Tartalomjegyzék

1 Bevezetés

Ez a jegyzet a Széchenyi István Egyetem Távközlési Tanszékének – Távközlés informatika szakirányán oktatott, Hálózati operációs rendszerek I. című tantárgyhoz (ta65vi) készült. Felhívnám a figyelmet, hogy a jegyzet magában nem elegendő, a tantárgy teljesítéséhez ajánlott a gyakorlatokon és az előadáson való részvétel!

1.1 A UNIX története

A UNIX első változatát 1969-ben készítette el Ken Thompson és Dennis 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

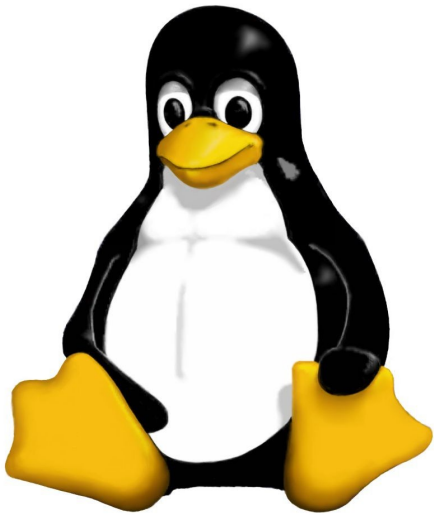
célja nem az volt, hogy felhasználókat szerezzen, hanem az, hogy segítséget kapjon a rendszermag (KERNEL) fejlesztésében.

A 0.11 verzió megjelenése új korszakot nyitott a Linux történetében.

A 0.11 verzió újdonságai:

- demand loading
- kód- és adat megosztás nem kapcsolódó processzek közt
- sokkal jobb floppy-vezérlők (most már többnyire működnek)
- hibajavítások
- Hercules/MDA/CGA/EGA/VGA támogatás
- a konzol hangot is ad (Óh! Fantasztikus rendszermag!)
- mkfs/fsck/fdisk (fájlrendszer-karbantartó programok)
- amerikai/német/francia/finn billentyűzet
- a soros portok sebessége beállítható

A fejlesztés a POSIX-nak való megfelelés felé terelte a tökéletesítést, ekkor már próbáltak elszakadni a MINIX-től. A szétválás elég szégyenletesen zajlott le Linus és a Minix atyja, Andrew Tanenbaum között egy internetes hírcsoportban.



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 ezen túl 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, és itt fontos megemlíteni, hogy ha ez a 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 [1] internet címen. A 2.6.x-os szériában valamelyest megváltozott a kernel fejlesztése. A 2.6.x a stabil kernelfa 2.7.x nincs, és előrelátható ideig nem is lesz. 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. Egyelőre úgy tűnik, ez nem megy a stabilitás rovására. Ha valaki az átlagosnál stabilabb kernelt szeretne, annak az Alan Cox -féle „-ac”[1] kernel patch-et javasolhatom. Szintén meg kell jegyezni, hogy az eddig megszokottaktól eltérően, nem „csak” három verziószám szerepelhet egy kernelfában, hanem négy is. A 4. verziószámmal rendelkező stabil kernelek kiadását, a sürgős, kritikus hibák teszik indokolttá, így nem kell megvárni, míg egy újabb stabil kernel kerül kiadásra a bugfixekkel (javításokkal).

1.3 Linux disztribúciók

A Linux 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 4.0 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.

1 „-ac” jelöli az Alan Cox általa készített foltot, mely a kernel verziójához is ezt az azonosítót fűzi.

Honlapja: www.ubuntu.com.

- Slackware: Az első disztribúciók egyike. Más csomagkezelőt használ, mint a Debian, frissebbek a stabil kiadások csomagjai. A régi szellemiségét sajnos elvesztette. 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. A legelterjedtebb live disztribúció a Knoppix (www.knoppix.org).

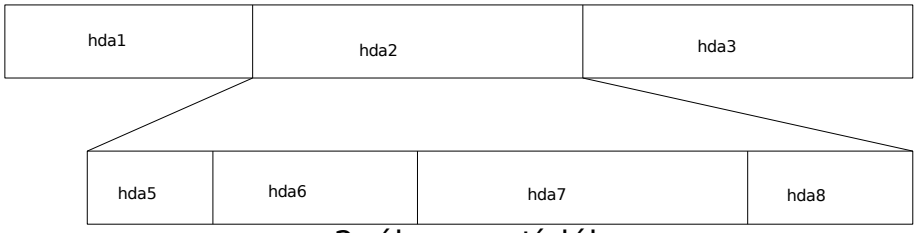
1.3.1 Debian GNU/Linux

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. Egy időben létezett FreeBSD kernelre épülő Debian terjesztés és létezik mind a mai napig 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 írásakor, a stable verzió a Etch, a testing az Lenny, és létrehoztak egy nem karbantartott oldstable ágat, ami a régi stabil kiadás volt, ennek a neve Sarge. Magyarországon több tükörszervere is létezik (a laborban egy lokális mirror-t is készítettünk) az ftp.hu.debian.org-ot célszerű használni, ez jelenleg a leggyorsabb Debian mirror.

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 egyik elsődlegesen belül több darab másodlagos partíció helyezkedhet el. Ezeket a Linux képes kezelni, és könyvtárakba felcsatolni különböző paraméterekkel: írási jog, futtatási jog, stb. Lehetőségünk van bootloaderek segítségével különböző partíción lévő más-más típusú operációs rendszerek betöltésére.



2. ábra: partíciók

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. 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.

1.4.5 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 [destination]`.

1.4.6 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[2]-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.

1.4.7 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.
```

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

```
~~: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

# 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.

1.4.8 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
- 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.

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.

2.1 Alapvető parancsok

`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 kiírás. `-a`, `--all` kapcsoló minden fájlt kilistáz, azaz a „.”-al kezdődő rejtett fájlokat is megjeleníti. Itt jegyezzük meg, hogy létezik két speciális fájl a könyvtárakban. Az első a „.”, ami a könyvtárat jelenti, a második a „..”, ami a könyvtár szülő könyvtárát jelenti. Természetesen a „..” a „/” esetében szintén önmagát jelenti. A kapcsolókat lehet egymás után megadni pl.: `ls -la /home`

`cd`: change directory = könyvtárváltás. Ugyanúgy, mint a DOS-ban 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: aktuális könyvtár kiíratása, ahol éppen tartózkodunk

cp:, azaz copy = másolás. cp [kapcsolók] <mit> <hova/milyen_néven>. -r rekurzív másolás. pl.:

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

mv: azaz move = mozgatás: mv <mit> <hova>,

```
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 a 16. oldalon)

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 mountolt 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. Kapcsolók (nincs „-” !!!): `a` minden 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
```

`kill`, `killall`: processzek vezérlésére szolgál. Ilyenek lehetnek a `-9`, vagy más néven `KILL`, a `CONT`, ami a `STOP` szignállal megállított processzt indítja újra, 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 proceszt 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. Pl.: `more hosszuszoveg.txt`

`less`: ugyan az, mint a `more`, csak lehet soronként és oldalanként fel/le lépkedni.

`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` becsomagolom az `ize` könyvtár tartalmát:

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

Kicsomagolom az `ize.tar.gz fájlból`

```
# tar -zvxf 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`, `depmod`, `modprobe`, `modinfo`, `insmod`, `rmmod`: a Linux kernel moduljait tudjuk betölteni, listázni, függőségi viszonyt beállítani, eltávolítani (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 változó beállításokkal vált át a másik felhasználókra.

`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.

2.2 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 használhatjuk az `apt-get` parancsot, ami a `dpkg` frontend-je. Szintaxis: `apt-get <kapcsolók> <parancs> <csomagnév>`

Ennek sok opciója 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 a`pt-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` 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 örök hívő 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 meg kell említenem a `tasksel` nevű programot, ami arra hivatott, hogy egy általunk kiválasztott funkciót/funkciókat ellátó gép dependenciáját (függőségét) beállítja, majd telepíti a csomagokat.

2.3 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
```

2.4 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 debian-binary control.tar.gz data.tar.gz
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 (pl.: ilyen szkriptek kérdezgetnek bennünket, postfix telepítésnél). A prerm, postrm parancsok a telepítés előtt illetve után eltávolítandó fájlokról gondoskodnak. Ezek szintén perl szkriptek.

2.5 A vi, vim kezelése

A `vi` egy alapvető szövegszerkesztő program UNIX rendszerekhez. A `vim` a `vi` utódja, lényegesen többet tud nála (A továbbiakban a régi `vi`-t `nvi`-nak a `vim`-et `vi`-nak nevezem!). Szintaxis kiemeléssel rendelkezik (színezi a beírt forrásokat). Néhány hasznos segédprogram van hozzá: `vimdiff`, `view`.. A `vi` 2 üzemmódban dolgozik: megjelenítő (visual) és szerkesztő módban. A kurzort a következő gombokkal tudjuk mozgatni:

↑ k

h ← → l

↓ j

Visual módban, a „:” gomb megnyomása után 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!
1

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 shell-t 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 ≈ 500Kbyte]
- csh (Pl.: IBM AIX, IRIX) [Mérete ≈ 320Kbyte]
- ksh (Pl.: AT&T UNIX, Solaris, HP-UX) [Mérete ≈ 630Kbyte]
- ash, dash (Pl.: Minimum shell, boot lemez esetén) [Mérete ≈ 60Kbyte]

3.1 Shell-ek fajtái, kezelésük

3.1.1 A bash

A Linux különböző kiadásai általában a bash nevű shell-t 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 fájl, parancs 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 bash esetén, 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='ls --color'
bash-2.05> alias dir='ls -la'
bash-2.05> alias cp='logout'
```

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='ls --color'
bash-2.05> unalias cp
alias dir='ls -la'
alias ls='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 Shell szkriptek írása

Akárcsak a DOS esetében (persze hibás a DOS-ra hivatkozni, hiszen a Unix típusú rendszerek és parancsértelmezői lényegesen korábban léteztek) a *nix típusú rendszerek parancsértelmezői tudnak szöveges fájlokat feldolgozni, amelyek a parancsértelmező számára érthető, végrehajtható parancsokat tartalmaznak. A DOS-nál a .bat kiterjesztés az, ami az ilyen fájlokat „jelölte”, a shellek esetében a magic number és a fájl `--x` (futtatható) joga teszi 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.

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

Jel	magyar jelentés	angol jelentés
{}	kapcsos zárójel	(curly) brace
( )	zárójel	parenthesis
~	tilde	tilde
[ ]	szögletes zárójel	(square) bracket
”	macskaköröm	double quote
’	aposztróf	(single) quote
\	vissza per	backslash

3.2.1 A kapcsos zárójel kifejtése (Brace expansion)

A kapcsos zárójel a bash-ban szövegrészek automatikus behelyettesítésére használható:

```
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
```

Lássunk egy összetettebb példát is:

```
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}}
```

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

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

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!

3.2.4 Eredmények helyettesítése (Command substitution)

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

```
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)
```

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 (2,3) közül mindig az utolsó érték érvényes
8
bash-2.05> echo $((2!=(a=3)))
1
```

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 szálból álló stringet vagy adatsort szavakra bont. A szavak határát egy 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-el 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
```

3.2.7 Elérési utak 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:

- * A helyén nulla vagy bármennyi tetszőleges karakter állhat
- ? A helyén egy darab tetszőleges karakter állhat
- [XYZ] A helyén a felsorolt karakterek bármelyikéből 1 darab állhat
- [a-g] A helyén a megadott karakter-tartományból való karakterek bármelyike állhat
- [^XYZ] A helyén megadott karakterek közül egyik sem állhat

```
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[1,2]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
```

3.2.8 Idézőjelek kifejtése (Quote removal)

Munkáink során az egybefüggő szöveges változókat (string-eket) általában idézőjelszerű karakterekkel védjük. Ilyenek a ’ és a ”. A ” között álló változók kiértékelésre kerülnek, míg a ' 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 a ' és ” között:

```
laptop:~# export valtozo="ertek"
laptop:~# echo "$valtozo"
ertek
laptop:~# echo '$valtozo'
$valtozo
laptop:~#
```

3.2.9 A standard ki és bemenetek irányítása

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 kimeneten (STDOUT) olvashatjuk a program kimenő üzeneteit. 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 file -ba 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 programok standard bemenetről veszik az adatot, ami viszont egy pipe (cső) kimenetéhez van csatlakoztatva:

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

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 crontabok használatakor fontos, ha nem akarjuk, hogy minden kis warn-ra e-mailt küldjön a rendszer.

```
laptop:~# cat > irok <<EOF
> Ez arra szolgál,
> hogy több sorba 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
laptop:~#
```

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

3.2.10 Bash specifikus fájlok

A bash-nek több üzemmódja létezik: interaktív login shell, interaktív, de nem login shell, valamint nem interaktív, nem login 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.

3.2.11 A bash néhány fontosabb környezeti változója

IFS # internal field separator, white space karakterek beállítása

PATH # elérési utak beállítása

HOME # a felhasználó home könyvtára

MAIL/MAILCHECK # mail a felhasználó postafiókja, check mennyi időközönként ellenőrizze van-e új levél

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

HISTSIZE # hány parancsot tároljon a bash

HISTFILE, HISTFILESIZE # ugyan ez csak a history fájlra vonatkozik

A következő környezeti változókat a bash saját maga állítja be:

PWD, OLDPWD # az aktuális és az azt megelőző könyvtár (cd - emlékeztek?)

UID, EUID # felhasználó user ID-je

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

3.3 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). Tökéletes választás, ha automatizálni akarunk különféle szerkesztéseket (pl.: logfájl kimenetének formázása). Néhány egyszerűbb majd bonyolultabb 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.

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

4. példa: zárójelek használata

```
laptop:~/sed# cat sorok.txt | sed 's/\(4.\) \(sor\)/\2 \1/'
1. sor
2. sor
3. sor
sor 4.
5. sor
6. sor
7. sor
laptop:~/sed#
```

5. példa: látható, hogy a \0 az egész keresett kifejezést kiírja, függetlenül, hogy van-e zárójellel „blokk” képezve, a második megoldásban a \0-át a & helyettesíti.

```
laptop:~/sed# cat sorok.txt | sed 's/\(4.\) sor/\0 \0/'
1. sor
2. sor
3. sor
4. sor 4. sor
5. sor
6. sor
7. sor
laptop:~/sed# cat sorok.txt | sed 's/\(4.\) sor/& &/'
1. sor
2. sor
3. sor
```

```
4. sor 4. sor
5. sor
6. sor
7. sor
```

6. példa: csere ismétlése a „g” kiterjesztés hatására

```
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#
```

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

3.4 AWK

Az awk szintén egy szövegfeldolgozó, ami soronként dolgozza fel a STDIN-ről érkezett adatokat és ezt a STDOUT-ra küldi. A kapott sztringen nagyon sok műveletet képes elvégezni, többek között például külső parancsok meghívására képes, ez hasznos, ha a feldolgozandó fájlból ki akarunk szedni hostnevet, IP címet, és az IP címet a host paranccsal feloldatjuk, majd a kapott értékeket újfent feldolgozzuk a szkriptünkkel. Erre majd óra keretén belül igyekszünk példákat mutatni. Természetesen sokkal egyszerűbb példák is vannak, melyeket használunk mindennapjaink során, például mikor egy `/etc/passwd` fájlból kiszedjük az 1000-65000 UID közé eső felhasználókat. Ezt a következő rövid szkript valósítja meg:

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

Az `awk` parancs után álló `-F \:` a field separator (mező elválasztó) beállítása, hiszen a `/etc/passwd` fájl mezőit a „:” választja el egymástól. A „:”-ot meg kell „védeni” a bash kiértékelésétől egy „\” (backslash) jellel. Mint említettük, az awk sorfeldolgozó. Az egész sorra, amire a keresési feltételünk illeszkedik, a „$0” változóval hivatkozhatunk. A sor első mezőjére a $1, a második $2 stb. hivatkozhatunk. A print parancs - akárcsak sok programozási nyelvben - a mögötte álló kifejezéseket írja ki. A print parancs szintaktikája a következő: print $1 „szöveg1” $2 „szöveg2” stb., A kapott sztringeket és változókat közvetlenül egymás mögé írja ki (tehát 2 változó kiíratása esetén nekünk kell gondoskodni az elválasztó karakterekről pl.: tabulator „\t”; szóköz „ ” stb.). Mint láthatjuk, lehetőségünk nyílik feltételes elágazás használatara is. Az if „()” közötti szintaktikája megegyezik a C

szintaktikájával (==;<;<=;>=;>;&&;||; stb). Ezek után nézzük, mit csinál a fenti kis rövid szkript: Ha a „$3” nagyobb egyenlő, mint 1000 és $3 kisebb egyenlő, mint 65000, akkor írd ki a „$1”-et, ahol a „$3” az UID mező a fájlban, a „$1” mező pedig a felhasználónév.

Tekintsünk egy másik, a fentihez hasonló példát azzal a kiegészítéssel, hogy a min és max változókkal megadott UID értékek közé eső felhasználók számát a szkript lefutásának végén adjuk meg.

```
cat /etc/passwd | awk -F \: 'BEGIN{min=1000; max=65000; count=0} {if ($3 >= min && $3
<= max) { print $1 ; count=count+1 }} END {print count}'
```

A fenti példában láthatjuk, hogy van egy BEGIN és egy END blokk. A BEGIN a bemenet feldolgozása előtt végrehajtódik, ide célszerű a változók deklarálását megadni. Az END blokkban pedig olyan műveleteket hajthatunk végre, melyek a bemenet feldolgozása után adnak eredményt.

Egy awk szkript a következő módon épülhet fel:

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

Folytassuk a korábbi szkript taglalását! A BEGIN részben beállítottuk a min, max és count változót (ez egyébként nem kötelező, hiszen alapértelmezetten 0 minden numerikus változó és üres string minden szöveges változó kezdőértéke). A második blokkban láthatjuk, hogy az if (feltétel) után szintén képezhetünk blokkot, amibe utasításokat téve a feltételtől függően kerülnek kiértékelésre. A szkriptünk legvégén pedig, az END-el kiíratjuk a count változót. Ha ezt nem itt tennénk meg, akkor a változó minden sor vizsgálata után kiírásra kerülne, ami számunkra teljesen felesleges.

3.5 FIND

A find parancs a fájlrendszerben hajt végre keresést. Segítségével listázhatunk különböző mintáknak megfelelő fájlokat, legyen az a fájl neve, egy regexpre 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. Ez a régebbi (és ha be van kapcsolva, akkor az újabb) rendszerek esetében a futás közben crash-el leálló 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 {} ami a fájlok „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 overflowhoz kihasználáshoz szükséges setuid bitek megkeresésével:

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

A régebbi find esetében a „/"-t a „+" 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 -en 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.

3.6 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 ha egy MAC addresst, 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#
```

3.7 Reguláris kifejezések

A reguláris kifejezésekkel (regexp-ekkel) sztringeket, karakterláncokat helyettesíthetünk egy, csak a keresett csoportra jellemző mintával.

karakter	jelentése
.	bármelyik karakter
\	a mögötte álló karaktert „védi” meg
$	az előtte álló kifejezés a sor végén áll
^	ha a regexp elején áll, a mögötte álló kifejezés akkor illeszkedik, ha a keresett kifejezés a sor elején áll
[...]	a bracketben felsorolt karakterek (...) közül bármelyik lehet
<	a keresett kifejezés szó elején áll
>	a keresett kifejezés a szó végén áll

karakter	jelentése
[^...]	a felsorolt karakterek (…) nem lehetnek

A reguláris kifejezéseknél, ha több karaktert keresünk, akkor a számosságot is megadhatjuk:

kifejezés	jelentése
*	bármennyi az előtte álló kifejezésből, amit legelsőnek talál
\+	legalább egy, de bármennyi (többszörös szóközök törlése)
\{n\}	pontosan n darab az előtte álló kifejezésből
\{n,\}	legalább n darab az előtte álló kifejezésből
\{n,m\}	legalább n, maximum m darab az előtte álló kifejezésből

Néhány egyszerű példa:

IP cím megfeleltetése (nem vizsgáljuk, hogy max 255 lehet a decimális szám értéke):

```
\([12]\{0,1\}[0-9]\{1,2\}\.\)\{3\}\([12]\{0,1\}[0-9]\{1,2\}\)
```

MAC address megfeleltetése (tegyük hozzá némi hibával, mert ha a MAC-en belül keverve használom a „:” és „-” jeleket a regexp illeszkedik rá, holott nem lenne szabad):

```
\([0-9a-zA-Z]\{2\}[-:]\)\{5\}[0-9a-zA-Z]\{2\}
```

Az `/etc/shadow` fájl első oszlopában lévő szavakra illeszkedő regexp. Tudjuk, hogy az oszlopokat „:” választja el egymástól, és nekünk bármennyi karakter lehet az első „:” -ig. Az emberek nagy része ezt írná:

```
.*:
```

A fenti kifejezés azt jelenti, hogy bármennyi karakter egy „:”-ig. Az elképzelés majdnem jó, de sajnos a regexp-ek a lehető legnagyobb illeszkedést támogatják, tehát a fenti kifejezés az utolsó „:”-ig mindent megjelenít:

```
laptop:~# cat /etc/shadow | grep -o '.*:' | tail -n3
sshd:!:13322:0:99999:7:::
proftpd:!:13329:0:99999:7:::
ftp:!:13329:0:99999:7:::
```

A helyes megoldás az lenne, ha azt mondanánk: minden olyan karakter ami nem „:” szerepelhet akárhányszor:

```
laptop:~# cat /etc/shadow | grep -o '[^:]*' | tail -n3
sshd
proftpd
ftp
```

Az ebben a fejezetben szereplő parancsok, és a reguláris kifejezések a *nix-os világban nélkülözhetetlenek lettek. Az ember hamar rájön, hogy rengeteg munkát spórolhat meg egy jól megírt szkript segítségével. Ezért kérek mindenkit, hogy ne csak a fenti parancsokat „magolják” be. Keressenek feladatokat, kihívásokat, amelyek segítségével sokkal mélyebben elmerülhetnek a reguláris kifejezések világában. Nélkülözhetetlenségüket 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 kártékony kódokat futtathatnának pl. a gépeinken) mivel valósítanánk meg, ha nem lenne kezünkben ez az eszköz?

3.8 Egyszerű shell szkript példák

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

Ha szeretnénk a könyvtárunkban található .htm fájlokat .html-re kicserélni, megtehetjük ezt a következő szkript segítségével:

```
# 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 *-ig számlál, amennyi fájl van a könyvtárban. A szkript minden egyes fájlt megpróbál átnevezni, mivel a nem .htm végződésű fájlokon a sed nem végez konverziót, az mv nem nevezi át ugyanarra a névre a fájlt, így hibaüzenetet ír ki. Már volt róla szó, hogy a „`” karakterek használata a végrehajtott parancs kimenetét helyettesíti. Másképpen a $( ) –el tudjuk megadni.

```
laptop:~# echo `which ls`
/bin/ls
laptop:~# echo $(which ls)
/bin/ls
laptop:~#
```

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 használt paraméter az [:upper:] a nagybetűket, míg a [:lower:] a kisbetűket jelöli.

```
# 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 ékezeteket, 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
árvíztűrőtükörfúró
próba
proba file
dev:/tmp/test# ../test.sh
dev:/tmp/test# ls -1
arvizturotukorfuro
proba
proba_file
dev:/tmp/test# cat ../test.sh
#!/bin/bash

export IFS=$' \t\n'

for i in *; do
```

```
mire=`echo $i| tr "öőóüúűéái " "oooouuueai_"`
mv "$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
```

3.9 Shell programok

Mint már említettem, a szkripteket egy futtatható fájlba írva is megoldhatók a feladatok. Sőt, a shell rendelkezik olyan beépített ciklusokkal, amelyek csak shell programokban használhatók, ilyenek a vezérlési szerkezetek, if..fi, case..esac, while..do, stb. A shell képes a program argumentumait kezelni, és változókba helyezni őket: $# $0 $1, stb. Nézzünk néhány egyszerű példát, melyekkel egy szöveget íratunk ki, majd argumentumszámot, és néhány argumentumot.

```
laptop:~/bash# ./hw.sh
Hello World!
laptop:~/bash# cat hw.sh
#!/bin/bash

echo "Hello World!"
laptop:~/bash#
```

A program futásának eredménye a programlista. Az első sor, mint már említettük, a „#!” -el kezdődik, ami a unix magic number-t állítja be, és az értelmező tudni fogja, hogy a futtatható állomány egy shell program. A „#!” után az értelmező teljes elérési útja szerepel. Az echo parancsot pedig már tanulmányaiból mindenki ismeri. A következő példa az argumentum számának kiírása és ellenőrzése.

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

A 2. sorban feltétel vizsgálata történik. A „$#” az argumentumok száma. A „[” a test parancs hardlinkje. A test parancs rengeteg feltételt tud vizsgálni, a sztringek egyezőségétől a számok egyenlőségi vizsgálatáig. A „`-lt`”, a less than vizsgálat, a „`-gt`”, a greather than, a „`||`” a logikai „vagy” kapcsolat. A fentiek ismeretében tehát, ha az argumentumok száma kevesebb, mint 1 (nem adunk meg egyáltalán argumentumot), vagy több, mint 3 akkor a 3-4. sor kerüljön végrehajtásra, ha a fentiek nem teljesülnek, akkor a 6. sor.

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.

4.1 Konfiguráció elkészítése

Mielőtt nekiállnánk, le kell töltenünk a kernel-package, bzip2 (ha ezt a típusú kernelt akarjuk letölteni), valamint a libncurses5-dev csomagokat. 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-2.6.14.3.tar.gz

Miután letöltöttük csomagoljuk ki a /usr/src könyvtárba:

```
# tar –xvzf linux-2.6.14.3.tar.gz; tar –jvzf linux-2.6.14.3.tar.bz2
```

Kibontás után készítsünk egy szimbolikus linket a könyvtárra.

```
root@pc0:/usr/src# ln –s linux-2.6.14.3 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.

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 (pl.: 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-ATA, 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 órai 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 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. Ezek a parancsok:

A helyi fordítókörnyezettel rendelkező gépen:

```
# make-kpkg --bzImage --initrd kernel_image
```

Ahol a –initrd kapcsolóval a kernel-package csomag, a deb állományt, initrd kompatibilisen készíti el,

A távoli gépen az install:

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

Fontos csomagok, parancsok:

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 rendszer lényege

A számítógépek szolgáltatásainak – felhasználási területtől függően – elérhetőnek kell lennie több felhasználó számára is. Ez az igény egyszerűen úgy jelentkezhet, hogy a felhasználók nem egyidejűleg, hanem felváltva használják a számítógépet. Ilyenkor már szükséges olyan szolgáltatásokat bevezetni, amely lehetővé teszi, hogy minden felhasználó önálló futtatási környezetben dolgozhasson, például mindig saját állományait futtathassa, 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 merül fel, 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, OS/2 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-elosztás 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 fájl művelet a naplóban előre eltárolódik, így 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, NT) 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 melyek nagy biztonsággal dolgoznak az NTFS-el). A hálózati meghajtókat is fájlrendszerként kezeli a UNIX, ilyen például az NFS és az SMBFS, CIFS is.

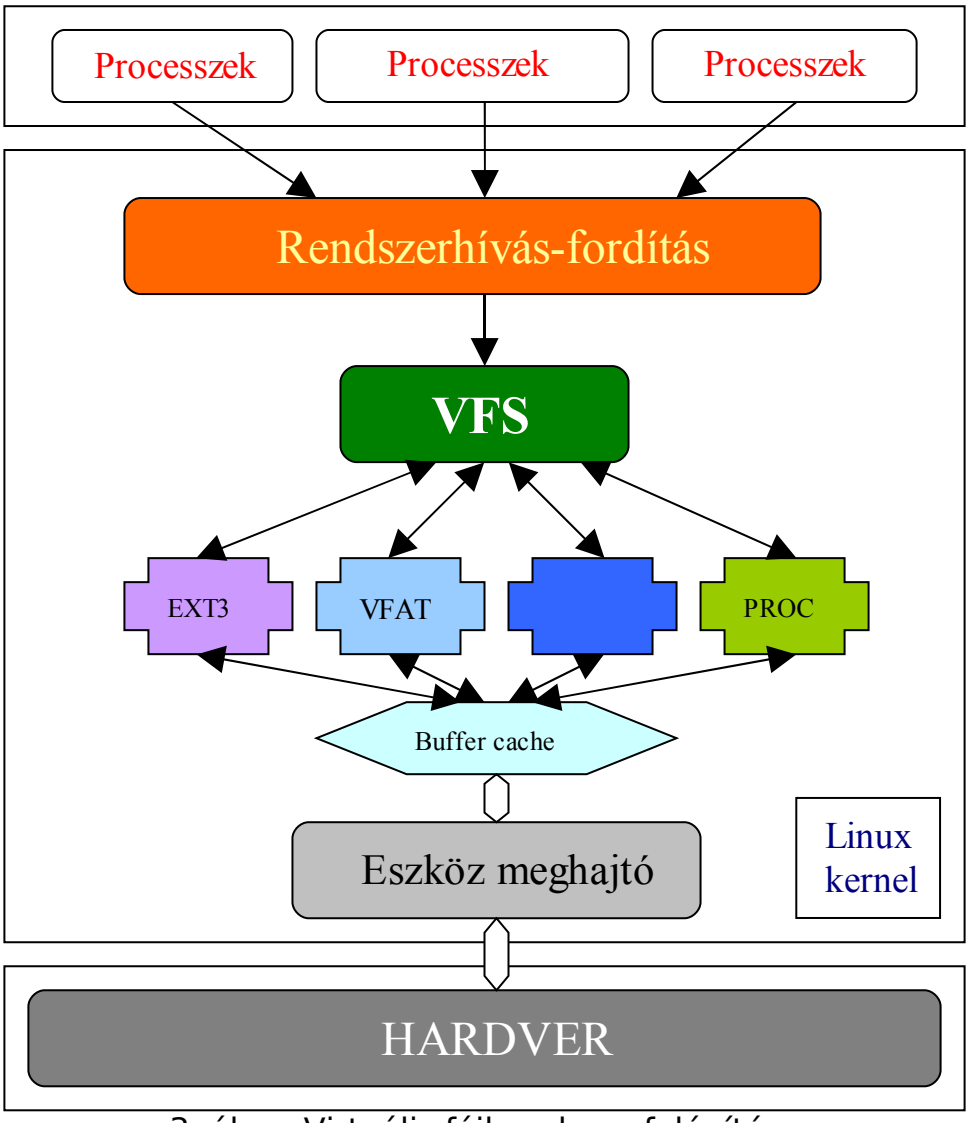
- ext2, ext3: A linux extended fájlrendszerei
- reiser3, reiserfs4: Hans Reiser csapata által fejlesztett fájlrendszerek
- xfs: A Silicon Graphics által fejlesztett nagygépes fájlrendszer
- jfs: Az IBM fájlrendszere
- NTFS, vfat: A Microsoft operációs rendszereinek a fájlrendszere. A pendrive-ok vfat

fájlrendszert használnak általában.

- NFS, SMBFS, CIFS: Hálózati fájlrendszerek
- minix: Andrew Tanenbaum operációs rendszerének a fájlrendszere
- cramfs: Read Only fájlrendszer
- JFFS2: beágyazott rendszerek kedvelt fájlrendszere

5.3 A VFS, virtuális fájlrendszer

A Linux - a különleges 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 2. 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>. Kapcsoló –t fájlrendszer típus a kernel által ismert típusokat a cat /proc/filesystems paranccsal nézhetjük meg. Ha az /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 kapcsolók nélküli kiadásával is megtekinthetjük. A kapcsolók az adott fájlrendszertől függenek. A mount paranccsal lehet loop eszközként CD, DVD imageket (képeket) felcsatolni a fájlrendszerünkhöz: Példa mount -o loop dvd.img /ahova/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 (menetközben) 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/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 1Kbyte többszöröse lehet (1024, 2048, 4096 byte a szokásos). Ha például egy 10Kbyte-os JPEG fájt helyezünk el egy 4096 byte-os blokkméretű partíción, akkor az valójában 12Kbyte-ot fog elfoglalni, mert 3 darab egyenként 4 Kbyte-os blokkot fog lefoglalni. Ennek a 3 darab blokknak a partíción belüli címét írjuk bele az inode megfelelő adatmezőjébe. Az inoderól lde (Linux disk editor) paranccsal kaphatunk információt:

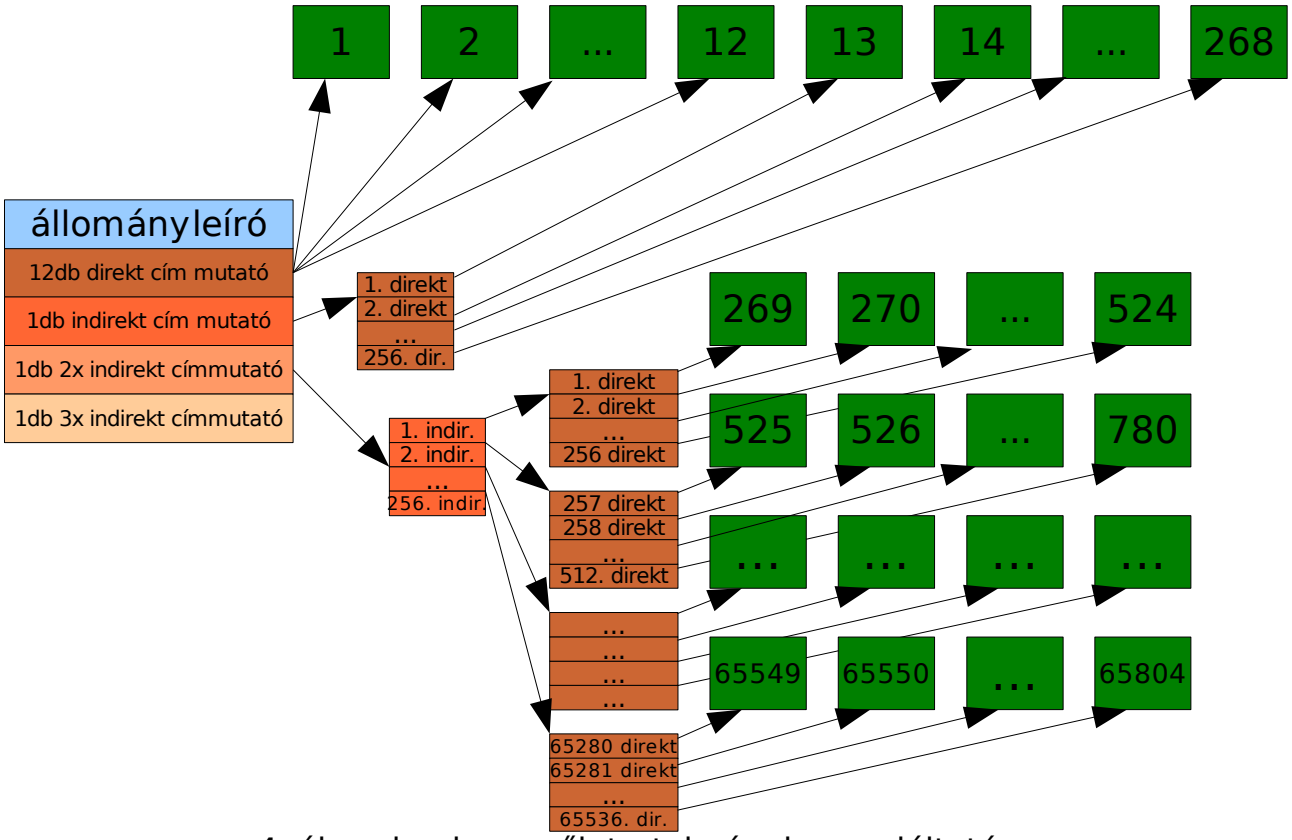
```
notebook:/mnt# lde -i 14 /dev/hde1
Device "/dev/hde1" is mounted, be careful
User requested autodetect filesystem. Checking device . . .
Found ext2fs on device.
------------------------------------------------------------------------
INODE: 14     (0x0000000E)
-rw-r--r--       root     root        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#
```

Az inode tartalmaz 12 db pointert, amelyek mindegyike egy-egy 1K-s diszk blokkra mutat. Ezek a direkt blokkok. A következő mutató egy olyan diszk blokkra mutat, amely további direkt mutatókat tartalmaz. Mivel egy blokkba 256 mutató fér el, így a fenti egyszeres indirekt címzéssel 256K+12K adatot lehet megcímezni. A következő mutató kétszeres indirekt mutató, vagyis egy olyan diszk blokkra mutat, amely 256 mutatót tartalmaz, amelyek mindegyike egy-egy további indirekt blokkra mutat. Vagyis most hivatkozáskor az első két lépésben indirekt blokkokat kapunk, és csak a harmadik lépésben jutunk el az adatokhoz. Ezzel a módszerrel 64M+256K+12K adatot lehet megcímezni. A következő,

egyben utolsó mutató egy háromszoros indirekt blokkmutató, vagyis még a harmadik hivatkozás is 256 pointert tartalmazó indirekt blokkra mutat, és onnan egy újabb referenciával juthatunk el az adatokhoz. Ezzel a módszerrel 16G+64M+256K+12K adatot lehet címezni. Tehát a fent vázolt struktúrával összesen:

- 12 db direkt blokk: 12K
- 1 db egyszeres indirekt blokk: 256K+12K
- 1 db kétszeresen indirekt blokk: 64M+256K+12K
- 1 db háromszorosan indirekt blokk: 16G+64M+256K+12K

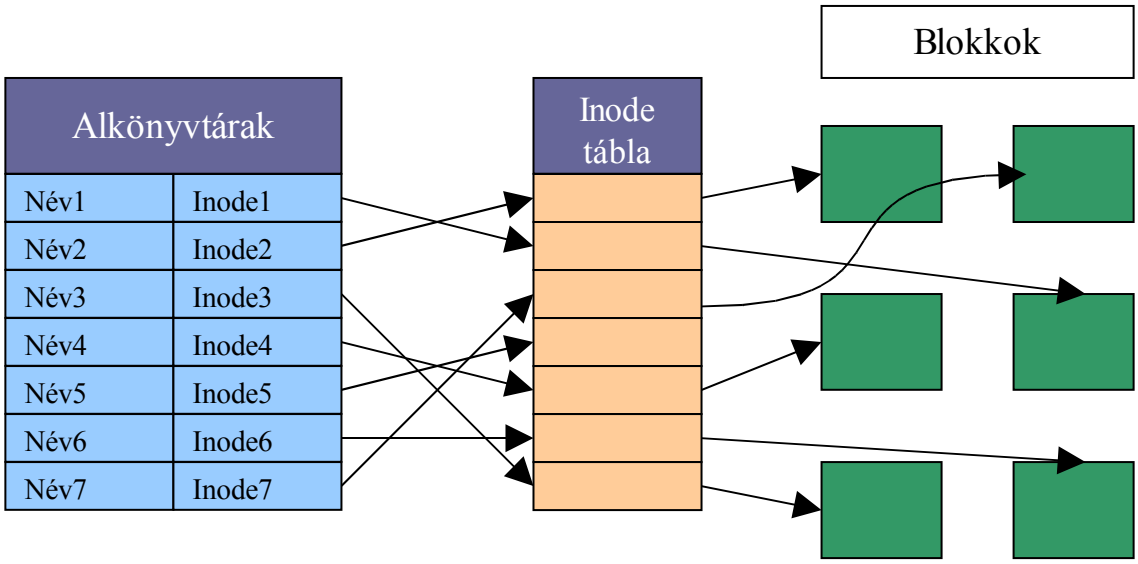
adatot lehet tárolni. A valóságban persze ennél kevesebbet, mert az inode állományhossz mezője is 4 byte, így az korlátozza a maximális állományméretet.



4. ábra: inode mezők tartalmának szemléltetése

5.8 Alkönyvtárak

Az alkönyvtárak tulajdonképpen speciális fájlok, ahol a fájl tartalma egy lista, melynek minden eleme egy fájlnevet, 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). Természetesen egy alkönyvtár bejegyzése mutathat másik alkönyvtár bejegyzésre is. Az alkönyvtár bejegyzéseket a 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 csökken. Amikor az utolsó hivatkozást is eltávolítottuk (pl.: 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 link–ek 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), ami tulajdonképpen egy fájl, amely egy fájl nevet és elérési útvonalát tartalmaz (létrehozása: ln –s <fájlnév> <link neve>). Ha egy szimbolikus linket szeretnénk elérni, 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) 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). 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 /dev/sda). 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.

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
/boot = BOOT partíciót szokták ide csatlakoztatni
/dev = DEVICES, azaz eszközfájlok
/etc = a rendszer konfigurációs fájljai
/home = felhasználók könyvtárai
/lib = library azaz beépülő programok
/proc = processz-, és rendszer-információk
/root = rendszergazda könyvtára
/tmp = ideiglenes könyvtár
/usr = alkalmazási és egyéb programok
/sbin = futtatható system bináris állomány
/var = egyéb, programok által használt könyvtárak

(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
- 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

Állomány típusok a következőek: reguláris, könyvtár, FIFO, socket, karakteres vagy blokkos berendezés.

A fájlok neve után nem kötelező, de célszerű megadni a „kiterjesztését”, hogy tudjuk, milyen fajta fájlról van szó (Pl.: alma.jpg – tudjuk, hogy kép fájl). Ez azonban a felhasználóknak és bizonyos felhasználói programoknak - gcc pl. ragaszkodik a .c kiterjesztéshez - 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 négy byte a tulajdonosokra, csoportokra és másokra vonatkozó jogokat határozza meg. Ennek az alapértelmezését az umask paranccsal állíthatjuk be, így az újonnan létrehozott fájlok, már ezekkel a jogokkal jönnek létre. Az első byte a setuid, setgid, sticky biteket állítja be. A második, harmadik, negyedik byte a tulajdonos, csoport és a „többi" felhasználó jogait határozza meg. A 2-4. byte bitjeit a következőképpen értelmezzük: első bit az olvasási, a második az írás, a harmadik a futtatás illetve könyvtárak esetében a browseolási (tallózás) jog.

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. Változtassuk meg, hogy mindenki számára böngészhető legyen, de csak olvasni lehessen! 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
```

Meg kell jegyezzem, hogy egyre inkább terjed az ún. POSIX ACL, ami arra szolgál, hogy sokkal „finomabb” beállításokat készítsünk a fájlrendszerünk jogaihoz.

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 (mindenki a szívéhez nőtt szerkesztő-programot használhatja: pl.: pico, nano, joe, mcedit, vagy vi) a /etc/passwd fájlt. 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 szkript, ezt kitöltve ugyanazt érhetjük el, mint a fent említett módszerrel ami, viszont minden UNIX rendszerben működik ugyanúgy. (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,o csak fake user,,:/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, nehogy visszaéljenek a jószívűségünkkel. Ez lehet akár tárolóhely, memória, vagy processzoridő.

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

6.2 Quota

Mivel egy adott rendszerben a háttértárak kapacitása véges, korlátozni, kell a felhasználók helyfoglalását. E célra a UNIX-okba 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, REISERFS, XFS [2.6.x] esetén is, JFS esetében azonban még nem!).

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 kvótázandó fájlrendszer gyökerébe (jelen esetben a /home) létrehozzuk a következő két fájlt: quota.user, quota.group (quota V4 esetén aquota.{group,user} lesz)

```
root@pc0:/home:# touch quota.user quota.group
```

Adjunk rá jogot, hogy csak root olvashassa és módosíthassa:

```
root@pc0:/home# chmod 600 quota.user
root@pc0:/home# chmod 600 quota.group
```

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 1   1
```

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

```
root@pc0:/# mount -o remount,rw /dev/hda4
root@pc0:/home# quotacheck -avug
Scanning /dev/hda4 [/home] done
Checked 4 directories and 72 files
Using quotafile /home/quota.user
Using quotafile /home/quota.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  200000  250000   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, akkor 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.

6.3 Ulimit

Előfordulhat, hogy egy futó program erőforrás-felhasználását szeretnénk korlátozni, pl.: nem szeretnénk, hogy egy program túlzottan sok processzoridőt vegyen el. Ezeket a paramétereket az általunk futtatott shell-re és az ebből futtatott programokra vonatkozóan az ulimit paranccsal állíthatjuk be (ez igaz mondjuk a bash esetén, de vannak „butább” shellek - ash, dash -, amik nem támogatják ezt a fajta korlátozást, ez esetben sajnos semmit nem ér).

Az ulimit 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

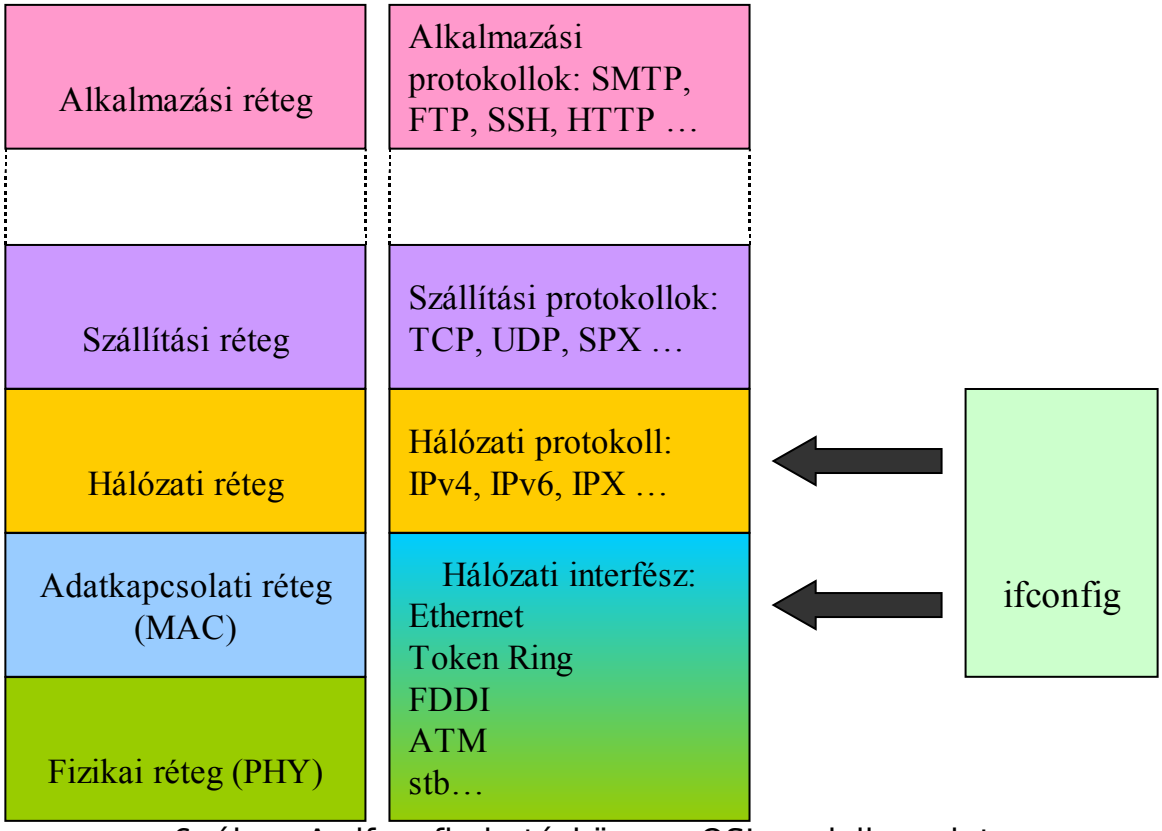
Ezeket az értékeket a /etc/security/limits.conf állományban lehet beállítani. A szintaktika együtt a fájl kommentezett soraiban olvasható!

Bővebb információ: man bash, /ulimit alatt.

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

Az 5. ábra szerint látható, hogy az ifconfig-gal a hálózati interfészünk adatkapcsolati és hálózati rétegbeli tulajdonságait paraméterezzük. Magát 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.

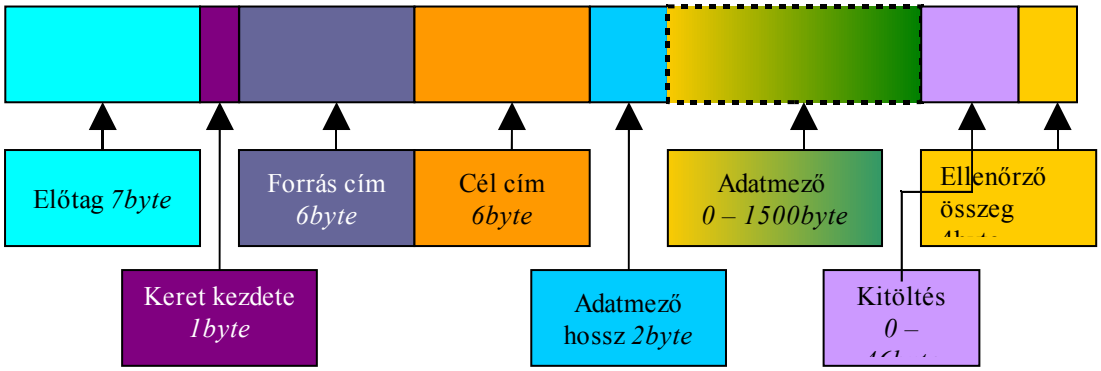
interfész típusa	linuxban a neve
ethernet	eth{0,1,2,...,n}
ethernet másodlagos	eth{0,1,2,..,n}:{0,1,2,..,n}
ethernet vlannal	eth{0,1,2,...,n}.{0,1,2,...,n}
bridge eszköz	br{0,1,2,...,n}
ethernet szintű virtuális eszköz	tap{0,1,2,...,n}
virtuális p-t-p eszköz	tun{0,1,2,...,n}
wireless	wifi{0,1,2,...,n}, 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

eth0   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.



7. ábra: Ethernet (IEEE802.3) keretszerkezete

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

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 jelzi.

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 IP cím> netmask <netmask> broadcast <broadcast> up

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

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 interfésznek 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 során) 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>
```

IPv6 felkonfigurálás: ifconfig <interface> inet6 <128bit-es IP cím / tartomány>. A hálózatcímünket és az alapértelmezett átjárót a route6 paranccsal adhatjuk meg.

IPX felkonfigurálás: ifconfig <interface> ipx <hálózatcím> <gépcím == MAC cím>. Az IPX átjárót az ipx_route paranccsal adhatjuk meg.

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 eth0 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 eth0
193.224.130.160 0.0.0.0         255.255.255.224 U     0      0        0 eth0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth3
193.224.128.0   0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.150.0   0.0.0.0         255.255.255.0   U     0      0        0 eth2
172.16.0.0      193.224.130.182 255.255.0.0     UG    0      0        0 eth0
0.0.0.0         193.224.128.9   0.0.0.0         UG    0      0        0 eth1
ns:~#
```

7.3 Az arp, arping, és a rarp

Egy kis elmélet: 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 szegmensünkön lévő interfész IP címét arping-elve a router felénk eső interfészének MAC címét fogjuk megkapni (lásd: Számítógép-hálózatok).

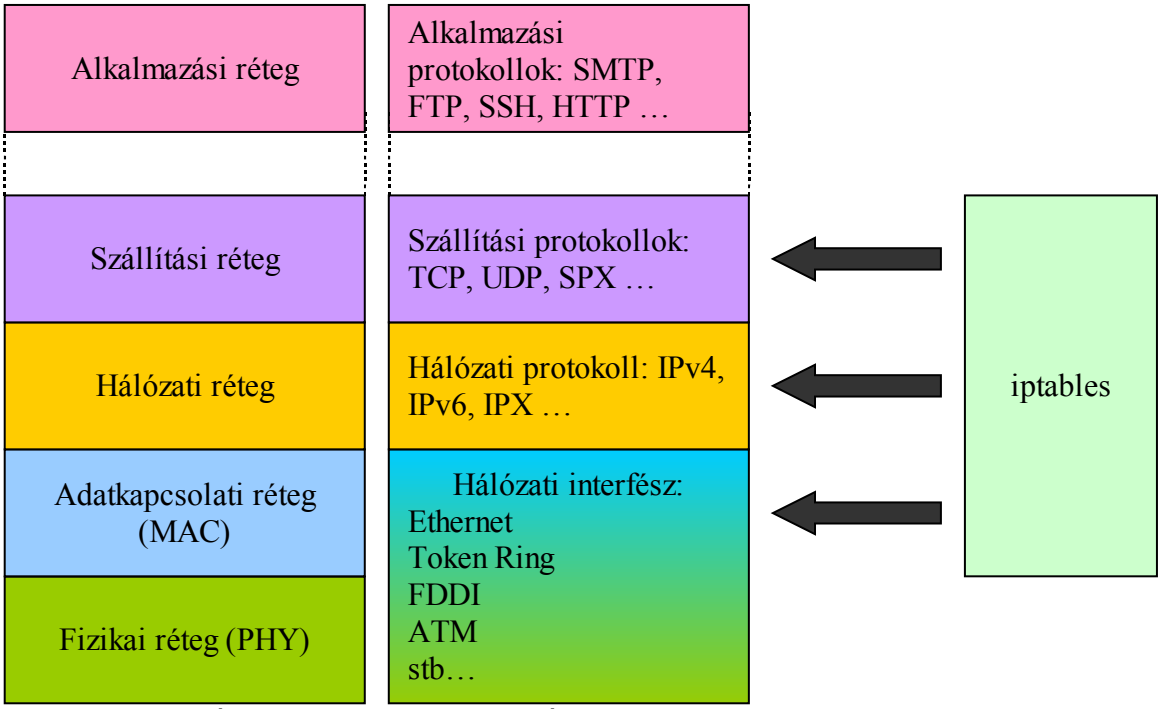
8 Netfilter

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 könnyen átlátható, állítható, módosítható, és elmenthető a konfiguráció.

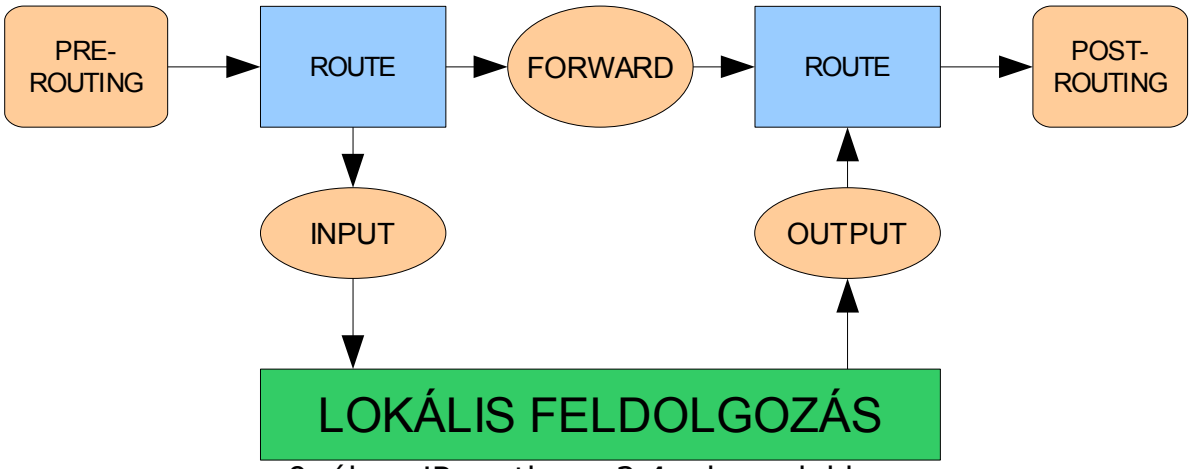
Az iptables parancson kívül az iptables-save az éppen futó konfigurációt jeleníti meg a standard outputra. Így könnyedén elmenthetjük konfigurációnkat pl.: iptables-save > my_ip_config. Az iptables-restore paranccsal viszont könnyen visszaállíthatjuk elmentett beállításainkat pl.: iptables-restore < my_ip_config.

Vizsgáljuk meg, hogy az iptables OSI modell szerint milyen 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 a gép tudjon forwardolni, engedélyezni kell kernel szinten az IP csomag-továbbítást, más néven a kernel routing-ot: echo ”1” > /proc/sys/net/ipv4/ip_forward. Debian sarge-ban a forwardot a /etc/network/options fájlban lehet engedélyezni, etch esetén a /etc/sysctl.conf állományban.



9. ábra: IP routing a 2.4.x kernelekben

8.1 Csomagszűrés működése és megvalósítása

A csomagszűrő (packet filter) figyeli a csomagok fejlécét, miközben azok keresztül haladnak 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 QUEUE listában tárolja, majd innen a listából a megfelelő programok (pl.: forgalomkorlátozó) kiveszik, és döntési mechanizmusuk szerint továbbítják egy részét, a többit eldobják.

Az iptables egy sor különböző funkciókkal rendelkezik (10.2 ábra). Három beépített lánccal indul, az INPUT, OUTPUT, és a FORWARD láncokkal, melyeket nem lehet törölni. A láncok alapértelmezett módja az ACCEPT, ami azt jelenti, hogy minden csomag áthaladhat bármelyik táblán.

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. A többi parancs (–I a beszúrásra és –R a cserére) egyszerű kiterjesztése ezeknek.

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 „célpontja, 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 érvényes-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 a 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 – 193.225.150.255 –ig terjedő IP címeket határoz meg. 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 maradék 0. Tehát a /24 egyenlő 255.255.255.0 –val, csak rövidítve fejezi ki.

Néhány kapcsoló, közöttük a ”–s” és a ”–d” flagek 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 lehet egy szám is (ha ismerjük a protokoll-számot) vagy névvel, mint például TCP, UDP, ICMP. Kis- és nagybetű nem számít. A protokoll elé is tehetünk „!”-t, ami az illeszkedést megfordítja. Tehát a –p ! TCP minden olyan protokollra vonatkozik, ami nem TCP.

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

Néha egy csomag túl nagy ahhoz, hogy egyszerre továbbítsuk. Ez esetben a csomagot töredékekre bontjuk, és több különböző csomagban küldjük el. A végponton ezek összeállnak, és újból rendelkezésünkre áll az eredeti csomag. Ezzel a töredékekkel az a baj, hogy csak az első csomag tartalmazza a komplett fejlécmezőket (IP+ TCP, UDP, és ICMP) a többi csomag pedig csak egy kivonatát a fejlécnek (IP hozzáadott protokoll mezők nélkül). Ezért ezeknek 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 – mivel az első csomagot ugyanúgy kezeli a csomagszűrő, mint a többit – a második és további töredékeket a csomagszűrő nem tudja vizsgálni. Így a szabály (-p TCP --sport www, forrás port meghatározása www-ként ”--sport www” azaz 80-as port) sosem fog illeszkedni a töredékre (legalábbis az elsőtől különbözőre). 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 valamennyi 192.168.1.1 IP címre érkező töredéket eldob.

```
root@pc0:/# iptables –A OUTPUT –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/2.4.21/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: (a töredékek kezelésének kivételével)

--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, PHS), de csak a SYN és az ACK flag lehet beállítva. 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,RST,ACK 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 „:”-al elválaszva, vagy ha a porttal egyenlő vagy annál nagyobb portokat akarjuk megváltoztatni a port neve „:”-al utána.

--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 indul.

8.7.2 UDP kiterjesztések.

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

ICMP kiterjesztések. Ezek a kiterjesztések is automatikusan betöltődnek –p icmp esetén.

--icmp-type 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 a 192.168.0.2 IP címről, és a 00:36:11:22:33:44 MAC című gépről fogadjon el csomagot a router!.

```
# 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.

A datagramm cím mezői (forráscím, célcím) eredetileg nem változtak, a protokollok ezt is várják el. Miért van szükség rá?

Mert elegendő egy IP címet megrendelni, mégis több gépen tudunk internetezni. Ez úgy lehetséges, hogy a külső hálózatra rátesszük a routerünket, és a belső hálózaton a többi gép belső IP címeket kap (ilyenek a 10.x.x.x, vagy a 192.168.x.x IP tartományok). Ezek a belső IP tartományok nem route-olhatóak.

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.

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 10.0.0.2
- Lehet több címet is megadni: --to-source 10.0.0.2 10.0.0.8
- Lehet portok alapján is: iptables –t nat –A POSTROUTONG –o eth0 –j SNAT --to-source 10.0.0.2:1-1023
- Pl.: Modemes kapcsolat megosztása NAT és MASQUERADE funkciókkal: 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 IP címek esetén NAT-ol a routerünk.

DNAT, azaz a Destination-NAT

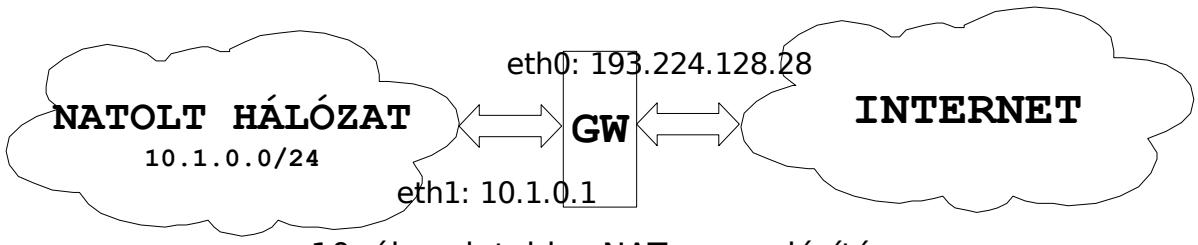
- Továbbítsa a router a 10.0.0.11 IP címre a datagramot: iptables –t nat –A PREROUTING –j DNAT --to-destination 10.0.0.11
- Küldjük a WEB kéréseket a 10.0.0.99 IP cím 8080 TCP portjára: iptables –t nat –A PREROUTING –p tcp --dport 80 –j DNAT --to-destination 10.0.0.99:8080

8.10 Protokoll segédek

A protokollok azt várják el, hogy a forráscím, és a célcím nem változik, ezért szükség van az ún. protokoll segédekre (protocol helpers). Ezek kernel modulként lefordíthatók, és megtalálhatók a /lib/modules/`uname -r`/kernel/net/ipv4/netfilter könyvtárban, és modprobe paranccsal betölthetőek. Ilyen például az FTP, ahol két portot használ a protokoll: 21-es portot a kommunikációra, a 20-as portot az adatátvitelre.

8.11 Tűzfal megvalósítások iptables segítségével

- NAT létrehozása a belső hálózatunk felé: iptables –t nat –A POSTROUTING –s 192.168.0.0/24 –j MASQUERADE
- Csak azokat a portokat engedélyezzük, amire ténylegesen szükségünk van (pl: HTTP): iptables –A INPUT –p tcp --dport 80 –j ACCEPT
- Alapesetben minden bejövő kapcsolatot szűrjünk ki: iptables –A INPUT –j DROP



10. ábra: iptables NAT megvalósítás

A fenti ábránkon, egy maximum 252 gépes hálózatot Ehhez a következő beállításokat kell megtennünk:

iptables támogatás a kernelbe

a forward engedélyezése a kernelben

masquerade-ing beállítása.

A forwardot, ahogy az előzőekben már említettük vagy a /etc/network/options, /etc/sysctl.conf vagy a következő paranccsal: echo „1” > /proc/sys/net/ipv4/ip_forward tehetjük meg. A maszkoláshoz 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 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
```

Nem minden processz szereti, ha nem „látja” a lokális portot amin dolgozik, 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 teljesen mindegy milyen sorrendben adjuk ki a fenti parancsokat.

8.12 Logolás 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 logoljunk, például a vállalati webszerver 80 portjára érkező összes kapcsolat kezdeményezést (a 80-as portra érkező SYN kéréseket) logolni. Két lehetőségünk van a logolásra. Az egyik a syslog melybe a rendszer minden más eseménye is belekerül, a másik az ULOG.

A logoláshoz egy szabályt kell definiálnunk valamelyik láncon, majd a -j kapcsoló után a LOG, vagy ULOG targetet kell megadnunk. Mind a két logolás 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 logoljuk a LOG targetbe, „VALLALATI_WEBSZ: ” prefixszel:

```
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 logolni, és a FORWARD láncon átmenő tcp syn kéréseket fogja logolni, „IPTABLES_ULOG1:” prefixszel:

```
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 a DROP, ACCEPT, REJECT targetekkel, a LOG, ULOG az illeszkedés esetén nem dobja el a csomagokat, hanem továbbengedi a szabályláncon.

9 Rendszernaplózás, logolás

A UNIX rendszerek alatt a syslog a rendszergazda legjobb barátja. A rendszerfolyamatok 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, 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 csomagokat stunnel segítségével lehet titkosítani, az UDP csomagokat alkalmazásszinten kellene titkosítani, de erre a rendszernaplózók nem képesek.

9.1 Facilityk

A linux a logokat különböző szintek szerint csoportosítja két fő csoportba. Az első csoport a facility-k csoportja. Ez tovább oszlik a levelek (severityk) csoportjára.

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

- auth - 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 deklarált (a local7-et a cisco es windows szerverek használjak általában)
- syslog - a syslog saját üzenetei.

9.2 Logolási szintek (loglevelek, severity, severities, priority)

Szemben a facility-kel, a logolási szintek "szabványosítottak", 8 szint van, a szintek prioritásuk szerint:

- 0 - emerg ; sürgős
- 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, debug szint képes „elárasztani” a logokat, hiszen minden üzenetet elment, hogy minél egyszerűbb legyen a hibák detektálása.

9.3 A sysklogd

A sysklogd 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.

A szintaktikája:

mit hova

A "hova" lehet FIFO, fájl vagy hosztnév. Ha a fájl "-"-al kezdődik, akkor nem rögtön írja a lemezre a változásokat, hanem gyorsítótárazza, ez crash eséten adatvesztést okozhat. Tipikusan nem gyorsítótárazzák a warning-tól kisebb prioritású szinteket. A "mit" kissé bonyolultabb. Szerepelhet benne a "*" wildchar, ha "," -vel választom el őket egész mást jelent, mint ha ";" -vel választanám. Ezeket példákon keresztül lehet leginkább megérteni:

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

minden facilityből a hibákat, és csakis a hibákat írja ki. (az *.emerg es a *.=emerg ugyanaz, hiszen az emerg-nél nagyobb severity nincs)

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

minden facilityből, az alert és nagyobb prioritású (*.emerg) üzeneteket írja ki.

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

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

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

az auth és kern facilitykből semmit, a többiből minden alert es alerttől nagyobb prioritású üzenetet logol.

Néhány 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 logol, gyorsítótárazással.

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

Az auth,authpriv,cron,daemon,mail,news kivételével, minden info,notice,warn szintű üzenetet logol, 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.

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 logolásokat képes megvalósítani vele. Nemcsak a facility és level szerint képes logolni, hanem ezeken belül regexp-ekkel is képes szűrni a log tartalmat. Az állomány 3 fő részre osztható (de nem szigorúan), a sorrendek felcserélődhetnek, de az átláthatóság érdekében érdemes meghagyni a felépítést. Az első rész 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 fájlokat, szűrőket (filter) állíthatjuk be. Az utolsó rész, ami a szűrőket, fájlneveket, forrásokat rendeli össze.

A logfájl-nevek szintaktikája:

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

Távoli logolás

```
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 logoló 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. Az IPTABLES jelöl minden olyan forgalmat melyet a nem PUBLIC tartományból, vagy tartomány felé forgalmaztak.

Az alkalmazott szabályláncok a távoli gépen:

```
-A FORWARD -d 193.224.130.160/255.255.255.224 -p tcp -m tcp --tcp-flags SYN,RST,ACK
SYN -j LOG --log-prefix "IPTABLES_PUBLIC:  "

-A FORWARD -s 193.224.130.160/255.255.255.224 -p tcp -m tcp --tcp-flags SYN,RST,ACK
SYN -j LOG --log-prefix "IPTABLES_PUBLIC:  "

-A FORWARD -s 192.168.100.0/255.255.255.0 -p tcp -m tcp --tcp-flags SYN,RST,ACK 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 -m tcp --tcp-flags SYN,RST,ACK 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 logolással elmentsük. Erre Linux alatt a logger parancs az, ami segítséget nyújt. A logger a STDIN-re érkező adatot, a log streamba helyezi az általunk megadott facility.severity 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 accesst a local4.info -ba teszi, amit mi egy külön logfájlba helyezhetünk. Persze meg kell jegyezni, hogy az Apache, amikor 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 logok mérete rohamosan növekedhet. Ezért, hogy ezt kiküszöböljük és, hogy egyszerűbb legyen keresni a logokban (pl.: ha tudjuk mikor történt az esemény nem kell 2 évet visszakeresni), bizonyos időközönként rotáljuk és tömörítjük. Ezt a logrotate könnyíti meg számunkra a Linux rendszer alatt. A /etc/cron.daily alatt található egy szkript, ami a /etc/crontab-ban beállított időközönként lefut. A /etc/logrotate.d könyvtár alatt lévő konfigurációs fájlok segítségével rotálja, tömöríti a logokat. (Egyéb hasznos dolgokra is lehet használni, ilyen például az időközönkénti mentés UML 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 futtatását. 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 szolgáltatásokhoz 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 konfigurációs állomány első oszlopában álló szolgáltatás-névhez a /etc/services fájlban vannak hozzárendelve az adott protokoll port számai. Vegyük észre, hogy ha ott például a POP3 protokollt ezután úgy neveznénk, hogy „kutya”, akkor az inetd.conf-ba is kutyát kellene írnunk! 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
```

Az inetd.conf hatodik oszlopában található tcpd daemon arra szolgál, hogy megszabhassuk, milyen IP címekről jelentkezhetnek be kliensek, hogy igénybe vehessék a szolgáltatásainkat (tcpwrap).

- /etc/hosts.allow – Ebben a fájlban azt mondhatjuk meg, kik azok, akik igénybe vehetik a szerver szolgáltatásait.
- /etc/hosts.deny – Itt mondhatjuk meg, hogy kik nem vehetik igénybe a szolgáltatásokat.

A különféle szolgáltatások inetd-ből történő indítása rendszerünket megbízhatatlanná teszi, amihez ezen felül, még biztonsági rések is társulnak (DoS támadás). Emiatt mostanában a rendszergazdák szívesebben alkalmazzák azt a módszert, hogy minden szolgáltatás számára

külön daemon-t futtatnak. Az inetd, xinetd gyakorlatilag obsolated (nem használatos) kategória. Napjainkban csak nagyon kevés szolgáltatás veszi igénybe. Debian GNU/Linux alatt bejegyzéseket a /etc/inetd.conf fájlba az update-inetd paranccsal tehetünk.

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 pl.: a rendszerindító szkriptek közül a /etc/init.d/bind fájl, ami például a DNS szerver indításáért felelős.

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
```

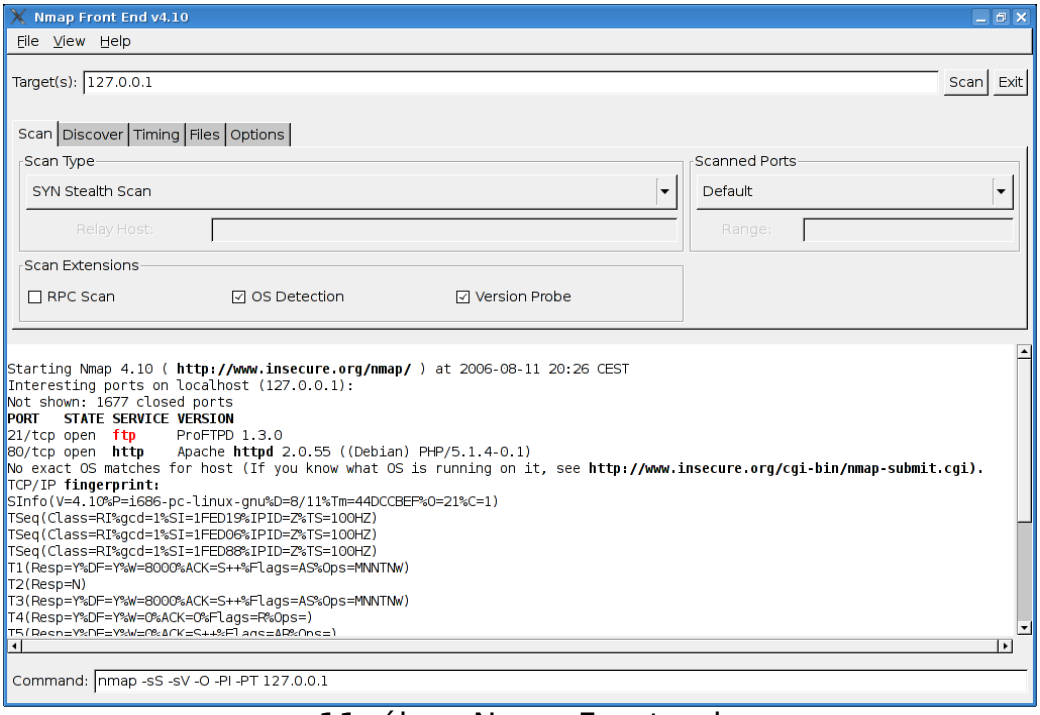
10.3 Szolgáltatások felderítése, rendszerbiztonság

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 felinstallálhatunk (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, például mit sikerült okoznunk az inetd átkonfigurálásával:



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 ez a protokoll? Mert kódolatlanul küldi a jelszavakat a hálózaton!

Miután tisztában vagyunk a nyitott portjainkkal, a fuser parancs segítségével a következő módon meggyőződhetünk, hogy milyen processz futtatja:

```
# fuser -vn tcp 80
                     USER        PID ACCESS COMMAND
```

```
80/tcp              root      22539 f....  apache2
                    root      22540 f....  apache2
```

Szintén hasznos parancs a netstat, ezzel az éppen nyitott 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 storage.tilb.sze.:41528 IGLD-80-230-209-2:49156 ESTABLISHED
tcp        0      0 storage.tilb.sze.:41436 dsl-lhtgw2-fea2dc:48018 ESTABLISHED
tcp        0      0 storage.tilb.sze.:41268 82-47-216-212.stb:65500 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 tulajdonságot. 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 nem csak IP címek kiosztására szolgál, 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.

A dhcpd program konfigurációs fájlja a /etc/dhcp3/dhcpd.conf alatt található. 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 általános-, 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.

Debian GNU/Linux alatt a /etc/default/dhcpd vagy /etc/default/dhcp3-server fájlban lehet beállítani mely interfészeken „hallgasson” a dhcp szerverünk.

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.

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 192.168.100.200 192.168.100.250;
            next-server             193.224.130.174;
            filename                "netboot/pxelinux.0";

            host teacherw {
                  hardware ethernet       00:03:47:c2:63:18;
                  fixed-address           192.168.100.15;
            }

            host teacherb {
                  hardware ethernet       00:11:11:C6:CA:9A;
                  fixed-address           192.168.100.205;
            }

            host pc4_alaplapi {
                  hardware ethernet       00:03:47:A0:56:93;
                  fixed-address           192.168.100.24;
            }
     }

     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, névfeloldásnak hívjuk (angolul: name resolving, mapping). Azt a műveletet, amikor egy IP címet host névre fordítunk visszafelé feloldásnak nevezzük (angolul: reverse mapping).

- Top Level Domain name (TLD): Ezek a hálózati név hierarchiában a legfelső helyen álló, gyökér domain nevek. (Pl.: .hu, .com, .net, .org, .tw, .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, ip.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 tilb.sze.hu már nem a sze.hu zónába tartozik, mert ő egy aldomain, ami saját zónát alkot. Az összes host, ami 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 alkotva host-okat jegyezhetünk be rá! (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. A névszolgáltatások lekérdezésének sorrendjét a /etc/nsswitch.conf fájl (régebben az /etc/host.conf) 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 hostnév-IP cím megfeleltetéseket tartalmazó fájl a /etc/hosts, melynek egy sora itt látható:

```
193.224.130.189   natsemi.tilb.sze.hu natsemi
```

Az első oszlopba a gép IP címét, a többibe a teljes nevét esetleg aliasokat írhatunk. Az alias megkönnyíti a gépre való hivatkozást. Pl.: ssh-nál nem kell beírni, hogy ssh natsemi.tilb.sze.hu, hanem egyszerűen csak ssh natsemi.

Ha a helyi fájlokban nem található meg a kért hostnévhez tartozó IP cím, egy DNS szerverhez kell fordulni. A /etc/resolv.conf nevű fájl határozza meg a feloldás sorrendjét:

```
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

Vegyünk egy egyszerű esetet. Otthoni internetes kapcsolatunkkal felkeressük az ns.tilb.sze.hu gépet. Az otthoni bejegyzett DNS szerverünk (Pl.: a 62.112.192.4). A névfeloldó szerver megkapja (az UDP/53-as portján) a kérést, hogy oldja fel a ns.tilb.sze.hu nevet és adjon vissza nekünk egy IP címet. Mivel ez a szerver még nem ismeri ezt a domaint, először lekérdezi a TLD szervertől, hogy ki a felelős a .hu feloldásáért. Majd erre a TLD válaszol, hogy a .hu domainért az NIIF szervere felel (host -t ns hu). Ezután az általunk használt névfeloldó szerver (62.112.192.4) megkérdezi az NIIF szerverétől, hogy ki felel a sze.hu domain feloldásáért. Az NIIF szerver választ ad, hogy a sze.hu-t az rs1.sze.hu (193.224.128.1) kezeli. A már hosszú utat bejárt név-feloldó szerverünk megkérdezi az rs1-től, hogy ki a felelős a tilb.sze.hu domain feloldásáért. Ő kidobja a választ, hogy a 193.224.128.28 IP című gép. Így megvan, hogy a keresett IP cím ez. A név-feloldó szerver (62.112.192.4) vissza is adja nekünk a választ, így mi el tudjuk érni a labor szervert. Mi történik, ha másodszor is lekérdezzük ezt a nevet? Semmi, mert a 62.112.192.4 eltárolta (el cache-elte) magának ezt a domain – IP cím párost, így legközelebb már nem kell ezt a hosszú folyamatot végig csinálni.

12.5 Caching-only name server

A caching only DNS szerverek elsődleges feladata, hogy egy átjárót biztosítsanak a kliensek

és a name szerverek között. Egy kliens nem képes arra, hogy pl. egy root name szerverhez fordulva feloldja a TLD-ket, ezért egy névszerverhez kell fordulnia, mely a kéréseket feloldja a kliens számára. Nem illik olyan name szervert használunk, 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.

A caching-only name server megvalósítható a named nevű program segítségével, amely megtalálható a Debian GNU/Linux csomagkészletében. (Installálása: 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 nagyteljesítményű gépeken futó DNS szerverek, melyek az Internet fő gerincvonalain ülve a Top Level Domain-ek feloldását végzik (A névszerverek helyei a világban). Természetesen ezeket az IP címeket nem kell tudnunk fejből, le lehet tölteni egy megfelelő fájl formájában az ftp.rs.internic.net címről vagy a host -t ns . paranccsal egy fájlba irányítani, amit a konfigfájlban majd egy hint típusú zónának kell megadni.

12.7 A named.conf fájl

A named alapkonfigurációs fájlja rendszerenként változó, de általában a /etc/bind/named.conf.

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

Az „options” részben a „directory” kulcsszó megadja, hogy melyik alkönyvtár legyen a

named alapértelmezett gyökér könyvtára, innen indul ki minden további elérési út, melyet megadunk. 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 „keys” részben meg is adunk. Ennek a kulcsnak egyeznie kell azzal a kulccsal, amelyet az adminisztrációs gép /etc/rndc.conf fájljában megadtunk. Ezután következik a root zóna megadása (zóna = domain), melyben a fentiekben említett root.hints fájl elérési útját kell megadni. A root zóna után meg kell adnunk a saját zónánkat leíró konfigurációs fájlokat, esetünkben ebből egy van, ez pedig a mydomain/127.0.0 nevű fájl.

12.8 A zónaleíró fájl

Nézzük meg a 12.7 fejezetben példaként hozott konfigurációs állományhoz tartozó ptr típusú zónaleíró fájlunkat, ha csak caching-only DNS-t építünk:

```
$TTL 3D
@                IN           SOA    ns.linux.bogus.     root.linux.bogus. (
                                      1      ; Serial
                                      8H     ; Refresh
                                      2H     ; Retry
                                      4W     ; Expire
                                      1D     ; Minimum TTL
}
                                NS     ns.linux.bogus.
1                               PTR    localhost
```

Az első sor a Time To Live (TTL) értékét adja meg három napban, amely azt jelenti, hogy ez a fájl három nap alatt elévül. A második sor az úgynevezett SOA rekord, mely a @ zónára vonatkozik. A @ a named.conf a fájlt tartalmazó zónát jelenti. A fenti fájl esetében a @ == 127.in-addr.arpa tartománnyal.

A SOA (Start Of Authority) időzítéseket tartalmaz, melyek a Refresh, Retry, Expire, Minimum TTL. Ezek az értékek a másodlagos névszerverek számára fontosak. A refresh azt mutatja meg a másodlagos domain szervereknek, hogy mennyi időnként kell frissíteni, a zónafájlokat. A retry mutatja meg, mennyi idő múlva próbálkozzon, ha elsőre nem éri el a másodlagos névszerver. Az expire, mennyi ideig érvényes a letöltött zónafájl. A a minimum TTL egy letöltött zóna vagy elutasított kérés elévülését jelenti.

Az NS sor adja meg a domain DNS szerverének nevét, esetünkben ez ugyanaz, mint a 2. sorban lévő név. Az utána álló pont fontos, mert így azt lezárja, és nem egészíti ki további DNS utótagokkal! Értsd: ns.linux.bogus ha nincs pont, a következőt fogja jelenteni: ns.linux.bogus.0.0.127.in-addr.arpa persze mi nem ezt akarjuk, tehát kénytelenek vagyunk a „lezáró” pontot alkalmazni.

Az utolsó sor megadja, hogy a 1(.0.0.127.in-addr.arpa) gép neve localhost.(linux.bogus).

Ahhoz, hogy a gépünkön lévő szoftverek a lokális DNS szervertől kérjék a nevek feloldását, meg kell adnunk a resolv.conf fájlban a következőket:

```
search          linux.bogus
nameserver      127.0.0.1
```

12.9 Egy egyszerű tartomány megvalósítása

Nézzük meg, mi is szükséges ahhoz, hogy végre legyen egy igazi DNS szerverünk! Először is, szükségünk van egy domain névre, amit szeretnénk ezzel kiszolgálni. Legyen ez mondjuk a laborunk domain-je a „tilb.sze.hu” domain. Ehhez a sze.hu tartományért felelős névszerverbe egy NS bejegyzésre lesz szükség, mely a tilb.sze.hu tartomány névszervere felé delegálja a *.tilb.sze.hu kéréseket.

Ezután szükségünk van még egy IP tartományra, amit a hálózatcímmel és a netmaszkkal adunk meg: 193.224.130.160/27 (másképp 193.224.130.160/255.255.255.224), melyre szintén bejegyzés szükséges a 193.224.130.0/24 tartományért felelős névszerverbe, hogy delegálja el felénk a reverse kéréseket.

A Debian GNU/Linux alapértelmezetten jól állítja be a localhostra vonatkozó zónafájlokat, így ezekkel nem foglalkozunk.

Tudjuk tehát, hogy a névszerverünkhöz olyan kérések érkezhetnek melyek tilb.sze.hu. vagy 27/160.130.224.193.in-addr.arpa. tartományokról érdeklődhetnek. Ezeket célszerű 2 külön zónafájlba tenni. Ehhez új zóna bejegyzéseket adunk a named.conf fájlhoz:

```
zone "tilb.sze.hu" {
        type master;
        file "/etc/bind/tilb/tilb.sze.hu";
};
zone "160/27.130.224.193.in-addr.arpa" {
        type master;
        file "/etc/bind/tilb/193.224.130";
};
```

Ezzel létrehoztuk az új zónákat.

Először a tilb.sze.hu zónafájlt beszéljük meg részletesen a reverse típusú feloldásokról később bővebben beszélünk majd.

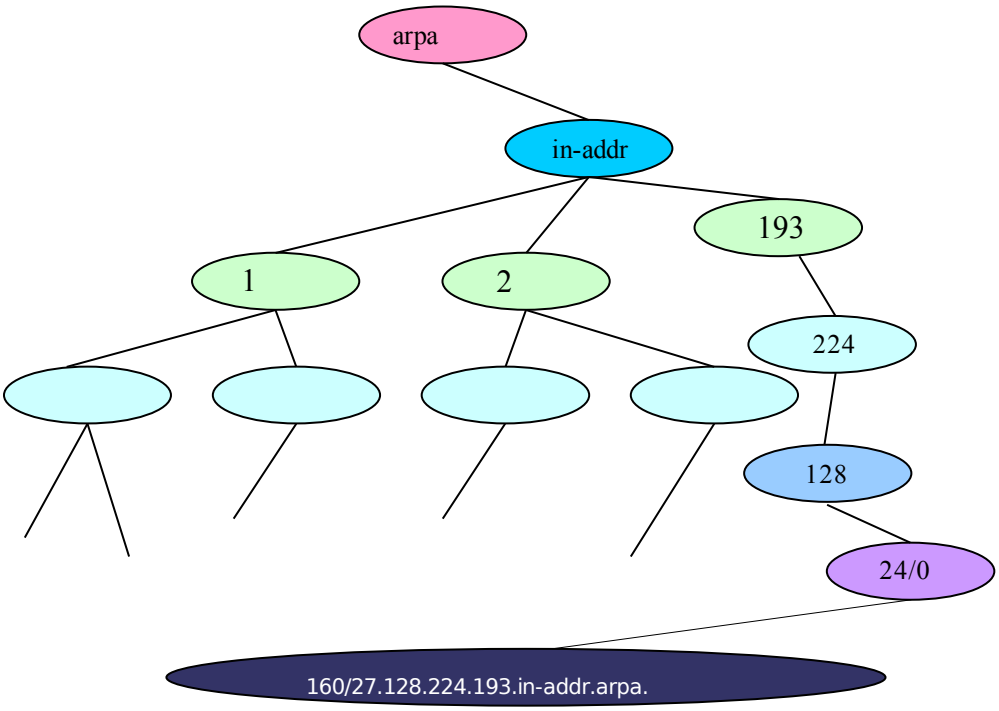
```
$TTL 3600
@       IN      SOA     ns.tilb.sze.hu. root.tilb.sze.hu. (
                        2007082801
                        8H
                        2H
                        4W
                        1D)
                NS      ns.tilb.sze.hu.

                MX      10      users.tilb.sze.hu.
                MX      20      www.tilb.sze.hu.

localhost       IN      A       127.0.0.1
apc             IN      A       193.224.130.164
camserv         IN      A       193.224.130.165
cam1            IN      A       193.224.130.166
ta65            CNAME           www.tilb.sze.hu.
ta13            CNAME           www.tilb.sze.hu.
```

Az MX rekord az alapértelmezett levelező szervert adja meg a hálózatunkon. Az „A” rekordok mindegyike egy IP címhez hozzárendelt domainnevet jelöl meg. Az „A” típusú rekordok segítségével érhetőek el domainnév alapján a zónafájlban megadott számítógépek. A „CNAME” rekordok a www.tilb.sze.hu IP címére hivatkozó bejegyzések, így például a ta65.tilb.sze.hu címre hivatkozva a bekonfigurált Apache szerver a tantárgy honlapját adja be (erre visszatérünk majd mikor az Apache-ot oktatjuk a tárgy keretein belül).

12.10 A reverse DNS beállítása

A reverse DNS feladata, hogy IP címből visszaadja a domain nevet. Egyértelműen egy IP címből csak egy domain nevet adhat vissza, annak ellenére is, hogy egy IP címre több domain név is mutathat!

A reverse DNS feloldását az arpa. TLD alatt található in-addr szerver segíti. Az itt található in-addr.arpa szerver decimális számok bejegyzéseit tartalmazza. Ezek a decimális számok az IP címek 8 bites számait jelentik. Ezek fastruktúra szerűen szétágazódnak (lásd: 12. ábra: reverse DNS feloldás). A feloldás menete fentről lefelé történik. Pl.: a 193.224.128.0/24 tartományhoz tartozó zónafájlt megfordítva kell bejegyezni a named.conf-ba: 0/24.128.224.193.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

12.11 Reverse DNS konfigfájl

A példánkban a named.conf fájlba bejegyeztük a /etc/bind/tilb/193.224.130 zónafájlt. Ez a zónafájl felel a reverse DNS beállításokért. A fájl a következő elemekből áll:

```
$TTL 1H
@       IN      SOA     ns.tilb.sze.hu. gecko.sid.sth.sze.hu. (
                        2007082801
                        28800
                        7200
                        604800
                        86400)
        IN      NS      ns.tilb.sze.hu.

$ORIGIN 160/27.130.224.193.in-addr.arpa.
161     PTR     tuzfal.tilb.sze.hu.
163     PTR     ns2.tilb.sze.hu.
164     PTR     apc.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 sorhoz hozzárendeli 130.224.193.in-addr.arpa. címet, így nekünk csak az utolsó (első) számot kell beírni, és a gép teljes nevét. A teljes domain név után tegyünk mindig pontot!

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.iif.hu.

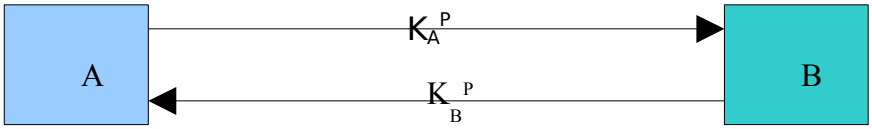
13 Az ssh

Az OpenSSH project 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)

Azonosítás folyamata:

Session key (gépkulcs csere)

A eljuttatja B-nek a publikus kulcsát, és B visszaküldi A-nak a saját publikus kulcsát.



13. ábra: Session key

Majd mindkét fél generál egy-egy véletlen számot (Unix rendszerekben a beépített véletlen szám generátor gondoskodik erről). „A” fél elkódolja ezt a véletlen számot „B” publikus kulcsával, majd átküldi „B”-nek. „B” fél is elkódolja ezt a véletlen számot „A” publikus kulcsával, majd átküldi „A”-nak. Mindkét oldalon előáll a $K_{SESSION}=f(r_a , r_b) = f(r_b , r_a)$. Ezután következik az authentikáció. Az authentikálás két módon futhat le. Az a, esetben erős (nyilvános kulcsú) titkosítással egy már előzetesen egyeztettet és megfelelő helyre beillesztett kulcsok alapján, illetve b, esetben a hagyományos Unix-os jelszó megadása során.

Az ssh-keygen programmal készítünk egy publikus és egy titkos kulcsot. A szerver oldalon elhelyezzük a publikus kulcsunkat a home könyvtárunkban a .ssh könyvtára alá (~/.ssh/{authorized_keys,authorized_keys2} – attól függően, hogy milyen ssh_protokollt használtunk). A titkos és a nyilvános kulcsot (Unix esetén) a kliens gépünk home könyvtárába szintén a .ssh könyvtár alá helyezzük el (~/.ssh/id_{rsa,dsa}, ~/.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.

13.1 Kulcsgenerálás

A következő egyszerű utasítással hozhatunk létre privát/publikus kulcs párokat:

```
ssh-keygen -t dsa/rsa -b 1024
```

Ez a parancs létrehoz a ~/.ssh könyvtár alatt id_{r,d}sa{,.pub} fájlokat, attól függően, hogy milyen típust adtunk meg. 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 SSH

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@gepneve
```

Parancs végrehajtás távoli gépen az ssh segítségével:

```
ssh felhasználónév@gépnév bash -i
```

A fenti parancs például egy interaktív shell-t indít számunkra, ami nem jelenik meg a log fájlokban.

13.3 SCP

Az ssh nemcsak biztonságos távoli bejelentkezést tud nyújtani számunkra, hanem titkosított adatátvitelt két fél között. Unix rendszerekben erre szolgál az scp parancs. Szintaktikája megegyezik a már tanult Berkley r* rcp paranccal.

```
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 konfig fájlt, majd a fontosabb beállítási lehetőségeket kiemelem:

```
$ 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!(verzió: 1, 2)
#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 graf felület alól jelentkezünk
be egy távoli graf 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.

EDDIG KESZ A JAVITAS!!!!!!

14 FTP szerver: 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 installálható fel.

A proftpd konfigurációs fájlja a /etc/proftpd.conf, felhasználó-regisztrációs file: /etc/ftpusers. Ebbe a fájlba azoknak a felhasználóknak a nevét kell bejegyezni, akiknek a hozzáférését le akarjuk tiltani.

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 anonymous-ként elérni.

A többi felhasználó, akit a rendszerünkben nyilvántartunk saját felhasználó nevével és jelszavával tudja elérni az FTP szolgáltatásunkat.

14.1 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, a chmod 022 (csoport, és mindenki által írható) jogokat
maszkolja ki FTP alatt.

MaxInstances            30
# Ezzel azt lehet beállítani, hogy maximum hány példányban fusson, tehát hány
kérelmet 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 első a könyvtárba történő váltáskor, a benne lévő szöveget megjeleníti az
erre alkalmas kliens.

<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 fájlt 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 létere illetve töröljön!

 </Anonymous>
# Itt ért véget az anonymous hozzáférés konfigurációja.
```

A fenti példában nincsen benne az összes számunkra fontos opció. Ezek a következőek:

```
RequireValidShell On|Off
```

A RequireValidShell engedi az olyan felhasználók belépését akik nem rendelkeznek shellel. Ez nagyon hasznos amikor a felhasználó „shell”-je pl: /bin/false.

```
DelayEngine on|off
```

A DelayEngine-t akkor érdemes kikapcsolni, amikor az FTP szerverünk véletlenszerűen bontja a kapcsolatot.

```
DefaultRoot path/a/homehoz [!] group
```

vagy

```
DefaultRoot ~
```

A felhasználók, ha nincs a DefaultRoot opció bekapcsolva, akkor bejelentkezés után a / könyvtárba jutnak. Be kell látni ez elég nagy probléma, pláne ha valahol elrontottuk a jogosultságokat. A DefaultRoot-al lehet szabályozni, hogy a felhasználó bejelentkezése után mi legyen az alapértelmezett „/” (root) könyvtára. Ha a saját home könyvtárát akarjuk beállítani a következőkép használjuk:

A proftpd képes nem csak PAM-ból, hanem SQL szerverről vagy ldapból authentikálni a felhasználókat, így nem kell unix accountokat létrehoznunk.

Minden konfigurálás után újra kell indítani a proftpd-t az /etc/init.d/proftpd restart paranccsal, ha nem inetd-ből fut.

15 HTTP szerver: Apache 2 és konfigurációja

Ebben a fejezetben megtanuljuk, hogyan lehet webszervert építeni és konfigurálni az apache, apache2 HTTP daemonok segítségével.

Az apache jelenleg a legelterjedtebb webszerver a világon, nem sokkal a Microsoft IIS-e előtt. A régi NCSA webszerverből származik, az apache2-t pedig, gyakorlatilag újraírták a fejlesztők. Az apache és az apache2 a Debian disztribúció része, így az apt-get install apache|apache2 parancs megadásával azonnal telepíthetjük rendszerünkre.

A Debian, az apache{,2} konfigurációs fájljait a /etc/apache{,2} könyvtárba, míg a futtatható fájljait a /usr/sbin könyvtárba helyezi el. Az apache{,-ssl,2} vezérlésére, a /etc/init.d/apache{,-ssl,2} scriptet használjuk. Alapértelmezetten a documentum-root a /var/www. Ide másolhatjuk be az általunk készített weblapokat. Telepítés után az apache 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.

15.1 Az Apache v1

Az Apache konfigurációs fájljai az /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 kezdőmoduljaik vannak itt felsorolva.
- modules.conf – a betöltendő modulok vannak itt felsorolva

Az access.conf és az srm.conf alap esetben üresek, mert jobb minden beállítást a httpd.conf fájlban tartani. Ez a két fájl a httpd.conf megfelelő sorában lenne include-olva, azonban ez a két sor ki van kommentezve, tehát ezt a két fájlt nyugodtan figyelmen kívül hagyhatjuk. Web szerver farm esetén a gépenként eltérő beállításokat tehetjük az srm.conf-ba.

15.2 Az Apache v2

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 konfiguráció újraszervezésével a fejlesztők elérték azt, hogy ehhez a fájlhoz csak akkor kell nyúlni, ha a teljesítményt befolyásoló beállításokat akarjuk konfigurálni.
- ports.conf – a portokat kell megadni, amin az apache2-nek figyelnie kell. Ez az egyik legnagyobb újdonság, hogy itt nincs külön apache-ssl es apache hanem egy program oldja meg a 2 szolgáltatást (erre természetesen az apache1 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
- mods-{available,enable} - a lehetséges (available) és a használandó (enable) modulok helye. Az available-ben az összes betölthető modul szerepel a modulnév.load formában és ha van konfigurációs állomány hozzá akkor az a modulnév.conf formában. Az enable könyvtárban linkek vannak az available-ben lévő fájlokra. Az apache indításakor azok az állományok töltődnek be, melyekre hivatkozunk az enable könyvtárból.
- sites-{available,enable} – hasonló mint a mods-{available,enable} azzal a különbséggel, hogy itt virtuális hosztokat tudunk megadni, amivel átláthatóbb a konfiguráció.
- ssl – a tanúsítványok és kulcsok találhatóak itt az ssl feletti kommunikációhoz. A kulcsot a következő egyszerű paranccsal generálhatjuk: apache2-ssl-certificate (ez valamilyen okból kifolyólag nem része az apache v2.2 common-nak. A fájl letölthető a http://dev.tilb.sze.hu/apache2-ssl-certificate URL-ről, a működéséhez szükséges még az ssleay.cnf állomány mely szintén itt http://dev.tilb.sze.hu/ssleay.cnf található meg, ezt a /usr/share/apache2 alá kell másolnunk). Ha futtatjuk a parancsot, feltesz néhány kérdést, melyekre értelemszerűen válaszolunk. Az egyetlen fontos dolog a Common Name. Itt, ha nem az ssl-es host nevét adjuk meg, akkor az apache figyelmeztetésekkel szemeteli tele a logot. A program futása után létrejön egy /etc/apache2/ssl/apache.pem szöveges állomány. Ez tartalmazza a kulcsot és a tanúsítványt.

15.3 Az Apache v1 részletes konfigurációja – a httpd.conf felépítése

A httpd.conf fájlt megnyitva láthatjuk, hogy alapvetően három fő részből áll (a konfigurációs állomány tagolása csak a v1 esetében igaz, a v2 mint említettük több kis konfigurációs fájlból áll):

- A globális opciók, melyek a szerverprogram működését adják meg, pl.: hány child processz fusson, stb…

- A szerverre vonatkozó opciók, melyek a nem-virtuális webszerverek működését adják meg. (Ezt később részletesen is elmagyarázzuk)
- A virtuális webszerverek konfigurációja, melyek alias-ként szerepelnek csak a DNS-ben és az alias ugyanerre a gépre mutat.

Amikor itt különféle „szerverekről” beszélünk, az nem azt jelenti, hogy ennyi gépünk van, hanem, hogy egy gépen belül több HTTP szolgáltatást is indíthatunk.

15.4 Globális opciók

```
ServerType standalone|inetd
```

Itt megadhatjuk, hogy a httpd szerver állandó processzként fusson-e, vagy az inetd szuperszerver indítsa-e el. Ha az előbbit szeretnénk, ide „standalone”-t, ha az utóbbit akkor „inetd”-t kell írni. Ha inetd-vel akarjuk indítani, akkor az inetd.conf-ban be kell állítanunk ennek a portnak a figyelését!

Ezentúl, ha ilyen jellegű konfigurációs opció van, ahol több lehetőség közül kell választani, így fogjuk jelölni: ServerType (standalone | inetd). Természetesen, csak egyiket szabad a fájlba beírni.

```
ServerRoot ”/etc/apache”
```

Megadja, hogy a httpd szerver számára mi legyen a root könyvtár. Ezután minden relatív útvonal ehhez képest értendő. Tehát ha a httpd.conf-ban egy fájlra hivatkozunk pl.: logs/error.log akkor ez valójában a /etc/apache/logs/error.log fájlt fogja jelenteni. Természetesen továbbra is működnek az abszolút elérési utak, pl.: /var/log/apache/error.log

```
PidFile /var/run/httpd.pid
```

Megadja azt a fájlt, ahol a szülő-httpd a saját processz-ID-jét tartja. Ne nyúljunk ehhez a beállításhoz, mert az apachectl indító szkript működését elronthatjuk vele.

```
Timeout 300
```

Egy kliensnek csatlakozás után ennyi ideje van, 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-k).

```
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 httpd forkol még child processzt. A maximum értéket se tegyük 5-nél nagyobbra, ha nincs extrém terhelésű szerverünk.

```
StartServers 5
```

A szülővel együtt ennyi processzt fogunk indítani elsőre.

```
MaxClients 150
```

Ez az összes futó httpd 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 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.

```
#Listen 3000
```

Itt beállíthatjuk, hogy a httpd szerver ne a 80-as porton figyeljen. Csak akkor állítsuk át, ha kimondottan ilyen szándékunk van.

```
LoadModule vhost_alias_module libexec/modvhost_alias.so
AddModule mod_vhost_alias.c
```

Ez a két sor egy Apache bővítőmodul betöltést végez el, számtalan ilyen van a httpd.conf-ban, csak akkor van rá szükségünk, ha egy új modult írunk vagy fordítunk az Apache-hoz (apache2 esetén ahány modul annyi fájl+némelyiknek konfigurációs fájl található a mods-available könyvtár alatt).

15.5 A szerver konfigurációja

Más néven globális opciók. Ebben a részben adhatjuk meg az alapértelmezett webszerver beállításait.

```
Port 80
```

Ezen a porton figyel a szerver.

```
User www-data
Group www-data
```

Itt megadhatjuk, milyen UID és GID alatt fusson a szerver. Ez biztonsági szempontból fontos, ha a webszerver törhető, akkor sem tudnak csak www-data jogokkal garázdálkodni a rendszerünkben az illetéktelenek.

```
ServerAdmin drmomo@tilb.sze.hu
```

Azt adja meg, hogy ki a szerver adminisztrátora.

```
ServerName www.tilb.sze.hu
```

Ez a szerver neve, amire hallgat, emellett ennek nyílván egy érvényes, DNS A rekordban szereplő névnek kell lennie.

```
DocumentRoot ”/var/www”
```

Itt megadhatjuk, melyik alkönyvtárban lesznek a honlap fájljai.

```
<Directory ”/var/www”>
```

Itt ugyanannak az elérési útnak kell szerepelnie, mind 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 Includes FollowSymLinks MultiViews
```

Opciók megadása, pl.: FollowSymLinks: szimbolikus linkek követése; Indexes: ha nincs a könyvtárban index.html akkor a benne lévő fájlokat listázza; stb.

```
    AllowOverride None
    Order allow,deny
    Allow from all
```

Az allowOverride-al tudjuk az alkönyvtárakon belül .htaccess-szel engedélyezni, hogy miket módosíthat a .htaccess fájl.

Az alapvető logika: Először az engedélyt adunk meg, majd a tiltást.

Az „allow from all” megengedi mindenkinek a kapcsolódást ehhez a könyvtárhoz.

```
</Directory>
```

A Directory-t lezáró tag.

```
<IfModule mod_userdir.c>
    UserDir public_html
</IfModule>
```

Ezek a sorok adják meg, hogy a felhasználók saját honlapjaik milyen nevű alkönyvtárban kell lenniük a home könyvtáron belül. Lehetőleg ne változtassunk rajta, ezzel sok kérdést előzhetünk meg a felhasználók irányából.

```
HostNameLookups (Off|On)
```

Ha bekapcsoljuk, a naplófájlokban megjelenik a hostoknak a neve is, amelyek kérést intéztek a szerverhez. Ha kikapcsoljuk, csak IP címet naplóz.

```
ErrorLog /var/log/apache/error_log
```

A hiba-naplófájl (errorlog) elérési útja. Ha a fájl nem létezik, létrejön, de az elérési útnak léteznie kell, a szerver nem hoz létre alkönyvtárat!

```
LogLevel (debug|info|notice|warn|error|crit|alert|emerg)
```

Azt a hibaszintet állítja be, amelynél már létrejön egy bejegyzés az errorlog-ban.

```
LogFormat <formátum-string>
```

Megadja a log fájlok egy sorának formátumát. Bővebb információk az Apache dokumentációban (http://www.apache.org)

```
CustomLog /var/log/apache/access_log common
```

Megadja az eléréseket tartalmazó log fájl útvonalát. Itt a letöltött fájlok adatai tárolódnak el (ki, mikor, mit szedett le)

```
ServerSignature (On|Off|Email)
```

Ha bekapcsoljuk, akkor a szerver által generált (tehát nem az általunk feltöltött dir. list.) oldalak aljához hozzáírja a szerver verziót, illetve az Email opció esetén odatesz egy linket a ServerAdmin-ban megadott e-mail címre.

```
<IfModule mod_alias.c>
    Alias /icons/ "/var/www/icons/"
    <Directory "/var/www/icons">
           Options Indexes MultiViews
           AllowOverride None
           Order allow,deny
           Allow from all
    </Directory>
    ScriptAlias /cgi-bin/ "/usr/lib/cgi-bin/"
    <Directory "/usr/lib/cgi-bin">
           AllowOverride None
           Options None
           Order allow, deny
           Allow from all
    </Directory>
</IfModule>
```

Az Alias opció segítségével becsatolhatunk egy, a rendszerben lévő alkönyvtárat a webszerver gyökeréhez viszonyított elérési útba. A ScriptAlias segítségével megadhatjuk, honnan lehet CGI szkriptet futtatni. A Directory környezetben megadhatjuk, hogy egy adott alkönyvtárra milyen elérési jogok vonatkozzanak. A különféle opciók jelentését lásd az Apache dokumentációban (apt-get install apache-doc apache2-doc).

15.6 Virtuális HTTP szerverek konfigurációja

Ha szeretnénk, hogy HTTP szerverünk több hostnévre is hallgasson, úgynevezett virtualhost-okat kell létrehoznunk. Ahhoz, hogy a www.tilb.sze.hu gépet pl.: ta65.tilb.sze.hu néven is nevezhessük, a tilb.sze.hu zónafájlban a következő sornak kell állnia:

```
ta65       CNAME        www.tilb.sze.hu.
```

Az apache konfigurációs fájljában a következőképp kell megadnunk a virtualhost-okat:

```
NameVirtualHost *
<VirtualHost *>
    ServerAdmin webmaster@ta65.tilb.sze.hu
    DocumentRoot /var/www/ta65/
    ServerName ta65.tilb.sze.hu
    ErrorLog /var/log/apache/ta65.error
```

```
    CustomLog /var/log/apache/ta65.custom
</VirtualHost>
```

A „*” a default beállítás, de megadhatnánk helyette a szerverünk ip címét és domain nevét is. (Pl.: NameVirtualHost 193.224.128.28:80 <VirtualHost 193.224.128.28:80>)

A „NameVirtualHost” kulcsszó bekapcsolja a virtuális webszervereket. Látható, hogy a virtuális szervereket ugyanúgy kell beállítani, mint a főszervert. A log-ok külön vannak választva, itt is érvényes, hogy a log-ot tartalmazó alkönyvtárnak léteznie kell.

15.7 Az apache v2 konfigurálása

Az alábbiakban megmutatjuk, hogyan kell telepíteni, beállítani egy apache2 webszervert, ami kezeli az ssl kapcsolatokat redirect-el, ha egy bizonyos kiterjesztése van az url-nek, valamint a virtuális hostot is támogassa. A célunk eléréséhez egy publikus IP-címmel rendelkező gépet használunk, amire van egy CNAME bejegyzés is a névszerveren:

Feltelepítjük az apache-unkat:

```
apt-get install apache2 apache2-doc
```

Installálás után létrejön a könyvtárszerkezet a /etc/apache2 alatt:

```
dev:~# ls -l /etc/apache2/
összesen 72
-rw-r--r-- 1 root root 12482 2006-08-05 23:46 apache2.conf
drwxr-xr-x 2 root root  4096 2006-08-16 01:40 conf.d
-rw-r--r-- 1 root root   748 2006-08-05 23:28 envvars
-rw-r--r-- 1 root root   268 2006-08-16 01:40 httpd.conf
-rw-r--r-- 1 root root 12441 2006-08-05 23:46 magic
drwxr-xr-x 2 root root  4096 2006-08-16 01:40 mods-available
drwxr-xr-x 2 root root  4096 2006-08-16 01:40 mods-enabled
-rw-r--r-- 1 root root    10 2006-08-16 01:40 ports.conf
-rw-r--r-- 1 root root  2266 2006-08-05 23:46 README
drwxr-xr-x 2 root root  4096 2006-08-16 01:40 sites-available
drwxr-xr-x 2 root root  4096 2006-08-16 01:40 sites-enabled
drwxr-xr-x 2 root root  4096 2006-08-05 23:45 ssl
```

Létrehozzuk az ssl-certet és kulcsot:

```
# apache2-ssl-certificate

creating selfsigned certificate
```

```
replace it with one signed by a certification authority (CA)

enter your ServerName at the Common Name prompt

If you want your certificate to expire after x days call this programm
with -days x
Generating a 1024 bit RSA private key
..++++++
.................++++++
writing new private key to '/etc/apache2/ssl/apache.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:HU
State or Province Name (full name) [Some-State]:GYMS
Locality Name (eg, city) []:GYOR
Organization Name (eg, company; recommended) []:SZE
Organizational Unit Name (eg, section) []:TILB
server name (eg. ssl.domain.tld; required!!!) []:dev.tilb.sze.hu
Email Address []:gecko@sid.sth.sze.hu
```

Hasonló eredményt kell, hogy kapjunk:

```
laptop:~# cat /etc/apache2/ssl/apache.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDctW2mQgcHH02L+Yl0dr2XNFg7V4QBkhdHETMjhhyCjj/0K609
/FLE6M4XGTTzBbRrjaIV9wt26BJD4BFjCJGg7v84QxM8oAGiB30NjRJKFzy3Ya/B
/JiLlTcjqT62YOXWGi3e51I94y0VScg3eYdVvjKJppzWzgQ3CoBKAkQd0wIDAQAB
AoGBALH8cb+ZEbfRq6TNCOdt9njgkqLQcbkeij0ahsYIGifC9ZpAOsZ5HHqFT+Xa
wxOUrynpHyRdhNZQUqKAKSVXPLAvgwV/CHB/Tf+4sg8ArhIZjGt7Y37VQ92XXzbN
3Z+kzhfayyedLpTg3sLzAQvOCDxlFvBWKo5DuCVnKiddE20pAkEA8KWWXg3rI+t1
nXIMy4x+xBE2R1KrD2lnlXRrmRrjTqOVwgmWHfDQuTY7JzyDOUhc9eOForHbU/7c
SDsD0n//9QJBAOrKMpQuzE9y02B6P0zpmNqxOObyXWormvd1k2AsDTlXA3fIC+1s
IYVW7gtLpA7H5Pq56V06gEIxrfqSgs0FcacCQEPmtTL8wmh0O8+dKrcUgYBZSvN9
A/9pQm0NWH7u80xxJASIJ+9yz7Iy9yXvESfMn05SUJbDkJt2wjkvTbjLnTECQEXP
ROamW30r6QmAj52wvxAXLZINchUFThhRzzYDIxNvecp9on4/bmAVDsYxAWCbssuR
x38+y6/i0ZX1RgrrmXUCQQDZKQQimGJDwMv2sGBeQEVZqKykXtUb0Mnyg6YqiMoV
xSQQhrnf8QA8fG3FEkIfA0B8Y2Nin/b7LeLl2OxLCqvJ
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIChzCCAfACCQDjoK4I8OulIjANBgkqhkiG9w0BAQUFADCBhzELMAkGA1UEBhMC
SFUxDTALBgNVBAgTBEdZTVMxDTALBgNVBAcTBEdZT1IxDDAKBgNVBAoTA1NaRTEN
MAsGA1UECxMEVElMQjEYMBYGA1UEAxMPZGV2LnRpbGIuc3plLmh1MSMwIQYJKoZI
hvcNAQkBFhRnZWNrb0BzaWQuc3RoLnN6ZS5odTAeFw0wNjA4MTUyMzQxMjlaFw0w
NjA5MTQyMzQxMjlaMIGHMQswCQYDVQQGEwJIVTENMAsGA1UECBMER1lNUzENMAsG
A1UEBxMER1lPUjEMMAoGA1UEChMDU1pFMQ0wCwYDVQQLEwRUSUxCMRgwFgYDVQQD
Ew9kZXYudGlsYi5zemUuaHUxIzAhBgkqhkiG9w0BCQEWFGdlY2tvQHNpZC5zdGgu
c3plLmh1MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDctW2mQgcHH02L+Yl0
dr2XNFg7V4QBkhdHETMjhhyCjj/0K609/FLE6M4XGTTzBbRrjaIV9wt26BJD4BFj
CJGg7v84QxM8oAGiB30NjRJKFzy3Ya/B/JiLlTcjqT62YOXWGi3e51I94y0VScg3
eYdVvjKJppzWzgQ3CoBKAkQd0wIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAAO3tG8w
Vt+GEDB8z2CLz7whOu8YuAmI6AgZm49zJa4kSnWabNP1OblU+2HSYqeN5oLB2s3s
K8BsrjftT1CUJMtrYeMOpSv7MjlUlgSj9WN/DtdLPkZvnN7v/VjSr/gYb6cDf3bG
Kr9T2S41mxkU6HGFkrDS0K6yIr4X5HzXJpx5
-----END CERTIFICATE-----
laptop:~#
```

Átnevezzük az apache.pem-et dev.pem-re, majd beszúrjuk a következő sort a ports.conf állományba:

```
Listen 443
```

Bekapcsoljuk az ssl modult:

```
cd /etc/apache2/mods-enabled
ln -s /etc/apache2/mods-available/ssl.conf
ln -s /etc/apache2/mods-available/ssl.load
```

A létrejött állományokon semmit nem kell állítgatni. Amit szerkesztenünk kell, az a sites-enable/000-default állomány:

```
NameVirtualHost *:80
NameVirtualHost *:443
<VirtualHost *:80>
     ServerAdmin webmaster@dev.tilb.sze.hu
     SSLEngine on
     SSLCertificateFile /etc/apache2/ssl/apache.pem
     ServerName nincsneki
.    ........
</VirtualHost>
```

A következőket tettük: 2 típusú virtuális hostot definiáltunk: a *:80 és az SSL-es *:443-at, ServerAdmin e-mail címét megváltoztattuk, bekapcsoltuk az SSLEngine opcióval az ssl támogatást. Az SSLCertificateFile -al megadtuk hol található a kulcs, amit legeneráltunk, valamint megadtunk egy fake szervernevet, így a későbbiek folyamán nem fog problémát okozni számunkra ez a konfigfájl, amikor is a virtualhostokat állítjuk be.

Következőkben létrehozzuk a /etc/apache2/site-available/dev állományt a következő tartalommal:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName dev.tilb.sze.hu
        SSLEngine on
        SSLCertificateFile /etc/apache2/ssl/apache.pem
        DocumentRoot /var/www/dev/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
```

```
        <Directory /var/www/dev/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
                RedirectMatch ^/(mail)$ https://dev.tilb.sze.hu/$1
        </Directory>
        ErrorLog /var/log/apache2/dev-error.log
        LogLevel warn
        CustomLog /var/log/apache2/dev-access.log combined
        ServerSignature On
</VirtualHost>
```

Amint látjuk a szintaktika egy az egyben ugyan az, mint a réginél csak kisebb állományokban. Talán a RedirectMatch szorul némi magyarázatra. A reguláris kifejezések tanulmányainkból emlékszünk még a „^” -ra ami azt jelenti, hogy az utána álló kifejezéssel kezdődjön. A „()” akárcsak a regexpeknél megint csak blokk-képzés azzal a különbséggel, hogy itt a záró jelet nem kell megvédeni, a „$” pedig, hogy ez álljon a sor végén. A második kifejezés az, amire majd cseréli az URL-t a RedirectMatch. A „$1” az első (esetünkben egyetlen) zárójellel képzett blokkra hivatkozik. Tehát ha jól megfigyeljük a sima plain http kéréseket amik a http://dev.tilb.sze.hu/mail címre érkeznek redirect-oljuk a https://dev.tilb.sze.hu/mail -re, ami SSL felett megy. Hasonló módon létrehozunk egy dev-ssl fájlt is azzal a különbséggel, hogy kiszedjük a RedirectMatch sort, valamint a *:80 at átírjuk *:443 -ra. Ezzel egyszerűen bekonfigoltuk a dev.tilb.sze.hu nevű virtualhostot. Hasonló módon hozhatunk létre virtual hostokat. Amire oda kell figyelni, hogy minden egyes virtualhost deklarációban a ServerName egyezzen meg a CNAME értékével.

16 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, installálása a következőképpen lehetséges: apt-get install samba samba-common.

A konfiguráció megkezdéséhez a /etc/samba/smb.conf-sample fájlt másoljuk át /etc/samba/smb.conf névre. Ez tartalmazza a szerverrel kapcsolatos beállításokat.

Nézzük a beállításokat a konfigurációs fájlon végighaladva!

A globális beállítások a [global] pontnál találhatók.

```
workgroup = tilb
```

A workgroup beállítással adhatjuk meg a munkacsoportot (Win9x, ME), illetve az NT Domain nevet (WinNT, 2000, XP).

```
server string = Samba Server on TILB
```

A server string tartalma a Windows ”Számítógép leírása” mezőjében jelenik meg.

```
hosts allow = 193.224.130. 193.225.150. 193.224.128.
hosts deny = 193.224.130.99
```

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.) 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
printcap name = /etc/printcap
loadprinters = yes
```

Ha szeretnénk, hogy a nyomtatónkat a /etc/printcap fájlból töltse be a Samba, és ne kelljen

őket egyenként megadni, akkor ezt kell engedélyezni.

```
printing = bsd
```

A szerverünk nyomtatási rendszerét itt adhatjuk meg (nem szükséges beállítani). A következő nyomtatási rendszereket támogatja a samba: bsd, sysv, lprng, aix, hpux, qnx, CUPS.

```
guest account = pcguest
```

Ha szeretnénk pcguest felhasználót, ezt engedélyezni kell, és felvenni a pcguest-et a /etc/passwd fájlba, egyébként a nobody felhasználó jogait használja.

```
log file = /var/log/samba/log.%m
```

Minden gép (%m) logja külön log fájlba kerül.

```
max log size = 100
```

A log fájlok maximális mérete Kbyte-ban.

```
security = user
```

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.

```
password server = <NT-server-name>
```

Az NT-s jelszó szerver nevét adhatjuk meg.

```
password level = 8
username level = 8
```

A felhasználónevek és jelszavak bonyolultsági szintjét adhatjuk meg.

```
encrypt passwords = yes
smb passwd file = /etc/smbpasswd
```

Megadhatjuk, hogy a Samba a jelszavakat titkosítsa. Ha ezt használni szeretnénk, akkor bővebb információt az ENCRYPTION.txt, Win95.txt és WinNT.txt fájlokban a samba dokumentációjában találhatunk.

```
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* .
```

Ez az opció engedélyezi, 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!

```
username map = /etc/smbusers
```

A Unix és samba felhasználónév lehet különböző is, ha az itt megadott fájlban bejegyezzük őket.

```
client code page = 852
character set = ISO8859-2
```

Ezzel a két sorral elérhetjük, hogy a Windows ékezetes fájl-nevei működjenek. A nemzetközi beállításokat részletesen a Samba dokumentációjában találjuk.

```
include = /etc/smb.conf.%m
```

Gépenként külön konfigurációs fájlt adhatunk meg a Samba számára, %m-et a gép Netbios nevével kell helyettesíteni.

```
socket options = TCP_NODELAY
```

Gyorsabb kiszolgálást érhetünk el ezzel az opcióval…

```
interfaces = 193.224.130.161/27 eth0
```

Több hálózati interfészen keresztüli kiszolgálás.

```
local master = yes
```

Tallózáskezelő opciók: állítsuk ezt ”no”-ra, ha nem szeretnénk, hogy szerverünk legyen a local master browser a hálózaton.

```
os level = 88
```

Ez adja meg a Master Browser kiválasztásánál a gépünk prioritását.

```
domain master = yes
```

Ezzel engedélyezhetjük, hogy a gépünk legyen a Domain Master a hálózatban. Ne használjuk ezt az opciót, ha van már NT Domain master a hálózatban.

A Local Master Browser és a Domain Master Browser az M$ 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.

```
preferred master = yes
```

Ha ezt engedélyezzük, akkor a szerver indításakor egy elekciós kört (master browser election) kezdeményez, és nagyobb esélyünk van, hogy a mi gépünk legyen az.

```
domain controller = <NT-Domain-Controller-SMBname>
```

Csak akkor használjuk, ha telepítéskor már van NT domain vezérlőnk.

```
domain logons = yes
```

Engedélyezzük, ha szeretnénk, hogy a Samba Domain logon controller legyen a Win95 kliensek számára.

```
logon script = /etc/smbscripts/%m.bat
```

Gépenként külön bejelentkező szkriptet engedélyezhetünk.

```
logon script = %u.bat
```

Felhasználónként is külön bejelentkező szkriptet engedélyezhetünk.

```
logon path = \\%L\Profiles\%U
```

Itt a felhasználói profilok elérhetőségét adhatjuk meg. A %L a szerver Netbios neve, %U a felhasználó név. (Alul a [Profiles] megosztást engedélyezni kell.

```
name resolve order = lmhosts wins host bcast
```

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.config, /etc/nsswitch.conf, /etc/resolv.conf fájlok függvényében.

```
wins support = yes
```

Ezzel engedélyezhetjük az nmbd WINS szerverét.

```
wins server = w.x.y.z
```

Ha a samba csak WINS kliens lesz, mert már van WINS szerverünk, akkor azt állítsuk be,

viszont az előbbi beállítást állítsuk „no”-ra!

```
dns proxy = no
```

Ha ezt engedélyezzük a DNS által feloldott Netbios neveket eltárolja egy pufferbe, hogy gyorsabb legyen a feloldás.

```
short preserve case = no
```

Kis és nagybetűk megőrzése. Alapértelmezett beállítás a „no”. (Ezt megosztásonként szabályozhatjuk.)

```
default case = lower
```

Kis és nagybetűk alapértelmezés.

```
case sensitive = no
```

Kis és nagybetűk megkülönböztetése, ha „no” beállítással használjuk, egy-két megosztás nem működik (unix típusú rendszerek alatt a Megosztas mEgosztas megoSztaS mást jelent)!

16.1 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.

16.2 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?
    writeable = no      #Lehet-e rá írni?
```

16.3 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
    gues 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
```

Ez a három beállítás az NT kompatibilitás miatt kell.

```
    default devmode = NULL
    nt smb support = yes
    nt status support = yes
```

16.4 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

```
smbpasswd <usernév>
```

parancsokkal változtathatjuk meg a jelszavát.

16.5 A Samba indítása

A samba indítását az smbd és nmbd programok indításával végezhetjük el:

```
# nmbd -D
# smbd -D
```

Illetve a /etc/init.d/samba start|stop|restart szkripttel.

16.6 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.

- smbmnt – Samba megosztáskezelő programja.
- nmblookup <gépnév> – Samba névfeloldó parancsa.

17 Proxy szerver: SQUID és konfigurációja

A squid beszerzése és installálása az apt-get install squid paranccsal lehetséges. A squid konfigurációs fájlja a /etc/squid.conf alatt található. Mivel a konfigurációnak nagyon sok beállítási lehetőség van, mi csak a legalapvetőbbet tárgyaljuk.

```
#cache peer
```

Ezt az opciót akkor használjuk, ha létezik az úgynevezett parent cache (szülő cache) gép, akihez kapcsolódni tudunk, 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-nak, cache céljára. (A squid kb. háromszorosát fogja használni a megadottnak, úgyhogy adjuk a harmadát, mint amennyit egyébként használni szeretnénk.) 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 /usr/local/squid/cache 100 16 256
```

Ez a bejegyzés a fájl cache alkönyvtárat adja meg, ide fogja a squid beszórni a becache-elt fájlokat. Az „ufs” a squid fájl-tárolási módja, ezt ne változtassuk! 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 /usr/local/squid/cache alatt, és mindegyik alatt még 256 darab.) A /usr/local/squid/cache alkönyvtárnak léteznie kell, és a squid számára írhatónak kell lenni, hogy a squid használni tudja.

```
icp_access deny all
```

Ezzel letilthatjuk, hogy más gépek a mi gépünket cache-peer-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 hostokról tölthetünk le WEB lapokat. A dst listát érdemes az alapértelmezett beállításon hagyni, hogy a squid mindent beengedjen. A minimum konfigurációt hagyjuk meg a config fájlban, ezt az egy sort adjuk hozzá a saját domain-ünk ip címeivel!

```
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 nobody
cache_effective_group nogroup
```

Az a felhasználó, illetve csoport, ami alatt futni fog a squid. Ne használjunk root-ot, ha nincs nobody a rendszerünkben, készítsünk egy squid user-t! A /usr/local/cache alkönyvtárat 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 WEB-oldalakon.)

```
cache_access_log /usr/local/squid/logs/access.log
cache_log /usr/local/squid/logs/cache.log
cache_storage_log /usr/local/squid/logs/storage.log
```

A log fájlok helyei. A logs alkönyvtárat is kell tudnia írni a squid-nek.

17.1 A squid indítása

Mielőtt elindítanánk a squid-et, be kell állítani a jogosultságokat azokhoz az alkönyvtárakhoz, amiket a squid el fog érni:

```
# mkdir /usr/local/squid/cache
# chown nobody /usr/local/squid/logs
# chmod 700 /usr/local/squid/logs
# chown nobody /usr/local/squid/cache
# chmod 700 /usr/local/squid/cache
```

Ezután a squid-el meg kell csináltatni a könyvtárstruktúrát, ahová cachelni fog. Ezt csak most kell megtenni, és soha többet!

```
# squid -z
2003/09/15 15:31:28| Creating Swap Directories
```

Ezután, ha belenézünk a /usr/local/squid/cache könyvtárba, láthatjuk, hogy ott a 16x256 db alkönyvtár. Ha ez lefutott, akkor a következő paranccsal indítjuk a squid-ot

```
# /usr/local/squid/bin/squid
```

A ps aux paranccsal ellenőrizhetjük a squid futását.

FONTOS!!!

Mivel a leírás ezen része még slackware Linux alapján készült, igen fontos megjegyeznünk, hogy Debian GNU/Linux alatt a squid a telepítés után létrehozza a cache könyvtárakat es automatikusan elindul!!

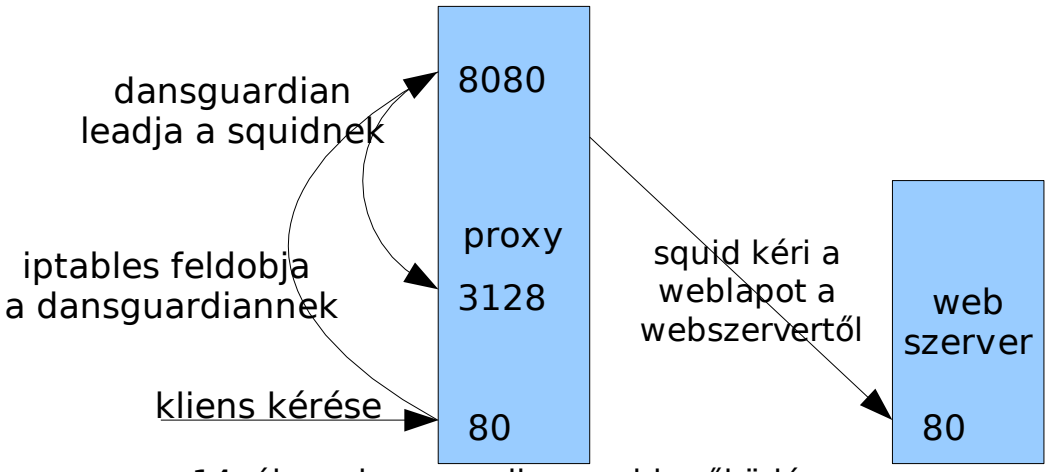
17.2 Dansguardian

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 van 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, ha minden rendben 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 vírusírtó 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 postfix szerint 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-be is az FQDN legyen beállítva.

Akárcsak a sendmail esetén, a postfixnek is meg lehet adni aliasokat. Esetünkben a /etc/aliases fájl tartalmazza a beállításokat. Ha módosítottuk, érvénybe léptetéséhez a newaliases parancsot kell futtatnunk.

18.1 A main.cf

```
# ezt írja ki az smtp szerver mikor betelnetelsz és varja utána a Helo -t
smtpd_banner = $myhostname ESMTP $mail_name (Debian/GNU)

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


# ???
recipient_delimiter = +

# az itt beallított név lesz a @ után. A trivial-rewrite daemon írja át a kimenő
# leveleket.
myorigin = /etc/mailname
#myorigin = tilb.sze.hu

# 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
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"

# ide fognak a leveleink megérkezni, ha a mailbox formátumot használjuk.
home_mailbox = Maildir/

# 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 *Protollok és szoftverek* tárgyból már megismert módot játszuk 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 rfcben 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 mbox

A régebbi Linux/Unix rendszerek a mbox formátumot részesítették előnyben, ha a levelezésről volt szó. A beérkező leveleket az MTA a /var/spool/mail/USERNAME fájlba tette. Ennek óriási hátránya, hogy ha megsérül a fájl, törlődik, akkor az összes levelezésünk odavész. Újabban problémát okozhat az elküldött levél mérete. Sok nagy levélből lesz egy nagyon nagy mbox fájl, amit az MUA-nek be kell olvasnia. Ha foldereket hozunk létre, azok is ebben az egy fájlban vannak. A Maildir formátum ezeket a hibákat igyekszik kiküszöbölni. A régi monolit levélfájl helyett a leveleinket egyesével külön fájlokba menti el. Hiba esetén nem fog elveszni az összes levelünk, ha Fortuna épp pikkel ránk, hanem csak egy-két levél. 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, imap, imaps szervere. Ezek tudnak autentikálni adatbázisból, vagy pam segítségével. A telepítésük egyszerű sok szakértelmet nem követelnek. Debian alatt telepítés után gyakorlatilag azonnal jól működik. A maildir formátumot használja, 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 : imaps szerver
- courier-imap: imap 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 localhostra 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 kiterjesztésű 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