az **awk** parancs után álló **-F \:** a mező elválasztó beállítása, itt a "**:**"-ot meg kellett „védeni” a bash kiértékelésétől egy "****" (backslash) jellel.

Az **awk** a kapott bemenet sorain nagyon sok műveletet képes elvégezni, többek között például külső parancsok meghívására is képes a **system("command line")** függvény segítségével. Ez nagyon hasznos például, ha a feldolgozandó fájlban szimbolikus nevek szerepelnek, amiket a **host** paranccsal fel tudunk oldani, majd a kapott IP címeket felhasználja a szkriptünk.

### 3.4.3 Find

A **find** parancs a fájlrendszerben hajt végre keresést. Segítségével listázhatunk különböző mintáknak megfelelő fájlokat, legyen az a fájl neve, egy *regexp*re való illeszkedése vagy a fájl jogainak, tulajdonosainak, csoportjának megadásával történő keresési feltétel. A **find** működését néhány példán mutatjuk be.

Első példánkban, a **core** nevű fájlokat keressük meg. Ezek a régebbi (és ha be van kapcsolva, akkor az újabb) rendszerek esetében a futás közben az operációs rendszer által *segmentation fault* hibával leállított programok memóriában lévő „képének” a fájlrendszerre írt – későbbi hibakeresés céljára szolgáló – fájlok.

```
# find / -name core
```

Ha a fenti parancs által megtalált fájlokon műveletet szeretnénk végrehajtani, arra is van lehetőségünk a find parancson belül a következő módon:

```
# find / -name core -exec rm -rf {} \;
```

A fenti utasításban a **{}** a fájlok (útvonalának és nevének) „behelyettesítése”, és mint látjuk a talált core fájlokat törölni szeretnénk. A „**\;**” a talált fájlra meghívott utasítás végét jelenti a **find** számára, de ha nem teszünk elé „****” (backslash) jelet akkor a shell kiértékeli, mint a **find** utasítás végét jelző karaktert.

A következő példánk a fájl jogait vizsgálja, egy esetleges *buffer overflow* kihasználáshoz szükséges setuid bitek megkeresésével:

```
# find / -perm /4000
```

A régebbi **find** esetében a „**/4000**”-t a „**+4000**” helyettesítette (sarge pl. a régit használja az etch az új jelölésmódot. A **-perm** után adjuk meg a keresett jogosultságot, amennyiben nem teszünk a jogok elé semmilyen jelet, úgy a pontos illeszkedést keressük. A mi példánkban a nem nulla értékű számokat veszi figyelembe a **find**. A jogok, hogy mindenki számára világosak legyenek a következőt jelentik: első szám a setuid, setgid, sticky bit beállításáért felel (ezek közül azt nézzük, hogy a setuid be van-e kapcsolva), a második a tulajdonos jogai, a harmadik a csoport, a negyedik a többi felhasználót jelenti.

Nézzük, hogyan viselkedik a következő parancsunk:

```
# find /usr/bin -name pass*
```

Sokan azt gondolnák, hogy ez a parancs megkeresi az összes **pass** kezdetű fájlt. Természetesen nem ezt teszi, hanem mivel a „*” a shell esetében is értelmes, kiértékelésre kerül. Ha a fenti parancsot nem a /usr/bin, hanem egy olyan könyvtárban állva adtuk ki, ahol nincs **pass** kezdetű fájl, akkor a parancs nem csinál semmi szokatlant. Ha könyvtár, ahol a parancsot kiadtuk tartalmaz például egy **passz** nevű fájlt akkor a **find** megkeresi az **/usr/bin** könyvtárban a **passz** nevű fájlt, hiszen a shell arra egészítette ki. A fenti probléma megoldása a következő példák valamelyike:

```
# find /usr/bin -name pass\*
/usr/bin/passwd
# find /usr/bin -name 'pass*'
```

Ezen megoldások bármelyike hatásos a shell kiértékelésével szemben. A **find** parancsról és kapcsolóiról bővebben a **find** manual-jában olvashatunk.

A következő feladatot mindenki próbálja meg egyénileg megoldani!

*Írjon bash shell szkriptet, amely megszámolja és kiírja, hogy a szkriptnek argumentumként megadott könyvtárban ÉS ALKÖNYVTÁRAIBAN összesen hány "***.jpg**" fájl van!*

Egy lehetséges megoldás:

```
#!/bin/bash
JC=0; # JpegCount
for i in `find $1 -name "*.jpg" 2>/dev/null`
do
  JC=$((JC+1));
done
echo $JC
```

Megjegyzés: A **find** hibaüzeneteit „eltüntettük”. Ez egyrészt célszerű, mert a programnak azt kell kiírnia, amit kértünk és nem egyéb olyan üzeneteket, hogy milyen könyvtárba nem

tudott a **find** belépni. Másrészt viszont ezzel a megoldással akkor sem kapunk hibaüzenetet, ha például hibásan adjuk meg a könyvtár nevét.

### 3.4.4 Grep

A **grep** a STDIN-ről vagy a paraméterként megadott fájlból a keresési feltételeknek megfelelő sorokat (**-l** kapcsoló esetén az ezeket tartalmazó fájlokat; -l: list) jeleníti meg.

1. példa: szöveg egy a feltételnek megfelelő sorát jelenítjük meg:

```
laptop:~/grep# cat szoveg.txt
elefánt
zsiráf
macska
kutya
oroszlán
laptop:~/grep# cat szoveg.txt | grep zsiráf
zsiráf
laptop:~/grep#
```

2. példa: csak a keresett szöveg sorát nem jelenítjük meg:

```
laptop:~/grep# cat szoveg.txt | grep -v zsiráf
elefánt
macska
kutya
oroszlán
laptop:~/grep#
```

3. példa: csak a keresett szöveget jelenítjük meg (**-o** kapcsoló nélküli hatást is bemutatjuk. A **-o** régebbi **grep** esetében nem működik):

```
laptop:~/grep# echo "paci zsiráf zebra kolibri" | grep -o zsiráf
zsiráf
laptop:~/grep# echo "paci zsiráf zebra kolibri" | grep zsiráf
paci zsiráf zebra kolibri
laptop:~/grep#
```

A fenti példa így értelmetlennek tűnhet, de például ha reguláris kifejezés segítségével egy MAC címet, vagy IP címet szeretnénk egy szövegben megkeresni akkor ez a megoldás nagyon hasznos lehet.

4. példa: az 1. példa megvalósítása kicsit másképp:

```
laptop:~/grep# cat szoveg.txt | sed -n '/zsiráf/p'
zsiráf
laptop:~/grep#
```

Magyarázat: A sed esetén a **-n** opció letiltja a bemeneten kapott sorok automatikus kiírását. A sed a "**/**" jel párban levő regexp-re illeszkedő sorokat választja ki, majd a **p** opció hatására kiírja a kiválasztott sorokat.

### 3.4.5 Tr

Egy újabb hasznos program a **tr**, ami string-ek „fordítását" végzi. A következő példánk a nagybetűből álló (pl.: MS-DOS partícióról másolt) fájlok neveit kisbetűsre nevezi át. A **tr**-hez is használhatjuk a reguláris kifejezéseknél megismert karakterosztály paramétereket.

```
# for i in *; do mv $i `echo $i | tr [:upper:][:lower:]` ; done
```

A **tr** képes kicserélni, törölni karaktereket a szövegből. Például nagyon gyorsan kicserélhetjük az ékezetes karaktereket, és szóközöket a segítségével:

```
laptop:~# echo "ékezetes szöveg amiben szóköz is van" | tr "öőóüúűéái " "ooouuueai_"
ekezetes_szoveg_amiben_szokoz_is_van
laptop:~#
```

vagy átnevezhetünk egy könyvtárnyi fájlt, hogy ne legyen ékezet és szóköz a fájlokban:

```
dev:/tmp/test# ls -1
árvíztűrőtükörfúrógép
próba
proba file
dev:/tmp/test# ../test.sh
dev:/tmp/test# ls -1
arvizturotukorfurogep
proba
proba_file
dev:/tmp/test# cat ../test.sh
#!/bin/bash
for i in *
do
  mire=`echo $i| tr "öőóüúűéái " "ooouuueai_"`
  mv "$i" $mire
done
dev:/tmp/test#
```

Az **ls -1** egy oszlopba listázza a könyvtár tartalmát.

Vagy a DOS-os szövegfájlból UNIX-osat formálni, ami annyit tesz, hogy a „kocsivissza" (return) speciális vezérlő karaktereket töröljük:

```
cat dos_szoveg.txt | tr -d '\r' > unix_szoveg.txt
```

A **tr** a **-s** kapcsolójával az ismétlődő karaktereket szünteti meg:

```
laptop:~# echo "a  a a    a     a" | tr -s ' '
a a a a a
```

## 3.5 További shell szkript példák

Néhány egyszerű példa szemlélteti, hogy a shell szkripttel milyen feladatok oldhatók meg, ami más módon nehezen megoldható, vagy időigényes lenne.

A korábbiakban általában tagoltan, több sorban írtuk a ciklusokat, segítve ezzel a megértést. Amennyiben ezeket parancssorból hajtjuk végre és később (a kurzort felfele mozgató nyíl segítségével) előhozzuk, akkor látjuk, hogy egyetlen sorban vannak, és ahol új parancs kezdődik ott pontosvessző áll. Most gyakorlásképpen ilyen formátumot használunk (ezzel helyet is megtakarítunk). Lássunk egy példát!

Ha szeretnénk a könyvtárunkban található **.htm** fájlokat **.html**-re átnevezni, megtehetjük ezt a következő ciklus segítségével:

```
bash-2.05> for i in *; do mv $i `echo $i | sed ”s/\.htm$/.html/”` ;done
```

Ahol a **for i in *; do <utasítások>; done** egy ciklus, a "****" a "**.**" –ot (mint helyettesítő karaktert) megvédi, a **.htm$** a **.htm**-re végződő string-eket jelöli. A ciklus az aktuális munkakönyvtárban levő fájlok nevein fut végig. A szkript minden egyes fájlt megpróbál átnevezni, de mivel a nem **.htm** végződésű fájlokon a **sed** nem végez konverziót, az **mv** nem nevezi át ugyanarra a névre a fájlt, így ezekben az esetekben hibaüzenetet ír ki. Ha ezt nem szeretjük, átnevezés előtt ellenőrizhetjük, hogy különbözőek-e a fájlnevek. Az alábbi szkript ezt mutatja be (és azt is, hogy mennyivel áttekinthetőbb a program, ha tördeljük):

```
#!/bin/bash
for i in *;
do
  from=$i;
  to=`echo $i | sed 's/\.htm$/.html/'`;
  if [ "$from" != "$to" ]; then
    mv "$from" "$to";
  fi
done
```

Megjegyzés: A figyelmes Olvasó észrevehette, hogy a fenti szkriptben a régi és új fájlneveket idézőjelek közé tettük! Vajon miért?

Megfejtés: ha esetleg a fájlnevekben szóköz van, akkor az idézőjelek használta nélkül hibát okozna!

A további feladatokhoz már minden szükséges előismeretet megadtunk, ezért javasoljuk, hogy először igyekezzenek azokat önállóan megoldani és csak utána nézzék meg a „kincstári” megoldást!

*1. feladat: Írjon bash shell szkriptet, amely kiszámítja az aktuális könyvtárban található HTML fájlok (pontosabban: "**.html**" végződésű névvel rendelkező fájlok) méretének összegét!*

Egy lehetséges megoldás:

```
#!/bin/bash
SUM=0;
for i in *.html;
do
  meret=`ls -l $i | awk '{print $5}'`
  SUM=$((SUM+meret))
done
echo $SUM
```

*2. feladat: Írjon bash shell szkriptet, amely kiírja az argumentumként megadott könyvtárban található "***.jpg**" fájlok közül azoknak a nevét, amelyeknek a mérete legalább 100kB és legfeljebb 200kB!*

Egy lehetséges megoldás:

```
#!/bin/bash
for i in $1/*.jpg
do
  size=`ls -l $i | awk '{print $5}'`
  if [ $size -ge 102400 ] && [ $size -le 204800 ]; then
    echo $i;
  fi
done
```

*3. feladat: Írjon bash shell szkriptet, amelyik megszámolja, hogy az első paraméterként megadott azonosítójú felhasználó hány reguláris fájllal (nem könyvtárral) rendelkezik a második paraméterként megadott könyvtárban és annak alkönyvtáraiban, és ezek összesen mennyi helyet foglalnak el a háttértárolón! (Segítség: a fájlok kiválogatásához: **man find**, diszkfoglaláshoz: **du** + **awk**).*

Egy lehetséges megoldás:

```
#!/bin/bash
# parameters: $1: user, $2: directory
FC=0; # FileCount
```

```
DU=0; # DiskUsage
for file in `find $2 -user $1 -type f 2>/dev/null`
do
  FC=$((FC+1));
  DU=$(($DU+$(du $file | awk '{print $1}')));
done
echo $FC " db fájl van a $1 tulajdonában a $2 könyvtárban"
echo $DU " blokkot foglalnak el összesen"
```

*4. feladat: Írjon bash shell szkriptet, amelyik sorszámozva kiírja (sorban egymás alá) az indításkor megadott paramétereit az alábbi formában:*

**1.: <az első paraméter>**

**2.: <a második paraméter>**

*(Segítség: a* ***$@*** *vagy a* ***$**** *segítségével előállítható a paraméterek listája, érdeklődőknek bővebben:* ***man bash /PARAMETERS /Special Parameters****)*

Egy lehetséges megoldás:

```
#!/bin/bash
n=1
for i in $*
do
  echo $n".: "$i
  n=$((n+1))
done
```

Vegyük észre, hogy mivel a ciklusváltozónk a paraméterek listájából képzett halmaz elemeit vette fel, ezért a sorszámozást „kézzel” állítottuk elő!

Továbbra is bátorítjuk hallgatóinkat arra, hogy találjanak ki maguknak feladatokat és oldják meg őket!

# 4 Linux kernel

A Linux kernel forrását C nyelven írják, szabadon terjeszthető, és bárki átírhatja, módosíthatja a GNU GPL licencet betartva, és persze ha rendelkezik megfelelő C programozási ismeretekkel. Ebben a fejezetben az alapvető ismereteket sajátítjuk el, hogy hogyan konfiguráljuk, fordítsuk le saját igényeink szerint a rendszermagunkat.

Ezt a fejezetet jelenleg nem frissítjük, a kernel fordítást a hallgatók laborgyakorlat keretében ismerik meg!

## *4.1 Konfiguráció elkészítése*

Mielőtt nekiállnánk, fel kell telepítenünk a `kernel-package`, `bzip2` (ha ezt a típusú kernelt akarjuk letölteni), valamint a `libncurses5-dev` csomagokat. Ezt egyszerűen megtehetjük ha feltesszük a `build-essential` csomagot, mely a kernelfordításhoz szükséges csomagokat tartalmazza. Szerezzük be a számunkra legmegfelelőbb kernelforrást! Ne terheljük az ország kimenő sávszélességét, használjuk a letöltésre a magyar tükörszervereket (a magyar tükörszerver az `ftp.kfki.hu`)!

Pl.: wget ftp://ftp.kfki.hu/pub/linux/ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.27.tar.gz

Miután letöltöttük, csomagoljuk ki a `/usr/src` könyvtárba:

```
# tar -xvfz linux-2.6.27.tar.gz; tar -xvfj linux-2.6.27.tar.bz2
```

Kibontás után készítsünk egy szimbolikus linket a könyvtárra.

```
root@pc0:/usr/src# ln -s linux-2.6.27 linux
```

Lépjünk be a könyvtárba, majd írjuk be a következő parancsot:

```
root@pc0:/usr/src/linux# make menuconfig
```

Több lehetőségünk is van a kernelt konfigurálni:

- config: parancssoros, minden opciót megkérdez, és úgy kell rá válaszolnunk
- menuconfig: egy curses based keretrendszer segítségével konfigurálhatunk

- xconfig: QT alapú grafikus menü segítségével
- gconfig: GTK alapú grafikus menü segítségével
- defconfig: az alapértelmezett értékeket állítja be
- randconfig: véletlen értékekkel állítja be a kernelt.(hardcore felhasználóknak!)

Fontos, hogy a fordítást mindig root-ként végezzük, a jogok miatt!

Rövid várakozás után megjelenik a konfigurációkészítő menürendszer. A menüben fel- illetve le-nyilakkal lépkedhetünk, az egyes almenükbe ENTER lenyomásával tudunk belépni. Visszafelé haladni az ESC billentyű megnyomásával tudunk. Az egyes meghajtó programok (program kódok) kiválasztásánál a SPACE gomb többszöri nyomogatásával lehet választani, hogy a rendszermagba, vagy modulként kívánjuk lefordítani azokat. Az „m” lenyomásával modulként, „y” hatására a kernel részeként fordul le, az „n” billentyű hatására pedig eltávolítható a kernelből a meghajtó program. Érdemes megjegyezni, hogy nem minden programkód fordítható modulba. (Például a kernelmodul betöltése funkciót nem tudnánk modulként engedélyeztetni!) Biztonságtechnikai szempontból a monolitikus kernel jobb, mint a moduláris, hiszen így nem tölthető be olyan kernelmodul, ami a rendszerünkre tekintve veszélyes.

A kernelbe fordított meghajtó programkódok előnye, hogy mindig szerepel a rendszerben, eszerint nem kell azt külön betöltögetni, így mindig rendelkezésre áll. Érdemes azokat a meghajtó programokat kernelbe fordítani, melyek a rendszerünk számára nélkülözhetetlenek (pl.: IDE-SATA, SCSI vezérlő, stb.).

Nem biztonságkritikus rendszerek esetén, modulként érdemes olyan eszközök meghajtó programjait fordítani, amiket nem mindig használunk, illetve használat után el szeretnénk távolítani a rendszerünkből. Tipikusan ilyenek az USB-s eszközök (UHCI, OHCI). A lefordított modulokat az `insmod` vagy a `modprobe` parancs segítségével tölthetjük be, az `rmmod` paranccsal távolíthatjuk el. Az `lsmod` paranccsal pedig a betöltött modulokat listázhatjuk ki. A lefordított modulok minden Linux disztribúció esetében a `/lib/modules/--kernel verziószám--` könyvtárban helyezkednek el (az aktuális kernelverziót az `uname -r` paranccsal kérdezhetjük le). Mivel a rendszerünk az éppen aktuális verziószámú könyvtárat automatikusan eléri (ehhez persze futtatnunk kell fordítás után, vagy ha új modult teszünk a modulok közé a `depmod -a `uname -r`` parancsot), elég a modul betöltéséhez csak a modul nevét megadni (pl.: Realtek 8139 típusú hálózati interfész modul betöltése: `modprobe 8139too`).

A 2.6.x.y kernelek esetében, nem a `modutils`, hanem a `module-init-tools` nevű programcsomagot használjuk a modulok betöltésére, eltávolítására. Ez fontos lehet olyan rendszereknél, ahol a disztribúció „öreg” és új kernelt akarunk használni. Megoldás lehet a modul nélküli kernel használata.

## 4.2 A kernel konfiguráló menü felépítése

A kernel konfigurálását gyakorlati óra keretek között mutatjuk be. Ennek oka a kernel gyors változása.

## 4.3 A kernel fordítása

A régebbi 2.4.x kernelek esetében a következő parancsokat kellett kiadni a kernel fordításához:

```
# make dep
# make
# make modules
# make install
# make modules_install
```

Az újabb 2.6.x.y kernelek esetén az első 3 lépést kiváltja egy sima `make` parancs. A továbbiak azonban nem elégségesek! Helyettünk inkább `.deb` csomagot fogunk készíteni, mivel így gondoskodunk az **initrd**-ről is. (Szerepét lásd korábban, a rendszer elindulásáról szóló részben.)

A Debian GNU/Linux esetében, ha telepítve van a `kernel-package` csomag, akkor egy paranccsal tudunk `.deb` állományt készíteni a forrásból, amit a helyi vagy akár egy távoli gépre átmásolva tudunk telepíteni úgy, hogy a távoli gépre nem kell fordító környezet.

Célszerű a már működő kernelünk konfigurációs állományát felhasználni alapértelmezettnek, majd ebben véghezvinni a módosításokat:

```
# cp /boot/config-$(uname -r) ./.config
```

Amennyiben a későbbiekben más modulokat is szeretnénk telepíteni, úgy szükségünk lesz a kernel headerre is. Ezek a parancsok:

A helyi fordítókörnyezettel rendelkező gépen:

```
# make-kpkg --initrd --revision=custom_kernel kernel_image kernel_headers
```

Ahol a `--initrd` kapcsolóval a `kernel-package` csomag, a `.deb` állományt, `initrd` kompatibilisen készíti el. FIGYELEM! Ekkor a kernel csomag az 1-el feljebb levő könyvtárban, esetünkben a `/usr/src/linux` helyett a `/usr/src`-ben jön létre. Jelen esetben a neve: `linux-image-2.6.27_2.6.27-10.00.Custom_kernel.deb` lett.

A helyi vagy akár a távoli gépen az telepítés:

```
# dpkg -i linux-image-$RELEASE.deb
```

# 5 A Unix felépítése

Ebben a fejezetben megismerkedünk a Unix felépítésével elméleti és gyakorlati szempontból. Megismerjük a több felhasználós és több feladatos rendszerek működési elvét. A gyakorlati alkalmazhatóság érdekében a Unix alapú rendszerek működési elvét a Linuxon keresztül fogjuk tanulmányozni.

## *5.1 Több felhasználós és több feladatos rendszer lényege*

A számítógépek szolgáltatásainak – felhasználási területtől függően – elérhetőnek kell lennie több felhasználó számára is. Ez az igény egyszerűen úgy is jelentkezhet, hogy a felhasználók nem egyidejűleg, hanem felváltva használják a számítógépet. Ilyenkor már szükséges olyan szolgáltatásokat bevezetni, amelyek lehetővé teszik, hogy minden felhasználó önálló futtatási környezetben dolgozhasson, például mindig saját állományait használhassa, másokét ne. Emiatt védelmi elemeket kell az operációs rendszerbe építeni: egy felhasználó csak a számára engedélyezett erőforrásokhoz és adatokhoz férhessen hozzá, a rendszer kritikus paraméterei pedig csak a rendszergazda, illetve az általa meghatározott csoportok számára legyenek hozzáférhetők. Fontos a felhasználók megkülönböztetése a skálázási szempontok miatt is.

A több feladatosság igénye például akkor is felmerül, amikor egy számítógépet szerverként üzemeltetünk, azaz szolgáltatásokat nyújtanak egyszerre több felhasználó számára (például WEB, FTP, E-MAIL). Ez annyit jelent, hogy a számítógép erőforrásait egy időben több program között kell felosztani és az erőforrások kiosztását/lefoglalását is ellenőrzötten kell végezni. Ezt csak az ún. multitaszkos, azaz több feladatos operációs rendszerek tudják teljesíteni. Ilyennek tekinthető pl. az összes UNIX klón, az OS/2 a Windows, stb. Tipikusan nem több feladatos, és nem több felhasználós operációs rendszer pl. a DOS.

A több feladatos végrehajtás legnagyobb veszélye, hogy egy hibás működésű program miatt leállhat egy másik vagy az összes többi program is, amely a számítógépen fut. Ezen nem kívánt „baleset” megelőzése érdekében az operációs rendszer beépített védelemmel rendelkezik.

Minden futó program külön memóriaszeletet használ, és a perifériákhoz történő hozzáférés is csak a rendszer hívásain keresztül lehetséges. Az operációs rendszer feladata az is, hogy figyelje azt, hogy egy program nem kezdeményez-e hibás vagy illetéktelen hozzáférést a rendszer valamely részéhez. Ilyen esetben az operációs rendszer közbeavatkozik, és a hibás működésű program futtatását megszünteti.

A több feladatos és több felhasználós rendszerek tulajdonságai összefoglalva:

- Egyszerre több program futhat
- A felhasználók egyedi azonosítóval rendelkeznek UserID (authentikáció)
- Az erőforrás-felhasználás szabályozott
- A rendszer működése védett mind a felhasználókkal, mind a hibás működésű programokkal szemben
- A rendszer ellenőrzött hozzáférést biztosít az egyes hardver elemekhez és egyéb erőforrásokhoz

## *5.2 Fájlrendszer típusok*

Egy unixos rendszer telepítésekor a legelső lépés a partíciók, fájlrendszerek létrehozása. Régebben, az EXT2 volt a legelterjedtebb fájlrendszer, a Linux rendszerek esetén. Ezzel lehet a legnagyobb sebességet és megbízhatóságot elérni a nem-naplózó fájlrendszerek között. Az újabb rendszermagok elterjedésével lehetőségünk van másfajta fájlrendszerre feltelepíteni az alaprendszerünket. Az EXT3, ReiserFS, XFS, JFS alapvető eltérése az EXT2-höz képest, hogy ezek már *naplózott fájlrendszerek*. Minden végrehajtandó fájlművelet a naplóban előre eltárolódik, így hibás rendszerleállás (crash) esetén nem kell a partíciónkat hosszasan végigellenőrizni, elég csak a napló bejegyzéseit végignézni: mi az amit végrehajtott, folyamatban van vagy végre kellett volna hajtania a rendszernek.

A Linux képes többek között FAT (DOS), FAT32 (windows9x) és NTFS/HPFS (windowsXP, Vista, 7, valamint Windows Server) partíciókat is kezelni.

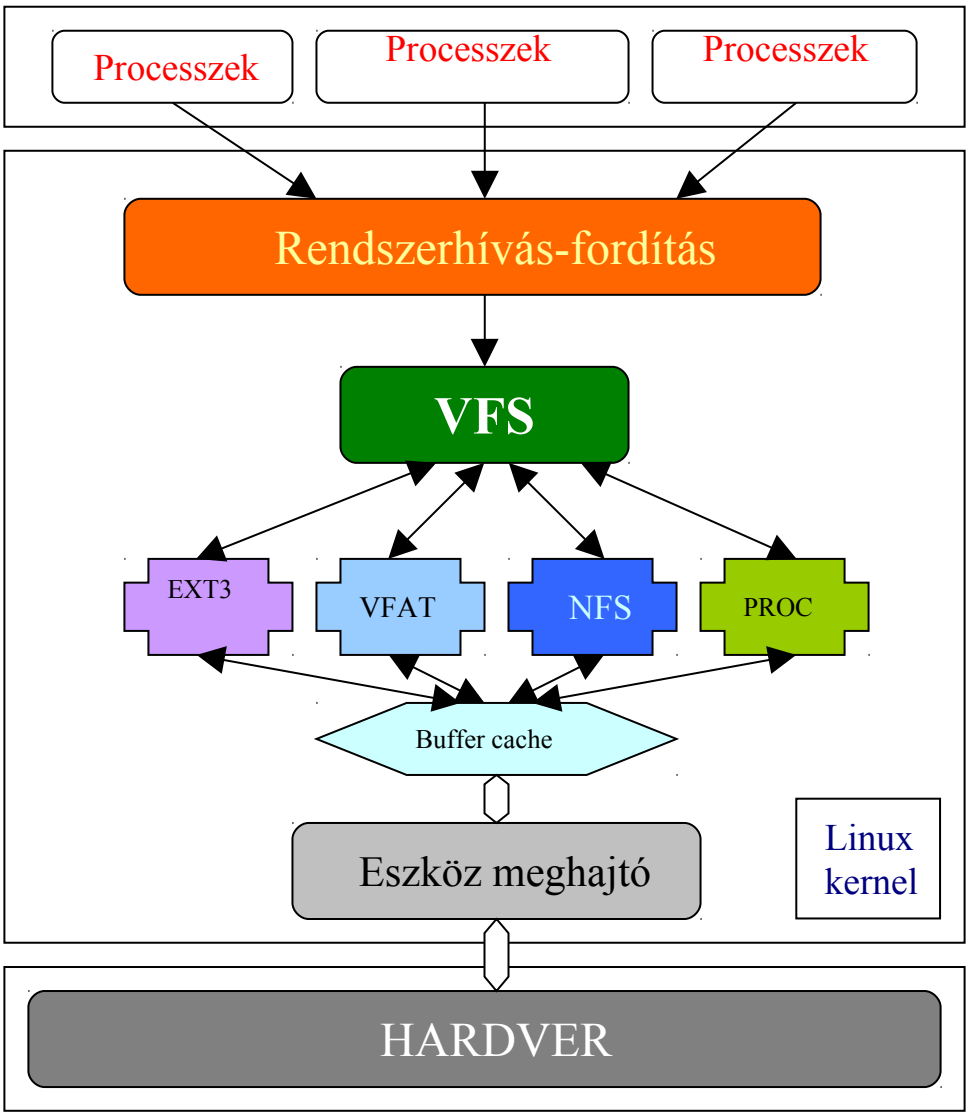
Az utóbbiaknál a partícióra való írás úgy történik, hogy a meglévő fájlokat módosítja a Linux (egy magyar fejlesztésnek FUSE (file system in userspace) köszönhetően már vannak olyan a FUSE-re épülő userspace 3. generációs NTFS modulok (ntfs-3g) melyek nagy biztonsággal dolgoznak az NTFS-el). A hálózati meghajtókat is fájlrendszerként kezeli a UNIX, ilyen például az NFS és az SMBFS, CIFS is.

- ext2, ext3: A Linux natív fájlrendszerei
- reiserfs, reiser4: Hans Reiser csapata által fejlesztett fájlrendszerek
- xfs: A Silicon Graphics által fejlesztett nagygépes fájlrendszer
- jfs: Az IBM fájlrendszere
- NTFS, vfat: A Microsoft operációs rendszereinek a fájlrendszere. A pendrive-ok vfat fájlrendszert használnak általában.(4-8 GiB-ig)
- NFS, SMBFS, CIFS: Hálózati fájlrendszerek
- minix: Andrew Tanenbaum operációs rendszerének a fájlrendszere
- cramfs: Read Only fájlrendszer
- JFFS2: beágyazott rendszerek kedvelt fájlrendszere

Az ext4 jelen állás szerint a leginkább terjedőben lévő fájlrendszer köszönhetően az ext3 nagy sikerének. Az ext4 támogatja az 1 exabyte tárhelyet (1 EiB=1024*1024 TiB), és a 16 TiB-s fájlméretet. Változtattak a block allokációs algoritmuson, mellyel gyorsulás érhető el. Valamint megdőlt a 32000-es limit az alkönyvtáraknál, ez az ext4-nél már 64000. Az ext4 kompatibilis a régebbi fájlrendszerekkel, sőt az ext2, ext3 fájlrendszer felcsatolható ext4 fájlrendszerként, ekkor sebességnövekedést érhetünk el, bár természetesen nem lesz az ext4 minden újdonsága elérhető.

## 5.3 A VFS, virtuális fájlrendszer

A Linux – a különböző fájlrendszerek egységes kezelhetőségének érdekében – tartalmaz egy szoftver réteget az ún. virtuális fájlrendszert. Ez a réteg elvonatkoztat az egyes fájlrendszerek típusától, egy egységes kezelési felületet biztosít a felhasználói alkalmazások számára. A 3. ábra szemlélteti a VFS elhelyezkedését a Linux rendszerstruktúrában.




3. ábra: Virtuális fájlrendszer felépítése

A VFS biztosítja, hogy a felhasználói alkalmazások minden kezelt fájlrendszert azonos módon lássanak. Ezt a gyakorlatban úgy tapasztaljuk meg, hogy az összes hardvereszközön lévő fájlrendszer tartalmát a root directory (gyökér könyvtár) alatt, külön alkönyvtárakban érhetjük el. E könyvtárak helyét a „mount” folyamattal tudjuk kijelölni. A VFS számon tartja a mount-olt fájlrendszerek helyét és állapotát (`cat /proc/mounts` vagy `cat /etc/mtab`). Így érhetjük el, hogy egy másik számítógép megosztását valamilyen hálózati fájlrendszer-protokoll segítségével (pl.: NFS, SAMBA) saját gépünk egy alkönyvtárában lássuk.

## 5.4 Mount

Partíciókat felcsatolni a `mount` paranccsal lehet. Szintaktikája: `mount [kapcsolók] <mit> <hova>`. A `-t` kapcsolóval a fájlrendszer típusát adhatjuk meg. A kernel által ismert típusokat a `cat /proc/filesystems` paranccsal nézhetjük meg. Ha a `/etc/fstab`-ba bejegyeztük a felcsatolni kívánt meghajtót és célkönyvtárat, akkor egyszerűen csak a kijelölt alkönyvtárat, vagy a forrást kell megadni. Pl.: cd-rom meghajtó felmountolása, ha bejegyeztük az `/etc/fstab`-ba akkor elég a `mount /cdrom`. A `/etc/mtab` az éppen felcsatolt meghajtókat jelzi, ezt egyébként a `mount` parancs argumentumok nélküli kiadásával is megtekinthetjük. A `mount` paranccsal lehet loop eszközként CD, DVD imageket (lemezképeket) felcsatolni a fájlrendszerünkhöz: Példa: `mount -o loop dvd.img /hova/akarom`.

## 5.5 A Proc fájlrendszer

A `/proc` könyvtárban lévő fájlok a futó processzekről, a kernel és a hardver bizonyos állapotairól szolgálnak információval. Néhány esetben nem csak információt ad, hanem lehetőséget arra, hogy „on the fly” (menet közben) módosítsuk a kernel bizonyos paramétereit (például `ip_forward`). Ezek az állományok szöveges fájlként jelennek meg a rendszerben, és bármikor olvashatjuk őket pl.:

`cat /proc/cpuinfo` a processzorunkról;

`cat /proc/partitions` a partícióinkról,

`cat /proc/meminfo` a memóriáról,

`cat /proc/filesystems` az elérhető fájlrendszerekről ad információt

*Érdemes egyszer végig böngészni a könyvtárat, hogy mit találhatunk még benne!*

## 5.6 Fájlrendszer, inode-ok, linkek

A Linux és az összes UNIX alapú rendszer legalapvetőbb védelme a fájlrendszer. Ez tárolja az összes fájl és alkönyvtár elérhetőségi szabályát, felhasználó-csoportosításokat, kezeli a linkeket. A fájlrendszer gondoskodik az adathalmazok tárolásáról és a szabad lemezterület menedzseléséről. A fájlrendszerrel rokon modul még a „Quota subsystem” (kvóta alrendszer), amely a felhasználók tulajdonában lévő fájlok maximális helyfoglalását (byte-ban és/vagy darabszámban) korlátozza.

## 5.7 Inode-ok

Minden fájlt egy inode-dal írhatunk le a fájlrendszerben, ez információkat tartalmaz a fájlról: típus, jogok, tulajdonságok, időbélyeg, méret, és az adatblokkok helye (mutatója, azaz pointere). Az adatblokkok ún. foglalási egységek, mérete 1 kilobyte többszöröse lehet (1024, 2048, 4096 byte a szokásos, az SSD meghajtóknál felépítésükből következően 64kB, illetve 128 kB). Ha például egy 10 kbyte-os JPEG fájt helyezünk el egy 4096 byte-os blokkméretű partíción, akkor az valójában 12 kbyte-ot fog elfoglalni, mert 3 darab egyenként 4 kB-os blokkot fog lefoglalni. (Képzeljük el ugyanezt SSD meghajtónál – a sebességnek ára van!) Ennek a 3 darab blokknak a partíción belüli címét írjuk bele az inode megfelelő adatmezőjébe. Az inode-ról az `lde` (Linux disk editor) paranccsal kaphatunk információt:

```
notebook:/mnt# lde -i 14 /dev/hde1
Device "/dev/hde1" is mounted, be careful
User requested autodetect filesystem. Checking device . . .
Found ext2fs on device.
------------------------------------------------------------------------
INODE: 14     (0x0000000E)
-rw-r--r--       root     root        697253 Mon Dec 19 20:30:31 2005
TYPE:                    regular file
LINKS:                   1
MODEFLAGS.MODE:          010.0644
SIZE:                    697253
BLOCK COUNT:             1370
UID:                     00000 (root)
GID:                     00000 (root)
ACCESS TIME:             Mon Dec 26 20:15:13 2005
CREATION TIME:           Mon Dec 26 21:42:32 2005
MODIFICATION TIME:       Mon Dec 19 20:30:31 2005
DELETION TIME:           Thu Jan  1 01:00:00 1970
DIRECT BLOCKS:           0x00001A01 0x00001A02 0x00001A03 0x00001A04
                         0x00001A05 0x00001A06 0x00001A07 0x00001A08
                         0x00001A09 0x00001A0A 0x00001A0B 0x00001A0C
INDIRECT BLOCK:          0x00001A0D
DOUBLE INDIRECT BLOCK:   0x00001B0E
TRIPLE INDIRECT BLOCK:

notebook:/mnt#
```

A következőkben – az egyszerűség kedvéért – 1 kB-os diszk blokkokkal számolunk.

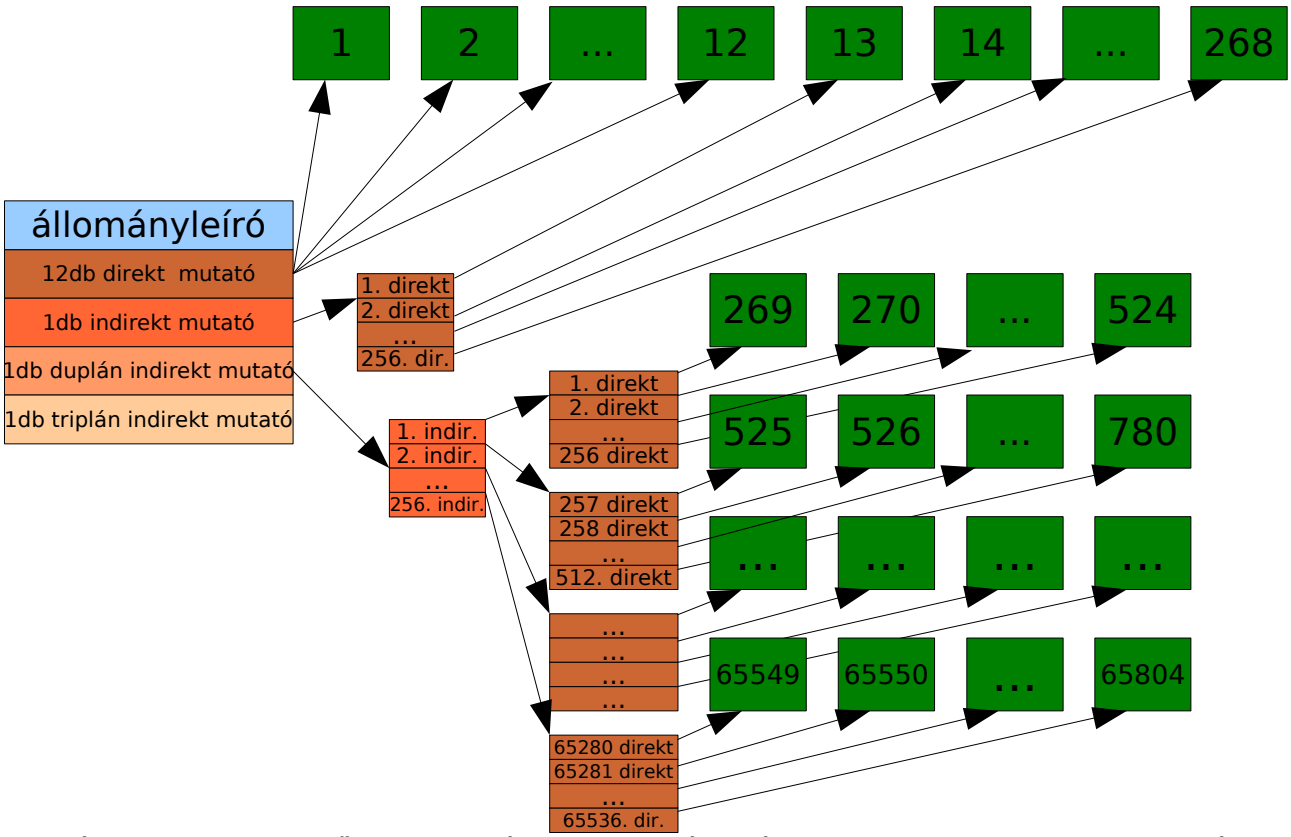
A számítások menete természetesen más méret esetén is ugyanez, és a blokkméretet paraméterként is használhatnánk, de könnyebben érthető, ha egy konkrét számmal dolgozunk.

Egy inode tartalmaz 12 db *direkt mutató*t (pointer), amelyek mindegyike egy-egy 1kB-os adatblokkra mutat. Ezek a *direkt blokkok*, bennük összesen 12 kB adatot tudunk tárolni. Az inode-ban a következő mutató egy *egyszeresen indirekt mutató*, azaz egy olyan diszk blokkra mutat, amely további olyan mutatókat tartalmaz, amik már a fájl adatblokkjaira mutatnak. Ezek az *egyszeresen indirekt blokkok*. Mivel egy 1 kB méretű blokkba 256 db 32 bites (azaz 4 byte-os) mutató fér el, így a fenti egyszeres indirekt címzéssel összesen 256 kB + 12 kB adatot lehet elérni. Az inode-ban a következő mutató *kétszeresen indirekt mutató*, vagyis egy olyan diszk blokkra mutat, amely 256 db egyszeresen indirekt mutatót tartalmaz. Vagyis most hivatkozáskor az első két lépésben mutatókat tartalmazó blokkokat

kapunk, és csak a harmadik lépésben jutunk el az adatokhoz. Az inode-ban a következő, egyben utolsó mutató egy *háromszorosan indirekt mutató*, vagyis még a harmadik hivatkozás is 256 db pointert tartalmazó blokkra mutat, és onnan egy újabb referenciával juthatunk el az adatokhoz. Tehát a fent vázolt struktúrával összesen:

- 12 db direkt blokkban: 12 kB
- 256 db egyszeresen indirekt blokkban: 256 kB
- 256*256 db kétszeresen indirekt blokkban: 64 MB
- 256*256*256 db háromszorosan indirekt blokkban: 16 GB

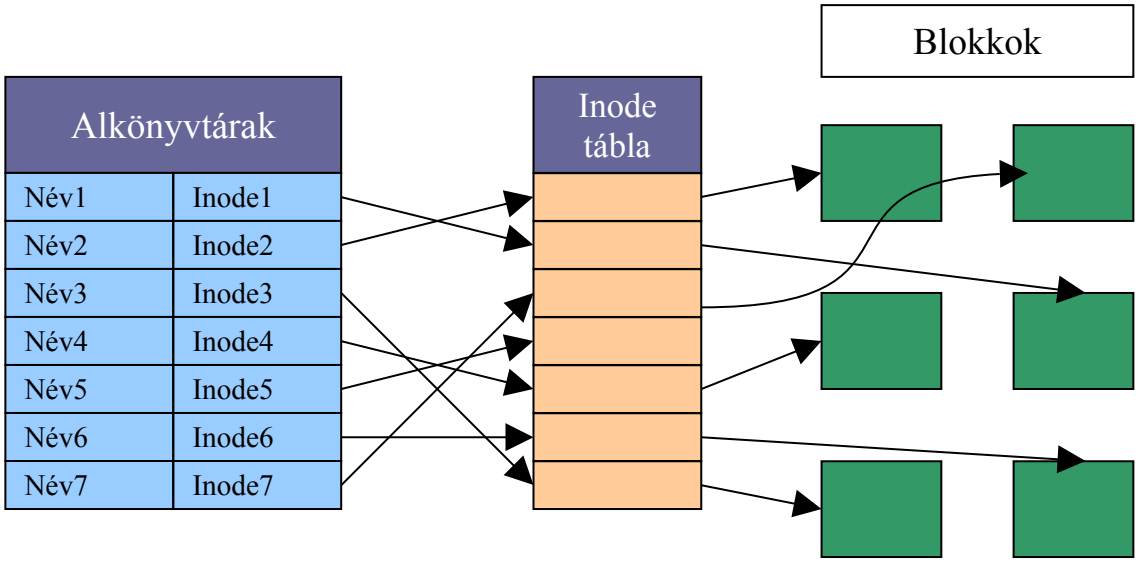
azaz mindösszesen 16GB+64MB+256kB+12kB adatot lehet tárolni.




4. ábra: inode mezők tartalmának szemléltetése (1KB-os diszk blokk esetén)

## 5.8 Alkönyvtárak

Az alkönyvtárak tulajdonképpen speciális fájlok, ahol a fájl tartalma egy lista, melynek minden eleme egy fájlnevet, és egy inode számot tartalmaz, amelyek a könyvtáron belül vannak. Az alkönyvtárakat is ugyanolyan inode-ok írják le, mint a fájlokat, csak a fájltípus mezőben alkönyvtár-jelzés van („d” bejegyzés lásd: `ls` parancs). Az alkönyvtár bejegyzéseket az 5. ábra szemlélteti.




5. ábra: Alkönyvtár inode mezőinek jelentése

## *5.9 Linkek*

A UNIX fájlrendszernél bevezették a link fogalmát, amely azt jelenti, hogy egy fájlt vagy alkönyvtárat leíró inode-ra több alkönyvtárbejegyzés mutathat.

Az egyik típus az ún. „hard link” úgy jön létre, ha egy alkönyvtárban készítünk egy új bejegyzést, amely már létező fájlra (inode-ra) mutat. Minden inode tartalmaz egy számlálót, amely mutatja, hány helyről hivatkozunk rá. Ha létrehozunk egy hard link-et, akkor ez a számláló érték nő, ha törlünk egy hard linket, akkor csökken. Amikor az utolsó hivatkozást is eltávolítottuk (pl.: az **`rm`** paranccsal), akkor az inode DELETION TIME (törlési idejét) beállítja, ezzel használaton kívül helyezi az inode-ot. A törlési időt (deletion time), azért állítja be, mert különben a fájlrendszer ellenőrzésekor megtalálnánk, mint „elveszett” fájlt. A hard linkek legnagyobb „hátránya”, hogy csak egy partíción belül használhatók. Pl. nem tehetünk hard link-et két külön partíció közé. Ezen felül nem engedi az operációs rendszer a könyvtárra mutató hard link készítését sem, nehogy véletlenül „végtelen rekurzió” alakuljon ki az alkönyvtárak között.

A linkek másik fajtája az úgynevezett szimbolikus link (symbolic link) vagy más néven „soft link”, ez egy olyan bejegyzés, amely egy fájl nevet és egy elérési útvonalat tartalmaz (létrehozása: **`ln -s <amire a link mutat> <link neve>`**). Ha egy fájl elérésekor szimbolikus linket adunk meg, az operációs rendszer kicseréli arra az elérési útra, amelyre a link mutat, és ehhez a névhez tartozó inode-ot keresi meg.

## 5.10 Eszközfájlok

A UNIX rendszerekben a támogatott hardver eszközöket (egér, winchester, cdrom, pendrive stb.) speciális, ún. eszközfájlokon keresztül lehet elérni. Ha egy ilyen fájl elérését kezdeményezzük, a kernel automatikusan meghívja az adott hardverelemet kezelő rutint, és szabványos I/O műveleten keresztül közvetíti az eredményt, mintha azt az eszközfájlból olvasnánk ki. Léteznek karakteres és blokk elérésű eszközök is. A karakteres elérésű eszközök getchar() és putchar() C függvényeken keresztül, karakterként olvashatóak és írhatóak, azaz egy műveletre egyetlen egy byte-ot adnak vissza. Ilyen eszköz például az egér (PS/2: **`/dev/psaux`**, újabban **`/dev/input/mice`** vagy **`/dev/input/mouse0`**). A blokk eszközök a **`read()`** és **`write()`** C függvényeken keresztül egyszerre több száz bájt írásával, olvasásával érhetők el. Ilyen eszköz például az összes lemezmeghajtó (IDE0: **`/dev/hda`**, SCSI0/SATA0: **`/dev/sda`** – de újabban az IDE-s eszközöket így jelölik). Az eszközfájlok két azonosítószámmal rendelkeznek. Ezek a „major number” és a „minor number”. Ez a két szám egyértelműen azonosít egy eszközt. A legtöbb UNIX rendszer nem hozza létre ezeket az eszközfájlokat, a felhasználó, vagy az operációs rendszer telepítője készíti el ezeket az **`mknod`** paranccsal. Debian GNU/Linux alatt az eszközfájlok létrehozásához a **`MAKEDEV`** parancsot használhatjuk. Az udev (Dynamic **`/dev`** directory) óta ez elavult (obsolate).

## 5.11 Hogy épül fel egy könyvtárrendszer

Mint minden operációs rendszer esetében megszokhattuk, itt is fájlokkal és könyvtárakkal kell dolgoznunk, viszont a már ismertebb operációs rendszerekkel szemben itt a meghajtókra nem betűjelekkel hivatkozunk. Az egész rendszerünk egy főkönyvtárból nyílik (főkönyvtár hivatkozása: „/”). Ezt más néven *root*nak (magyarul: gyökérnek) hívjuk. Ezekben találhatók az alábbi könyvtárak.

| | | |
|---|---|---|
| **`/`** | = | főkönyvtár |
| **`/bin`** | = | futtatható bináris állományok (alapvető parancsok) |
| **`/boot`** | = | BOOT partíciót szokták ide csatlakoztatni |
| **`/dev`** | = | DEVICES, azaz eszközfájlok |
| **`/etc`** | = | a rendszer konfigurációs fájljai |
| **`/home`** | = | felhasználók könyvtárai |
| **`/lib`** | = | library, azaz dinamikusan szerkesztett könyvtárak |
| **`/proc`** | = | processz-, és rendszer-információk |
| **`/root`** | = | rendszergazda könyvtára |
| **`/tmp`** | = | ideiglenes könyvtár |
| **`/usr`** | = | alkalmazási és egyéb programok |

**/sbin** = futtatható bináris állományok (rendszer karbantartása céljaira)

**/var** = egyéb, programok által használt könyvtárak, ahova írnak is...

(bővebben: Filesystem Hierarchy Standard: **http://www.pathname.com/fhs/** ; Linux Standard Base: **http://www.linux-foundation.org/en/LSB**)

## 5.12 Fájlok és jogaik a UNIX-ban

Mivel a UNIX rendszer többfelhasználós rendszer, az esetleges felhasználóink adatait egymástól védeni kell. A UNIX-ban a védelem háromszintű (ez függ az alkalmazott fájlrendszer típusától is). Első szint a felhasználó jogai, ugyanis a fájlok és könyvtárak hozzá vannak rendelve a felhasználóinkhoz (ezek a USER-ek). Továbbá hozzá van rendelve, hogy milyen csoportba tartozik (ezek a GROUP-ok), ez a második szint. A harmadik szint azt szabályozza, hogy hogyan férhet hozzá a többi felhasználó (olyanok, akik nem a tulajdonosai, illetve nincsenek a csoportban sem). Egy fájlhoz és könyvtárhoz információkat jegyez be egy inodeba a fájlrendszer:

- tulajdonos azonosító (UID, GID)
- állomány típusa (reguláris fájl, könyvtár, szimbolikus link, named pipe, socket, karakteres vagy blokkos berendezés)
- állomány hozzáférési jogosultságok (tulajdonos, csoport, ill. a világ számára, olvasási, írási ill. végrehajtási jogok)
- időcímkék (time stamps):
- az utolsó állomány-hozzáférés ideje
- az utolsó állomány módosítás ideje
- az utolsó attribútum módosítás ideje (inode módosítás)
- linkek száma (hány néven lehet hivatkozni az adott fizikai állományra)
- címtábla (mutatók adat blokkokra 12 db direkt, 1 db indirekt, 1 db kétszeres indirekt és 1 db háromszoros indirekt blokkra mutató mutató)
- állomány méret

A fájlok neve után nem kötelező, de célszerű megadni a típusra utaló végződését (máshol „kiterjesztés”-nek hívják), hogy tudjuk, milyen fajta fájlról van szó (Pl.: alma.jpg – tudjuk, hogy kép fájl). Ez azonban a felhasználóknak és bizonyos felhasználói programoknak (a **gcc** pl. ragaszkodik a „.c” vagy „.cc” végződéshez) nyújt információt. A DOS/Windows rendszerekkel ellentétben a Unix rendszerek NEM a fájl nevének a végződését használják a fájl típusának a megállapításra, hanem a fájl tartalmának az elejét! A kezdő 2 bájt, az ún. „magic number” (bűvös szám) alapján ismeri fel az operációs rendszer, hogy pl. egy ELF vagy egy a.out típusú végrehajtható fájl-e az adott állomány.

Az inode-ban tárolt jogosultságokat négy oktális (8-as számrendszerbeli) számmal írhatjuk

le. Ennek az alapértelmezését az *umask* paranccsal állíthatjuk be, így az újonnan létrehozott fájlok, már ezekkel a jogokkal jönnek létre. Az első oktális számjegy a setuid, setgid, sticky biteket adja meg. A második, harmadik, negyedik oktális számjegy a tulajdonos, csoport és a „többi" felhasználó jogait határozza meg. A 2-4. oktális számjegy bitjeit a következőképpen értelmezzük: fájloknál első bit az olvasási, a második az írási, a harmadik a futtatási jog, könyvtáraknál az első bit a listázási (könyvtárban levő bejegyzések megjelenítése), a második módosítási (pl. bejegyzések létrehozása, törlése), a harmadik pedig a seek („keresési", pontosabban: elérési) jog (a könyvtárbejegyzések elérése).

Egy példa: a diak nevű könyvtár jogai a következők: rwx r-x --- ennek bináris értéke a következő 111 101 000, ez oktálisan 7 5 0. Adjuk meg mindenki számára a listázás és a tartalom elérés jogát! A jogok megváltoztatására a chmod parancsot használjuk.

```
# chmod 755 diak
```

Lehet másképpen is megadni:

```
# chmod +r+x diak
```

Setuid beállítása:

```
# chmod 4755 diak
```

Tulajdonos megadása: **chown** tulajdonos bejegyzés

```
# chown diak /home/diak
```

Csoport megadása: **chgrp** csoport bejegyzés

```
# chgrp users /home/diak
```

Legegyszerűbben egy parancsból lehet megadni a felhasználót és csoportot:

```
# chown diak:users /home/diak
```

vagy:

```
# chown diak.users /home/diak
```

# 6 Felhasználók kezelése a UNIX-ban

Nézzük meg a **/etc/passwd** fájlt! Ez a fájl tartalmazza a UNIX rendszerekben a felhasználók adatait. Régebben itt helyezkedett el a felhasználó elkódolt jelszava is. Biztonsági okokból áttették a későbbi verziókban a jelszavakat a **/etc/shadow** fájlba. Könnyen belátható, hogy miért: Nézzük meg a jogokat a **/etc/passwd** fájlra (-rw-r--r--). Látható, hogy mindenki által olvasható, mert nem minden processz fut root jogokkal, és néhánynak szüksége van a felhasználók azonosítására, és ezek ezt a fájlt használják. A fájlban a jelszót ugyan egyirányú (azaz nem invertálható) kódolással tárolták, de próbálgatással (szótáras törés vagy akár kimerítő keresés), így is lehetőség volt a jelszó megfejtésére.

Mit tartalmaz a **/etc/passwd** fájl?

```
root:x:0:0:Ez a rendszergazda account,,,:/root:/bin/bash
drmomo:x:1000:100:Molnár Zoltán,laboros,L1-7,tel.szám,:/home/drmomo:/bin/bash
```

Nézzük meg részletesen lépésről lépésre (a mezőelválasztó a „:” jel):

1. Felhasználói név (user account): **root**
2. Régen itt volt a jelszó, de ez már a shadow fájlban található: **x**
3. Felhasználói azonosító szám (UID = User ID): **0**
4. Csoportazonosító szám (GID = Group ID): **0**
5. Név, és további információk (a mező elválasztó a „**,**” jel)
6. Home könyvtár: **/root**
7. A felhasználó által bejelentkezés után használt shell: **/bin/bash**

Ha szeretnénk létrehozni új felhasználót, akkor root-ként szerkeszteni kell a **/etc/passwd** fájlt (mindenki a szívéhez nőtt szerkesztő-programot használhatja: pl.: **pico**, **nano**, **joe**, **mcedit**, vagy **vi**). Értelemszerűen be kell jegyezni az új felhasználó adatait. Ezután szinkronizálni kell a **passwd** fájlt a **shadow** fájllal. Erre szolgáló parancs: **pwconv**

Ezután hozzuk létre a megadott home könyvtárat. Adjunk rá jogokat, és adjuk meg a felhasználói jelszót a **passwd** paranccsal.

Erre szolgáló (egyszerűbb módszer) a Debian GNU/Linux-ban egy **adduser** nevű (**perl**-ben készült) szkript, ezt kitöltve ugyanazt érhetjük el, mint a fent említett módszerrel. (Az **adduser** szkript egy interaktív program, amely átveszi a rendszer adminisztrátorától a létrehozni kívánt felhasználó paramétereit, és meghívja a **useradd** futtatható bináris programot, ami elvégzi a felhasználó létrehozását.)

Egy példa a **buksi** felhasználó felvételére:

```
# vi /etc/passwd
```

Beírjuk a következő sort a fájlba:

**buksi:x:1077:100:Kamu Bela,,,:/home/buksi:/bin/bash**

```
# pwconv
# mkdir /home/buksi
# chown buksi:users /home/buksi
# passwd buksi
Changing local password for buksi.
New password:
Retype new password:
```

## 6.1 Felhasználói korlátozások

A UNIX rendszerekben lehet és kell is a felhasználóinknak a rendszerünk adta lehetőségeket korlátozni. Ez lehet akár tárolóhely, memória, vagy processzoridő.

Először nézzük meg, hogyan lehet a tárolóhelyet korlátozni, azaz kvótázni.

### 6.1.1 Quota

Mivel egy adott rendszerben a háttértárak kapacitása véges, korlátozni kell a felhasználók helyfoglalását. E célra a UNIX-okban beépített támogatás van, az ún. kvóta alrendszer (Quota subsystem). A kvóta alrendszer minden lemezre íráskor ellenőrzi egy adott felhasználó helyfoglalását, és ha eléri a rá kiszabott határt (hard limit) a rendszer „write error” hibaüzenettel megtagadja a további írást a lemezre. A kvóta beállításokat felhasználókra és csoportokra egyaránt vonatkoztathatjuk, illetve az összes külön felcsatolt lemezegységre vagy partícióra megadhatjuk azokat.

Ahhoz, hogy a kvóta működni tudjon, a kernelbe be kell fordítani a quota támogatását (a kvóta működik EXT2, EXT3, EXT4, REISERFS, JFS, XFS [2.6.x] esetén is.

Tegyük fel, hogy a rendszerünk **/home** könyvtárban lévő felhasználói fájlok helyfoglalását szeretnénk korlátozni. A következő műveleteket kell elvégeznünk a kvóta rendszer üzembe állításához:

Ellenőrizzük, hogy a **/home** könyvtár külön partíción helyezkedik-e el:

```
root@pc0:/# mount
/dev/hda1 on /boot type ext2 (rw)
/dev/hda3 on / type ext2 (rw)
/dev/hda4 on /home type ext2 (rw)
none on /dev/pts type devpts (rw, gid=5, mode=620)
none on /proc type proc (rw)
```

Ezután a **/etc/fstab**-ban megjelöljük, hogy a **/home** partíció kvótázva lesz.

```
root@pc0:/# pico /etc/fstab
```

A megfelelő sort a következőképpen módosítsuk:

```
/dev/hda4  /home  ext3  defaults,usrquota,grpquota  0  2
```

Figyeljünk oda rá, hogy a vesszők előtt és után nem állhat szóköz! A szóközök (és tabulátorok) ugyanis mezőszeparátornak számítanak: az összes csatolási opció egyetlen mező része!

Miután módosítottuk az **fstab**-ot, mountoljuk újra a **/home** partíciót, majd aktiváljuk a kvóta rendszert.

```
root@pc0:/# mount -o remount /dev/hda4
root@pc0:/home# quotacheck -avug
Scanning /dev/hda4 [/home] done
Checked 4 directories and 72 files
Using quotafile /home/aquota.user
Using quotafile /home/aquota.group
root@pc0:/home# quotaon -avug
/dev/hda4: group quotas turned on
/dev/hda4: user quotas turned on
```

Ha most megnézzük az általunk létrehozott fájlokat, láthatjuk, hogy már nem 0 a méretük, a kvótázással kapcsolatos információkat tartalmazzák.

Ezután ha a felhasználó beírja a **quota** parancsot, megnézheti a rendelkezésre álló tárhelyet:

```
diak@pc0:~> quota
Disk quotas for user diak (uid: 1000): none
```

Természetesen, még senkinek nem állítottunk be kvóta értékeket, ezért továbbra is korlátlan tárhellyel rendelkeznek felhasználóink. A korlátozást **edquota** paranccsal állíthatjuk be:

```
root@pc0:/home# edquota -u diak
Disk quotas for user diak (uid 1000):
Filesystem   blocks      soft        hard     inodes    soft      hard
/dev/hda4    263288     50000      100000     2129     15000     17000
```

A fenti mezőben a **soft** jelenti azt a határt, aminél több adatot a felhasználó nem tárolhat huzamosabb ideig a lemezen. Az **edquota -t** paranccsal beállíthatjuk az ún. *grace period* időtartamot: ezen időtartam alatt még írhat a felhasználó a lemezre, de legfeljebb a *hard limit* erejéig. Ha a grace period lejár, a soft limit is „hard limitté” válik.

### 6.1.2 Korlátozások az ulimit segítségével

Előfordulhat, hogy egy futó program erőforrás-felhasználását szeretnénk korlátozni, pl.: nem szeretnénk, hogy egy program túlzottan sok processzoridőt vegyen el. Ezeket a

paramétereket bash shell esetén az általunk futtatott shellre és az ebből futtatott programokra vonatkozóan az **ulimit** paranccsal állíthatjuk be. De vannak olyan shellek, amelyek nem támogatják ezt a fajta korlátozást (pl. **csh**), ez esetben a beállítás sajnos semmit nem ér. (Kérdéses esetben a shell kézikönyvében keressünk rá!)

Az **ulimit** (többek között) a következő paramétereket fogadja el:

- **-a**: – megmutatja az aktuális határokat
- **-c**: – a „core" fájl maximális mérete
- **-d**: – a maximális adatszegmens mérete
- **-f**: – a maximális fájlméret
- **-n**: – nyitott fájlok maximális száma
- **-s**: – maximális veremméret
- **-t**: – maximális processzoridő másodpercben
- **-u** – az adott felhasználó által futtatott processzek maximális száma
- **-v** – a shell által használható maximális virtuális memória mérete

Bővebb információ: **man bash**, majd **/ulimit** alatt.

### 6.1.3 Korlátozások a pam_limits segítségével

A **pam_limits** PAM modul számára a **/etc/security/limits.conf** állományban lehet beállítani a korlátozásokat. A szintaktika a következő:

**<domain> <type> <item> <value>**

Ezek jelentése a következő:

**domain**: Itt megadható egy felhasználói név (pl. **diak**), egy „**@**" karakter után egy felhasználói csoport tagjai (pl. **@users**), illetve a ***** karakter, ami az összes felhasználót jelenti.

**type**: A korlátozás típusa lehet **hard**, amit egyszer beállítva a kernel kikényszerít, a felhasználó nem tudja ennél feljebb állítani; **soft**, amit alapértelmezett értékként kap a felhasználó, de értékét a hard limitig változtathatja; végül a „**-**" típus a soft és a hard együttes beállítását szolgálja.

**item**: Ez adja meg a korlátozni kívánt jellemzőt, ami sokféle lehet, közülük néhány fontosabb:

- **core**: – a „core" fájl maximális mérete (kB)
- **data**: – a maximális adatszegmens mérete (kB)
- **fsize**: – a maximális fájlméret (kB)
- **nofile**: – nyitott fájlok maximális száma

- **stack**: – maximális veremméret
- **cpu**: – maximális processzoridő percben
- **nproc**: maximális processz szám
- **maxlogins**: maximális bejelentkezések száma ezzel a userid-vel (kivétel: uid=0)

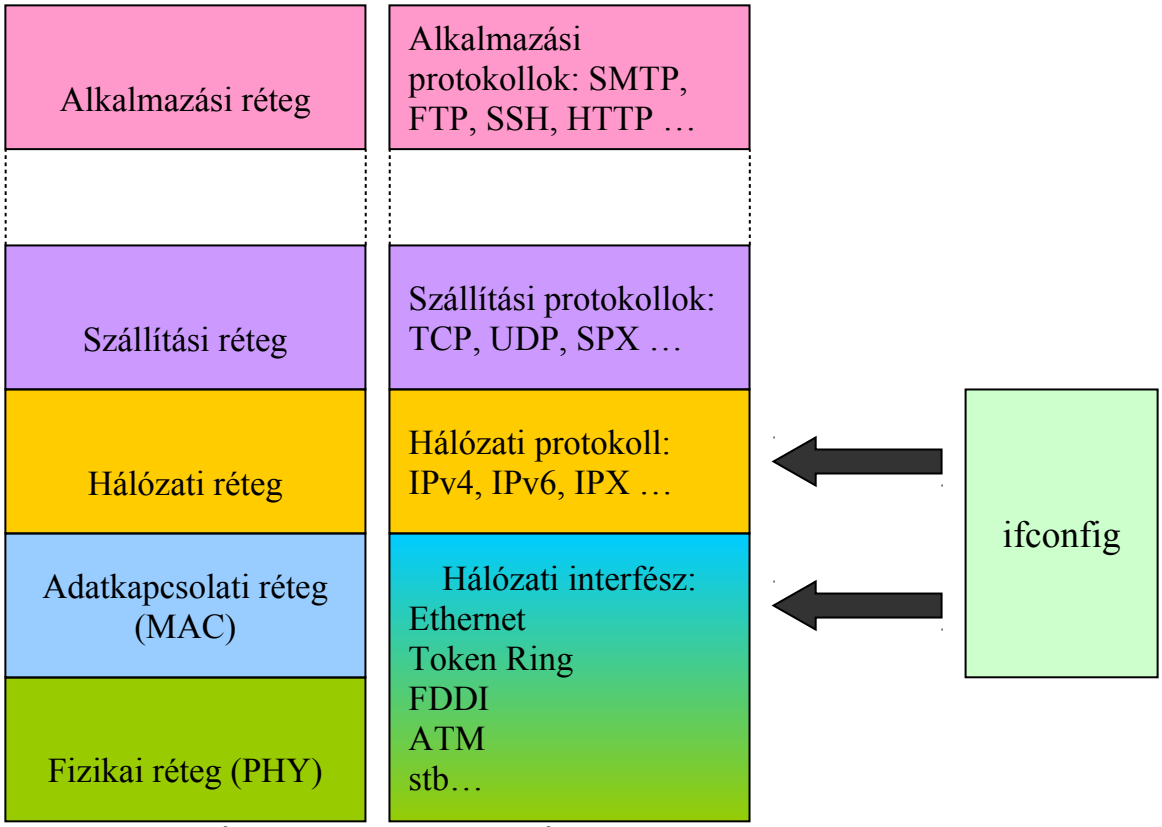
**value**: A beállítandó érték. A „-1” érték általában azt jelenti, hogy korlátlan.

Bővebb leírás: **man limits.conf**

# 7 Hálózati interfészek konfigurációja

## *7.1 Interfészek paraméterezése*

A hálózati beállításokat (ahogy gyakoroltuk is számítógép-hálózatokból) majdnem minden UNIX rendszerben az **`ifconfig`** paranccsal el tudjuk végezni (van rá más lehetőség is). Emlékezzünk vissza a számítógép-hálózatok tantárgyban tanultakra: a hálózati interfészeknek szükségük van felsőbb szintű protokollokra, hogy használhatóak legyenek számunkra (szolgáltatásokat nyújthassunk rajta).




6. ábra: Az ifconfig hatásköre az OSI modell szerint

A 6. ábra szerint látható, hogy az **`ifconfig`**gal a hálózati interfészünk adatkapcsolati és hálózati rétegbeli tulajdonságait paraméterezzük. Természetesen a beállításokat a kernel végzi el, az **`ifconfig`** csak a kernelnek ad utasítást a paraméterezésekre.

## 7.2 Adatkapcsolati réteg paraméterezése

A UNIX rendszerekben a hálózati interfészekre hivatkozni kell. A Linux esetében az Ethernet típusú hálózati csatolókártyák hivatkozása: **eth0**, **eth1**, stb. attól függően, hogy hány darab Ethernet hálózati kártya van felkonfigurálva a rendszerünkben. PSTN kapcsolatú eszközökre (telefonos modem, ADSL modem) például **ppp0** –ként hivatkozunk. Vezeték nélküli interfészek esetén **wlan0**. Az alábbi táblázatban összefoglaljuk a leggyakrabban használt linuxos elnevezéseket.

| interfész típusa | linuxban a neve |
|---|---|
| Ethernet | eth{0,1,2,...,n} |
| Ethernet másodlagos | eth{0,1,2,..,n}:{0,1,2,..,n} |
| Ethernet VLAN-nal | eth{0,1,2,...,n}.{0,1,2,...,n} |
| bridge eszköz | br{0,1,2,...,n} |
| Ethernet szintű virtuális eszköz | tap{0,1,2,...,n} |
| virtuális p-t-p eszköz | tun{0,1,2,...,n} |
| wireless | wifi{0,1,2,...,n}, wlan{0,1,2,...,n}, ra{0,1,2,...,n} |
| modem | ppp{0,1,2,...,n} |
| atheros kártya | ath{0,1,2,...,n} |
| **nameif** név MACADDR | név |

Vizsgáljuk meg Ethernet esetén az adatkapcsolati réteg paramétereit!

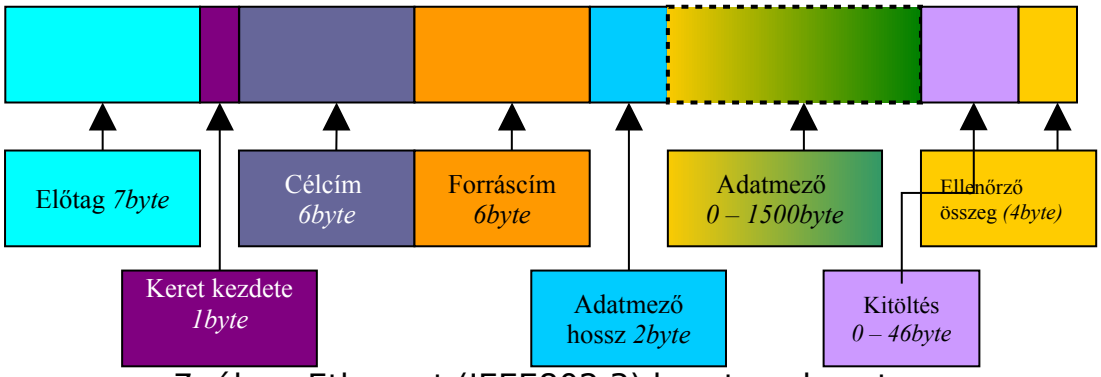
A labor gépein rendszerindításkor konfigurálódik valamelyik hálózati interfész, vizsgáljuk meg az `ifconfig` kimenetét!

```
root@pc0:/# ifconfig

eth4   Link encap:Ethernet  HWaddr 00:06:25:0B:03:36
    inet addr:193.224.130.170  Bcast:193.224.130.191  Mask:255.255.255.224
    UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
    RX packets:1208 errors:0 dropped:0 overruns:0 frame:0
    TX packets:819 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:100
    RX bytes:111845 (109.2 KiB)  TX bytes:379338 (370.4 KiB)
    Interrupt:3 Base address:0x100
```

Nézzük milyen paraméterek tartoznak az adatkapcsolati rétegbe. Idézzük fel az Ethernet

keretszerkezetét (az alábbi ábrán az első két mező a fizikai réteghez tartozik).




7. ábra: Ethernet (IEEE802.3) keretszerkezete

Az MTU, azaz *Maximum Transmission Unit* (maximálisan átvihető byte-ok száma egy keretben) értéke jelen esetben 1500 byte, tehát a maximum adatmező hossz. Egyes esetekben szükséges lehet ennek a módosítása, ezt az **ifconfig <interface> mtu <MTU mérete byte-ban>** paranccsal tehetjük meg lekapcsolt interfész esetén (tipikusan ilyen eset: egyes ADSL szolgáltatók más MTU-t használnak, ezért a belső hálózatból érkező csomagok 1500-as MTU-ja problémát okozhat).

Az **ifconfig**-gal állíthatjuk a hálózati csatolókártyánk MAC címét is (ha a kártya meghajtó modulja támogatja ezt a funkciót). A jelenlegi MAC címünket az **ifconfig** kimenetének jobboldali legfelső sorában láthatjuk. Az interfész MAC címét átállíthatjuk kedvünk szerint, de figyelni kell a szabályokra: nem adhatunk olyan MAC címet, ami már szerepel a hálózati szegmensben, illetve nyilván van tartva a switch címlistájában. A cím 6byte-os (6 darab kétszer 0 – F hexadecimális számokból állhat, 2 szám után kettősponttal választjuk el őket). Pl.: 00:06:36:88:44:3F. A címet az **ifconfig <interface> hw ether <új MAC cím>** paranccsal változtathatjuk meg, de csak lekapcsolt interfész esetén. A rendszer arra is ügyel, hogy például csoportcímet nem enged beállítani az interfész címének!

Az **ifconfig** továbbá a kernel által mért, átvitt adatok mennyiségét is megjeleníti az adott interfészen: RX packets, RX bytes, TX packets, TX bytes. Az RX packets a fogadott csomagokat, az RX bytes a fogadott byte-okat, míg a TX packets a küldött csomagokat, a TX bytes pedig a küldött byte-okat jelenti.

A kernel méri az elveszett csomagokat és az ütközéseket is.

Érdemes összehasonlítani több gép esetén, hogy milyen eredmények mérhetőek HUB és Switch használatakor.

A hálózati réteg paraméterezése során meg kell adnunk, hogy milyen protokoll szerint szeretnénk konfigurálni a hálózatunkat. Az **ifconfig**gal lehetséges IPv6-ot és IPX-et is konfigurálni az IPv4 mellett.

IPv4 konfigurálás: **ifconfig <interface> [inet] <32bit-es IPv4 cím> netmask <netmask> broadcast <broadcast> up**

```
ifconfig eth0 192.168.0.12 netmask 255.255.255.0 broadcast 192.168.0.255 up
```

IPv6 konfigurálás: **ifconfig <interface> [inet6 add] <128bit-es IPv6 cím>/<a prefix hossza> up**

```
ifconfig eth4 inet6 add 2001:470:1f0b:1110::12/64
```

Alternatív megoldás az **iproute** csomag **ip** parancsának használata. Ez sokkal elterjedtebb, rengeteg finom beállításra van lehetőségünk, amire az **ifconfig** nem képes, ilyen például, hogy egy hálózati interfészhez több IP címet rendeljünk, de nem az ethX:X használatával:

```
laptop:~# ip addr sh eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0a:e4:af:75:9c brd ff:ff:ff:ff:ff:ff
laptop:~# ip addr add 192.168.0.12/24 dev eth0
laptop:~# ip addr sh eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0a:e4:af:75:9c brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.12/24 scope global eth0
laptop:~# ip addr add 192.168.5.12/24 dev eth0
laptop:~# ip addr sh eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0a:e4:af:75:9c brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.12/24 scope global eth0
    inet 192.168.5.12/24 scope global eth0
laptop:~#
```

Az **iproute** használatával többek között a routingot is tudjuk állítani. Bátran állíthatjuk, hogy az **ip** egy „hálózati svájci bicska”.

Az **ifconfig** (helyes paraméterezése esetén) a hálózatcímet automatikusan kiszámítja. A **route** paranccsal meggyőződhetünk erről. Az alapértelmezett átjárót szintén a **route** paranccsal adhatjuk meg.

```
route add default gw <32bit-es IP cím>
```

Debian GNU/Linux alatt a hálózati interfészt lényegesen könnyebb beállítani. Csak ki kell töltenünk, vagy új bejegyzést kell tennünk az **/etc/network/interfaces** fájlba, majd **/etc/init.d/networking restart** paranccsal a hálózatot újrakonfigurálni.

```
# ez egy dhcp-s hálózati kártya:
iface eth0 inet dhcp
# a következő statikusan beállított hálózati kártya:
iface eth1 inet static
address 192.168.100.1
network 192.168.100.0
broadcast 192.168.100.255
```

```
netmask 255.255.255.0
#amennyiben átjárót is szeretnénk megadni:
gateway 192.168.100.254
# Az auto bejegyzések után lévő hálózati interfészeket az indítószkriptek linux
# indulásakor beállítják, egyébként saját magunknak kell kiadni az
# ifup <hálózatiinterfész>, ifdown <hálózatiinterfész> parancsokat:
auto eth0 eth1
```

Lásd még: **ipcalc**, **ipv6calc**, **man interfaces**

Az **ns.tilb.sze.hu** routing táblája:

```
ns:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
193.224.131.128 0.0.0.0         255.255.255.240 U     0      0        0 bond0
193.225.151.64  0.0.0.0         255.255.255.240 U     0      0        0 bond0
193.224.129.160 0.0.0.0         255.255.255.240 U     0      0        0 bond0
193.224.130.160 0.0.0.0         255.255.255.224 U     0      0        0 bond0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 bond0.11
193.224.128.0   0.0.0.0         255.255.255.0   U     0      0        0 eth6
10.9.0.0        0.0.0.0         255.255.255.0   U     0      0        0 br0
0.0.0.0         193.224.128.9   0.0.0.0         UG    0      0        0 eth6
ns:~#
```

## *7.3 Az arp, arping, és a rarp*

Egy kis ismétlés: Az ARP (Address Resolution Protocol = Cím feloldó protokoll) OSI 2. rétegbeli protokoll. A Routerek a hálózati szegmensben úgy juttatják el a kereteket a címzettnek, hogy ARP-t használva felderítik a szegmensen lévő hálózati interfészek MAC címeit. Ezeket az IP cím, MAC cím párosokat eltárolják. Ezt hívják *ARP caching*nek. A UNIX-ban használt **arp** parancs arra szolgál, hogy ezt az ARP cache-t kiírassuk. Az arping paranccsal megnézhetjük az adott IP című interfész MAC címét. Figyelem: a routerek OSI 3. rétegében kötik össze a hálózati szegmenseket, így egy nem a mi szegmensünkön lévő interfész IP címét **arping**-elve a router felénk eső interfészének MAC címét fogjuk megkapni (lásd: Számítógép-hálózatok).

# 8 Netfilter

A Linux fejlődése során a kernelbe építették, és még ma is fejlesztik a hálózati forgalom figyelését, és irányítását. A 2.0.x kernelverziók esetén **ipfwadm** paranccsal lehetett úgynevezett FORWARD, azaz csomagtovábbítási paramétereket adni. A 2.2.x verzió megjelenésével az **ipchains** már újabb funkciókat tudott az új kernelből kicsikarni. Ennek előnye, hogy könnyedén lehetett paraméterezni egy csomag útját, hátránya viszont, hogy átláthatatlanná, és kezelhetetlenné vált sok paraméter esetén.

A 2.4.x verzió megjelenése hozta az **iptables** megoldást. Az **iptables** segítségével egyszerűen lehet a csomagokat kezelni, és sok paraméter esetén is viszonylag könnyen átlátható, állítható, módosítható, és elmenthető a konfiguráció.

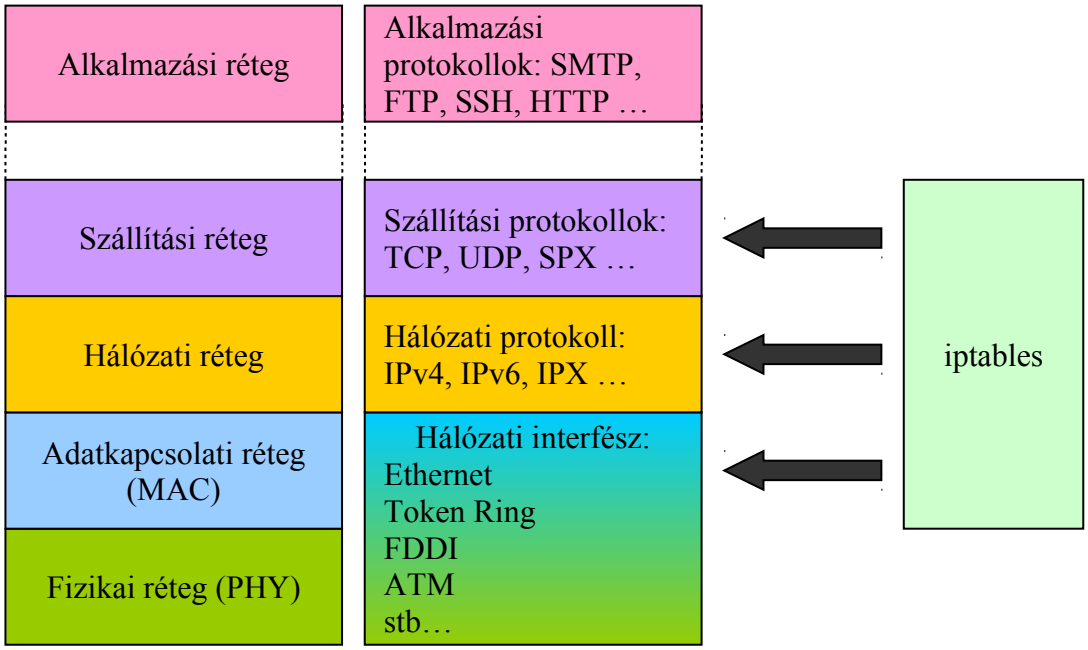
Az **iptables** parancson kívül az **iptables-save** az éppen futó konfigurációt jeleníti meg a standard outputra. Így könnyedén elmenthetjük konfigurációnkat:

```
iptables-save > my_ip_config
```

Az iptables-restore paranccsal viszont könnyen visszaállíthatjuk elmentett beállításainkat:

```
iptables-restore < my_ip_config
```

Vizsgáljuk meg, hogy az iptables az OSI modell szerint mely rétegekben tevékenykedik!

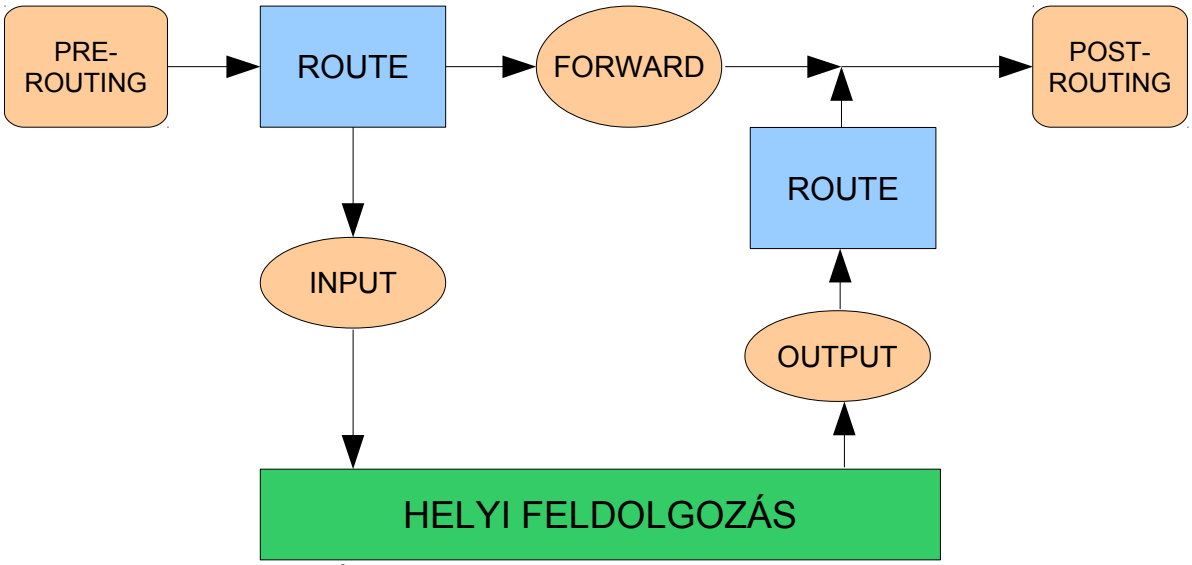



8. ábra: Az iptables hatásköre az OSI modell szerint

Mint azt később látni fogjuk, az **iptables** egészen az adatkapcsolati rétegtől a szállítási rétegig tudja befolyásolni a csomagok útját.

Ahhoz, hogy egy gép tudjon más gépek között csomagokat továbbítani, engedélyezni kell kernel szinten az IP csomag-továbbítást, más néven a kernel routing-ot: **`echo "1" > /proc/sys/net/ipv4/ip_forward`**. Debian lenny-ben és squeeze-ben a forwardot a **`/etc/sysctl.conf`** fájlban lehet engedélyezni: **`net.ipv4.ip_forward=1`**

A 9. ábrán egy nagyon leegyszerűsített képet mutatunk be arról, hogy egy csomag életébe az **`iptables`** segítségével mely pontokon lehet beavatkozni. A pontos helyzet ennél sokkal bonyolultabb, érdeklődőknek ajánlott: http://jengelh.medozas.de/images/nf-packet-flow.png




9. ábra: IP routing a 2.4.x kernelekben

Érdeklődőknek ajánlott olvasmány: Rusty Russell: Linux 2.4 Packet Filtering HOWTO

http://netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html

http://www.szabilinux.hu/iptables/index.html (magyar fordítás)

A fejezet további részei ezeknek a felhasználásával készültek.

Amint a neve is mutatja, az egyes feladatokra az **`iptables`** különböző táblázatokat használ. A használni kívánt táblázat nevét a **`-t`** kapcsolóval lehet megadni. Az alapértelmezett táblázat – ezt használja, ha nem adunk meg táblázatot – a **`filter`**. Ezt használjuk csomagszűrésre. Hálózati címfordítás (NAT, NAPT) céljára a **`nat`** táblát használjuk. A **`mangle`** táblát különböző csomag manipulációkra (az egyes protokollok adategységeiben levő mezők megváltoztatására) használhatjuk. És van még egy tábla, a **`raw`**, aminek a neve azt fejezi ki, hogy a „nyers” IP csomagokkal foglalkozik. Ezt használhatjuk például akkor, ha a kapcsolatkövetést kihagyva szeretnénk a csomaggal valamit kezdeni.

A különböző táblázatok esetén a fenti ábrán megadott beavatkozási pontok (hook) közül némelyek használhatók, mások nem (bővebben: **`man iptables`** parancs után **`/TABLES`**). Az egyes beavatkozás pontokra megadott szabályokat láncokba (chains) szervezi, így a

beavatkozási pontok nevét a láncok neveként is használjuk.

## 8.1 Csomagszűrés működése és megvalósítása

A csomagszűrő (packet filter) figyeli a csomagok fejlécét, miközben azok keresztülhaladnak rajta, és eldönti az adott csomag további sorsát. Ez lehet DROP és a REJECT melyek a csomag eldobását jelentik (REJECT annyiban más, hogy egy ICMP replyt küld a hosztnak, hogy az adott szolgáltatás nem érhető el), ACCEPT, mely a csomag elfogadását jelenti (hagyja továbbhaladni), és QUEUE azaz csomagkésleltetés. Az IP QUEUE egy kísérleti stádiumban lévő funkció. A kernel a csomagot egy várakozási sorban tárolja, majd innen a sorból a megfelelő programok (pl.: forgalomkorlátozó) kiveszik, és döntési mechanizmusuk szerint továbbítják egy részét, a többit eldobják. A sor feldolgozását már nem a kernel, hanem valamely felhasználói módban (user space) futó program végzi.

Csomagszűréshez az **`iptables`** alapértelmezett, **`filter`** nevű táblázatát használjuk. Ebben három beépített lánc van, az INPUT, OUTPUT, és a FORWARD láncok, melyeket nem lehet törölni. A láncok alapértelmezett módja (irányelv, policy) az ACCEPT, ami azt jelenti, hogy minden csomag áthaladhat.

Nézzük a lehetséges műveleteket a láncokkal:

- Új lánc alkotása (–N)
- Üres lánc törlése (–X)
- Irányelv megváltoztatása beépített láncon (–P)
- Egy lánc szabályainak listázása (–L)
- A lánc összes szabályának törlése (–F)
- A csomag és byte számlálók nullázása a lánc valamennyi szabályában (–Z)
- Új szabály hozzáfűzése a lánchoz (–A)
- Új szabály beszúrása a láncba adott pozíción (–I)
- Adott pozíción lévő szabály cseréje újjal (–R)
- Adott pozíciójú szabály törlése a láncból (–D láncnév szám)
- Az első, erre illeszkedő szabály törlése a láncból (–D)

## 8.2 Műveletek egy egyszerű szabályon

A csomagszűrés alapja: szabályok alkotása és módosítása. A leggyakrabban a szabály hozzáfűzése (–A) és a szabály törlése (–D) parancsokat fogjuk használni. Néhány más parancs (–I a beszúrásra és –R a cserére) ezeknek egyszerű kiterjesztése.

Minden szabály meghatároz bizonyos tulajdonságokat, melyeknek illeszkedniük kell a csomagra, és persze meghatározza azt is, hogy mit kell tenni a csomaggal, ha a tulajdonság illeszkedik (ez a szabály „célpont”-ja, angolul „TARGET”). Például: tételezzük fel, hogy minden a 127.0.0.1 címről érkező ICMP csomagot el akarunk dobni. Ez esetben a tulajdonságok közül kettőnek kell illeszkednie: a protokollnak ICMP-nek kell lennie, a csomag forráscímének pedig a 127.0.0.1-nek. A szabály célpontja pedig a „DROP” lesz. A 127.0.0.1 a visszacsatolt interfész (loopback device), mely akkor is működik, ha nincs tényleges hálózati kapcsolat. Adatcsomagok generálása a **ping** paranccsal történik. Ez egy 8-as típusú (echo request) ICMP csomagot küld, melyre alapesetben a célállomás egy 0-ás típusú (echo reply) ICMP csomaggal válaszol.

```
root@pc0:/# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
```

Szűrjük ki a 127.0.0.1 forráscímről érkező ICMP csomagokat, majd kíséreljük meg újra a ping parancsot!

```
root@pc0:/# iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
root@pc0:/# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

Láthatjuk, hogy az első **ping** eredményes volt (a ” -c 1” azt jelenti, hogy csak 1 csomagot küld). Azután hozzáfűztünk (–A) az ”INPUT” lánchoz egy szabályt, mely azt mondja, hogy a 127.0.0.1 forráscímről érkező (–s 127.0.0.1) ICMP protokollal rendelkező (–p icmp) csomagokat dobja el (–j DROP). A második **ping** használatával leteszteltük, hogy működik-e a szabály.

Két módon tudjuk a szabályokat törölni a láncból. Először is, mivel tudjuk, hogy ez az egyetlen szabály az INPUT láncban, használhatunk számozott törlést, mégpedig:

```
root@pc0:/# iptables -D INPUT 1
```

Ezzel az INPUT lánc első szabályát töröltük. A második mód a –A (hozzáfűzés) parancs tükörképe, de a –A parancsot –D -re cseréltük. Ezt akkor használhatjuk, ha komplexebb láncunk van. Ez esetben használjuk így:

```
root@pc0:/# iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
```

A –D szintaxisának pontosan egyeznie kell a –A (vagy –I, –R) szintaxisával. Ha több erre a szabályra illeszkedő szabály van a láncban, csak az első fog törlődni.

## 8.3 Forráscím és célcím meghatározása

Láthattuk, hogy a –p kapcsoló határozza meg a protokollt, és a –s való a forráscím meghatározására. Vannak azonban további opciók, melyekkel pontosabban meghatározhatjuk a csomagokat.

A forrás (–s, vagy --source, vagy --src) és a cél (–d, vagy --destination, vagy --dst) IP címeit négy módon adhatjuk meg. A legnépszerűbb a teljes domain név megadása, mint például az rs1.sze.hu . Második mód az IP cím megadása pl.: 193.224.128.1.

A harmadik és negyedik móddal az IP címek csoportjait tudjuk megadni. Pl.: 193.225.150.0/24 vagy másképpen: 193.225.150.0/255.255.255.0. Mindkettő meghatározás a 193.225.150.0-tól 193.225.150.255-ig terjedő IP címeket jelenti. A ”/” jel utáni szám a netmaszkot jelenti. A ”24” például azt jelenti, hogy a netmaszkban 24 darab egymást követő bináris jelentésű 1-es van a többi 0. Tehát a /24 ugyanazt fejezi ki, mint a /255.255.255.0, csak az előbbit rövidebb leírni.

Van néhány kapcsoló, közöttük a ”–s” és a ”–d” flagek, amelyek alkalmazhatják argumentumukat a „!” előtaggal is. Ez a logikai tagadásnak felel meg. Ez minden olyan csomagot meghatároz, amely az adott feltételnek nem felel meg. Pl.: *! -s 127.0.0.1* minden csomagra illeszkedik, amely nem a localhost-tól érkezik.

## 8.4 Protokoll meghatározása

A protokollt a –p (vagy a --protocol) kapcsolóval határozhatjuk meg. A protokoll megadható számmal is (ha ismerjük a protokoll-számot) vagy névvel, mint például TCP, UDP, ICMP. A **/etc/protocols** fájlban található protokollnevek használhatók; kis- és nagybetű nem számít. A protokoll elé is tehetünk „!”-t, ami az illeszkedést megfordítja. Tehát a ! –p TCP (régebben: a –p ! TCP is elfogadott volt) minden olyan protokollra vonatkozik, ami nem TCP.

Újabban az **iptables** jelzi, hogy a felkiáltójelet ugyan most még elfogadja a kapcsoló és annak paramétere között, de írjuk inkább a felkiáltójelet a kapcsolótól balra!

**Using intrapositioned negation (`--option ! this`) is deprecated in favor of extrapositioned (`! --option this`)**

Ez a sorrend egy kicsit jobban tükrözi a **!**-et használó kifejezés jelentését. Nézzünk erre egy példát: Mit sugall a következő kifejezés? **-p icmp --icmp-type ! echo-request**

A tapasztalatlan olvasóban könnyen azt a hatást keltheti, hogy "*olyan ICMP üzenetről van szó, ami nem echo-request*", pedig a valódi jelentése: "*bármi, ami nem ICMP echo-request*"; és ebbe beletartozik az összes nem ICMP protokollt használó csomag is!

Persze sajnos a parancssor-kiértékelés szabályai és az **iptables** modulbetöltés-kezelése miatt a logikailag igazán tökéletes **! (-p icmp --icmp-type echo-request)** szintaxis használata szóba sem jöhet, így marad a szintén nem igazán pontos szemantikát sugárzó **-p icmp ! --icmp-type echo-request** szintaxis.

## 8.5 Interfész meghatározása

A ”–i” (vagy --in-interface) és a ”–o” (vagy --out-interface) kapcsolók egy interfész nevével való egyezést határoznak meg. Az interfész fizikai eszköz, melyen keresztül a csomag bejön „–i” vagy kimegy „–o”. Az INPUT láncra érkező csomagoknak nincs kimeneti interfészük ”–o”, ezért az INPUT láncon ilyen szabály semmilyen csomagra nem fog illeszkedni (nem is fogadja el az **iptables**). És hasonlóképpen az OUTPUT láncon kimenő csomagok bemeneti interfésszel nem rendelkeznek „–i” ezért ezen a láncon a „–i” kapcsolókkal meghatározott szabályokra nem fog illeszkedni a csomag (nem is fogadja el az **iptables**). Csak a FORWARD láncon átfutó csomagok rendelkeznek mind kimeneti, mind bemeneti interfésszel.

Ha jelenleg nem működő interfészre határozunk meg szabályt, az helyes szabály lesz, de csak akkor lép érvénybe, ha felkonfiguráljuk az interfészt. Ez igen jól használható például ppp-s interfészek esetén (telefon, ADSL modem, általában **ppp0**).

Egy speciális opció, ha az interfész név után ”+” jelet teszünk, az valamennyi interfészre illeszkedni fog (függetlenül attól, hogy az interfész fel van-e konfigurálva, vagy sem) melynek a neve a + előtti szöveggel kezdődik. Például egy olyan szabályt, amely az összes Ethernet interfészre illeszkedni fog a „–i eth+” kapcsolóval használhatunk. Az interfész elé „!” –t téve az összes csomagra illeszkedni fog, melyek nem az adott interfészre érkeznek, vagy nem az adott interfészről távoznak.

## 8.6 Töredékek meghatározása

Egy útvonalválasztó szükség esetén egy IP csomagot tördelhet. Ezek a töredékek a végponton összeállnak, és újból rendelkezésünkre áll az eredeti csomag. Addig azonban a töredékeket egyenként kell kezelnünk. A töredékekkel az a baj, hogy csak az első csomag tartalmazza a komplett fejlécmezőket (IP + TCP/UDP/ICMP) a többi csomag pedig csak az IP fejrészt tartalmazza, a felette levő protokoll fejrészét nem. Ezért ezeknek további a csomagoknak a fejlécét nem tudjuk protokoll szempontjából vizsgálni.

Ha NAT (Network Address Translation, azaz hálózatcím-fordítás) használatával kapcsolódunk a hálózathoz, a csomagok újból összeállnak, mielőtt elérnék a csomagszűrőt. Ez esetben tehát nem kell a töredékektől tartani.

Minden más esetben fontos megérteni, hogy hogyan bánnak a csomagszűrők a töredékekkel. Minden olyan szabály, mely létező tulajdonságot vizsgál nem illeszkedőnek tekintendő. Ez azt jelenti, hogy a második és további töredékeket a csomagszűrő nem tudja vizsgálni. Így a szabály (-p tcp --sport www – forrás port meghatározása www-ként, azaz 80-as port) sosem fog illeszkedni egy második vagy azt követő töredékre. Csakúgy, mint az „ellentétes” szabály (–p tcp --sport ! www) sem. Mégis meg tudunk határozni szabályt a második és azt következő töredékekre a ”–f” (vagy ”--fragment”) kapcsoló segítségével. Általában biztonságosnak tekintjük a második és további töredékek átengedését a csomagszűrőn, mivel a szűrés az első töredék alapján is egyértelműen meghatározza a csomag sorsát. Ennek ellenére vannak olyan hibák, melyek kihasználásával a töredékek alkalmasak

lehetnek számítógépek feltörésére.

Példa: a következő szabály a 192.168.1.1 IP címre érkező össze töredéket eldobja.

```
root@pc0:/# iptables –A INPUT –f –d 192.168.1.1 –j DROP
```

## *8.7 Kiterjesztések az iptables-hez: új illeszkedések*

Az **iptables** kiterjeszthető, azaz mind a kernel, mind az iptables alkalmazás alkalmassá tehető újabb tulajdonságok vizsgálatának ellátására. A kernel kiterjesztések általában a kernel modulok alkönyvtárban találhatók (ha le lettek fordítva), mint például a **/lib/modules/2.6.32-5-amd64/kernel/net/ipv4/netfilter**, és a **modprobe** paranccsal tölthetjük be őket.

### 8.7.1 TCP kiterjesztések.

A TCP kiterjesztések automatikusan betöltődnek, amint a ”–p tcp” ezt meghatározza. Ez a következő opciókat teszi lehetővé:

--tcp-flags (esetleg „!”-el) majd közvetlenül utána maszk a vizsgált tcp flagekről, ezután a maszkban megadott flagek, amelyek beállítva kell, hogy legyenek. Ezek lehetővé teszik, hogy speciális vizsgálatokat végezzünk egy TCP csomagon.

```
root@pc0:/# iptables –A INPUT –p tcp --tcp-flags ALL SYN,ACK –j DROP
```

Ez azt jelenti, hogy valamennyi flaget vizsgálunk (ALL ugyanazt jelenti, mint a SYN,ACK,FIN,RST,URG,PSH), de ezek közül pontosan a SYN és az ACK flag kell, hogy be legyen állítva. Van még egy beállítási lehetőség: „NONE” azt jelenti, hogy egy flag sincs beállítva.

--syn (használható a „!”-el előtte) ez a rövidítés ugyanazt jelenti, mint a „--tcp-flags SYN,ACK,FIN,RST SYN”.

--source-port (használható a „!”-el utána) majd vagy egy port, vagy egy TCP port tartomány. A portok megadhatók számmal vagy névvel, ahogy az a **/etc/services** fájlban le van írva. A tartomány két port név vagy port szám, egy „:”-tal elválasztva, vagy ha a porttal egyenlő vagy annál nagyobb portokat akarjuk megadni, akkor a port neve „:”-tal a végén.

--sport ugyanaz, mint a --source-port

--destination-port hasonló, mint feljebb, csak ez a csomag célját határozza meg

--dport ugyanaz, mint a --destination-port

Néha hasznos, ha egyik irányban engedélyezzük a TCP forgalmat, a másik irányba pedig nem. Például, ha el akarunk fogadni csomagokat egy külső szerver felől, de nem akarjuk, hogy a szerverről hozzánk kapcsolódjanak. Elsőre azt gondolnánk, elég blokkolni a szerverről felénk érkező TCP csomagokat. Azonban sajnos a TCP kapcsolatok működéshez mindkét irányban adatforgalmat igényelnek. A megoldás az, ha csak azokat a csomagokat blokkoljuk, melyek kapcsolat igénylésére szolgálnak. Ezeket a csomagokat SYN csomagoknak hívjuk. Ezen csomagok tiltásával meg tudjuk akadályozni a gépünkre való kapcsolódást. Példa: Tiltsunk le minden 192.168.1.1-ről érkező kérelmet!

```
root@pc0:/# iptables –A INPUT –p TCP –s 192.168.1.1 --syn –j DROP
```

Ez a flag invertálható ha „!”-t teszünk elé, ez minden olyan csomagra érvényes lesz, amely nem kapcsolat kezdeményezésére irányul.

### 8.7.2 UDP kiterjesztések

Ezek a –p udp esetén automatikusan betöltődnek. --sport, --dport kiterjesztések hasonlóképpen működnek, mint TCP kiterjesztések esetében.

### 8.7.3 ICMP kiterjesztések

Ezek a kiterjesztések is automatikusan betöltődnek –p icmp esetén.

--icmp-type segítségével megadhatjuk az ICMP fajtáját névvel vagy akár a számával. Ez a mód is invertálható „!”-el előtte.

## *8.8 MAC cím alapján való vizsgálat*

Felmerülhet az a lehetőség, hogy az általunk üzemeltetett hálózatban vannak olyan gépek is, melyek számára nem akarunk forgalmat engedélyezni (vagy épp ellenkezőleg csak bizonyos gépeknek akarunk). Ha ezt IP cím alapján szeretnénk megtenni, könnyen kijátszható, mert csak az IP címet kell átállítani a gépen. Az **iptables** képes MAC cím szerint a kereteket, és az azokban utazó csomagokat vizsgálni. Ehhez a ”–m mac” kapcsoló szükségeltetik. A MAC kiegészítőt automatikusan betölti az **iptables** (amennyiben a socket filtering le van fordítva).

--mac-source forrás MAC címet lehet kijelölni

--mac-destination célzott MAC címet lehet kijelölni

Egy példa: Valósítsuk meg, hogy a belső hálózatunkon (**eth1** csatoló) csak arról a gépről fogadjon el csomagot a router, amelynek az IP címe 192.168.0.2 és a MAC címe 00:36:11:22:33:44!

```
# iptables –A INPUT –s 192.168.0.2 –m mac --mac-source 00:36:11:22:33:44 –j ACCEPT
# iptables –A INPUT –i eth1 –s 192.168.0.0/24 –j DROP
```

A 192.168.0.2 IP című és 00:36:11:22:33:44 MAC című géptől fogadja a kéréseket. Ezután minden kérést a 192.168.0.0/24 hálózatról eldob.

## *8.9 Belső hálózatok route-olása, NAT megvalósítása*

A NAT jelentése: Network Address Translation. Azaz címfordítást végzünk az IP csomagokban.

Az IP datagram cím mezői (forráscím, célcím) eredetileg nem változtak, a protokollok ezt is várják el. Miért van szükség a címfordításra?

Mert IPv4-es címből kevés van, és a NAT használatával elegendő egy darab publikus IP címet megrendelni, mégis több gépről tudjuk az internetet elérni. Ez úgy lehetséges, hogy a külső hálózatra rátesszük a routerünket, és a belső hálózaton a többi gép nem publikus IP címeket kap. Ilyenek a 10.0.0.0/8, 192.168.0.0/16 és a 172.16.0.0/12 IP tartományok. Ezek a belső IP tartományok nem route-olhatóak. A router a kimenő forgalomban kicseréli a nem publikus forrás IP címet a saját publikus IP címére. A válasz tehát a router címére fog megérkezni. A válasz csomagokba a korábban eltárolt információ alapján beírja azt a nem publikus IP címet, ahonnan az eredeti kérés jött, és továbbítja az illető gépnek.

Megvalósítások:

- MASQUERADE (2.2.x kernel)
- NAT (2.4.x kernel)

Másik esetleges probléma, hogy a szolgáltatásaink különböző szervereken futnak, viszont mi csak egy IP címre fizettünk elő, ilyenkor a szolgáltatás portját továbbítani lehet az adott szerver belső IP címe felé. Ezt port forwardnak hívják.

Technikailag az alábbi lehetőségek állnak rendelkezésünkre:

SNAT, azaz a Source-NAT

- A kifelé tartó csomagokban a forrás IP címét cseréljük ki sajátunkra:
  iptables –t nat –A POSTROUTING –o eth0 –j SNAT --to-source 193.224.130.161
- Lehet több címet is megadni: --to-source 10.0.0.2 10.0.0.8
- Lehet portok alapján is:
  iptables –t nat –A POSTROUTING –o eth0 –j SNAT --to-source 10.0.0.2:1-1023
- Például modemes kapcsolat megosztása:

iptables –t nat –A POSTROUTING –o ppp0 –j MASQUERADE

- Ha változik az internetre kapcsolódó interfész (pl.: a modemes kapcsolat csak tartalék, ha esetleg leáll a fővonal):
  iptables –t nat –A POSTROUTING –s 10.0.0.0/24 –j MASQUERADE
  Ilyenkor a 10.0.0.0 – 10.0.0.255 forrás IP címek esetén NAT-ol a routerünk.

DNAT, azaz a Destination-NAT

- Továbbítsa a router a 10.0.0.11 IP címre a datagramot:
  iptables –t nat –A PREROUTING –j DNAT --to-destination 10.0.0.11
- Küldjük a WEB kéréseket a 10.0.0.99 IP cím 8080 TCP portjára:
  iptables –t nat –A PREROUTING –p tcp --dport 80 –j DNAT --to-destination 10.0.0.99:8080

Vegyük észre, hogy az SNAT és a MASQUERADE helye mindig POSTROUTING lánc, a DNAT helye pedig mindig PREROUTING lánc! Gondolkozzunk el rajta, hogy miért!

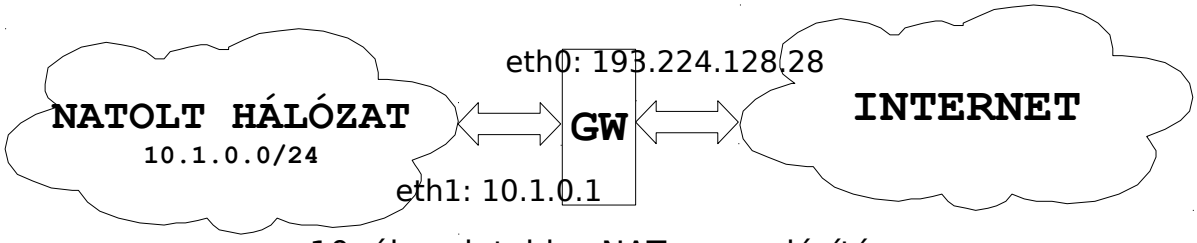
## *8.10 Protokoll segédek*

A protokollok azt várják el, hogy a forráscím, és a célcím nem változik, ezért bizonyos protokolloknál szükség van az ún. protokoll segédekre (protocol helpers). Ilyen protokoll például az FTP, ami két portot használ a kliens-szerver kommunikációhoz: a 21-es portot a vezérlő kapcsolatra, a 20-as portot az adatátvitelre. Aktív módban a kliens küldi el a szervernek a vezérlő kapcsolaton keresztül az IP címét és az adatkapcsolat számára megnyitott sockethez tartozó port számát, amire kapcsolódva a szerver kezdeményezi az adatkapcsolat felépítését. Egy nem routolható IP cím természetesen erre a célra a szervernek teljesen haszontalan. A protokoll segédnek dekódolnia kell a vezérlő kapcsolat üzeneteit és a kliens IP címénél is el kell végeznie a címfordítást! (Egy másik megoldás természetesen az FTP-nél a passzív mód használata.)

A protokoll segédek kernel modulként lefordíthatók, és amennyiben az megtörtént, megtalálhatók a **`/lib/modules/`uname -r`/kernel/net/ipv4/netfilter`** könyvtárban, és a **`modprobe`** paranccsal betölthetőek.

## *8.11 Tűzfal megvalósítások iptables segítségével*

A 10. ábrán szereplő összeállítást szeretnénk megvalósítani.




10. ábra: iptables NAT megvalósítás

A fenti ábránkon egy maximum 252 gépes hálózatot tudunk kiszolgálni. Ehhez a következő beállításokat kell megtennünk:

- iptables támogatás a kernelbe
- a forward engedélyezése a kernelben
- masquerade-ing beállítása.
- a tűzfalas védelemhez szükséges szabályok létrehozása

A forwardot – ahogy az előzőekben már említettük – vagy a **`/etc/sysctl.conf`** fájlban vagy a következő paranccsal engedélyezhetjük.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

A címfordításhoz az **`iptables`** MASQUERADE opcióját kell alkalmaznunk annak figyelembevételével, hogy a NAT-olni kívánt tartomány a 10.1.0.0/24 és a kimenő interfész az eth0:

```
iptables -A POSTROUTING -t nat -s 10.1.0.0/24 -o eth0 -j MASQUERADE
```

A NAT működéséhez, már csak a NAT box mögötti hálózatban lévő gépeken kell beállítanunk az IP-címeket, az átjárót és a DNS szervereket.

A fenti konfiguráció már majdnem jó lenne számunkra, de a tűzfalunk túlságosan nyitva van. A „*mindent tiltunk kivéve amit engedélyezünk*” elvet fogjuk alkalmazni az INPUT és a FORWARD láncon. Ehhez a két szabálylánc alapértelmezett szabályát (policy) kell DROP-ra beállítani:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

A helyes szintaktikára figyeljünk oda!

Ha a fenti utasításokat távolról adjuk ki, igen veszélyesek, hiszen az INPUT láncra kiadott DROP paranccsal azonnal kizárjuk magunkat a rendszerből amit konfigurálunk, ezért ezt megelőzendő, engedélyezzük a már felépült kapcsolatokat:

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Ezzel elértük, hogy a már meglévő pl. ssh kapcsolatunk nem szakad meg. Ahhoz, hogy ne csak a meglévő kapcsolatunk éljen, engedélyeznünk kell a 22/TCP-re érkező NEW állapotú csomagokat:

```
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
```

Bizonyos programok megkívánják, hogy a loopback interfész működjön, ezért célszerű ha a 127.0.0.1 címről érkező összes csomagot engedélyezzük, de ezt szeretnénk a második helyre beszúrni:

```
iptables -I INPUT 2 -s 127.0.0.1 -j ACCEPT
```

Tehát távolról a parancsok kiadásának helyes sorrendje:

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -s 127.0.0.1 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j ACCEPT
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

Ezzel garantáltuk a gépünk helyes működését. Amennyiben a konzol előtt ülve van lehetőségünk konfigurálni, úgy „majdnem teljesen mindegy”, milyen sorrendben adjuk ki a fenti parancsokat.

Azért csak „majdnem teljesen mindegy”, mert ha lényegesen hosszabb szabálylánc aink vannak, akkor nem elég pusztán a logikai hibátlanságra törekednünk (mit engedélyezünk és mit nem), hanem a teljesítmény is számít. Márpedig teljesítmény szempontjából nem mindegy, hogy a nagyobb gyakoriságú csomagok sorsa a szabálylánc elején vagy végén dől-e el.

## *8.12 További iptables illeszkedések*

Aki mélyebben meg akarja ismerni az `iptables` működését, lehetőségeit, annak erősen ajánlott a `man iptables` tanulmányozása. Ízelítőül érdekes néhány lehetőség:

`-m state --state` `ESTABLISHED|INVALID|RELATED|NEW` (a csomagok állapotának vizsgálata)

`-m connlimit --connlimit-above 100 -j REJECT` (az egyidejűleg nyitott kapcsolatok számának korlátozása)

`-m - ipv4options --any-opt` (IPv4 flag-ek vizsgálata; itt: van-e bármi

opció)

`-m owner --uid-owner 0 -j LOG -log-prefix "root kapcsolatok"` (UID alapján való szűrés, CSAK OUTPUT láncon működik)

`-m quota --quota 65535 -j DROP` (megadott mennyiségű csomag után letiltja a processzt)

`-m string --string .mp3 -j DROP` (a megadott sztringre keres a csomag data mezőjében.)

`-m ttl --ttl-eq` (egyenlő), `--ttl-gt` (nagyobb), `--ttl-lt` (kisebb) (a TTL (Time To Live) értéket figyeli, a `--ttl-set` egy adott értékre állítja be (csak **`mangle`** táblánál))

`-m unclean` (nem szokványos vagy hibásan formázott csomag)

## *8.13 Rendszernaplózás iptables segítségével*

Egy vállalat hálózatán és szerverein szükség lehet az átmenő és a szerver felé nyitott kapcsolatok nyomon követésére. Ezzel például kiszűrhetjük a túlságosan gyakran látogatott oldalakat, melyek a munkavégzés hatékonyságát csökkentik, stb. Az **`iptables`** lehetőséget ad számunkra, hogy általunk megadott minta alapján naplózzunk, például naplózzuk a vállalati webszerver 80 portjára érkező összes kapcsolat kezdeményezést (a 80-as portra érkező SYN kéréseket). Két lehetőségünk van a naplózásra. Az egyik a syslog, melybe a rendszer minden más eseménye is belekerül, a másik az ULOG.

A naplózáshoz egy szabályt kell definiálnunk valamelyik láncon, majd a -j kapcsoló után a LOG, vagy ULOG targetet kell megadnunk. Mind a két naplózás esetében lehetőségünk van prefix megadására, mely a tárolt bejegyzések elé kerül. A későbbiekben ezek alapján szűrhetünk.

Első példánkban egy 192.168.10.21 IP címmel rendelkező webszerver 80-as portjára érkező SYN kéréseket fogjuk figyelni az INPUT láncon és naplózzuk a LOG targetbe, „**`VALLALATI_WEBSZ:`** ” prefix használatával:

```
iptables -A INPUT -p tcp --dport 80 --syn -d 192.168.10.21 -j LOG --log-prefix
"VALLALATI_WEBSZ: "
```

Második példánk az **`ulogd`**-be fog naplózni, és a FORWARD láncon átmenő TCP SYN kéréseket fogja naplózni, „**`IPTABLES_ULOG1:`** ” prefix használatával:

```
iptables -A FORWARD -p tcp --syn -j ULOG --ulog-prefix "IPTABLES_ULOG1: "
```

A fenti példákban a **`--syn`** a **`-p tcp`** modul része, ha ezt nem használjuk, az **`iptables`** nem tudja értelmezni a kapcsolót és hibaüzenettel leáll.

Szintén fontos megemlíteni, hogy ellentétben az ACCEPT, DROP, REJECT, QUEUE targetekkel, a LOG, és az ULOG targetek használatakor a szabály illeszkedés esetén nem jelenti a csomag feldolgozásának végét, hanem továbbengedi a csomagot a szabályláncon!

## 8.14 Gyakorló feladatok

A gyakorló feladatokhoz már minden szükséges előismeretet megadtunk, ezért javasoljuk, hogy először igyekezzenek azokat önállóan megoldani és csak utána nézzék meg a „kincstári” megoldást! Természetesen lehetséges, hogy van más helyes megoldás is!

*1. feladat: Az iptables segítségével engedélyezze az INPUT láncon a 00:11:22:aa:bb:cc MAC címről érkező ICMP redirect csomagokat, de tiltsa le az összes többi beérkező redirectet!*

```
iptables -A INPUT -m mac --mac-source 00:11:22:aa:bb:cc -p icmp --icmp-type redirect
-j ACCEPT
iptables -A INPUT -p icmp --icmp-type redirect -j DROP
```

*2. feladat: Az iptables segítségével utasítsa vissza az adott gépre az eth1 interfészen keresztül beérkező FTP kapcsolatfelépítési kéréseket! (Az esetlegesen fennálló FTP kapcsolatok nem szakadhatnak meg!)*

```
iptables -A INPUT -i eth1 -p tcp --dport 21 --syn -j REJECT
```

*3. feladat: Írjon olyan tűzfal szabályt, amely a FORWARD láncon minden töredéket eldob!*

```
iptables -A FORWARD -f -j DROP
```

*4. feladat: Írjon olyan tűzfal szabályt, amely a FORWARD láncon az összes a 10.1.12.0/24 hálózatba irányuló web kérést átengedi!*

```
iptables -A FORWARD -d 10.1.12.0/24 -p tcp --dport 80 -j ACCEPT
```

*5. feladat: Írjon olyan tűzfal szabályt, amely az INPUT láncon eldobja az összes olyan TCP szegmenst, amelyben a FIN és a SYN bitek mindegyike be van állítva*!

```
iptables -A INPUT -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
```

# 9 Rendszernaplózás

A UNIX rendszerek alatt a syslog a rendszergazda legjobb barátja. A rendszerfolyamatok naplóüzeneteinek jelentős részét a syslog tárolja. Két rendszernaplózó terjedt el a hosszú évek folyamán. Az első a **sysklogd**, a másik a magyar fejlesztésű és sokkal újabb **syslog-ng**. A rendszernaplózók a logokat a **/dev/log** socketen keresztül kapják, majd ezt fájlokba, és/vagy távoli gépre mentik. Mind a két naplózó képes lokális és távoli naplózásra. A **syslog-ng** már nemcsak UDP, hanem TCP protokollon keresztül is kommunikál, ami azért fontos, mert a TCP forgalmat **stunnel** segítségével lehet titkosítani, míg az UDP csomagokat alkalmazásszinten kellene titkosítani, de erre a rendszernaplózók nem képesek.

A naplóüzenetek forrása valamilyen *szolgáltatás* (facility) és minden üzenet valamilyen *prioritási osztály*ba tartozik (level/priority/severity).

Megjegyzés: az egyes kifejezések magyar fordítását a **man syslog.conf** magyar fordításához igazítottuk. Ebben a jegyzetben korábban a facility-re magyarul az *eszköz* szót használtuk.

## *9.1 Szolgáltatások*

A facilityk sehol nincsenek rögzítve, de van néhány elterjedt, ezek:

- **auth** (authpriv) – authentikációs
- **cron** - cronjobs (cron daemon) üzenetei
- **daemon** - minden egyéb daemon üzenet
- **kern** - a kernel üzenetei
- **lpr** - nyomtatással kapcsolatos üzenetek
- **mail** - levelezéssel kapcsolatos
- **user** - felhasználói
- **local0**-**local7** – lokális, előre nem definiált (a **local7**-et a Cisco és Windows szerverek használják általában)
- **syslog** - a **syslog** saját üzenetei.

Az RFC 3164 és az azt felváltó RFC 5424 mintegy utólagosan dokumentálja a **syslog** protokollt, de nem ad kötelező érvényű előírást a facility megnevezésekre. A nekik megfelelő numerikus értékekről annyit mond, hogy a [0, 23] tartományba esnek.

## 9.2 Prioritások

Szemben a facilitykkel, a prioritások (naplózási szintek) szabványosítottak, 8 szint van, a szintek prioritásuk szerint (a kisebb számérték jelenti a magasabb prioritást) a következők:

- 0 - **emerg** ; vészhelyzet
- 1 - **alert** ; riasztás
- 2 - **crit** ; kritikus
- 3 - **err** ; hiba
- 4 - **warning** ; figyelmeztetés
- 5 - **notice** ; megjegyzés
- 6 - **info** ; tájékoztatás
- 7 - **debug** ; nyomkövetés

Értelemszerűen, egy jól működő rendszer esetében **emerg** nem fordul elő vagy csak ritkán. A **debug** szint képes „elárasztani" a naplókat, hiszen minden üzenetet elment, hogy minél egyszerűbb legyen a hibák detektálása.

## 9.3 A sysklogd

A **sysklogd** csomag két részből áll: **syslogd** és **klogd**. A **syslogd** a **/dev/log**-ot dolgozza fel, a **klogd** a **/proc/kmsg**-et használja, és a kernel üzeneteit kezeli.

A konfigurációs állomány a **/etc/syslog.conf** fájl. Ennek minden sora egy szabályt ad meg. A „**#**" kezdetű (megjegyzést tartalmazó) és az üres sorokat a rendszer figyelmen kívül hagyja. Egy sor szintaktikája a következő:

**<selector> <action>**

A *kiválasztó* (selector) kiválaszt bizonyos naplóüzeneteket, a *végrehajtó* (action) pedig azt fejezi, ki, hogy mi történjen a selector által kiválasztott üzenetekkel. Ez utóbbi az egyszerűbb, ezért ezzel kezdjük.

### 9.3.1 Végrehajtó

Az action mezőben megadható:

- fájlnév teljes útvonallal ellátva (ebbe íródik be az üzenet)
- named pipe (más néven: FIFO, ennek neve előtt egy „**|**" jelnek kell állnia)
- terminál vagy konzol (pl. **/dev/console**)

- hosztnév (előtte egy „**@**” karakternek kell állnia: erre a gépre továbbítódik az üzenet)
- felhasználói nevek listája vesszővel elválasztva (ezek a felhasználók kapnak értesítést, ha be vannak jelentkezve), illetve egy „*****” karakter (az összes bejelentkezett felhasználó értesítést kap)

Fájlnév megadása esetén, ha közvetlenül a fájlnév megadás előtt „**-**” van, akkor nem rögtön írja a lemezre a változásokat, hanem gyorsítótárazza; ez rendszerösszeomlás esetén adatvesztést okozhat. Tipikusan NEM gyorsítótárazzák a warningnál kisebb prioritási számértékű (azaz fontosabb!) üzeneteket.

### 9.3.2 Kiválasztó

A selector mező két részből áll, ezeket pont választja el egymástól:

**facility.priority**

Megjegyzés: a **priority** helyett a **severity** (súlyosság) kifejezés is használatos.

Ezek felvehetik a korábban ismertetett értékeket, illetve használható még néhány speciális jelölés: szerepelhet benne a "*****" karakter, megadható egy sorban több bejegyzés is, és ha vesszővel ("**,**") választjuk el őket, az egészen mást jelent, mintha pontosvesszővel (";") választanánk el. Példákon keresztül mutatjuk be őket:

```
*.alert      /var/log/hiba.log
```

minden facilityből, az **alert** és nagyobb prioritású (***.emerg**) üzeneteket írja a megadott fájlba.

```
*.=err      /var/log/hiba.log
```

minden facilityből a hibákat, és csakis a hibákat naplózza fájlba. (Megjegyzés: a ***.emerg** és a ***.=emerg** ugyanazt jelenti, hiszen az **emerg**-nél nagyobb severity nincs.)

```
*.alert;kern.none /var/log/hiba.log
```

a **kern** facilityből semmit (**none**), egyébként minden **alert** és **alert**-től nagyobb prioritású (azaz ***.emerg**) üzenetet ír ki a logba.

```
*.alert;auth,kern.none /var/log/hiba.log
```

az **auth** és **kern** facilitykből semmit, a többiből minden **alert** és **alert**-től nagyobb prioritású üzenetet naplóz.

Néhány további példa a konfigurációs fájlból:

```
*.=debug;\
    auth,authpriv.none;\
    news.none;mail.none     -/var/log/debug
```

Az **auth**, **authpriv**, **news**, **mail** facilityk kivételével minden **debug** és csak **debug** szintű üzenetet naplóz, gyorsítótárazással.

```
*.=info;*.=notice;*.=warning;\
    auth,authpriv.none;\
    cron,daemon.none;\
    mail,news.none          -/var/log/messages
```

Az **auth**, **authpriv**, **cron**, **daemon**, **mail**, **news** kivételével, minden **info**, **notice**, **warning** szintű üzenetet naplóz, gyorsítótárazással.

```
*.*         @logszerver
```

Minden üzenetet elküld a **logszerver** nevű hosztnak, ami az UDP/514-es porton fogadja a logokat, amennyiben a **logszerver** nevű gépen a **syslog** szerver a **-r** kapcsolóval lett elindítva; ezt Debian alatt a **/etc/init.d/sysklogd** szkriptben lehet beállítani.

```
*.=emerg   *
```

Minden **emerg** szintű üzenetet elküld az összes bejelentkezett felhasználónak. Természetesen nem levélben, hanem kiírja a felhasználó terminájára, úgy mint a **wall** (write all) parancs.

## *9.4 A syslog-ng*

A konfigurációs állománya a **/etc/syslog-ng/syslog-ng.conf** fájl. Első ránézésre sokkal bonyolultabb, mint a **syslog.conf**, de ha az ember megszokta, és

megtanulta kezelni, pontosan szűrt naplózásokat képes megvalósítani vele. Nemcsak a facility és level szerint képes logolni, hanem ezeken belül regexp-ekkel is képes szűrni a log tartalmat. Az állomány három fő részre osztható (de nem szigorúan), a sorrendek felcserélődhetnek, de az átláthatóság érdekében érdemes meghagyni a felépítést.

- Az első rész (**options**) az, ahol a szerver beállításait végezhetjük. A kapcsolatokra, a logfájlok jogaira, a logforrásokra, a cache méretre, a könyvtárak létrehozására, DNS használatára való beállításokat tehetjük meg többek között.
- A következő rész a deklarációs rész, ahol a forrásokat (**sources**), a naplófájlokat és egyéb naplózási célokat (**destinations**) és a szűrőket (**filters**) állíthatjuk be.
- Az utolsó rész (**logs**) az, ami a szűrőket, fájlneveket, forrásokat rendeli össze.

*A Debian Lenny disztribúcióban olyan **syslog-ng.conf** fájl szerepel, ami a **syslog** alapbeállításai szerinti naplózást valósít meg, ráadásul megjegyzésben szerepelnek az ekvivalens **syslog** beállítások is. A fájl a tárgy oldalán elérhető, önálló tanulmányozását erősen ajánljuk!*

A naplófájlnevek szintaktikája:

```
destination hivatkozási_név { file("/var/log/logfile_neve"); };
```

Távoli naplózás

```
destination loghost {
        tcp("loghost" port(514));
};
```

Ez a **loghost** 514/TCP portjára küldi a logokat. A loghosztnál nem a szervert kell speciális módban indítani, hanem a konfigurációs állományban kell source-nak beállítani például az 514/TCP portot a következő módon:

```
source remote {
        tcp(
            port(514)
            keep-alive(yes)
# a kapcsolat felépülve marad
            max-connections(100)
# maximum 100 gép (kapcsolat) épülhet ki a szerverrel.
        );
};
```

Ahhoz, hogy a távolról jövő logok megjelenjenek, a log {} szekciókba forrásként be kell jegyezni source(remote); forrást.

```
log {
        source(s_all);
        source(remote);
        filter(f_ipt);
        destination(df_ipt);
};
```

A filterek szintaktikája:

```
filter hivatkozási_név { facility(facility_lista); };
```

```
filter hivatkozasi_nev { facility(facility_lista)
[and|or] level(level_lista)
[and|or] (
    facility(facility_lista)
    [and|or]
    level(level_lista)
 ); };
```

Mint láthatjuk, elég összetett filtereket is létrehozhatunk, a facilityn és levelen kívül a szövegben előforduló kifejezésekre is kereshetünk, ezzel specifikus logolások is megvalósíthatóak. Erre példát itt nem mutatunk.

Források, filterek, fájlnevek összerendelése:

Miután felépítettük a filter és fájldeklarációinkat, hogy működni is tudjanak, logfájlonként össze kell őket rendelni. Íme, egy példa:

```
log {
    source(source_azonosito_1);
    source(source_azonosito_2);
    filter(filter_azonosito_1);
    filter(filter_azonosito_2);
    destination(logfile_azonosito);
}
```

Amint látjuk, egy logfájlba több filter szerint is tehetünk dolgokat, és bármelyik filterre illeszkedik, a log belekerül a fájlba. Természetesen több forrás is megadható (gondoljunk bele, ha egy közös – a távolról naplózó gépek bejelentkezéseit is egy fájlba szeretnénk a könnyebb és jobb átláthatóság kedvéért – **`auth.log`** fájlra van szükségünk, vagy egy közös fájlra, ami csak az **`ssh`** logolásokat figyeli).

```
log {
        source(s_all);
        destination(loghost);
};
```

Távoli gépre küldi az összes source-ot.

Egy egyszerű példa, mely távolról kapott, **iptables**-szel logoltatott, SYN kéréseket tartalmazó logot menti el a **/var/log/ipt.log** fájlba. Azokat a kapcsolat-kezdeményezéseket szeretnénk eltárolni, melyet nem a PUBLIC tartományból, vagy tartomány felé indítottak.

Az alkalmazott szabályláncok a távoli gépen:

```
-A FORWARD -d 193.224.130.160/255.255.255.224 -p tcp --syn -j LOG --log-prefix
"IPTABLES_PUBLIC:  "

-A FORWARD -s 193.224.130.160/255.255.255.224 -p tcp --syn -j LOG --log-prefix
"IPTABLES_PUBLIC:  "

-A FORWARD -s 192.168.100.0/255.255.255.0 -p tcp --syn -j LOG --log-prefix
"IPTABLES_LABOR:  " --log-tcp-options --log-ip-options

-A FORWARD -s 192.168.150.0/255.255.255.0 -p tcp --syn -j LOG --log-prefix
"IPTABLES_WIFI:  "
```

Ezek után az egyszerű log részlet:

```
options {
        chain_hostnames(0);
        time_reopen(10);
        time_reap(360);
        log_fifo_size(2048);
        create_dirs(yes);
        group(adm);
        perm(0640);
        dir_perm(0755);
        use_dns(no);
};

source remote {
        tcp(
            port(514)
            keep-alive(yes)
            max-connections(100)
        );
};

# a destination a /var/log/ipt.log fájl lesz:
destination df_ipt { file("/var/log/ipt.log"); };

# Ez azokat a sorokat szűri, melyek tartalmazzák az IPTABLES sort, de nem
tartalmazzák a PUBLIC sort
filter f_ipt { match("IPTABLES") and (not match("PUBLIC") };

log {
        source(remote);
```

```
        filter(f_ipt);
        destination(df_ipt);
};
```

## 9.5 A logger

Sajnos nem minden alkalmazás gondolja úgy, hogy neki a syslogba kellene naplóznia. Ilyen az Apache **access.log**, és a proftpd **xferlog** része. Ezekre néha szükség lehet, hogy a syslogba tegyük, és akár helyi, akár távoli naplózással elmentsük. Erre Linux alatt a logger parancs az, ami segítséget nyújt. A logger a STDIN-re érkező adatot a **/dev/log** streambe helyezi az általunk megadott **facility.priority** beállításokkal, ezekre mi a későbbiekben már tudunk szűrni. Példa az Apache **access.log**-jának a syslogba helyezésére:

```
tail -n0 -f /var/log/apache/access.log | logger -p local4.info
```

Ez az összes web hozzáférési bejegyzést a **local4.info**-ba sorolja, amit mi egy külön logfájlba helyezhetünk. Persze meg kell jegyezni, hogy amikor az Apache rotálja a logokat, a **tail**-ünk még mindig a „régi” **access.log** fájlt fogja figyelni, amibe naplózás már nem történik, tehát meg kell oldani, hogy logrotálás után az új fájlt figyeljük.

## 9.6 A logrotate

A naplófájlok mérete rohamosan növekedhet. Ezért, hogy ezt kiküszöböljük és, hogy egyszerűbb legyen keresni bennük (pl.: ha tudjuk mikor történt az esemény nem kell 2 évet visszakeresni), bizonyos időközönként rotáljuk és tömörítjük a naplófájlokat. Ezt a **logrotate** könnyíti meg számunkra a Linux rendszer alatt. A **/etc/cron.daily** könyvtárban található a **logrotate** nevű szkript, ami a **/etc/crontab**-ban beállított időközönként lefut. Ez a szkript indítja el **/usr/sbin/logrotate** programot, amely a **/etc/logrotate.d** könyvtár alatt lévő konfigurációs fájlok segítségével rotálja, tömöríti a naplófájlokat. (Egyéb hasznos dolgokra is lehet használni, ilyen például az időközönkénti mentés, virtuális gépek fájlrendszerének rotálása, médiára való kiírása.)

# 10 Hálózati szolgáltatások UNIX alatt

## *10.1 Szolgáltatások indítása inetd segítségével*

Ahhoz, hogy egy adott szolgáltatást nyújtsunk az Internet felé UNIX segítségével, futtatnunk kell egy olyan programot, mely az adott protokollal érkező kérésre válaszolni tud. Például, ha egy WEB szervert (HTTP kiszolgálót) szeretnénk üzemeltetni, állandóan futnia kell egy olyan programnak, mely figyeli a 80/TCP porton érkező kéréseket, és képes azokat megfelelően megválaszolni/kiszolgálni. Régebben nem voltak olyan nagy teljesítményűek a számítógépek, mint napjainkban, ezért UNIX esetében is igyekeztek erőforrást kímélő módon megoldani a szolgáltatások nyújtását. A módszer lényege, hogy nem futtatunk minden szolgáltatáshoz külön programot, hanem csak akkor indítjuk el őket, amikor ténylegesen szükség van az adott szolgáltatásra. Ehhez semmi másra nincs szükségünk, csak egy úgynevezett „szuperszerverre", amely az összes, általunk definiált szolgáltatáshoz tartozó hálózati kérést figyeli, és ha szükséges, elindít egy daemon-t, ami végül lekezeli a kérést ténylegesen. A szuperszerver neve általában minden UNIX rendszerben: **inetd**. Az általa nyújtott szolgáltatásokat a **/etc/inetd.conf** fájlban állíthatjuk be. (A szintaxist csak röviden ismertetjük, mert ma már egyre ritkábban használják.) A fájlban minden sor egy-egy szolgáltatást ad meg az alábbiak szerint:

```
service-name socket-type protocol {wait,nowait} user server-program server-prg-args
```

Egy konkrét példa:

```
pop3 stream tcp nowait root /usr/sbin/tcpd ipop3d
```

A konfigurációs állomány első oszlopában álló szolgáltatásnevekhez a **/etc/services** fájlban vannak hozzárendelve az adott protokollok port számai. Jelen esetben például:

```
# service-name  port/protocol  [aliases ...]   [# comment]
pop3            110/tcp         pop-3           # POP version 3
```

Vegyük észre, hogy ha a **/etc/services** fájlban a **pop3** protokollt ezután úgy neveznénk, hogy „kutya", akkor az **inetd.conf**-ba is kutyát kellene írnunk!

Visszatérve a **/etc/inetd.conf** fájlra: a TCP kapcsolat stream típusú socketet igényel, ebben az esetben nowait a szokásos beállítás, míg datagram esetén: wait. A root felhasználó nevében történő szolgáltatásnyújtást kifejezetten ROSSZ példaként mutatjuk be, erre a célra mindig létre kell hozni egy erősen korlátozott jogkörökkel rendelkező felhasználót!

Példánkban az **inetd.conf** hatodik oszlopában a tényleges szerver program helyett található **tcpd** daemon (TCP wrapper) arra szolgál, hogy megszabhassuk, hogy milyen IP

címekről jelentkezhetnek be kliensek, hogy igénybe vehessék a szolgáltatásainkat. (A tényleges szerver program a **tcpd** argumentuma, és a **tcpd** fogja elindítani, ha úgy találja, hogy a szolgáltatás igénybevétele engedélyezett.)

Az engedélyezés folyamata a következő:

- Az **/etc/hosts.allow** fájlban megadott gépek számára a hozzáférés engedélyezett.
- Az előző fájlban nem, de a **/etc/hosts.deny** fájlban megadott gépek számára a hozzáférés tiltott.
- Amely gépek egyik fájlban sem szerepelnek, azokra a hozzáférés engedélyezett.

Példánkban a 7. oszlopban áll a tényleges szerver program, aminek további argumentumai is lehetnének, de itt most nincsenek.

Ha szeretnénk megszüntetni egy szolgáltatást, annyit kell tennünk, hogy egy komment jelet („#”) írunk az **inetd.conf** megfelelő sorába, és újraindítjuk az **inetd** daemon-t:

```
# /etc/init.d/inetd restart
```

A különféle szolgáltatások **inetd**-ből történő indítása biztonsági szempontból veszélyeket rejt magában. (Például ha valamelyik szolgáltatást feltörik, az hatással lehet a többire is.) Emiatt mostanában a rendszergazdák szívesebben alkalmazzák azt a módszert, hogy minden szolgáltatás számára külön daemon-t futtatnak – ennek a módszernek a nagyobb erőforrásigénye ma már nem okoz problémát. Az **inetd**, és annak újraírt változata az **xinetd** gyakorlatilag *obsolated* (elavult, már nem használatos) kategória. Napjainkban csak nagyon kevés szolgáltatás veszi igénybe.

Debian GNU/Linux alatt a **/etc/inetd.conf** fájlba az **update-inetd** paranccsal tehetünk bejegyzéseket.

## *10.2 Szolgáltatások egyedi indítása*

Amennyiben precízebben szeretnénk kontrollálni szolgáltatásainkat, egyenként kell elindítanunk a szolgáltatást végző *daemon*okat. Ez persze nem azt jelenti, hogy kézzel kell elindítani őket, hanem vannak rá megfelelő szkriptek, amelyek a rendszer elindulásakor meghívásra kerülnek. Például a rendszerindító szkriptek közül a DNS szerver indításáért felelős a **/etc/init.d/bind**.

Ha kíváncsiak vagyunk, hogy egy adott szolgáltatás (pl.: named) fut-e, a ps programmal ellenőrizhetjük:

```
# ps ax |grep named
135 ?           S          0:07   /usr/sbin/named -u  daemon
6707 pts/1      S          0:00   grep named
```

Az, hogy egy adott daemon indításakor mit kell beírni a parancssorba, a program típusától függ, pl.: az NFS szerver szolgáltatásait a /etc/init.d/nfs-kernel-server szkript segítségével ki-be kapcsolhatjuk.:

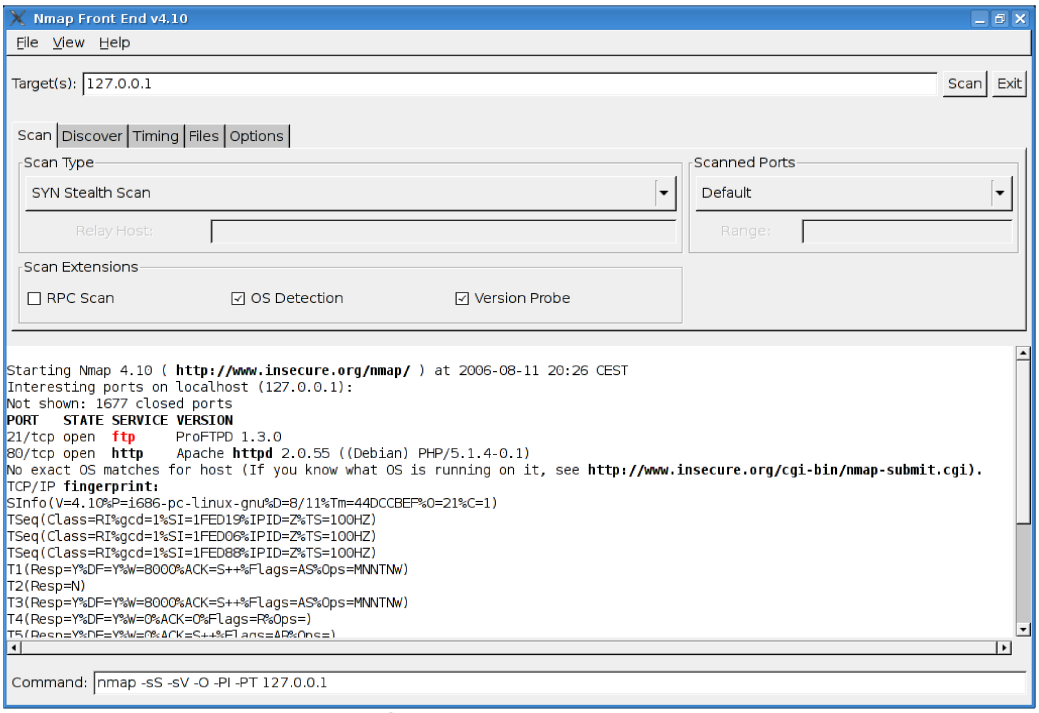
```
# /etc/init.d/nfs-kernel-server start
Starting NFS services:
     /usr/sbin/exportfs -r
     /usr/sbin/rpc.rquotad
     /usr/sbin/rpc.nfsd 8
     /usr/sbin/rpc.mountd --no-nfs-version 3
     /usr/sbin/rpc.lockd
     /usr/sbin/rpc.statd
# ps ax|grep nfs
6733 pts/1        SW           0:00   [nfsd]
6734 pts/1        SW           0:00   [nfsd]
6735 pts/1        SW           0:00   [nfsd]
6745 ?            S            0:00   /usr/sbin/rpc.mountd --no-nfs-version 3
# /etc/init.d/nfs-kernel-server stop
Stopping NFS kernel daemon: rquotad nfsd mountd lockd statd
Unexporting directories for NFS kernel daemon… done
```

## *10.3 Szolgáltatások felderítése, rendszerbiztonság*

Mint látható, a **ps** program segítségével könnyen ellenőrizhetjük, fut-e egy adott daemon. Ha ehhez hozzávesszük, hogy milyen szolgáltatásokat nyújt az **inetd**, nagyjából képet kapunk arról, hogy milyen szolgáltatásokat nyújt saját szerverünk. Ellenben mi van azokkal a programokkal, amelyek esetleg egyszerre több porton szolgáltatnak, vagy azokkal, amelyeket olyan szkript indít el, aminek a létezéséről nem is tudunk, és pl. holnap fogják vele feltörni a szerverünket? Ezeket csak bizonyos szolgáltatásokat felderítő programok segítségével „fülelhetjük” le.

Az egyik ilyen rendkívül hasznos program az **nmap**, melyet a Debian GNU/linux csomagkészletében is megtalálhatunk, és könnyedén fel is telepíthetünk (**apt-get install nmap**).

Grafikus módban, amihez fel kell telepítenünk az **nmapfe** (fe=front end) nevű csomagot (**apt-get install nmapfe**) a program kimenete színes, és egyből láthatjuk a kritikus beállítási pontokat. Nézzük meg egy példát:



11. ábra: Nmap Frontend

A „State” oszlopban lévő open állapot azt jelzi, hogy szerverünk az adott porton kiszolgálja a portra érkező kéréseket, ezen felül a színekből látszik, hogy melyik protokollokat tekinti a szoftver különösen veszélyesnek: **`ftp`**. Hogy miért éppen ezt a protokollt? Mert az FTP kódolatlanul küldi a jelszavakat a hálózaton!

Miután tisztában vagyunk a nyitott portjainkkal, a **`fuser`** parancs segítségével a következő módon meggyőződhetünk, hogy melyik processz figyel az adott porton:

```
# fuser -vn tcp 80
                     USER        PID ACCESS COMMAND
80/tcp               root      22539 f....  apache2
                     root      22540 f....  apache2
```

Szintén hasznos parancs a **`netstat`**, ezzel a fennálló hálózati kapcsolatainkat tekinthetjük meg:

```
# netstat |head -n 5
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 dev.tilb.sze.hu:10050   tuzfal.tilb.sze.h:50201 TIME_WAIT
tcp        0      1 dev.tilb.sze.hu:45991   log.tilb.sze.hu:shell   SYN_SENT
tcp        0     52 dev.tilb.sze.hu:ssh     host-94-248-131-20:1919 ESTABLISHED
```

# 11 DHCP (Dynamic Host Configuration Protocol)

Képzeljük el azt az egyébként igen gyakori esetet, amikor több száz számítógép van egy hálózatban. Ha ezeken a számítógépeken az egyes hálózati beállításokat egyesével kellene végrehajtani, az óriási munka lenne. Az ilyen nagy hálózatok címkiosztásához találták ki a DHCP-t, mely egy szerver-kliens alapú protokoll. A DHCP protokoll segítségével az egyes gépek be tudják állítani maguknak az IP-címüket, a hálózati átjárót, névszervereket és egyéb hálózati tulajdonságokat. Ezeket a beállításokat DHCP szervertől veszik.

A DHCP szerver segítségével IP címeket lehet kiosztani dinamikusan, vagy Ethernet cím alapján. A kiosztás csak egy meghatározott időre szól, azután újra kell igényelni. Ez természetesen teljesen automatikusan zajlik, a felhasználó ebből nem vesz észre semmit. A szerver nemcsak IP címek kiosztására szolgál, hanem a hálózathoz tartozó beállításokat is képes átadni a kliens gépnek (hálózatcím, maszk, átjáró, DNS szerver, tftp szerver a remote boothoz, ntp szerver időszervernek, stb.). A protokoll leírása a 2131-es számú RFC-ben található meg.

Debian Lenny alatt a DHCP szerver programot az **apt-get install dhcp3-server** paranccsal telepíthetjük fel.

Debian Squeeze alatt a **dhcp3-server** csak egy ún. *áttérést segítő csomag* (transitional package), aminek a telepítésekor egyben az új az **isc-dhcp-server** nevű csomag is telepítésre kerül, ami valójában a DHCP szerver 4-es verziója! Mivel az **apt-get remove dhcp3-server** parancs után gépünkön fent maradt az **isc-dhcp-server**, ezért tanuljuk meg, hogy a csomag új neve **isc-dhcp-server** és mind telepítésekor, mind eltávolításakor azt kell használni!

Ellentétben más szerver programokkal, a DHCP szerver indítása feltelepítés után sikertelen lesz. Miért? Mert még semmilyen beállítással sem rendelkezik, és így nincs mit szolgáltatnia! (Az olvasó gondolja végig, hogy például egy web vagy FTP szerverrel szemben miért nincs értelmesen használható alapbeállítás DHCP szervernél!)

Indítás előtt tehát a DHCP szervert konfigurálni kell! A **dhcpd** program konfigurációs fájlja Lenny alatt a **/etc/dhcp3/dhcpd.conf**, Squeeze alatt viszont a **/etc/dhcp/dhcpd.conf**.

A maximális frissítési (bérleti) idő megadása lehetséges a **max-lease-time**, az alapbeállítás pedig a **default-lease-time** opcióval. Itt másodpercben kell megadni az időtartamokat. A leggyakrabban használt bejegyzések az **option**, **subnet**, **host**. Az **option** általánosan, illetve alhálózatra/gépre vonatkozóan adhat meg paramétereket:

```
option domain-name ”tilb.sze.hu”;
option domain-name-servers 193.224.130.161, 193.224.128.1;
option routers 193.224.130.161;
```

A **subnet** egy alhálózaton belüli paramétereket határoz meg. A címek dinamikusan kerülnek kiosztásra, a **range** által megadott tartományokból. Itt is használható az **option** kulcsszó, ez az alhálózatra adja meg az opciókat. Amennyiben nincs megadva ilyen, akkor az általános részben definiáltak kerülnek értelmezésre. A tisztánlátás érdekében érdemes

minden alhálózathoz megadni ezeket a paramétereket, amint az a példában is látható:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
     range 192.168.1.20 192.168.1.30;
     option broadcast-address 192.168.1.255;
     option routers 192.168.1.1;
     option domain-name servers 192.168.1.1, 193.225.12.58, 193.224.128.1;
}
```

A **host** kulcsszóval egyetlen gépre vonatkozóan adhatjuk meg az IP címet, az átjárót, a DNS szervert. A hivatkozás Ethernet cím alapján történik:

```
host pc0 {
     hardware ethernet       00:50:04:34:A7:5F;
     fixed-address           192.168.1.7
     next-server             193.224.130.174;
     filename                "netboot/pxelinux.0";
}
```

A fenti példa egy MAC címhez hozzárendel egy IPv4 címet, valamint ha a hálózati interfész támogatja a hálózatról való bootolást (PXE), akkor a **next-server** megmondja, mely szerverhez csatlakozzon és mit (**filename**) szedjen le a hálózati boothoz.

Azt is be kell állítanunk, hogy mely interfészeken „hallgasson” (azaz szolgáltasson) a DHCP szerverünk. Ezt Lenny alatt a **/etc/default/dhcp3-server**, Squeeze alatt pedig a **/etc/default/isc-dhcp-server** fájlban tehetjük meg. Az több interfészt is meg szeretnénk adni, ezeket a szóközzel kell egymástól elválasztani. Nézzünk egy példát:

```
INTERFACES="eth0 eth1"
```

FONTOS, hogy amennyiben egy intézményi hálózatnak csak egy részéért vagyunk felelősek, akkor csak azon belül szolgáltassunk DHCP-t – tehát az intézményi gerinc felé néző interfészen szigorúan TILOS!

A kliens oldalon operációs rendszertől függően kell beállítani, hogy DHCP szervertől kapjunk IP címet. Debian GNU/Linux alatt a **/etc/network/interfaces** fájlban kell hivatkozni rá, például így (**eth1**):

```
auto lo eth1
iface lo inet loopback
iface eth1 inet dhcp
```

A DHCP szerver indítása Lenny alatt a **/etc/init.d/dhcp3-server start**, Squeeze alatt pedig természetesen a **/etc/init.d/isc-dhcp-server start** paranccsal történik.

Az **ns.tilb.sze.hu** konfigurációs állományának kivonata:

```
default-lease-time 600;
max-lease-time 600;

    subnet 192.168.100.0 netmask 255.255.255.0 {
          option subnet-mask 255.255.255.0;
          option domain-name "tilb.sze.hu";
          option routers 192.168.100.1;
          option domain-name-servers 192.168.100.1, 193.224.128.28;
          range 192.168.100.220 192.168.100.250;
          next-server             193.224.130.174;
          filename                "netboot/pxelinux.0";

          host teacherw {
                hardware ethernet       00:1b:fc:ae:38:cf;
                fixed-address           192.168.100.15;
          }

          host teacherb {
                hardware ethernet       00:11:11:C6:CA:9A;
                fixed-address           192.168.100.205;
          }
    }

    subnet 192.168.150.0 netmask 255.255.255.0 {
          option subnet-mask 255.255.255.0;
          option domain-name "tilb.sze.hu";
          option routers 192.168.150.1;
          option domain-name-servers 193.224.128.28, 193.224.128.1;
          range 192.168.150.10 192.168.150.200;
    }

    subnet 193.224.130.160 netmask 255.255.255.224 {
          option subnet-mask 255.255.255.224;
          option domain-name "tilb.sze.hu";
          option routers 193.224.130.161;
          option domain-name-servers 193.224.130.161, 193.224.128.28;
          next-server             193.224.130.174;
          filename                "netboot/pxelinux.0";

          host apc {
                hardware ethernet       00:c0:b7:42:79:f1;
                fixed-address           193.224.130.164;
          }

          host kamera1 {
                hardware ethernet       00:40:8C:69:1E:00;
                fixed-address           193.224.130.166;
          }

          host switch {
                hardware ethernet       00:0d:54:1a:c9:00;
                fixed-address           193.224.130.169;
          }
    }
```

# 12 DNS szerver: named és konfigurációja

## *12.1 A DNS alapjai*

A DNS (Domain Name System) feladata, hogy a hálózaton lévő számítógépek számára kijelölt neveket IP címre, illetve az IP címeket nevekre fordítsa. Ezen funkciónak a felhasználó számára teljesen „átlátszónak” kell lennie. Azt a műveletet, amikor egy hálózati nevet (pl.: **`ns.tilb.sze.hu`**) IP címre fordítunk, *névfeloldás*nak hívjuk (angolul: name resolving, mapping, forward mapping). Azt a műveletet, amikor egy IP címet host névre fordítunk, visszafelé feloldásnak nevezzük (angolul: reverse mapping).

Néhány fogalom, amire remélhetőleg mindenki emlékszik Számítógép-hálózatok tárgyból:

- Top Level Domain name (TLD): Ezek a hálózati név hierarchiában a legfelső szinten álló domain nevek. (Pl.: **`hu`**, **`com`**, **`net`**, **`org`**, **`at`**, **`it`**, stb…)
- Domain name: Ezek a TLD-k alatt álló névrészletekkel együtt vett nevek, (Pl.: **`sze.hu`**, **`debian.org`**, stb.)
- Host name: A domain névhez ragasztott név azonosít egy adott gépet. (Pl.: **`ta.sze.hu`**, **`rs1.sze.hu`**, stb.)

## *12.2 Domain, Zóna*

Vegyük példának a **`sze.hu`** domain-t. A **`sze.hu`** egy zónaként is funkcionál, hiszen a **`www.sze.hu`**, illetve az **`rs1.sze.hu`**, a **`sze.hu`** zónában található. Viszont a **`dev.tilb.sze.hu`** már nem a **`sze.hu`** zónába tartozik, mert a **`tilb.sze.hu`** egy aldomain, ami saját zónát alkot. Az összes host, ami a **`tilb.sze.hu`** aldomain-be tartozik a **`tilb.sze.hu`** zónában van, de ha például létrehozunk egy **`proba.tilb.sze.hu`** zónát a **`tilb.sze.hu`** aldomain-ben, akkor az egy saját zónát alkot, és host-okat jegyezhetünk be a **`proba.tilb.sze.hu`** zónába! (Pl.: **`gep1.proba.tilb.sze.hu`**)

## *12.3 Helyi feloldás*

Ahhoz, hogy a DNS működjön, először bizonyos helyi szabályokat kell meghatároznunk, hogy mit, milyen sorrendben, és honnan kérdezzünk le.

Régebben erre a célra a **`/etc/host.conf`** fájlt használták, melynek tipikus tartalma a következő volt:

```
order hosts,bind
multi on
nospoof on
```

Ebből az első sor adta meg a sorrendet: először a **/etc/hosts** fájlban kell keresni, majd ha abban nincs, akkor a **/etc/resolv.conf** nevű fájlban beállított névkiszolgálóhoz kell fordulni.

A második sor azt mondja meg, hogy egy szimbolikus névhez tartozhat-e több IP cím is (ez a beállítás kizárólag a fájlból való feloldásra vonatkozik). A harmadik sor az *IP spoofing* elleni védekezést kapcsolja be.

Bár a **host.conf** man page még Lenny alatt is a régi, és a **/etc/host.conf** fájl is létezik, de a lekérdezés sorrendjét a tartalma nem befolyásolja!

A névszolgáltatások lekérdezésének sorrendjét az utóbbi jó néhány év óta Debian Linux alatt a **/etc/nsswitch.conf** fájl két sora adja meg:

```
hosts:          files  dns
networks:       files  dns
```

Ez annyit jelent, hogy a host és domain nevek feloldását előbb a helyi fájlokból, majd a DNS szerverből próbáljuk meg elkérni. A hosztnév → IP cím megfeleltetéseket tartalmazó fájl a **/etc/hosts**, melyben benne szokott lenni a loopback interfész, saját gépünk elsődleges interfésze és esetleg még néhány más gép is. Például:

```
fekete0:~# cat /etc/hosts
127.0.0.1 localhost
192.168.100.210 fekete0.tilb.sze.hu fekete0
193.224.130.174 ftp.tilb.sze.hu
10.9.0.73       ftp.mgmt.tilb.sze.hu
```

Az első oszlopba a gép IP címét, a többibe a teljes nevét, esetleg aliasokat írhatunk. Az alias megkönnyíti a gépre való hivatkozást.

Ha a helyi fájlokban nem található meg a kért hosztnévhez tartozó IP cím, egy DNS szerverhez kell fordulni. A **/etc/resolv.conf** nevű fájl tartalmazza a helyi névkiszolgáló(ka)t IP címmel megadva.

```
search tilb.sze.hu sze.hu
nameserver 193.224.130.161
nameserver 193.224.128.1
```

A search sorba írt domain nevekkel egészíti ki a rendszer a domain név nélkül beírt host neveket. A „nameserver” sorokban adhatjuk meg a DNS szerverek IP címeit, az első sorban megadott lesz az elsődleges.

## 12.4 A névfeloldás folyamata

Ismétlésként tekintsük át a névfeloldás folyamatát egy egyszerű példán! Otthoni internetes kapcsolatunkkal felkeressük a **users.tilb.sze.hu** gépet. Az otthoni bejegyzett DNS szerverünk legyen például a 62.112.192.4. A szerver megkapja (az UDP/53-as portján) a kérést, hogy oldja fel a **users.tilb.sze.hu** nevet, és adjon vissza nekünk egy IP címet (*recursive query*). Játsszuk el a folyamatot úgy, mintha a szerver még egyáltalán nem rendelkezne a feladat szempontjából releváns ideiglenesen tárolt (cache-elt) információval! A szerver tehát először lekérdezi az egyik TLD szervertől, hogy ki a felelős a **hu** zónáért (*iterative query*). Majd erre a TLD válaszol, hogy a **hu** zónáért az NIIF (valamelyik) szervere felel (*referral*). (**host -t ns hu.**) Ezután az általunk használt névfeloldó szerver (62.112.192.4) megkérdezi az NIIF szerverétől, hogy ki felel a **sze.hu** zónáért. Az NIIF szerver választ ad, hogy a sze.hu zónát az **rs1.sze.hu** (193.224.128.1) kezeli. A már hosszú utat bejárt DNS szerverünk megkérdezi az rs1-től, hogy ki a felelős a **tilb.sze.hu** zónáért. Ő kidobja a választ, hogy a 193.224.128.28 IP című gép. Végül tőle a névkiszolgálónk megtudja a keresett IP címet: 193.224.130.172 (*authoritaive answer*). Ezt a DNS szerver vissza is adja nekünk (*authoritaive answer*), így mi el tudjuk érni a **users.tilb.sze.hu** gépet. Mi történik, ha másodszor is lekérdezzük ezt a nevet? Mivel a 62.112.192.4 eltárolta (el cache-elte) magának ezt a hosztnév – IP cím párost, így legközelebb már nem kell ezt a hosszú folyamatot végig csinálnia, hanem a tárolt információ alapján megadja nekünk a választ.

## 12.5 Caching-only name server

A *caching only* DNS szerverek elsődleges feladata, hogy egy átjárót biztosítsanak a kliensek és a zónákért felelős névkiszolgálók között. Egy kliens *recursive query*vel fordul egy helyi névszerverhez, ami a *iterative query*k segítségével a kéréseket feloldja és megválaszolja a kliens számára. Nem illik olyan DNS szervert használunk (*recursive query*vel megszólítanunk), mely nem a saját tartományunk névszervere.

A caching only névszerverek előnye, hogy miután már egyszer feloldotta egy host nevét, megjegyzi, és a következő kérés esetén már ez szolgál ki minket, rendkívül felgyorsítva a DNS feloldás folyamatát. (Ez a módszer különösen hasznos, ha lassú vonalon keresztül kapcsolódunk az Internethez és a caching only name server helyben van.)

A caching-only name server megvalósítható a **named** nevű program segítségével, amely megtalálható a Debian GNU/Linux csomagkészletében. (Telepítése: **apt-get install bind9**)

## 12.6 A root DNS szerverek

Ahhoz, hogy a DNS szerverünk tudja, hogy egy adott TLD feloldásához hová kell fordulnia, meg kell adnunk neki az ún. root DNS szerverek IP címeit. A root DNS szerverek nagy teljesítményű gépeken futó DNS szerverek, melyek a Top Level Domain-ek feloldását végzik. Természetesen ezeket az IP címeket nem kell tudnunk fejből, le lehet tölteni például az **ftp://ftp.internic.net/domain/named.cache** címről, és aztán a konfigurációs fájlban majd a root (".") zónához hint típusú bejegyzésben kell megadni. Debian alatt a névkiszolgálót telepítve a root névkiszolgálók listáját is megkapjuk a **/etc/bind/db.root** fájlban. (Régebben a fájl neve **root.hints** volt!)

## 12.7 A named.conf fájl

A **named** alapkonfigurációs fájlja rendszerenként változó, de általában a **/etc/bind/named.conf**. Ez Lenny esetén még az érdemi információkat tartalmazó fájl, Squeeze esetén azonban csak hivatkozások vannak benne az érdemi leírásokat tartalmazó három fájlra:

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

A továbbiakban egy monolitikus **named.conf** fájl példáján mutatjuk be a legfontosabb beállításokat. A hallgatókat bátorítjuk, hogy gyakorlaton vizsgálják meg, ezek közül melyek hova kerülnek Squeeze alatt!

```
options {
        directory "/var/named";
};
controls {
        inet 127.0.0.1 allow { localhost; } keys { rndc_key; };
};
key "rndc_key" {
        algorithm hmac-md5;
        secret "c3Ryb25nIGVu … tYW4K";
};
zone "." {
        type hint;
        file "/etc/bind/db.root";
};
zone "localhost" {
        type master;
        file "tilb/localhost";
};
zone "0.0.127.in-addr.arpa" {
        type master;
        file "tilb/127.0.0";
};
zone "tilb.sze.hu" {
        type master;
        file "tilb/tilb.sze.hu";
```

```
        allow-transfer { 193.224.128.89; };
        also-notify { 193.224.128.89; };
        notify yes;
};
zone "160/27.130.224.193.in-addr.arpa" {
        type master;
        file "tilb/193.224.130";
        allow-transfer { 193.224.128.89; };
        also-notify { 193.224.128.89; };
        notify yes;
};
```

Az *options* részben a *directory* kulcsszó megadja, hogy melyik alkönyvtár legyen a **named** alapértelmezett gyökér könyvtára, innen indul ki minden további relatív elérési út.

A *controls* megadja az **rndc** szoftverrel való kapcsolat módját, ahol az *allow* kulcsszó azokat a gépeket jelenti, ahonnan a DNS szerver adminisztrálható. A *keys* megadja, hogy lesz egy titkosítási kulcsunk, melyet a *key* részben meg is adunk. Ennek a kulcsnak egyeznie kell azzal a kulccsal, amelyet az adminisztrációs gép **/etc/rndc.conf** fájljában megadunk. Mi a továbbiakban kézzel (szövegszerkesztővel) fogjuk a szervert adminisztrálni; de téma iránt alaposabban érdeklődő hallgatók számára további információ: http://www.debian-administration.org/article/Configuring_Dynamic_DNS__DHCP_on_Debian_Stable.

Ezután következik a root zóna megadása. Ezt *hint* típussal kell megadni. Azért hint, mert a root névkiszolgálók IP címe is változhat időnként, de természetesen nem egyszerre az összes. Így a root DNS szervereket tartalmazó fájlban talált névkiszolgálók között biztosan lesz olyan, ami elérhető (a legtöbb ilyen). A névkiszolgálónk induláskor ezek egyikétől lekéri az aktuális listát, majd a benne szereplők mindegyikét letesztelve kideríti, hogy melyiknek a legrövidebb a válaszideje: ezt fogja használni.

Esetünkben a fentiekben említett **/etc/bind/db.root** fájlt teljes elérési úttal kell megadnunk, hiszen nem a korábban (directory opcióval) megadott könyvtárban van.

A root zóna után meg kell adnunk a saját zónáinkat leíró konfigurációs fájlokat, példánkban esetünkben ezekből négy van, ezek útvonala **directory** kulcsszóval megadott könyvtárhoz képest relatív, tehát a fájlokat a **/var/named/tilb** könyvtárban találjuk! Mivel a saját zónáinkról van szó, a típus: master.

Mind a négy zónafájlnál van még 3 sor, aminek a megértéshez tisztáznunk kell néhány fogalmat! A megfelelő rendelkezésre állás érdekében minden zónához legalább két olyan névkiszolgálóra van szükség, amely *authoritative answer*t képes adni. Kell egy *Master* névkiszolgáló: ez az, ahol a zónafájl eredetije található. És kell még legalább egy *Slave* is, ami megfelelő időközönként a zónafájlt áttölti a Mastertől. Az áttöltést hívják *zóna traszfer*nek (zone transfer). Jegyezzük meg, hogy a zóna transzfer során a Master és a Slave névkiszolgálók a kommunikációhoz TCP protokollt használnak! Régen ezeket a névkiszolgálókat a Primary és a Secondary nevekkel illették, DNS leírásokban ma is találkozhatunk ezekkel, de ma inkább a Master és a Slave kifejezések használatosak.

A fentiekben tehát a 193.224.128.89 IP című névkiszolgáló számára engedélyezzük, hogy Slaveként a zónafájlokat áttöltse, illetve értesítjük is, ha a zóna megváltozik.

## 12.8 A zónaleíró fájl

Nézzük meg a 12.7 fejezetben példaként hozott konfigurációs állományhoz tartozó egyik zónaleíró fájlunkat; legyen ez a **/var/named/tilb/tilb.sze.hu** fájl egy részlete:

```
$TTL 3600
@       IN      SOA     ns.tilb.sze.hu. root.tilb.sze.hu. (
                        2010102301 ; Serial
                        8H         ; Refresh
                        2H         ; Retry
                        4W         ; Expire
                        1D )       ; Negative Cache TTL
                NS      ns.tilb.sze.hu.
                NS      ns.maxwell.sze.hu.

                MX      10      users.tilb.sze.hu.
                MX      20      www.tilb.sze.hu.

localhost               IN      A       127.0.0.1
ns.tilb.sze.hu.         IN      A       193.224.128.28
;
; 193.224.130.160/27 es tartomany.
;

tuzfal          IN      A       193.224.130.161
ovpn            IN      A       193.224.130.162
```

Az első sorban a **$TTL** direktíva a Time To Live értékét adja meg 1 órában (az időt jelentő, mértékegység nélküli számérték mindig másodpercben értendő), amely azt jelenti, hogy ez a fájl 1 óra alatt évül el. A második sor az úgynevezett **SOA** rekord, mely a „@” zónára vonatkozik. A „@” azt a zónát jelenti, amelynek a **named.conf** a fájlban a zónafájljaként az adott fájlt megadtuk, azaz példánkban **tilb.sze.hu** zónát jelenti. A **SOA** string után a névkiszolgáló neve és az adminisztrátor e-mail címe szerepel, ez utóbbiban (mivel a „@” karakter a zónafájlban speciális jelentéssel bír) a „@” karakter helyett pontnak kell állnia, de ettől még a cím egyértelműen dekódolható: **root@tilb.sze.hu**.

A **SOA** (Start Of Authority) rekordban a *Serial* mező a zónafájl verziószáma, ami konvencionálisan egy 10 számjegyből álló szám, aminek az első 8 jegye a dátum és azt követi még 2 számjegy a napon belüli változásokra. A fájl verziószámát a fájl minden módosításakor kötelezően növelni kell! Utána időzítések következnek: *Refresh*, *Retry*, *Expire*, *Negative Cache TTL* (más források szerint: *Minimum TTL*). Ezek az értékek a másodlagos névszerverek számára fontosak. A *Refresh* azt mutatja meg a Slave DNS szervereknek, hogy mennyi időnként kell az eredeti zónafájlt tároló Master névkiszolgálótól áttölteni a zónafájlt. A *Retry* mutatja meg, mennyi idő múlva próbálkozzon, ha elsőre nem sikerül a frissítés. Az *Expire* pedig azt jelenti, hogy (további sikertelenség esetén) mennyi ideig érvényes a letöltött zónafájl. A *Negative Cache TTL* a korábbi verziókban (BIND 4 és 8) *Default TTL* jelentéssel bírt, azaz alapértelmezett elévülési idő volt minden olyan erőforrásrekordra (RR) nézve, ahol mást nem adtunk meg. Ezt a feladatot BIND 9 esetén a (már megismert) **$TTL** direktíva látja el. BIND 9 esetén ez a negatív lekérdezési eredmények (nincs ilyen host vagy domain) cache-elésének idejét adja meg. Érdeklődőknek bővebben: http://www.zytrax.com/books/dns/ch8/soa.html

Az első **NS** sor adja meg a domain Master (régi kifejezéssel: Primary) DNS szerverének

nevét, esetünkben ez ugyanaz, mint a fájl 2. sorában lévő név. Az utána álló pont fontos, mert így azt lezárja, és nem egészíti ki további DNS utótagokkal!

Ha nem lenne pont az **ns.tilb.sze.hu** után, akkor következőt jelentené: **ns.tilb.sze.hu.tilb.sze.hu.** – persze mi nem ezt akarjuk, tehát kénytelenek vagyunk a „lezáró" pontot alkalmazni.

A másik **NS** sor egy Slave DNS szervert ad meg. (Ebből több is lehetne.)

A zónafájlban levő üres sorok a tagolást szolgálják, ezeket a **named** figyelmen kívül hagyja. A „**;**" utáni részek megjegyzésnek számítanak, amit a **named** ugyanígy kezel.

A következő két érdemi sor **MX** rekord: a zónához tartozó levelező kiszolgálókat (*Mail Exchanger*) adják meg. Ha egy olyan e-mail címre küld valaki levelet, ahol a „**@**" karaktertől jobbra nem egy gép szimbolikus neve, hanem egy zóna szerepel, akkor ennek alapján derül ki, hogy melyik gépnek kell átadni a levelet. Mivel itt kettő is van, ezért preferencia érték alapján előbb a kisebb értékkel bíró (jobban preferált) gépnek próbálja a küldő SMTP szerver a levelet átadni, majd ha az nem elérhető, akkor próbálkozik a másikkal. (A 10 és 20 szokásos értékek, de elvileg 0-65535 között bármi lehet a preferencia érték.)

Ezután 4 db „**A**" rekord következik, ami szimbolikus nevekhez IPv4 címet ad meg. Vegyük észre, hogy ahol a nevek után nincs „**.**", ott a DNS szerver a nevet kiegészíti a **tilb.sze.hu.**-val.

## *12.9 Bejegyzés a szülő zónában*

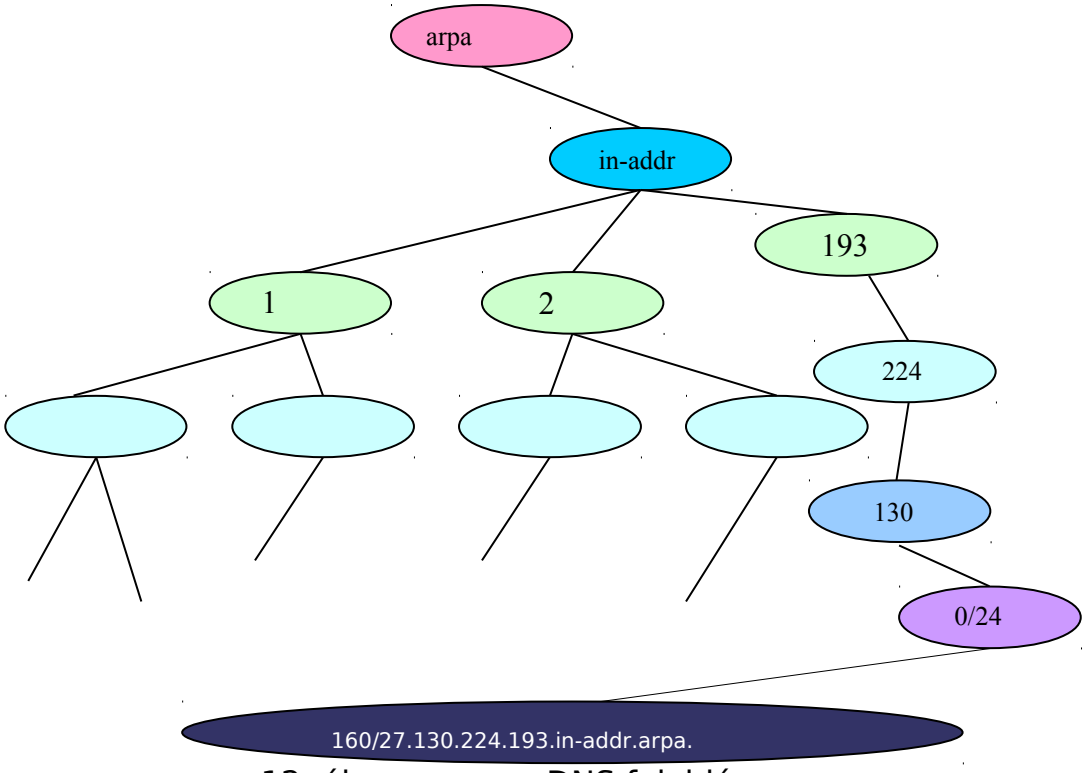
Nézzük meg, mi is szükséges a fentieken kívül még ahhoz, hogy végre legyen egy működő DNS szerverünk! Már megvan a laborunk domain-je a **tilb.sze.hu**. Ahhoz, hogy a világ többi része is tudjon róla, hogy mely névkiszolgálók felelnek ezért a zónáért, a **sze.hu** zónáért felelős névszerveren a **sze.hu** zóna zónafájljába egy NS bejegyzésre lesz szükség, mely a **tilb.sze.hu** tartomány névszervere felé *delegálja* a ***.tilb.sze.hu** kéréseket.

Természetesen szükségünk van még arra is, hogy a fent már megismert 193.224.130.160/27 IP tartományra vonatkozóan a 193.224.130.0/24 tartományért felelős névszerver delegálja felénk a reverse kéréseket.

A Debian GNU/Linux alapértelmezetten jól állítja be a localhostra vonatkozó zónafájlokat, így ezekkel nem foglalkozunk. Azt azért megemlítjük, hogy bár terjedelmi okokból a példaként bemutatott **named.conf** fájlból a locahost-ra vonatkozó bejegyzéseken kívül a többi, még megkövetelt zónát (broadcast zónák, lásd RFC 1912) kitöröltük, ezeket egy élő rendszerben benne kell hagyni.

## 12.10 A reverse DNS működése

A reverse DNS feladata, hogy az IP címből visszaadja a szimbolikus nevet. Ezt ugyanaz az infrastruktúra szolgáltatja, mint a forward DNS feloldást, azzal a különbséggel, hogy a reverse feloldás gyökere a névfának az **`in-addr.arpa.`** ága. Első közelítésben itt a 0-255 decimális számok bejegyzéseit találjuk, ezek mindegyike alatt fastruktúra szerűen szétágazva szintén a 0-255 decimális számok bejegyzései vannak, ami még további két szinten folytatódik (lásd: 12. ábra: reverse DNS feloldás). A feloldás menete fentről lefelé történik. Pl.: a 193.224.130.0/24 tartományhoz tartozó zónafájlt megfordítva kell bejegyezni a **`named.conf`**-ba: 0/24.130.224.193.in-addr.arpa. Mint láthatjuk, egy 0/24-es tartomány egy részét tovább lehet delegálni – példánkban 160/27-es tartományt.




12. ábra: reverse DNS feloldás

És természetesen a valóságban a delegálás határa nem feltétlenül esik 8 bites határra, tehát a reverse DNS fa korántsem ilyen szabályos.

## 12.11 Reverse DNS zónafájl

A példánkban a **`named.conf`** fájlba a **`160/27.130.224.193.in-addr.arpa`** zónához bejegyeztük a **`tilb/193.224.130`** zónafájlt. A fájl a következő elemekből áll:

```
$TTL 1H
@       IN      SOA     ns.tilb.sze.hu. gecko.sid.sth.sze.hu. (
                        2010090302
```

```
                        28800
                        7200
                        604800
                        86400)
        IN      NS      ns.tilb.sze.hu.
        IN      NS      ns.maxwell.sze.hu.

$ORIGIN 160/27.130.224.193.in-addr.arpa.
161     PTR     tuzfal.tilb.sze.hu.
162     PTR     ovpn.tilb.sze.hu.
163     PTR     ns2.tilb.sze.hu.
164     PTR     zorp.tilb.sze.hu.
```

Itt látható, hogy a reverse DNS zónafájl nem sokban különbözik a DNS zóna fájltól. Itt is megtalálható a ”**@**” karakter, amely minden bejegyzett sort első tagját kiegészíti a **`130.224.193.in-addr.arpa.`** címmel, így nekünk csak az utolsó (első) számot kell beírni, és a gép teljes nevét. A teljes domain név után tegyünk mindig pontot!

Néhány megjegyzés:

Bár egy IP címhez elvileg akár több **`PTR`** rekord is tartozhat, hacsak nincs rá külön okunk, akkor egynél többet ne használjunk! Bővebben lásd:

http://en.wikipedia.org/wiki/Reverse_DNS_lookup#Multiple_pointer_records

A **`$ORIGIN`** direktíva azt fejezi ki, hogy az utána következő rész mely ponton kapcsolódik a DNS fába.

A zónafájlban **`$`**-ral kezdődő szavak direktívák. Még egy ilyet használnak: **`$INCLUDE`**, ami egy másik fájlnak a zónafájlba való beszúrását jelenti.

A zónafájlok sorait erőforrás rekordoknak (Resource Record, RR) nevezik.

Még egy fontos erőforrás rekord: **`AAAA`** – IPv6 címek megadására szolgál.

Ha domain nevet szeretnénk regisztrálni, arra vannak cégek, akik azzal foglalkoznak, hogy pénzért domain nevet regisztrálnak. Regisztrálni csak olyan nevet lehet, ami nincs még felhasználva. Bővebb információ: http://www.domain.hu

Egy link a reverse mapping témához: http://www.zytrax.com/books/dns/ch3/

# 13 Az ssh

Az OpenSSH projekt lehetővé tette számunkra az ssh protokoll ingyenes használatát. Az ssh-nak két ismert verziója létezik: v.: 1.x, 2.x, de ezek nem összetévesztendőek az ssh protokoll verziókkal! (SSH Protocol v.1 = RSA, SSH Protocol v.2 = RSA/DSA)

Az SSH kliens és szerver kommunikációja során először a „Hálózati alkalmazások” c. jegyzetben leírt módon a kliens és a szerver egy biztonságos csatornát hoz létre, ami egy szimmetrikus kulcsú rejtjelezéssel valósul meg. Ehhez a két fél a gépkulcsok felhasználásával létrehoz egy kapcsolatkulcsot. A további kommunikáció ezzel a kapcsolatkulccsal titkosítva történik. A felhasználó autentikációjára pedig az ugyanott felsorolt három lehetőség közül a gyakorlatban a nyilvános kulcsú módszerrel történő erős azonosítást és a jelszavas azonosítást használjuk. Az említett jegyzetben leírt ismereteket itt nem ismételjük meg, de építünk rájuk! A továbbiakban feltételezzük, hogy a gépkulcsok már rendelkezésre állnak.

## *13.1 Kulcsgenerálás és elhelyezés a szerverre*

Ahhoz, hogy az **ssh**-t nyilvános kulcsú erős azonosítással használhassuk, az **ssh-keygen** programmal készítünk magunknak egy publikus és egy titkos kulcsot. A titkos kulcs helye **~/.ssh/id_{rsa,dsa},** a nyilvános kulcsé **~/.ssh/id_{rsa,dsa}.pub**. Itt van lehetőségünk a titkos kulcsunkat elkódolni egy jelszóval. Ilyenkor a belépésnél ezt a jelszót kell megadnunk ahhoz, hogy kikódoljuk a titkos kulcsunkat. Soha ne felejtsük el az **id_{rsa,dsa}** fájlunkra 600 jogot adni! (Bár ez rendszerint automatikusan jól jön létre.)

A következő egyszerű utasítások valamelyikével hozhatunk létre privát/publikus kulcs párokat:

```
ssh-keygen -t rsa -b 1024
ssh-keygen -t dsa -b 1024
```

Ezek a parancsok létrehozzák a **~/.ssh** könyvtár alatt az **id_{r,d}sa{,.pub}** fájlokat. A parancsokban beállított 1024 bites kulcsméret kellő biztonságot nyújt.

Megjegyzés: DSA esetén megadható ugyan az 1024-es kulcsméret, de felesleges, mert a DSA kulcsok mindig ilyen hosszúak, egyéb érték megadása esetén hibaüzenetet kapunk.

A szerver oldalon elhelyezzük a publikus kulcsunkat a home könyvtárunkban a **~/.ssh** könyvtára alá (**~/.ssh/{authorized_keys,authorized_keys2}** – attól függően, hogy milyen verziójú ssh protokollt használtunk).

A következő paranccsal a legegyszerűbb felmásolni a távoli gépre a publikus kulcsunkat:

```
ssh-copy-id -i ~/.ssh/id_dsa.pub felhasznalo@gepnev
```

Ez akkor is helyesen teszi a kulcsot a megfelelő fájlba, ha a fájl nem létezett vagy már tartalmazott más kulcsokat.

## 13.2 Az ssh parancs

Az **ssh** parancs segítségével léphetünk be a távoli, ssh daemon-t futtató gépre, vagy adhatunk ki utasításokat, a következő parancsokkal:

Belépés távoli gépre:

```
ssh -l felhasznalo gepnev
```

vagy:

```
ssh felhasznalo@gepnev
```

A következő parancssor az **ssh** segítségével a távoli gépen egy **ls** parancsot futtat le, aminek az eredménye a helyi gépen jelenik meg.

```
ssh felhasznalo@gepnev ls
```

Az alábbi parancs pedig egy interaktív shell-t indít számunkra, ami nem jelenik meg a log fájlokban:

```
ssh felhasznalo@gepnev bash -i
```

## 13.3 Az scp parancs

Az **ssh** csomag nemcsak biztonságos távoli bejelentkezést tud nyújtani számunkra, hanem titkosított adatátvitelt is két fél között. Unix rendszerekben erre szolgál az **scp** parancs. Szintaktikája megegyezik a Berkley r* **rcp** paranccsal, de az **scp**-nél a forrás és a cél közül legfeljebb az egyik lehet távoli fájl.

```
root@teacher:~/# scp kep.jpg lencse@vip.tilb.sze.hu:~/pic001.jpg
```

## 13.4 Az sshd konfigurációja

Az **sshd** konfigurációs fájlja az **/etc/ssh/sshd_config**. Az alábbiakban bemutatunk egy példa konfigurációs fájlt, melyben a fontosabb beállítási lehetőségeket kiemeljük:

```
$ cat /etc/ssh/sshd_config
# Package generated configuration file
# See the sshd(8) manpage for details

# A port és az IP cím amire a szerver figyel.
Port 22
# ListenAddress 0.0.0.0
# ListenAddress ::
# Titkosítási protokollok, amit az sshd használ. Csak a 2es verziót szabad használni!
#Protocol 2,1
Protocol 2
# RSA, DSA kulcsok helye.
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
# … nagyobb biztonság érdekében alapértelmezetten engedélyezve van.
UsePrivilegeSeparation yes

# az egyes verziójú szerverkulcsnak élettartama és mérete
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logolás, logolási szint
SyslogFacility AUTH
LogLevel INFO

# Azonosítással kapcsolatos beállítások (türelmi idő a csatlakozáshoz, ne engedjen
root-ként belépést, …).
LoginGraceTime 600
PermitRootLogin no
StrictModes yes

# RSA azonosítás
RSAAuthentication yes
# RSA publikus kulcsok azonosítása
PubkeyAuthentication yes
# RSA titkos kulcsok helye a hoszt gépeken
#AuthorizedKeysFile     %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Az X11 forwardolasa miatt lehet rá szükség. Pl. Linux grafikus felület alól
jelentkezünk be egy távoli, grafikus felülettel rendelkező gépre, az ott kiadott
parancs a saját X szerverünkön jelenik meg közvetlenül. (olyan mint egy távoli asztal
csak nem az egész asztalt visszük át
X11Forwarding yes
X11DisplayOffset 10

# A „napi” üzenetet (Message Of The Day), és az utolsó bejelentkezést írja-e ki.
PrintMotd no
PrintLastLog yes
# A futó processzek, csak addig működnek, amíg van TCP kapcsolat a gép között. Alap a
yes. Ha no ra állítjuk, előfordulhat, hogy szakadás után a processzek beragadnak és
tovább futnak.
```

```
KeepAlive yes
# környezeti változók
AcceptEnv LANG LC_*

# Betölti az sftp modult
Subsystem sftp /usr/lib/openssh/sftp-server

# PAM alapú authentikáció. Ha ezt no-ra állítjuk, akkor a rendszer csak kulcs alapú
authentikációval enged be.
UsePAM yes
```

A **/etc/init.d/ssh [start|stop|reload|force-reload|restart]** opciókkal irányíthatjuk az **sshd** futását.

# 14 FTP szerverek

## *14.1 A proftpd és konfigurációja*

A **proftpd** egy igen elterjedt FTP szerver program a Linux rendszerek világában. Debian GNU/Linux alá az **apt-get install proftpd** paranccsal telepíthető fel.

A **proftpd** konfigurációs fájlja a **/etc/proftpd/proftpd.conf**, a felhasználó-regisztrációs fájl pedig a **/etc/ftpusers**. Ebbe a fájlba azoknak a felhasználóknak a nevét kell bejegyezni, akiknek a hozzáférését le akarjuk tiltani. (Tipikusan ilyenek a root és egyéb nagy erejű felhasználók, de a kifejezetten gyengék is, például a nobody.)

Az anonymous hozzáféréshez létre kell hozni egy felhasználót, jelen esetben **ftp** nevűt, és adni kell neki egy home könyvtárat (Debian alatt automatikusan létrejön). Értelemszerűen csak ezt a home könyvtárat lehet majd anonymousként elérni.

A többi felhasználó, akit a rendszerünkben nyilvántartunk, saját felhasználó nevével és jelszavával tudja elérni az FTP szolgáltatásunkat.

### 14.1.1 Az /etc/proftpd.conf felépítése

```
ServerName        "Ez az ftp szerverünk neve"
ServerType        standalone
# A proftpd daemon futtatható a háttérben és inetd alatt is. Ha csak simán a
háttérben szeretnénk futtatni akkor standalone -t kell megadni.

DefaultServer     on
# Ez az alapértelmezett szerver, ha esetleg van a rendszerünkön másik FTP szerver
program.

Port              21
# Adhatunk más portot is, de az alapértelmezett a 21-es TCP port.

Umask             022
# Védelem a könyvtárakra, az umask 022 (csoport, és mindenki által írható) jogokat
maszkolja ki FTP alatt.

MaxInstances      30
# Ezzel azt lehet beállítani, hogy maximum hány példányban fusson, tehát hány
kapcsolatot lásson el egyszerre. Ez csak standalone módban működik.

User              nobody
Group             nobody
# Milyen felhasználó és csoport jogaival induljon el a szerverünk.

SystemLog         /var/log/proftpd.log
TransferLog       /var/log/xferlog
# Hová logoljon. System log a program futással kapcsolatos információkat logolja, míg
a transfer log egy külön fájlba logolja, hogy kik mit szedetek le tőlünk, illetve
raktak fel.

<Directory /*>
```

```
    AllowOverwrite            on
</Directory>
# Az összes file felülírható legyen, ha van rá engedély.

<Anonymous ~ftp>
RequireValidShell on     # csak olyan usereket enged be, akinek van érvényes shellje.

User ftp    # milyen user névvel rendelkezzen az anonymous belépés (ezt a felhasználót
létre kell hozni a /etc/passwd fájlban.)

Group      ftp    # milyen group -hoz tartozzon az anonymous belépés (ezt is létre
kell hozni a /etc/group fájlban, ha nincs ilyen csoport.)

UserAlias  anonymous ftp# hozzárendeli az anonymous hozzáférést az ftp felhasználóhoz

DisplayLogin     welcome.msg  # üdvözlő fájl megadása (ez a file az anonymous home
könyvtárában kell, hogy elhelyezkedjen)

# DisplayFirstChdir     .message     # Ha egy könyvtárban ilyen nevű fájlt helyezünk
el, akkor az abba a könyvtárba történő első belépéskor a benne lévő szöveget
megjeleníti az erre alkalmas kliens. DEPRICATED! HELYETTE:

DisplayChdir     .message true # true nélkül minden belépéskor megjeleníti!

<Limit WRITE>     # írás jog korlátozás anonymous -ként a gyökér könyvtárra.
DenyAll           # alapértelmezetten tiltva van, engedélyezés: AllowAll
</Limit WRITE>    # mint a HTML -nél le kell zárni a blokkot.

# Ha például szeretnénk egy olyan könyvtárat is csinálni amibe bárki tud feltölteni a
következő képen kell eljárnunk (ezt a könyvtárat létre kell hozni az anonymous home-
jában: /home/ftp/upload):
<Directory upload/*>
<Limit WRITE>
    AllowAll
</Limit WRITE>
</Directory upload/*>
# Erre a könyvtárra chmod 777 jogokat kell adni. Az anonymous felhasználónak nem lesz
joga, hogy könyvtárakat hozzon létre, illetve töröljön!

 </Anonymous>
# Itt ért véget az anonymous hozzáférés konfigurációja.
```

A fenti példában nincsen benne az összes számunkra fontos opció. Ezek a következőek:

A **RequireValidShell** segítségével tilthatjuk meg az olyan felhasználók belépését, akik nem rendelkeznek valódi shell-lel, hanem a beállított shell-jük például: **/bin/false**.

```
RequireValidShell On|Off
```

A **DelayEngine**-t akkor érdemes kikapcsolni, amikor az FTP szerverünk véletlenszerűen bontja a kapcsolatot.

```
DelayEngine on|off
```

A felhasználók, ha nincs a **DefaultRoot** opció bekapcsolva, akkor bejelentkezés után a **/** könyvtárba jutnak. Ez elég nagy probléma, pláne ha valahol elrontottuk a jogosultságokat. A **DefaultRoot**-tal lehet szabályozni, hogy a felhasználó bejelentkezése után mi legyen az

alapértelmezett „**/**” (root) könyvtára. Utána megadható még egy vagy több csoport, amelynek tagjaira vonatkozzon, illetve ne vonatkozzon (csoport neve előtt: "**!**") a beállítás:

```
DefaultRoot path/a/homehoz [!] group
```

Ha a saját home könyvtárát akarjuk beállítani, azt a következőkép használjuk:

```
DefaultRoot ~
```

A **proftpd** képes nemcsak PAM-ból, hanem SQL szerverről vagy LDAP-ból is authentikálni a felhasználókat, így nem kell unix accountokat létrehoznunk.

Minden konfigurálás után újra kell indítani a **proftpd**-t, ha nem **inetd**-ből fut. Ezt megtehetjük a **/etc/init.d/proftpd restart** paranccsal.

## *14.2 A TFTP szolgáltatás*

A TFTP tipikus felhasználási területe a hálózati bootimage-ek letöltése, valamint aktív eszközök firmware-ének, konfigurációs állományainak elérése.

Debian Linux alatt a legelterjedtebb TFTP szerver az **atftpd** (Advanced Trivial File Transfer Protocol Daemon)

Konfigurálása igen egyszerű. A **/etc/default/atftpd** nevű fájlt szerkesztve beállíthatjuk, hogy önálló szolgáltatásként fusson vagy az **inetd** szuperszerver indítsa, valamint a kiajánlani kívánt könyvtár teljes elérési útját.

Példa a konfigurációs állományra:

```
teacherb:~# cat /etc/default/atftpd
USE_INETD=false
OPTIONS="--daemon --tftpd-timeout 300 --retry-timeout 5 --mcast-port 1758 --mcast-
addr 239.239.239.0-255 --mcast-ttl 1 --maxthread 100 --verbose=5 /tftpboot"
teacherb:~#
```

Az **atftpd** a többi hálózati szolgáltatáshoz hasonlóan **init.d** szkriptel indítható:

```
bash# /etc/init.d/atftpd {start|stop|restart}
```

A TFTP protokoll nem használ autentikációt, ezért ha a kiajánlott könyvtárra írási jog van, bárki írhat/módosíthat benne!

# 15 Microsoft networks kezelése Linuxszal: Samba és konfigurációja

Ha a Linuxot Microsoft Windows gépekkel szeretnénk fájlmegosztó hálózatba összekapcsolni, a samba szervert kell használnunk, mely a Server Message Block (SMB) protokoll Linux alatti megvalósítása. A Samba megtalálható a Debian GNU/Linux csomagkészleteiben. Telepítése: **`apt-get install samba`**. A parancs feltelepít még egy néhány más csomagot is, ilyen például a **`samba-common`**.

Más Linux disztribúciókban (régebben) egy **`/etc/samba/smb.conf-sample`** fájlt mellékeltek, amit át kellett másolni **`/etc/samba/smb.conf`** névre – ez tartalmazza a szerverrel kapcsolatos beállításokat. Debian Lenny alatt ez a fájl telepítéskor létrejön.

A telepítő (beállítástól függően) egy (vagy két) kérdést tesz fel. Bekéri a munkacsoport vagy tartomány nevét (jelentését lásd később), és esetleg megkérdezi, hogy a Netbios névkiszolgálók listáját is lekérje-e DHCP-vel. Ezekre értelemszerűen válaszoljunk. Utána a kiszolgáló elindul, megosztás még nem lesz elérhető, de a gépünk látható lesz a megadott nevű munkacsoport más gépei között.

A „kincstári” **`smb.conf`** fájl viszonylag sok megjegyzést tartalmaz, megjegyzésnek számítanak a „**`#`**” és „**`;`**” utáni részek. Segítségként a nagyon sok lehetséges beállítás közül megjegyzésben megadnak fontosabb paramétereket. Hasznos konvenció, hogy amit „**`#`**” után adnak meg, az megegyezik az alapértelmezett beállítással, amit „**`;`**” után, az eltér tőle!

A kívánt beállítások elvégzése után erősen ajánlott a konfigurációs fájl szintaktikai ellenőrzése a következő paranccsal:

```
testparm /etc/samba/smb.conf
```

A beállított paraméterek érvényre juttatása érdekében a kiszolgáló újraindítása a szokott módon történik:

```
/etc/init.d/samba restart
```

Nézzük a beállításokat a konfigurációs fájlon végighaladva!

A globális beállítások a **`[global]`** pontnál találhatók.

```
workgroup = TILB
```

A workgroup beállítással adható meg mind a *munkacsoport*, mind *tartomány* név.

Mind a munkacsoport, mind a tartomány windows-os számítógépek egy logikai csoportját jelenti. Ezek tipikusan közös erőforrásokat használnak (pl. nyomtatók, fájl-megosztások). A fő különbség az, hogy míg a tartomány adminisztrációja központosított, az ún. *domain controller* lépteti be mind a gépeket, mind a felhasználókat; a munkacsoportról csak kijelenti egy gép, hogy ő beletartozik, és a felhasználókat is magának kell beléptetnie.

Érdeklődők a két megoldás előnyeiről és hátrányairól bővebben olvashatnak:

http://www.ntcompatible.com/thread29911-1.html

A **server string** tartalma a Windows ”Számítógép leírása” mezőjében jelenik meg.

```
server string = Samba Server on TILB
```

A **hosts** opcióknál megadhatjuk, melyik IP tartományokból lehessen elérni a szerver szolgáltatásait és melyikből nem. (A host engedélyező fájlok érvényességi sorrendje a UNIX-ban: 1. hosts.allow, 2. hosts.deny, és ha egyikben sem szerepel a cím, akkor meg van engedve a hozzáférés.)

```
hosts allow = 193.224.130. 193.225.150. 193.224.128.
hosts deny = 193.224.130.99
```

Ezenkívül megadhatunk még netmaszkkal vagy az „except” kifejezéssel alhálózatot is például:

```
hosts allow = 193.224.130.160/27
hosts allow = 193.224.130. except 193.224.130.99
```

Ha szeretnénk, hogy a nyomtatónkat a **/etc/printcap** fájlból töltse be a Samba, és ne kelljen őket egyenként megadni, akkor ezt kell engedélyezni.

```
printcap name = /etc/printcap
loadprinters = yes
```

A következő nyomtatási rendszereket támogatja a samba: **bsd**, **sysv**, **lprng**, **aix**, **hpux**, **qnx**, **CUPS**. A szerverünk nyomtatási rendszerét itt adhatjuk meg (nem szükséges beállítani):

```
printing = bsd
```

Ha szeretnénk saját magunk által definiált vendég felhasználót (itt: **pcguest**), ezt engedélyezni kell, és felvenni a **pcguest**-et a **/etc/passwd** fájlba. Amennyiben nincs megadva guest account, alapértelmezetten a **nobody** felhasználó jogait használja.

```
guest account = pcguest
```

Naplózásnál megadható, hogy minden gép (**%m**) logja külön log fájlba kerüljön:

```
log file = /var/log/samba/log.%m
```

A log fájlok maximális mérete (kB-ban) a következőképpen adható meg:

```
max log size = 100
```

A megosztásokhoz biztonsági szintet lehet megadni. Ha ezt ”**share**” értékre változtatjuk, akkor nem kell jelszó a megosztások eléréséhez, ”**user**” esetén kell. Ha a ”**server**” biztonsági módot választjuk, akkor külön NT szerver fogja a jelszavakat szolgáltatni.

```
security = user
```

Az NT-s jelszószerver nevének megadása:

```
password server = <NT-server-name>
```

A következő beállítással a felhasználónevek és jelszavak ellenőrzésének mikéntjét adhatjuk meg: amennyiben a felhasználó által megadott felhasználónév és a jelszó az itt megadott számértéknél nem több helyen tér el a kisbetű/nagybetű voltában (de egyébként helyes), akkor a rendszer elfogadja. A 0 érték esetén csupa kisbetűvel és csupa nagybetűvel próbálkozik. Jelszavak esetén ez csak akkor van így, ha a jelszavakat nem titkosítjuk – ennek beállítását lást lent; a 3.0.0. Samba verzió óta a jelszavakat alapértelmezetten titkosítjuk.

```
password level = 8
username level = 8
```

Megadhatjuk, hogy a Samba a jelszavakat titkosítsa. Bővebb információt a samba dokumentációjában találhatunk.

```
encrypt passwords = yes
smb passwd file = /etc/smbpasswd
```

Ez az opcióval engedélyezhetjük, hogy Windows alól változtatni tudjuk a Linux-os rendszerjelszót is. Ezeket csak az ”**encrypt passwords**” és az ”**smbpasswd**” fájl opciókkal használjuk! Figyelem! Ha csak az Samba jelszavakat akarjuk Windows alól változtatni, akkor ezekre nincs szükség!

```
unix passwd sync = Yes
passwd program = /usr/bin/passwd %u
passwd chat = *Enter\snew\sUNIX\spassword:* %n\n *Retype\snew\sUNIX\spassword:* %n\n
*password\supdated\ssuccessfully* .
```

A Unix és samba felhasználónév lehet különböző is, ha az itt megadott fájlban bejegyezzük őket.

```
username map = /etc/smbusers
```

Gépenként külön konfigurációs fájlt adhatunk meg a Samba számára, **%m**-et a gép NetBIOS nevével kell helyettesíteni.

```
include = /etc/smb.conf.%m
```

Több hálózati interfész esetén megadhatjuk, hogy mely intfész(ek)en keresztül történjen kiszolgálás; IP cím és interfész név egyaránt használható.

```
interfaces = 193.224.130.161/27 eth0
```

A Local Master Browser és a Domain Master Browser az MS Windows világban az úgynevezett tallózáskezelők, melyek a hálózaton lévő számítógépek neveit és egyéb adatait gyűjtögetik, ennek az az értelme (a M$ szakemberei szerint), hogy így (elvileg) nem lesz annyi broadcast csomag a hálózatban, amikor pl.: egy titkárnő megnyomja a „Teljes hálózat” ikont :). A Local Master csak egy adott alhálózaton gyűjtöget, a Domain Master pedig az egész NT domain-ben.

Tallózáskezelő opciók: állítsuk ezt ”**no**”-ra, ha nem szeretnénk, hogy szerverünk legyen a local master browser a hálózaton:

```
local master = yes
```

A Master Browser kiválasztásánál a gépünk prioritását így adhatjuk meg:

```
os level = 88
```

Az alábbi módon engedélyezhetjük, hogy a gépünk legyen a Domain Master a hálózatban. (Ne használjuk ezt az opciót, ha van már NT Domain master a hálózatban.)

```
domain master = yes
```

Az alábbi beállítás esetén a Samba szerver indításakor egy választási kört (master browser election) kezdeményez:

```
preferred master = yes
```

Ha telepítéskor van már működő NT tartomány vezérlőnk (Domain Controller), akkor használhatjuk azt autentikációra:

```
domain controller = <NT-Domain-Controller-SMBname>
```

Ha pedig azt szeretnénk, hogy a Samba Domain Controller legyen, az így adhatjuk meg:

```
domain logons = yes
```

A következő három beállításnak csak **domain logons = yes** esetén van értelme.

Gépenként külön bejelentkező szkriptet engedélyezhetünk:

```
logon script = /etc/smbscripts/%m.bat
```

Felhasználónként is külön bejelentkező szkriptet engedélyezhetünk:

```
logon script = %u.bat
```

A felhasználói profilok elérhetőségét az alábbiak szerint adhatjuk meg. A **%L** a szerver Netbios neve, **%U** a felhasználó név. (Alul a [Profiles] megosztást engedélyezni kell.)

```
logon path = \\%L\Profiles\%U
```

A NetBIOS neveket IP címekre kell fordítani, és a „**Name Resolve Order**” segítségével megadhatjuk a feloldás sorrendjét. Az alap sorrend „**lmhosts wins host bcast**”. A „**host**” azt jelenti, hogy a Unix gethostbyname() rendszerhívással próbálja meg feloldani a nevet, melyet akár a **/etc/hosts** vagy a DNS vagy a NIS feloldhat, a **/etc/host.conf**, **/etc/nsswitch.conf**, **/etc/resolv.conf** fájlok függvényében.

```
name resolve order = lmhosts wins host bcast
```

A WINS (Windows Internet Name Service) a NetBIOS Name Service implementációja. Az **nmbd** WINS szerverét így engedélyehetjük:

```
wins support = yes
```

Ezzel engedélyezhetjük az **nmbd** WINS szerverét.

Ha a samba csak WINS kliens lesz, mert már van WINS szerverünk, akkor WINS szerver IP címét így adhatjuk meg neki (viszont az előbbi beállítást állítsuk „**no**”-ra):

```
wins server = w.x.y.z
```

Ha a DNS proxyt engedélyezzük, akkor a DNS által feloldott NetBIOS neveket eltárolja egy pufferbe, hogy gyorsabb legyen a feloldás:

```
dns proxy = no
```

A Windows (és a DOS) nem különbözteti meg a fájlnevekben a kis- és nagybetűket, a Unix/Linux viszont igen. A Samba számára beállíthatjuk, hogyan járjon el:

```
case sensitive = no
```

Ha azt szeretnénk, hogy az új fájlok kis/nagybetű szempontjából úgy jöjjenek létre, amint a kliens megadja a fájlnevet, akkor az alábbiakban „yes”-t adjunk meg; „no” esetén a „default case”-t használja.

```
preserve case = no
```

Az alapértelmezés pedig megadható:

```
default case = lower
```

A hagyományos rövid fájlnevekre (8+3 karakter) külön beállítás adható meg:

```
short preserve case = no
```

## *15.1 Megosztások kezelése*

Ha szeretnénk, hogy a felhasználók home könyvtára automatikusan megosztásra kerüljön a homes név alatt, akkor ezt így jegyezzük be:

```
[homes]
comment = Home directories     #megjegyzés
    browseable = yes    #tallózható
    writeable = yes     #írható
```

Netlogon megosztás a netlogon kliensek számára:

```
[netlogon]
    comment = Network Logon Service
    path = /home/netlogon
    guest ok = yes
    writeable = no
    share modes = no
```

Felhasználói profilok könyvtára:

```
[Profiles]
    path = /home/profiles
    browseable = no
    guest ok = yes
```

Kommentezzük ki, ha szeretnénk megadni külön könyvtárat a felhasználói profiloknak. Az alapértelmezés, hogy a felhasználó home könyvtárát használjuk.

## 15.2 Standard megosztások

```
[bigspace]
    comment = Home
    path = /bigspace    #A megosztott könyvtár elérési útja
    valid users = lencse tbalazs drmomo     #Kik használhatják
    create mask = 700   #Fájlok létrehozási jogai
    read only = yes     #Csak olvasható
    public = no  #Mindenki láthatja?
    printable = no      #Lehet-e rá nyomtatni?
```

Megjegyzés a **`read only`** és a **`writeable`** egymásnak inverz szinonimái.

## 15.3 Nyomtató megosztása

```
[HP_LaserJet]
    comment = HP LaserJet 2200DN
    printer name = HP LaserJet 2200DN       #Nyomtató neve a Windowsban
    path = /var/spool/lpd/ljet4-a4-auto-mono#Nyomtató spool elérési útja
    browseable = yes    #Tallózható
    printable = yes     #Lehet rá nyomtatni
    writeable = yes     #Lehet rá írni
    guest ok = no#Guest user használhatja-e?
    print command = echo '/bin/date' %U nyomtatja a %s fajlt. >> \
 /tmp/print.log; lpr %s; rm %s #Nyomtató parancssor, a UNIX hajtja végre %U user, %s
fájl
```

## 15.4 Felhasználók kezelése

Ha már létezik egy felhasználó a Unix-ban, a Samba-hoz akkor a következő paranccsal hozhatunk létre smb felhasználót:

```
smbpasswd –a <usernév>
```

Illetve az alábbi paranccsal változtathatjuk meg a jelszavát:

```
smbpasswd <usernév>
```

## 15.5 A Samba indítása

Egyéb Linux disztribúciók alatt a samba indítását az **smbd** és **nmbd** programok indításával végezhetjük el:

```
# nmbd -D
# smbd -D
```

Debian Linux alatt a szokott módon a **/etc/init.d/samba start|stop|restar**t szkripttel.

## 15.6 A Samba parancsai

A Samba használatához tartoznak különböző utasítások.

- **smbmount <//Gépnév/Megosztás> <Könyvtár>** – Windowsos megosztások mountolása (Kernelnek tudnia kell az smbfs-t kezelni)
- **smbclient <opció> <//Gépnév/Megosztás>** – Windowsos megosztásokra csatlakozás. Opció: **-L** egy gép megosztásainak kilistázása. (Az **smbclient** csomagot telepíteni kell.)
- **smbmnt** – Samba megosztáskezelő programja.
- **nmblookup** <gépnév> – Samba névfeloldó parancsa.

## 15.7 Grafikus konfigurációs felületek

Megemlítjük, hogy létezik egy **system-config-samba** nevű csomag, ami más, Debian alapú disztribúcióból (pl. Ubuntu) kis ügyeskedéssel feltehető. Ez a grafikus felületen működő eszköz nem támogat minden lehetőséget, de egyszerűbb esetben a képességei elégségesek lehetnek.

Egy másik megoldás a SWAT (Samba Web Administration Tool) használata. Ez szerepel a Debian csomagkészleteiben is.

# 16 Az Apache HTTP szerver

Ebben a fejezetben megtanuljuk, hogyan lehet webszervert építeni és konfigurálni az **apache2** HTTP daemon segítségével.

Az Apache jelenleg a legelterjedtebb webszerver a világon, jóval a Microsoft IIS-e előtt. (Érdeklődők számára egy felmérés: http://news.netcraft.com/archives/2011/.) A régi NCSA webszerverből származik, az **apache2**-t pedig gyakorlatilag újraírták a fejlesztők. Az 1.3 verziót a fejlesztők már nem tartják karban, helyette az **apache2** telepítését javasolják. Az **apache2** a Debian disztribúció része, így az **apt-get install apache2** parancs megadásával azonnal telepíthetjük rendszerünkre. (Telepítéskor jó néhány más csomag is felkerül vele együtt.)

A Debian az **apache2** konfigurációs fájljait a **/etc/apache2** könyvtárba, míg a futtatható fájljait a **/usr/sbin** könyvtárba helyezi el. Az **apache2** vezérlésére a **/etc/init.d/apache2** szkriptet használjuk, ami viszont a műveletek elvégzésére meghívja a **/usr/sbin/apache2ctl** szkriptet. Alapértelmezetten a dokumentum root a **/var/www**. Telepítés után az **apache2** placeholder oldalát találjuk itt. Ha ide bemásoljuk a saját WEB-lapunk fájljait úgy, hogy a WEB-lap **index.html**-je a **/var/www** alkönyvtárban van, és az összes többi tartalom az ez alatt lévő könyvtárakban, akkor honlapunk működőképes lesz.

## *16.1 Konfigurációs fájlok*

Mivel működnek még Apache v1 szerverek, ezért közöljük mindkét verzió konfigurációs fájljainak rendszerét, de a 2010. évtől már az Apache v2-n mutatjuk be a részletes konfigurációt.

### 16.1.1 Az Apache v1 esetén

Az Apache konfigurációs fájljai a **/etc/apache** könyvtárban vannak:

- **httpd.conf** – Az Apache fő konfigurációs fájlja, ebben van majdnem minden beállítás.
- **access.conf** – Az Apache-hoz történő hozzáféréseket lehet benne szabályozni.
- **srm.conf** – egyéni konfigurációs utasításokat adhatunk itt meg.
- **magic**, **mime.types** – A dokumentumtípusok és kezelőmoduljaik vannak itt felsorolva.
- **modules.conf** – a betöltendő modulok vannak itt felsorolva

Az **access.conf** és az **srm.conf** alap esetben üresek. Ez a két fájl a **httpd.conf** megfelelő sorában lenne include-olva, azonban ez a két sor ki van kommentezve, tehát alap

esetben az összes beállítás a **httpd.conf** fájlba kerül. Web szerver farm esetén a gépenként eltérő beállításokat tehetjük az **srm.conf**-ba.

### 16.1.2 Az Apache v2 esetén

A v2-vel a fejlesztők szakítottak az „egy fájl központú" konfigurációval. Az **apache2** rengeteg fájlt használ, melyeket linkekkel való hivatkozással tehetünk aktívvá. Később példával illusztráljuk ezt. A főbb konfigurációs állományok:

- **apache2.conf** – ez vette át a régi **httpd.conf** szerepét. A fájl végén több fájlt is be-include-olnak, többek között a telepítéskor teljesen üres **httpd.conf** fájlt is.
- **ports.conf** – azokat a portokat kell benne megadni, amin az apache2-nek figyelnie kell. Ez az egyik legnagyobb újdonság, hogy itt nincs külön **apache-ssl** és **apache** hanem egy program oldja meg a két szolgáltatást (erre természetesen az 1-es verzió esetében is volt lehetőség, de a Debian alatt általában azt választották, hogy 2 db webszerver futott, az egyik a 80-as portra, a másik a 443-asra figyelt).
- **conf.d** – egyéb betöltendő konfigurációs állományok helye: ilyenek a **charset** és a **security**
- **mods-{available,enabled}** – a lehetséges (available) és a használandó (enabled) modulok helye. A **mods-available** könyvtárban az összes betölthető modul szerepel a modulnév.load formában és ha van konfigurációs állomány hozzá akkor az a modulnév.conf formában. A **mods-enabled** könyvtárban linkek vannak a **mods-available** könyvtárban lévő fájlokra. Az Apache indításakor azok az állományok töltődnek be, melyekre hivatkozunk a **mods-enabled** könyvtárból.
- **sites-{available,enabled}** – hasonló mint a modulok esetén, csak itt virtuális webszervereket tudunk megadni.
- **envvars** – környezeti változókat definiálnak benne (lásd később).
- **magic** – az egyes fájlok tartalomtípusának megállapításához használt *magic number* definíciók találhatók benne.

## *16.2 Az Apache v2 részletes konfigurációja*

### 16.2.1 Az `apache2.conf` fájl legfontosabb direktívái

A kincstári **apache2.conf** fájlt megnyitva azt olvashatjuk, hogy az alapvetően három fő részből áll:

- A globális opciók, melyek a szerverprogram működését adják meg, pl.: hány gyerek processz (child process) fusson, stb…
- A fő (main/default) szerverre vonatkozó opciók, melyek a nem-virtuális webszerverek működését adják meg.

(Ezek egyben alapértelmezett értékek a virtuális webszerverek számára is.)

- A virtuális webszerverek konfigurációja, melyek alias-ként (CNAME) szerepelnek csak a DNS-ben, és az alias ugyanerre a gépre mutat.

Ez valóban így is volt 1-es verzió esetén, de továbbiakban azonban ezek közül csak az elsőt találjuk meg:

```
### Section 1: Global Environment
#
```

A másik kettő nem véletlenül hiányzik: ezek a beállítások máshol találhatók. Ha valakinek volt már tapasztalata az 1-es verzióval, és nem talál valamilyen direktívát a megszokott helyen, akkor célszerű rákeresnie az alábbi módon:

```
feher10:/etc/apache2# grep Listen * */*
ports.conf:Listen 80
ports.conf:    Listen 443
```

Most bemutatjuk az **apache2.conf** fájlban található fontosabb direktívákat.

```
ServerRoot ”/etc/apache2”
```

Megadja, hogy az **apache2** web szerver számára mi legyen a root könyvtár. Ezután minden relatív útvonal ehhez képest értendő.

Tehát ha az **apache2.conf**-ban egy fájlra relatív útvonallal hivatkozunk pl.: **logs/error.log** akkor ez a **/etc/apache2/logs/error.log** fájlt fogja jelenteni. Természetesen továbbra is működnek az abszolút elérési utak, pl.: **/var/log/apache/error.log**

```
PidFile ${APACHE_PID_FILE}
```

Megadja azt a fájlt, ahol a szülő **apache2** a saját processz-ID-jét tartja. Ne nyúljunk ehhez a beállításhoz, mert az **apache2ctl** szkript működését elronthatjuk vele! Esetünkben a tényleges beállítás a **/etc/apache2/envvars** fájlban történik.

```
Timeout 300
```

Egy kliensnek csatlakozás után ennyi ideje van (másodpercben mérve), hogy kérést adjon a szervernek, különben a szerver timeout üzenetet küld, és bezárja a TCP socketet.

```
KeepAlive On | Off
```

Ha ez „On”, akkor egy kliens egy TCP kapcsolat alatt több kérést is küldhet, ha „Off”, akkor a kapcsolat a kérés teljesítése után megszakad. Az „On” állapot akkor hasznos, ha a html oldalaink több részből állnak (pl.: képek, frame-ek).

Ezentúl, ha ilyen jellegű konfigurációs beállítás van, ahol több lehetőség közül kell választani, így fogjuk jelölni: *Direktíva_neve egyik | másik | harmadik*. Természetesen ezek közül csak egyet szabad a fájlba beírni. (A választást jelző függőleges vonal csak metanyelvi elem, azt nem szabad beírni!)

```
MaxKeepAliveRequests 100
```

Ha az előbbi „On”, akkor ez adja meg a TCP kapcsolatonkénti maximális kérések számát. Ha teljesítményre kell optimalizálni egy szervert, akkor érdemes ezt viszonylag magas értéken tartani (néhány százas nagyságrend).

```
KeepAliveTimeout 15
```

Ennyi másodpercet várunk egy nyitott TCP kapcsolatban a következő kérésre.

```
MinSpareServers 5
MaxSpareServers 10
```

A „tartalék” szerver processzek számának minimális és maximális értéke. Kis szervereknél érdemes a minimumértéket 1-en tartani, ez indítás után 2 processzt fog eredményezni: 1db szülő és 1db tartalék (gyerek). Ha a webszerver terhelése növekszik a szülő **apache2** forkol még child processzt. A maximum értéket se tegyük 10-nél nagyobbra, ha nincs extrém terhelésű szerverünk.

```
StartServers 5
```

A szülővel együtt ennyi processzt fogunk indítani elsőre.

```
MaxClients 150
```

Ez az összes futó HTTP kérések maximális száma. Ha ezt elérjük, a webszerver nem szolgál ki több klienst. Nem érdemes túl alacsonyra állítani, mert szerverünk el fog utasítani klienseket. Ha viszont túl nagyra vesszük, és jön egy túlterhelés a hálózat felől, rendszerünk alól elfogy a memória, és a szerver kiakad (DoS).

```
MaxRequestsPerChild 0
```

Ennyi klienst szolgálhat ki egy processz, utána kihal. Ha 0-ra van állítva, ez a szám korlátlan.

```
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}
```

Itt megadhatjuk, milyen UID és GID alatt fusson a szerver. Esetünkben a tényleges érték megadása a **`/etc/apache2/envvars`** fájlban történik. A GID és UID beállítása biztonsági szempontból fontos; ha a webszerver törhető, akkor sem tudnak root jogokkal garázdálkodni a rendszerünkben az illetéktelenek.

```
HostNameLookups Off | On
```

Ha bekapcsoljuk, a naplófájlokban megjelenik a hosztoknak a neve is, amelyek kérést intéztek a szerverhez. Ha kikapcsoljuk, csak IP címet naplóz. (A döntéskor vegyük figyelembe, hogy a hosztok nevének a kiderítéséhez reverse DNS lekérdezésre van szükség – ez terhelést jelent a webszervernek, a helyi névkiszolgálónak és a hálózatnak is.)

```
ErrorLog ${APACHE_LOG_DIR}/error.log
```

A hibanaplófájl (**`error.log`**) és elérési útja. Ha a fájl nem létezik, létrejön, de az elérési útnak léteznie kell, a szerver nem hoz létre alkönyvtárat!

```
LogLevel debug | info | notice | warning | error | crit | alert | emerg
```

Azt a hibaszintet állítja be, amelynél már létrejön egy bejegyzés a hibanaplófájlban.

```
LogFormat <formátum-string>
```

Megadja a log fájlok egy sorának formátumát.

Bővebb információk az Apache dokumentációban:
http://httpd.apache.org/docs/2.2/mod/mod_log_config.html#logformat

(A tényleges naplófájlok nevét szerverenként szokás megadni. A direktívát lásd ott.)

Fontos megjegyezni, hogy bizonyos beállításokat külön fájlokban szokás megadni. Ezeket az alábbiakban látható módon érjük el:

```
# Include module configuration:
Include mods-enabled/*.load
Include mods-enabled/*.conf

# Include all the user configurations:
Include httpd.conf

# Include ports listing
Include ports.conf

# Include generic snippets of statements
Include conf.d/

# Include the virtual host configurations:
Include sites-enabled/
```

Ezek közül néhány fontosabbal tudunk csak foglalkozni.

### 16.2.2 A `ports.conf` fájl legfontosabb direktívái

Amint a fájl neve is mutatja, itt adjuk meg, hogy a web kiszolgáló milyen portokon várja a kliens csatlakozását. Azt is említettük már, hogy egyetlen szerver képes mind HTTP, mind HTTPS kiszolgálásra is.

Van azonban még egy fontos direktíva, amit ide szoktak helyezni: a `NameVirtualHost`, ami a *Name-based Virtual Host* engedélyezésére szolgál. Mit is jelent ez? A HTTP/1.1 protokoll lehetővé teszi, hogy a kliens a GET kérés után elküldje a kiszolgáló szimbolikus nevét is. Így az összes kívánt szimbolikus névhez ugyanazt az IP címet rendelhetjük (tipikusan CNAME segítségével), az **`apache2`** mégis tudni fogja, hogy a kliens melyik szimbolikus névhez tartozó weboldalt kéri. (Enélkül arra lenne szükség, hogy annyi IP címet húzzunk fel a kiszolgáló interfészére, ahány különböző szimbolikus néven el akarjuk érni: Ez hívják *IP-based Virtual Host*nak.)

Érdeklődőknek bővebben: http://httpd.apache.org/docs/2.2/vhosts/name-based.html

Az **`apache2`** kincstári port beállításai a következők:

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
```

```
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

A **NameVirtualHost** direktívánál a „*****” azt jelenti, hogy minden aktív interfész 80-as portján szolgáltatunk. Amennyiben itt egy IP címet adunk meg, akkor csak azon az interfészen nyújtunk Name-based Virtual Host szolgáltatást, amelyhez az IP cím tartozik. FONTOS, hogy majd a virtuális hosztok megadásánál ugyanazt kell írnunk, mint itt! Különben induláskor hibaüzenete(ke)t (warning) fog adni az **apache2**!

A **Listen** direktíva teljes szintaxisa a következő:

```
Listen [IP-address:]portnumber [protocol]
```

A fentiekhez hasonlóan itt is megadható IP cím, ennek az értelmezése is azonos. Megadható még protokoll is, ami **http** vagy **https** lehet. A 443 porthoz az alapértelmezett a **https**, az összes többihez a **http**.

### 16.2.3 Az `envvars` fájlban beállított környezeti változók

Mivel bizonyos beállításokat több szkript is használ (például: **apache2ctl**, **/etc/init.d/apache2**, **/etc/logrotate.d/apache2**, stb.) és ezeknek szkripteknek meglehetősen kellemetlen lenne az **apache2.conf** fájlból kihámozni a kívánt beállításokat, ezért a fejlesztők azt a megoldást választották, hogy környezeti változókat használnak. Az **apache2** számunkra érdekes kincstári beállításai az **envvars** fájlban:

```
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
export APACHE_PID_FILE=/var/run/apache2.pid
export APACHE_LOG_DIR=/var/log/apache2
```

Jelentésüket az **apache2.conf** elemzésénél már tárgyaltuk.

### 16.2.4 A `conf.d/security` fájl legfontosabb direktívái

Ide olyan beállítások kerültek, amik biztonsági kérdésekkel kapcsolatosak. Az általuk engedélyezett információ visszaélésre ad lehetőséget, például a szerver jellemzőinek ismerete a támadókat segíti, az e-mail cím kéretlen levelek küldését teszi lehetővé.

```
ServerTokens (Full|OS|Minimal|Minor|Major|Prod)
```

Ez a direktíva azt adja meg, hogy a HTTP protokoll válasz fejrészébe (*response header*) milyen információk kerüljenek bele. A felsoroltak közül a Full ad a legtöbb információt, a Prod a legkevesebbet.

```
ServerSignature Off | On | Email
```

A szerver által generált lapokra – ilyen például egy könyvtárhoz generált tartalomjegyzék (ha engedélyezve van), valamint az összes hibaüzenet virtuális dokumentuma – milyen információ kerüljön rá: semmi, csak a szerver neve, a webgazda e-mail címe is (ez utóbbit majd a **`ServerAdmin`** direktívával lehet megadni).

### 16.2.5 A `conf.d/charset` fájl direktívája

Amennyiben ez a szándékunk, megadhatjuk az összes oldalhoz a karakterkészlet típusát az alábbi módon:

```
AddDefaultCharset UTF-8
```

Ezzel azonban legyünk óvatosak, mert ha megadunk valamit, azzal felülírjuk az egyes weboldalak forrásában (pl. **`META HTTP-EQUIV`** beállítással) megadott karakterkészletet!

### 16.2.6 A fő szerver és a virtuális szerverek definiálása

Fő szervernek azt szoktuk nevezni, amelyiknek a szimbolikus neve a szolgáltatást nyújtó gép elsődleges neve (tipikusan ezt kapjuk vissza az IP cím reverse DNS feloldásakor) és virtuális szervereknek nevezzük mindazokat, amelyek szimbolikus neve CNAME a fő szerver szimbolikus nevére.

Természetesen a fenti megoldás pusztán egy konvenció. Akár azt is meg lehet csinálni, hogy a zónafájlban az összes virtuális szerver szimbolikus nevéhez **`A`** rekorddal ugyanazt az IP címet rendeljük. Ez persze meglehetősen ügyetlen megoldás; ha az IP cím változik, akkor mindenhol át kell írnunk. E jegyzet írásakor a tesztelésre használt rendszerhez nem is volt zónafájl, hanem csak a **`/etc/hosts`** fájlt használtuk, amiben a loopback interfész IP címét (127.0.1.1) adtuk meg a virtuális szerverekhez: ettől még azok kiválóan működtek!

A szervereket a **`sites-available`** könyvtárban szoktuk definiálni. A fő szerver fájlneve konvenció szerint: „**`default`**”, a többieké célszerűen az adott virtuális szerver szimbolikus neve. Ezek közül az **`apache2`** csak azokat fogja ténylegesen kiszolgálni, amelyeknek a

szervert definiáló fájljára a **sites-enabled** könyvtárban szimbolikus linket helyeztünk el.

A szimbolikus linket a korábban megismert módon az **ln -s mire milyen_neven** parancs segítségével kézzel is létrehozhatjuk, de az **a2ensite** parancs kényelmessé teszi ezt. A jegyzet írásakor a tesztelésre használt gépen a szimbolikus linkek így néztek ki (a két virtuális webszerver neve **kutya** és **macska** volt):

```
feher10:/etc/apache2/sites-enabled# ls -l
Összesen 0
lrwxrwxrwx 1 root root 26 nov 13 14.21 000-default -> ../sites-available/default
lrwxrwxrwx 1 root root 24 nov 13 21.44 kutya -> ../sites-available/kutya
lrwxrwxrwx 1 root root 25 nov 13 20.53 macska -> ../sites-available/macska
```

### 16.2.7 A fő szerver és a virtuális szerverek legfontosabb direktívái

Az **apache2** esetén a fő szervert éppen úgy definiáljuk, mint a virtuális szervereket; a fő szervert definiáló **default** nevű fájlban is egy virtuális szerver leírása van:

```
<VirtualHost *:80>
# … itt van a leírása
</VirtualHost>
```

A fő szerver és a virtuális szerverek definícióját ezen direktívák között kell megadnunk.

Természetesen a fő szerver és a virtuális webszerverek definiálásakor ugyanazokat a direktívákat használhatjuk. Nézzük meg ezek közül a legalapvetőbbeket!

```
ServerName Szerverem_neve
```

Ezzel KELL megadni a virtuális webszerverek szimbolikus nevét. (Ennek életszerű használat esetén természetesen egy érvényes, DNS A vagy CNAME rekordban szereplő névnek kell lennie.) A fő szervernél ezt nem kötelező megadni.

Vegyük észre, hogy a fájlnevek ésszerű használta (azaz: a szervert definiáló fájl neve a virtuális webszerver neve) egy nagyon célszerű konvenció, de a virtuális webszerverek nevének megadása nem azzal, hanem a fenti direktívával történik! Például a korábban bemutatott virtuális webszerverekre a következő módosítást alkalmazva a virtuális webszerver neve továbbra is **macska** marad!

```
feher10:/etc/apache2/sites-enabled# rm macska
feher10:/etc/apache2/sites-enabled# cd ../sites-available/
feher10:/etc/apache2/sites-available# mv macska macsek
feher10:/etc/apache2/sites-available# a2ensite macsek
Enabling site macsek.
Run '/etc/init.d/apache2 reload' to activate new configuration!
feher10:/etc/apache2/sites-available# /etc/init.d/apache2 reload
```

Folytassuk a direktívák megismerését!

```
ServerAdmin webgazda@tilb.sze.hu
```

Azt adja meg, hogy ki a szerver adminisztrátora. (Ez az e-mail cím jelenik meg a szerver által generált weblapokon, ha a **ServerSignature Email** direktívával engedélyeztük.)

```
DocumentRoot ”/var/www”
```

Itt megadhatjuk, melyik alkönyvtárban lesznek a honlap fájljai. A fő szerver esetén szokásos a fenti, a virtuális webszervereknél célszerű a szerverrel azonos nevű alkönyvtárat használni.

```
<Directory ”/var/www”>
```

Itt ugyanannak az elérési útnak kell szerepelnie, mint a **DocumentRoot**-nál.

Ez a **Directory** tag a **DocumentRoot-ot** jelöli meg egészen a **/Directory** tag-ig. Itt az alapvető elérhetőségek és jogok beállítását tehetjük meg.

```
    Options Indexes FollowSymLinks MultiViews
```

Opciók megadása:

- Indexes: ha nincs a könyvtárban **index.html** fájl, akkor a benne lévő fájlokat listázza (automatikusan elkészít egy kulturáltan kinéző HTML dokumentumot)
- FollowSymLinks: szimbolikus linkek követése
- MultiViews: egy weboldalak több reprezentációja is lehetséges, például nyelv, médiatípus stb. szerint. Ezek közül a kliens kérésének legmegfelelőbbet szolgáltatja a szerver.

```
    AllowOverride None
    Order allow,deny
    Allow from all
```

Az **AllowOverride**-al tudjuk az alkönyvtárakon engedélyezni, hogy a **.htaccess** fájlban foglaltak módosíthassák a jogosultságokat. (Ennek a fájlnak a neve az **AccessFileName** direktívával meg is változtatható.)

A sorrendben az alapvető logika: Először az engedélyt adunk meg, majd a tiltást.

Az „**Allow from all**” megengedi mindenkinek a kapcsolódást ehhez a könyvtárhoz.

```
CustomLog CustomLog ${APACHE_LOG_DIR}/access.log combined
```

Megadja az eléréseket tartalmazó log fájl útvonalát és nevét. Itt a letöltött fájlok adatai tárolódnak el (ki, mikor, mit szedett le).

```
</Directory>
```

A **Directory**-t lezáró tag.

Természetesen a fentieken túl még jó néhány egyedi beállítást tehetünk.

### 16.2.8 Felhasználói honlapok engedélyezése

Amennyiben szeretnénk, hogy a szokásos módon az egyes felhasználók a saját home könyvtárukban elhelyezett **public_html** könyvtárban honlapot készíthessenek, és az elérhető legyen a szokásos **http://szervernev/~usernev** formájú URL-lel, akkor engedélyeznünk kell a **userdir** modult:

```
feher10:/etc/apache2# a2enmod userdir
Enabling module userdir.
Run '/etc/init.d/apache2 restart' to activate new configuration!
feher10:/etc/apache2# /etc/init.d/apache2 restart
```

Természetesen az **a2enmod** használata helyett a szimbolikus linket kézzel is be lehet állítani, de akkor sem szabad elfeledkezni az **apache2** újraindításáról!

Ezután a felhasználók honlapja már működik is.

De érdemes még egy pillantást vetni a **mods-available/userdir.conf** fájl kincstári beállításaira:

```
<IfModule mod_userdir.c>
        UserDir public_html
        UserDir disabled root

        <Directory /home/*/public_html>
                AllowOverride FileInfo AuthConfig Limit Indexes
                Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
                <Limit GET POST OPTIONS>
                        Order allow,deny
```

```
                        Allow from all
                </Limit>
                <LimitExcept GET POST OPTIONS>
                        Order deny,allow
                        Deny from all
                </LimitExcept>
        </Directory>
</IfModule>
```

A **UserDir public_html** direktíva adja meg, hogy a felhasználók saját honlapjának milyen nevű alkönyvtárban kell lennie a saját home könyvtárában. Lehetőleg ne változtassunk rajta, ezzel sok kérdést előzhetünk meg a felhasználók irányából! :-)

A **UserDir disabled root** direktíva tiltja a root felhasználó honlapjának hasonló módon történő elérését. (Itt több felhasználót is felsorolhatunk, illetve tilthatjuk általában és engedélyezhetjük csak a **UserDir enabled** listában felsorolt felhasználóknak.)

Érdemes odafigyelnünk a **<Directory /home/*/public_html>** jelölésre! Ezt meg KELL változtatnunk, ha a felhasználók home könyvtárát történetesen nem a **/home** könyvtárban tároljuk!

A további beállításokból még annyit említünk meg, hogy a HTTP protokoll GET, POST és OPTIONS metódusait minden kliens számára engedélyezi, és minden más metódust minden kliens számára tilt.

A további direktívákat az érdeklődők megkereshetik az **apache2** aktuális verziójának a dokumentációjában: http://httpd.apache.org/docs/2.2/mod/directives.html

Amennyiben valaki az Internetről is elérhető webszervert készít, mindenképpen javasoljuk az **apache2** dokumentációjának mélyebb tanulmányozását!

# 17 Proxy szerver: SQUID és konfigurációja

A HTTP proxy feladata, hogy a web kliens és szerver közé ékelődve web cache-ként szolgáljon: az egyszer már lekért oldalakat tárolja és új kérés esetén a kliensnek kiszolgálja. Amennyiben a kliens hálózatán van, akkor a kliens hálózatának bemenő forgalmát jelentősen csökkentheti. Szerver oldalra is helyezhető, feladata ekkor csupán a szerver terhelésének csökkentése, ezt *reverse proxy*nak nevezik. Kliens oldalon hierarchikusan is szokták szervezni, például az egész szervezet is üzemeltet egy proxyt a teljes bejövő forgalomra és az egyes részlegek külön-külön egyet-egyet az ő bejövő forgalmukra.

Egy-két évtizeddel korábban (a 90-es években) igen nagy volt a jelentősége, azóta egyrészt nagy mértékben nőtt a hálózatok átviteli sebessége, másrészt a mai dinamikus weblapok miatt csökkent a cache-elés hatásfoka is.

Amennyiben nem transzparens módban használjuk, akkor a web kliensben kell beállítani, hogy a szerver helyett a proxy-hoz csatlakozzon. Transzparens mód esetén a kliens mit sem tud a proxyról, a szerverhez próbál csatlakozni, de a kapcsolat a proxynál végződik. Mindkét esetben kliens a proxyhoz kapcsolódik és a kliens kérését a proxy értelmezi, majd a proxy kéri le a szervertől a kliens által kért oldalt és továbbítja a kliensnek. A transzparens működéshez (a proxy megfelelő beállításán kívül) kell még valami (tipikusan **iptables**), ami „eltéríti” a kliens csatlakozását. Transzparens proxy esetén hogyan tudjuk kideríteni mégis, hogy egy oldalt proxy-n keresztül látunk? Például úgy, hogy hibás HTTP kérést küldünk, és megnézzük, hogy a válasz honnan származik (a szervertől vagy egy proxytól).

A legismertebb proxy a **squid**, ami a HTTP-n kívül FTP forgalom cache-elésére is használható.

A **squid** beszerzése és telepítése az **apt-get install squid** paranccsal lehetséges. E sorok írásakor (2010. 11. 21.) a 2.7-es verziót használjuk. A **squid** konfigurációs fájlja a **/etc/squid/squid.conf** alatt található. Nagyon sok beállítási lehetőség van, mi csak a legalapvetőbbeket tárgyaljuk.

```
#cache_peer
```

Ezt az opciót akkor használjuk, ha létezik a cache-hierarchiában fölöttünk álló úgynevezett parent cache (szülő cache) gép, amihez kapcsolódni tudunk (rajta keresztül kapcsolódunk a web szerverekhez), ha nincs ilyen, mint esetünkben, akkor ezt hagyjuk kikommentezve.

```
cache_mem 16 MB
```

Ez az opció adja meg, hogy mennyi memóriát szeretnénk adni a squid-nek cache céljára. Összességében a squid ennél több memóriát fog használni, ez pusztán a cache-elésre fordítható memóriát határozza meg! Nem szabad többet megadni, mint amennyi ténylegesen szabad RAM memóriánk van, mert semmi értelme, hogy a system swap-en legyen a cache.

```
cache_dir ufs /var/spool/squid 100 16 256
```

Ez a bejegyzés a cache alkönyvtárat adja meg, ide fogja a squid beszórni a becache-elt fájlokat. Az „ufs” a squid fájl-tárolási módja, ezt ne változtassuk, ha nincs rá okunk. Az első szám a maximális cache méretet adja meg MB-ban, a második kettő, pedig az alkönyvtár-rendszer méretét, itt tehát 16x256 könyvtárat fog a squid készíteni. (16 alkönyvtár lesz a **/var/spool/squid** alatt, és mindegyik alatt még 256 darab.) A **/var/spool/squid** alkönyvtárnak léteznie kell, és a **squid** számára írhatónak kell lennie, hogy a **squid** használni tudja.

```
icp_access deny all
```

Ezzel letilthatjuk, hogy más proxyk a mi gépünket parent cache-ként használják: senkinek nem engedjük, hogy rajtunk keresztül cache-eljen.

```
acl labor src 192.168.100.0/24
```

Ez az úgynevezett Access Control List, ilyen **acl** kezdetű sorokkal csoportokat adhatunk meg, hogy honnan, mit lehessen elérni a proxy-n keresztül. Ha **src**-t adunk meg, akkor azokat soroljuk fel (esetünkben a labor gépeit), akik igénybe vehetik a proxy szolgáltatást, ha **dst**-t, akkor azt adjuk meg, hogy milyen szerverekről tölthetünk le weblapokat.

```
http_access allow labor
http_access deny all
```

Ezzel megadhatjuk, hogy a fentiekben elkészített listák közül milyen szabály vonatkozik. (Itt a labor gépeit engedjük a cache-hez hozzáférni.)

```
cache_mgr drmomo@tilb.sze.hu
```

A szerver gazdájának e-mail címe.

```
cache_effective_user proxy
#cache_effective_group
```

Az a felhasználó, illetve csoport, ami alatt futni fog a **squid**. Ne használjunk root-ot! Alapértelmezett értéke a **proxy** nevű felhasználó, és az ő az alapértelmezett csoportja. (Lásd: **/etc/passwd** fájl.) A **/var/spool/squid** könyvtárat adjunk ennek a user-nek a tulajdonába!

```
visible_hostname cache.tilb.sze.hu
```

A név, ami látszani fog a cache neveként. (Pl.: a cache által generált weblapokon.)

```
access_log /var/log/squid/access.log squid
cache_log /var/log/squid/cache.log
cache_store_log /var/log/squid/store.log
```

A log fájlok helyei. A **/var/log/squid** könyvtárat is kell tudnia írni a squidnek.

Debian GNU/Linux alatt a **squid** a telepítéskor létrehozza a **/var/log/squid** és a **/var/spool/squid** könyvtárakat (valamint az utóbbin belüli könyvtárakat, és mindet a **proxy** felhasználó tulajdonába adja) és automatikusan elindul!

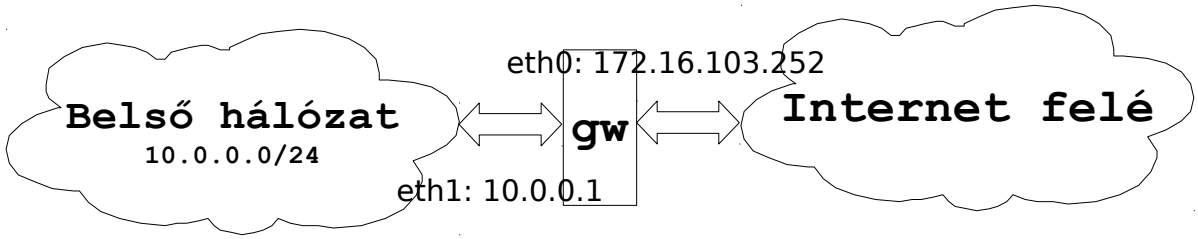
## *17.1 Transzparens proxy készítése*

A squid 2.6 verziója előtt a transzparens proxy építése viszonylag bonyolult volt, több opciót kívánt, most már egyetlen egy opcióval beállítható. Bővebben:

http://www.deckle.co.za/squid-users-guide/Transparent_Caching/Proxy

Az alábbiakban bemutatunk egy példa rendszert, hogy hogyan lehet ezt megoldani.

Tekintsük az alábbi hálózatot:




13. ábra: minta rendszer transzparens proxy megvalósításához

Itt a **gw** jelzésű gép az **eth0** interfészével a 172.16.103.0/24-es (privát IP címtartományú) hálózaton keresztül kapcsolódik az internet felé. Ahhoz, hogy az **eth1** interfészén keresztül a gépet transzparens proxyként használhassuk, előbb be kellett állítani, hogy egyáltalán internet elérést nyújtson. A parancsok (reméljük) magukért beszélnek:

```
ifconfig eth1 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.0.255 up
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth0 -s 10.0.0.0/24 -j MASQUERADE
```

Ezután a tesztelésre használt notebookon beállítottuk a 10.0.0.2 IP címet, a 255.255.255.0 netmaszkot, a 10.0.0.1 átjárót és a 172.16.103.1-es névkiszolgálót (ami egyben a 172.16.103.0/24 hálózat routere is). Ezután a notebookon ellenőriztük: az internet elérhető volt.

Ellenőrzésül a **gw** jelű gép routing táblázata:

```
feher10:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.103.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth1
0.0.0.0         172.16.103.1    0.0.0.0         UG    0      0        0 eth0
```

Most adjuk át a HTTP kéréseket a squid-nek!

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination
10.0.0.1:3128
```

A squid.conf-ban elég egyetlen egy opciót beállítani:

```
http_port 3128 transparent
```

Újraindítás után a transzparens proxy működik, amit többek között a naplófájlokból is láthatunk:

```
feher10:/var/log/squid# tail -n 3 access.log
1290361717.960     92 10.0.0.2 TCP_MISS/200 475 GET http://dev.tilb.sze.hu/ -
DIRECT/193.224.130.179 text/html
1290361721.046    572 10.0.0.2 TCP_MISS/404 653 GET
http://dev.tilb.sze.hu/favicon.ico - DIRECT/193.224.130.179 text/html
1290361801.698     78 10.0.0.2 TCP_REFRESH_HIT/304 313 GET http://dev.tilb.sze.hu/ -
DIRECT/193.224.130.179 -
```

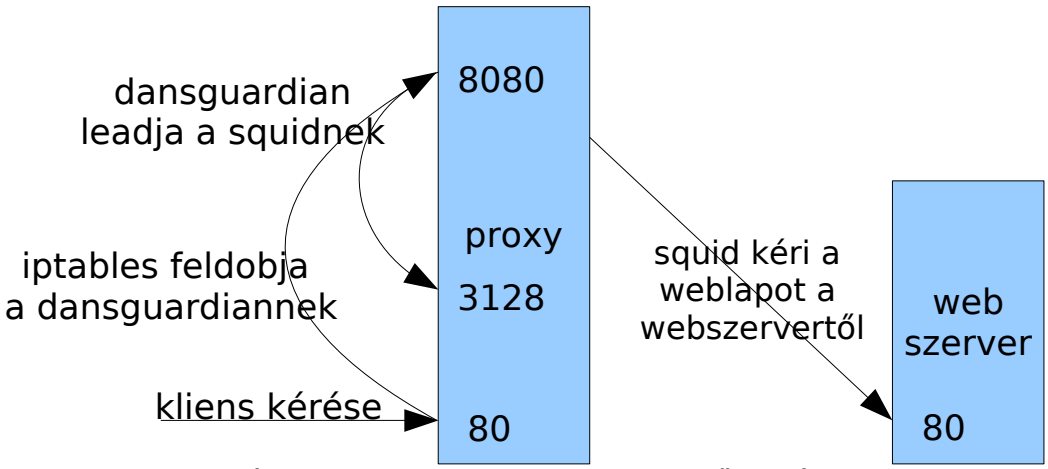
Az első két bejegyzés a honlap letöltésekor, a harmadik a böngészőben való frissítéskor keletkezett.

## *17.2 Dansguardian*

A **squid** egy nagyszerű eszköz, mely feladatát tökéletesen ellátja. De az idő múlásával, az igények változnak, melyeknek a **squid** nem felel meg teljesen. Ilyenek például a tartalomszűrés szavakra – előfordulás gyakorisága szerint –, vagy a vírus szűrés, hiszen mostanában mindenkinek lehet ingyenes webmail-je, amit a szolgáltatók nem ingyenesen szűrnek. A vírusok jelentős része a weben keresztül kerül hálózatunkba. Ezen problémák miatt felmerült az igény olyan proxy megoldásokra, melyek a tartalmat is vizsgálják mindamellett, hogy cache-elnek is. Ezt 3 eszköz kombinációjával érhetjük el: **dansguardian**+**squid**+**clamav**. A **clamav** egy nagyszerű és ingyenes vírusirtó, melynek van Windows portja is, a **squid** a web proxy, a **dansguardian** pedig a tartalomszűrő.

## 17.2.1 A dansguardian működése

A **dansguardian** a **squid** nélkül nem képes működésre.




14. ábra: dansguardian+squid működése

A lekért adatokat visszafele először a **squid** kapja meg, mely elcache-eli, aztán a **dansguardian**, mely a cachelt adatot ellenőrzi, és ha minden rendben, akkor az ellenőrzött adatot továbbítja a kliens felé. Ha a **dansguardian** valamely feltételének nem felel meg a kapott adat, akkor a **dansguardian** egy hibaüzenetet ad vissza.

## 17.2.2 A dansguardian konfigurálása

Meglepő módon a dansguardian a telepítés után használatra kész, bár nagyon szigorúak a beállításai. Az UNCONFIGURED kezdetű sort kell csak kommentezni a **/etc/dansguardian/dansguardian.conf** állományban. Ha feltelepítettük a **clamd** vírusirtó démont, akkor a dansguardian vírusszűrést is végez a kapott adatokon. A tiltásokhoz, engedélyezésekhez használt szavakat, url-eket, reguláris kifejezéseket **/etc/dansguardian** könyvtárban módosíthatjuk, ezekre külön nem térünk ki, mert az egyes verziók között nagy különbségek lehetnek, ugyanakkor egyszeri áttekintéssel is rájöhetünk, hogy melyik fájl mire való a könyvtárban.

# 18 MTA POSTFIX

A **postfix** jelenleg az egyik legelterjedtebb MTA a linuxos világban. Könnyedén kezeli a maildir formátumot, nagyon minimális konfigurációval is tökéletesen, megbízhatóan működik. Tud authentikálni adatbázisból, kezeli a virtuális felhasználókat és virtuális hosztokat. Könnyedén konfigurálhatjuk spam és vírusszűrésre is. A konfigurációs állományai a **/etc/postfix** könyvtárban találhatóak.

A következő fejezetben egy teljes konfigurációs állományon keresztül bemutatjuk a **postfix** legfontosabb tulajdonságait.

Előtte azonban fontos megemlítenünk néhány dolgot. A **postfix** nagyon kínosan ügyel arra, hogy az FQDN hostnevet használjuk. Azt, hogy mi a gépünk FQDN-je, a **hostname -f** paranccsal kapjuk meg. Ha itt nem FQDN van (hanem csak a hostnév), akkor a **/etc/hosts** fájlban kell beállítanunk az IP címünket, majd whitespace karakterrel elválasztva az FQDN-t:

```
193.224.130.172 users.tilb.sze.hu users
```

Szintén fontos a myorigin beállítása miatt, hogy a **/etc/mailname**-ben is az FQDN legyen beállítva.

Akárcsak a **sendmail** esetén, a **postfix**nek is meg lehet adni aliasokat. Esetünkben a **/etc/aliases** fájl tartalmazza a beállításokat. Például:

```
postmaster:    root
```

Ezt azt jelenti, hogy a **postmaster** leveleit a **root** fogja megkapni. Ha módosítottuk a fájlt, az új beállítások érvénybe léptetéséhez a **newaliases** parancsot kell futtatnunk.

## *18.1 A main.cf*

```
# ezt írja ki az smtp szerver mikor betelnetelsz és várja utána a Helo -t
smtpd_banner = $myhostname ESMTP $mail_name (Debian/GNU)

# a biff szolgáltatás új levélkor „new mail” üzenetet küld azoknak, akik kérték
biff = no
```

```
# appending .domain is the MUA's job.
append_dot_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h

# A szerver hostneve. Itt van szükség először az FQDN-re
myhostname = users.tilb.sze.hu

# az alias fájl elérési útja.
# minden aliasba tett bejegyzés után futtatnunk kell a newaliases parancsot
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases

# felhasználói fiók mérete
mailbox_size_limit = 0

# felhasználók max. ekkora levelet fogadhatnak byte-ban. Alapértelmezetten
# 10M körül van
# message_size_limit = 102400000

# az itt beállított név lesz a @ után. A trivial-rewrite daemon írja át a kimenő
# leveleket.
myorigin = /etc/mailname

# mely interfészen működjön a postfix
inet_interfaces = all

# milyen címekre érkező levelet tekintsen sajátjának a mail szerverünk
mydestination = users.tilb.sze.hu, users, tilb.sze.hu


# központi relay szerver használata, ha nem a local-t akarjuk használni
# levélküldésre (ez nagyon useful ha a cég belső hálózatán van egy
# levelezőszerver, amin keresztül kötelesek vagyunk kimenni, vagy ha nem a saját
# gépünkről akarjuk küldeni. Az egyetemen belül lehetőségünk van arra, hogy az
# rs1.sze.hu -t használjuk kimenő smtp szervernek
#relayhost = rs1.sze.hu

#milyen hálózatokból, hostokról engedélyezzük a levélküldést (nekik relay-ezünk)
mynetworks = 127.0.0.0/8, tok.sth.sze.hu, 10.1.6.67

# ez az alapértelmezett lenne, bár nem minden esetben lesz ez beállítva
#mailbox_command = procmail -a "$EXTENSION"

# Egy fájl/könyvtár neve a user home könyvtárában: ide fognak a leveleink érkezni.
# home_mailbox = Mail    # ez egy fájl, tehát mailbox formátum!
home_mailbox = Maildir/ # ez egy könyvtár, tehát maildir formátum!

# az authentikációhoz szükséges beállítások
smtpd_sasl_local_domain = $myhostname
smtpd_sasl_auth_enable = yes
smtpd_sasl_security_options = noanonymous
broken_sasl_auth_clients = yes
smtpd_recipient_restrictions =
           permit_sasl_authenticated
           permit_mynetworks
           reject_unauth_destination
```

## *18.2 A postfix korlátozásai*

Napjainkban egyre inkább felmerül az igény, hogy a levelező szerver ne továbbítson minden levelet, köszönhető ez a spammerek igen hatékony működésének. Bizonyára sokan kaptak már spammet viagráról vagy más hasznos létfontosságú dologról. Eltekintve néhány esettől, a spammek nagy része zombi gépekről jön, melyeket valamilyen backdoor vagy egyéb programmal megfertőztek, és alkalmassá tették levelek küldésére. Az ilyen gépek, levelek szűrésére ad lehetőséget a postfix néhány beállítása.

Ahhoz, hogy megértsük, hogyan is történik mindez, a *Számítógép-hálózatok* tárgyból már megismert levélküldést játsszuk el:

```
users:~# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 users.tilb.sze.hu ESMTP Postfix (Debian/GNU)
mail from: gecko
250 Ok
rcpt to: gecko@sid.sth.sze.hu
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
salala
.
250 Ok: queued as A4AFE3A0B9
quit
221 Bye
Connection closed by foreign host.
users:~#
```

Amint látjuk, ez elég egyszerűen ment. Ha jól megnézzük az RFC-t, akkor rájövünk, hogy elég banális hibákat vétettünk. Nem HELO/EHLO-ztunk. Nem valid e-mail a sender stb. A spam programok jó része ugyanezt a hibát elköveti. Illesszük a **`/etc/postfix/main.cf`** fájlba a következő sorokat:

```
smtpd_helo_required = yes

smtpd_recipient_restrictions =
        reject_unknown_sender_domain
        reject_non_fqdn_sender
        permit_mynetworks
        reject_unauth_destination
        reject_non_fqdn_hostname
        reject_invalid_hostname
        reject_rbl_client sbl-xbl.spamhaus.org
        permit
```

A szerver újraindítása után próbáljuk ki, mit csinálnak a fenti beállítások (és most szándékosan másik hostról jelentkezünk be).

```
dev:~# telnet users.tilb.sze.hu 25
Trying 193.224.130.172...
Connected to users.tilb.sze.hu.
Escape character is '^]'.
220 users.tilb.sze.hu ESMTP Postfix (Debian/GNU)
mail from: gecko
503 Error: send HELO/EHLO first
helo envagyok
250 users.tilb.sze.hu
mail from: gecko
250 Ok
rcpt to: gecko
504 <gecko>: Sender address rejected: need fully-qualified address
helo envagyok
250 users.tilb.sze.hu
mail from: gecko@mashol.hu
250 Ok
rcpt to: gecko
450 <gecko@mashol.hu>: Sender address rejected: Domain not found
helo envagyok
250 users.tilb.sze.hu
mail from: gecko@sid.sth.sze.hu
250 Ok
rcpt to: gecko
504 <envagyok>: Helo command rejected: need fully-qualified hostname
helo sid.sth.sze.hu
250 users.tilb.sze.hu
mail from: gecko@sid.sth.sze.hu
250 Ok
rcpt to: gecko
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
hoppamegyez
.
250 Ok: queued as E019B3A0B9
quit
221 Bye
Connection closed by foreign host.
```

Vegyük sorra a hibaüzeneteket, és hogy mi idézte elő azokat:

```
503 Error: send HELO/EHLO first
```

Mint láthatjuk, hiányzik a szervernek, hogy a kliens az RFC-ben előírt HELO/EHLO utasításokat használja. Ezt a **`smtpd_helo_required = yes`** paraméterrel értük el.

```
504 <gecko>: Sender address rejected: need fully-qualified address
```

Itt a sender (mail from:) e-mail címével van a gond. Ez nem rendes e-mail cím. A hatást a **`reject_non_fqdn_sender`** paraméterrel értük el.

```
450 <gecko@mashol.hu>: Sender address rejected: Domain not found
```

Ezt a küldő host nevének DNS feloldásában rejlő hiba miatt kaptuk. A postfix a küldő (és ha úgy állítjuk be a címzett) host nevét ellenőrzi, hogy az valóban rendelkezik-e **A** illetve **PTR** rekordokkal. A kapcsoló: **reject_unknown_sender_domain**.

```
504 <envagyok>: Helo command rejected: need fully-qualified hostname
```

Ez, mint látjuk egy újabb hiba a helo-ban, amit azért kaptunk, mert a helo után írt domainnév nem FQDN név. A szükséges kapcsoló: **reject_non_fqdn_hostname**.

Hasonló hatást érünk el, csak más hibaüzenetet kapunk a **reject_invalid_hostname** kapcsolóval, ha FQDN-nek látszó de nem **A** illetve **PTR** rekordokkal nem rendelkező hostnevet írunk.

Röviden a kapcsolók és azok, amiknek a kimenetét nem láttuk:

```
# az smtpdhez érkező kapcsolatokra vonatkozó korlátozások
smtpd_recipient_restrictions =
# dobja el ha a küldő hostneve nem feloldható
        reject_unknown_sender_domain
# dobja el ha a küldő e-mail címében a @ után nem FQDN áll
        reject_non_fqdn_sender
# Engedélyezze a levélküldést a mynetworks-ben lévő hostokról. Ha nem állítunk
# be semmit az smtpd_recipient_restrictions-nek ez alapértelmezett
        permit_mynetworks
# Hibaüzenettet ír ki, hogy a rendszerből tilos a relayezés, (kivéve a mynetworksnek
# hiszen így továbbítja a levelet) Ha nem állítunk be semmit az
#  smtpd_recipient_restrictions-ek ez alapértelmezett
        reject_unauth_destination
# dobja el, ha a(z) helo/ehlo után nem FQDN van
        reject_non_fqdn_hostname
# dobja el, ha a(z) helo/ehlo után nem feloldható hostnév van
        reject_invalid_hostname
# ellenőrizze le a kliens ipcímét, hogy spamlistán van-e.
        reject_rbl_client sbl-xbl.spamhaus.org
# azoknak akik a fenti követelménynek megfeleltek, engedélyezett a levélküldés.
        permit
```

Az **smptd_recipient_restrictions** után álló értékek sorrendje nem lényegtelen!!! A szabályok egymás után következnek, és csak akkor mennek tovább, ha az előtte lévő OK vagy DUNNO üzenetet adott. Tehát ha véletlenül a legelső bejegyzésünk reject, akkor az összes levelet eldobja a rendszer.

Szintén érdemes megemlíteni, hogy ha SMTP-nél nem HELO hanem EHLO-t használunk, akkor a rendszerről információkat kap a kliens. Például megtudhatja, hogy van VRFY és a

VRFY usernév SMTP paranccsal, hogy létezik-e az adott felhasználó. Ez azért kellemetlen, mert információt kaphatnak, hogy milyen felhasználók vannak a rendszerben. Ezt a **`disable_vrfy_command = yes`** paraméterrel tilthatjuk.

## *18.3 Maildir vs mailbox*

A régebbi Linux/Unix rendszerek a mailbox formátumot részesítették előnyben, ha a levelezésről volt szó. A beérkező leveleket az MTA a **`/var/spool/mail/$USERNAME`** fájlba tette. Ennek elvi hátránya, hogy ha megsérül a fájl vagy törlődik, akkor az összes levelezésünk odavész (ez nem szokott gyakran előfordulni). Újabban problémát postafiók mérete okoz. Sok nagy levélből lesz egy nagyon nagy mailbox fájl, amit az MUA-nek be kell olvasnia (ez például az rs1.sze.hu szerveren valóságos probléma, ráadásul a milbox fájl mérete legfeljebb 2GB lehet). A Maildir formátum ezeket a hibákat igyekszik kiküszöbölni. A régi monolit levélfájl helyett a leveleinket egyesével külön fájlokba menti el. A Maildir alapértelmezetten a felhasználó home könyvtárában található, de természetesen megoldható, a levelek **`/var/spool/mail/$USERNAME`** alá helyezése is, a különbség annyi lesz, hogy a **`$USERNAME`** ebben az esetben nem fájl, hanem könyvtár.

# 19 Courier POP3, IMAP szervercsalád

A courier egy nagyon sokrétű szervercsalád. Van smtp, pop3, pop3s, imap4, imap4s szervere. Ezek tudnak authentikálni adatbázisból, vagy pam segítségével. A telepítésük egyszerű, sok szakértelmet nem követelnek. Debian alatt telepítés után gyakorlatilag azonnal jól működnek. A Maildir formátumot használják, ezért szükséges az MTA-t helyesen beállítani, hogy a leveleinket a megfelelő módon szortírozza. Telepítése **apt-get**-tel történik. Az alábbi courier csomagok állnak rendelkezésünkre:

- **courier-imap-ssl** : imap4s szerver
- **courier-imap**: imap4 szerver
- **courier-pop-ssl**: pop3s szerver
- **courier-pop**: pop3 szerver
- **courier-mta-ssl**: ESMTP ssl felett
- **courier-mta**: ESMTP

Célszerű telepítés után az SSL nélküli daemonokat tiltani, vagy megoldani, hogy csak a **localhost**ra figyeljenek (bindeljenek). Ezt a konfigurációs állományon belüli ADDRESS=127.0.0.1 beállítással tehetjük meg. A courier szerverek konfigurációs állományai a **/etc/courier** könyvtár alatt találhatóak. A **.cnf**-re végződő nevű fájlok az ssl protokollt támogató szerverek tanúsítvány beállító fájljai, ebből generáljuk a pem kiterjesztésű fájlokat a **/usr/lib/courier/mk[imapd|pop3d|stb]cert** szkriptekkel.

# 20 Ábrajegyzék