

Széchenyi István Egyetem

Műszaki Tudományi Kar

Informatikai és Villamosmérnöki Intézet

Távközlési Tanszék

Villamosmérnöki szak

Távközlés-informatika szakirány

Hálózati szolgáltatások nyújtása Linux alatt

Hálózati operációs rendszerek I. jegyzet

Írta és szerkesztette: Molnár Zoltán Vilmos

Javították: Jónás Zsolt, Lencse Gábor, Sinkó Gergely

2005.

2006. október 30.

1 Bevezetés

Ez a jegyzet, a Széchenyi István Egyetem Távközlési Tanszékének – Távközlés informatika szakirányán oktatott, Hálózati operációs rendszerek I. című tantárgyhoz (ta65vi) készült. Felhívnom 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, a másik az AT&T hivatalos változata a System V (System Five, Release 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 fordíthatta 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 szerezzen a rendszermag (KERNEL) fejlesztésében.

A 0.11 verzió megjelenése új korszakot nyitott a Linux történetében.

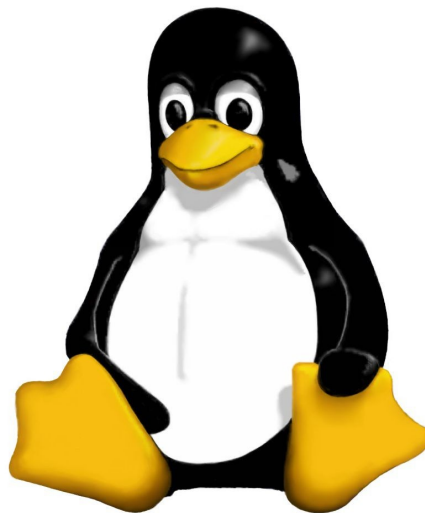
A 0.11 verzió újdonságai:

- demand loading
- kód és adatmegosztás nem kapcsolódó processzek közt
- sokkal jobb floppy-vezérlők (most már többnyire működnek)
- hibajavítások
- Hercules/MDA/CGA/EGA/VGA támogatás
- a konzol hangot is ad (Óh! Fantasztikus rendszermag!)
- `mkfs/fsck/fdisk` (fájlrendszer karbantartó programok)
- amerikai/német/francia/finn billentyűzet
- a `com1/2` sebessége beállítható

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

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 <http://www.kernel.org> 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 előre nem látható 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”-ot ha a patch a várakozási sorba került, és egy „nak”-ot ha elutasították. Az áttekintési ciklusban az „áttekintő bizottság” eldönti, hogy a patch „ack”-ot vagy „nak”-ot 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 BitKeeper-t, 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 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).



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

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/Linuxsal (a 3.1 verzióval) fogunk foglalkozni. A Debian GNU/Linuxról bővebbet a <http://www.debian.org> címen olvashatunk. Magyarországi tükörszervert pedig az <ftp://ftp.hu.debian.org> címen találhatunk.

Más disztribúciók

- Ubuntu: Debian alapú disztribúció, a fejlesztők a legfrissebb csomagokat teszik bele. Honlapja: www.ubuntu.com.
- Slackware: Az első disztribúciók egyike. Más csomagkezelőt használ, mint a debian, frissebbek a stabil kiadások csomagjai. A régi szellemiségét sajnos elvesztette. Honlap: www.slackware.org.
- Fedora Core, Red Hat: A Red Hat az egyik legelterjedtebb pénzes Linux disztribúció az üzleti szférában. A szupportja miatt az Oracle is támogatja. A Fedora a

Red Hat által szponzorált, de nem támogatott nyílt forrású 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. A 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).

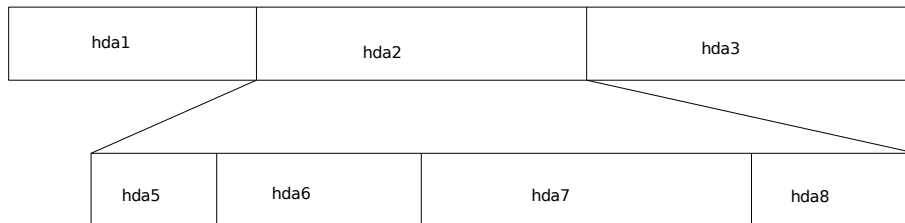
1.3.1 Debian GNU/Linux

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-ra 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 sarge, a testing az etch, és létrehoztak egy nem karbantartott oldstable ágat ami a régi stable volt, ennek a neve woody. Magyarországon több tükörszervere is létezik (a laborban egy lokális mirror-t is csináltunk) 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 lemezén 4 db elsődleges 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 partícióon lévő más-más típusú operációs rendszerek betöltésére.



1.2. 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 a MBR-ben (master boot record) vagy ha itt nincs akkor a partíciós tábla szerinti (a MBR felső 64 byte-ja) aktív partícióon (a nagy bootloaderek kicsit máshogy működnek). A bootszektor memóriába történő töltődése után, az betölti a kernelt a memóriába, és adja át 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 floppyra másoljuk és arról bootolunk. A kernel elfog indulni és csak a root „/” partíció felcsatolásakor fog hibaüzenettel megállni, ha nem talál ilyet. A kernel különböző paramétereit, így a root partíciót az rdev parancssal módosíthatjuk. Bővebben erről lásd: <https://sid-sec.sth.sze.hu/lfs/inditolemez-HOGYAN/Inditolemez-HOGYAN.html>)

1.4.2 Boot metódusok: PXE (Preboot Execution Environment)

A PXE az Intel fejlesztése, melynek célja, hogy a rendszer hálózatról DHCP, és TFTP segítségével bootoljon fel. A számítógép bekapcsolás után, egy DHCPDISCOVER-t küld, amire a DHCP szerver DHCPOFFER-rel válaszol, melyben megad egy TFTP szerveret és egy rajta található NBP-t (network bootstrap program). Linux alatt ilyet a syslinux csomag tartalmaz. Az NBP egy futtatható állomány mely minimális funkciókkal rendelkezik, UDP kezelés ha a kernel is a távoli gépről szedjük le, ami végül is a célunk. Innen minden „ugyanúgy” zajlik mintha lemezzel 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 Boot metódusok: LILO

A LILO (LInux LOader) az első elterjedtebb összetett bootloader. A rendszer a POST után meghívja az első fokozatot (first stage), mely majd a második fokozatot (second-stage bootloader nek 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 Boot metódusok: 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, melyekről a Linux képes bootolni. A kernelnek nem kell fix helyen lenni a partíción, mert a loader second-stage-e 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 kernel 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 vagy nem 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 ramdisk-be 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 eszköz fájlokat 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-el hozhatunk létre. Mind a cramfs mind a cpio-s megoldás esetében gzippel tömörítve van az initrd, melyet a kernel csomagol majd ki induláskor. A cramfs-es initrd-t gunzip után mount -o loop paranccsal tudjuk felcsatolni. A cpio-st pedig kicsomagolhatjuk a következő paranccsal: `gzip -dc < [file] | cpio -i -d [destination]`. Mindkét esetben miután az initrd elvégezte feladatát, a vezérlést a root partícióra teszi át, a `pivot_root` parancs segítségével. Ezekután a Linux az `init` meghívásával folytatja a bootolást.

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 PID¹-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 /etc/inittab

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

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

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

# futási szintekhez tartozó parancsok

10:0:wait:/etc/init.d/rc 0 # halt
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2 # Debian alapértelmezett futási szintje
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6 # reboot
```

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


```
# 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>
# ezeket a parancsokat 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 `syslogd` 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 indítanak 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 linuxok 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 initet (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 lehet a `bash`. A `bash` kezelése egyszerű, és nagyon felhasználóbarát. A kiadott parancsokat visszahívhatjuk a `fel` és `le` gombot nyomogatva. A parancsok, fájlnevek és könyvtárnevek 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 szülőkönyvtárát jelenti. Természetesen a „..” a „/” esetében szintén önmagát jelenti.) A kapcsolókat lehet egymás után megadni pl.: `ls -la /home`

`cd`: change directory = könyvtár váltás. Ugyan úgy mint a dos-ban a „.” az aktuális könyvtárat jelöli a „...” egygyel 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 „-” egygyel 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 kiírása, 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.

Felhívnam a figyelmet arra, hogy a labor gépén root-ként kiadott `rm -rf /*` parancsért fenékberúgás jár (tavalyi példából kiindulva az `rm -rf /home /*` is törli a gyöker könyvtárat, mert felsorolás jelleggel a home könyvtár törlése után a `/ -t` is letörli) !!!

`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 23. 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, ha nincs kapcsoló akkor lexikálisan, `-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`: fájl létrehozás (0 byte méretű), vagy ha már létezik a fájl 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 kiírása, `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 másnéven `KILL`, a `CONT` ami a `STOP` szignálnak a folytatása, esetleg a `HUP` (HangUP) ami egy processzt lelő majd el is indítja. A `kill` után a PID-et kell megadni a `killall` a processz nevét kéri és lehetősége (jogoultáság) 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 processt 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 kiírása (megadott `-n` vagy tíz sor).

`tail`: ugyanúgy szöveges állomány kiírása (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 = ha gzip-el` (ki/be)tömöríteni akarjuk, `j = ha bzip2-vel` (ki/be)tömöríteni akarjuk.

```
# ize.tar.gz be csomagolom az ize könyvtár tartalmát
# tar -zvcf ize.tar.gz ize/
# kicsomagolom az ize.tar.gz
# tar -zxvf ize.tar
```

`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 „-” ha használjuk (`su - user`) akkor a környezeti változók alapértelmezett beállításokkal vált át.

`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. A 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 rendszer időt kérdezhetjük, állíthatjuk 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, fájlnevre; a találatokon pedig parancsokat hajthatunk végre.

Ráérő időnkben (pl.: gyakorlaton) próbálgassuk az alap parancsokat, és szokjuk meg ezek használatát.

Fontos jó tanács: Soha ne használjunk ékezetes karaktereket fájlok és könyvtárak nevének, mert csak magunkkal vagy másokkal szűrhatunk ki! Lencse Gábor a szóközök mellőzését is javasolja. Használjunk helyette aláhúzást.

2.2 Az `dpkg`, `apt`, `aptitude`, `dselect` 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 nézzünk meg.

`update`: csomaglista frissítése. A rendszerünk installálásakor az `install` script megkérdezi, hogy milyen médiáról 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 *available* 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 megpecsélve (csak előtte bizonyosodjunk meg róla, hogy a csomagok számára fenntartott cache könyvtárban, nincs az `ssh` `.deb` csomagja)
- `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`: csomag keresés 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 (`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. Ezt megtehetjük kézzel, de hatékonyabb az `apt-setup` programmal, ami automatikusan megteszi helyettünk. Ha minden áron ragaszkodunk a kézi megoldáshoz, akkor rendszerint annyit kell tennünk, hogy a Debian forrás sort másoljuk és a `deb` szócskát `deb-src`-re egészítjük ki.

Szintén hasznos program a `deborphan`, ami a gépünkön lévő obsolated (már nem használt) csomagokat listázza ki, ezt egy kis shell script 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 csodát, ami arra hivatott, hogy egy általunk kiválasztott funkciót/funkciókat ellátó gép dependenciáját (függőségét) beállítja, majd telepíti a csomagokat.

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 mely a csomagból telepítésre kerül, 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 script. `Preinst` esetén a csomag telepítése előtt, a `postinst` esetén pedig, a telepítés után (pl.: ilyen scriptek kérdeztnek bennünket, postfix telepítésnél). A `prerm`, `postrm` parancsok a telepítés előtt illetve után eltávolítandó fájlokról gondoskodik szintén perl script esetünkben.

2.3 A VI kezelése

A *VI* egy alapvető szövegszerkesztő program UNIX rendszerekhez. 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 elé), `i` = beszúrás (az aktuális karakter után), `r` = felírás mód 1 karakter erejéig, `R` = felülírás mód, `o` = új sor, `x` betű törlés, `q` = kilép, `w` = írás (lemezre), `del` = sortörleszt, `v`: kijelölést kezd, `y`: kijelölést befejez másolásra, `p`: beilleszt, `q!` mindenképpen lépjen ki mentés nélkül (így nem rontjuk el az eredeti fájlt).

Példa: kilépés, mentéssel = `:wq`

kilépés mentés nélkül = `:q!`

2.4 VIM (*Vi Improved*)

A `vi` utódja, lényegesen többet tud nála. Szintaxis kiemeléssel rendelkezik (színezi a beírt forrásokat). Néhány hasznos segédprogram van hozzá: `vimdiff`, `view`.

3 Shell scriptek

3.1 Shell-ek fajtái, kezelésük

Azt a programot, amely a rendszer használata során parancsainkat várja, és végrehajtja shell-nek (parancsértelmezőnek, héjnak) hívjuk. 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 bemutatunk röviden néhány shell típust, 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 $\approx$ 500Kbyte]
- `csh` (Pl.: IBM AIX, IRIX) [Mérete $\approx$ 320Kbyte]
- `ksh` (Pl.: AT&T UNIX, Solaris, HP-UX) [Mérete $\approx$ 630Kbyte]
- `ash`, `dash` (Pl.: Minimum shell, boot lemez esetén) [Mérete $\approx$ 60Kbyte]

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 $\uparrow$ és $\downarrow$ billentyűkkel böngészhetjük a régebben kiadott parancsokat (history)

- A [TAB↔] megnyomásával egy fájl, parancs vagy könyvtár nevet 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 következőképp 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 a 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 névvel 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 shell típus, 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 Unixszal 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 scriptek 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 tartalmaz. 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 scriptelési lehetőségeit mutatja be közel sem teljesen, hiszen erről külön könyvek vannak melyek egyenkénti terjedelme is ezen jegyzet sokszorosa.

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

<i>Jel</i>	<i>Magyar jelentés</i>	<i>Angol jelentés</i>
{ }	Kapcsos zárójel	(Curly) brace
()	Zárójel	Parenthesis
~	Tilde	Tilde expansion
[]	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 {Smith,Taylor}@ieee.org
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 életszerűbb 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 felhasználó nevet 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 nagy betűvel 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 változóban 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 tárolását egy IFS (Internal Field Separator) nevű változóban megadott karakterek alapján dönti el. Ezek alapesetben a szóköz, TAB, új sor karakterek (`IFS=$' \t\n'`). A szavakra bontás miatt azokat a fájl neveket, 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 fájlnev 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 idézőjel szerű 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 idéző jelek 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 OUT 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 első sor > file #az „első sor” stringet a file -ba teszem
# echo második sor >> file #a „második sor” -t hozzáfűzöm
# cat file
első sor
második 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 | sort -r
uborka
korte
dinnye
alma
bash-2.05> ls | grep alma
alma
```

A standard ki és bemeneteket, egymásba fűzhetjük, vagy fájlba tehetjük erre látunk példákat a következőkben:

```
1. 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 láthatjuk a 8. sorban a hibaüzenetet megkapjuk, 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 mindent 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 megsemmisítünk. Ez például `crontab`ok 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 ujsorba EOF -et nem irok
> A másodlagos promptnál sorszerkesztőt használunk,
> így nincs lehetőségünk a már beírt sorokat ezzel a módszerrel
> javítani
> EOF
laptop:~#
```

A fenti egyszerű parancs arra szolgál, hogy a 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 a

/etc/bash.bashrc, majd a ~/.bashrc kerül kiértékelésre. Nem interaktív shell a BASH_ENV-et kifejti, és azt használja fájlnevként, 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 ID-je

HOSTNAME # gép hostneve (hostname -F hostnévfájl paranccsal módosítható /etc/hostname az alapértelmezett fájl)

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 nagyon intelligens 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 például hasznos, ha a feldolgozandó fájlból kiakarunk szedni hostnevet, IP címet, és az IP címet a `host` paranccsal feloldatjuk, majd a kapott értékeket újfent feldolgozzuk a scriptünkkel. Erre óra keretében majd igyekezünk 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 script 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 akár csak 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 kiírása esetén nekünk kell gondoskodni az elválasztó karakterekről pl.: tabulator „\t”; szóköz „ ” stb.). Mint láthatjuk lehetőségünk nyílik feltételes elágazás használatára is. Az `if „( )”` közötti szintaktikája megegyezik a C szintaktikájával (`==;<;<=;>;>=&&;||`; stb). Ezek után nézzük mit csinál a fenti kis rövid script: Ha a „\$3” nagyobb egyenlő mint 1000 és \$3 kisebb egyenlő mint 65000 akkor ír 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 értékek közé eső felhasználók számát a script 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 script lefutása előtt végrehajtódik, ide célszerű a változók deklarálását megadni. Az `END` blokkban pedig olyan műveleteket hajthatunk végre melyek a script lefutása után adnak eredményt.

Egy `awk` script 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;
}

```

Folytatva a fenti script taglalását. A BEGIN részben beállítottuk a min, max és count változót (ez egyébként nem kötelező, hiszen alapértelmezetten 0 minden változó). A második blokkban láthatjuk, hogy az if (feltétel) után szintén képezhetünk blokkot, amibe utasításokat téve a feltételtől függően kerülnek kiértékelésre. A scriptünk 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 nevének 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 történő buffer overflow-val leálló programok memória képeinek fájlrendszerre kiírt 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 „\,” az utasítás végét jelenti a `find` számára, de ha nem teszünk elé „\,” (backslash) jelet akkor a shell kiértékeli, mint 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 régiit 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 biteket veszi figyelembe a `find`. A jogok csak hogy mindenki számára világosak legyenek a következőt jelentik, első bit a `setuid`, `setgid`, sticky bit beállításáért felel, a második az olvasás, harmadik az írás a negyedik a futtathatóság vagy könyvtárak esetében a browse (kereshetőséget) jelenti (balról jobbra haladva).

Nézzük mit csinál a következő parancsunk:

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

Sokan azt gondolnák hogy ez a parancs megkeresi az összes `pass` kezdetű fájlt. Természetesen nem ezt csinálja, 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 semmit 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 parancsok bármelyike hatásos a shell kiértékelésével szemben. A `find` parancsról és kapcsolóiról bővebben a `find` manjában 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) 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 szöveget 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, ez 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#
```

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 (regexpekkel) sztringeket, karakterláncokat helyettesíthetünk egy, csak a keresett csoportra jellemző mintával.

<i>karakter</i>	<i>jelentése</i>
.	bármelyik karakter
\	a mögötte álló karaktert „védi” meg
\$	az előtte álló kifejezés a sor végén áll
^	ha a regexp elején áll, a <u>mögötte</u> 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
[^...]	a felsorolt karakterek (...) <u>nem</u> lehetnek

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

<i>kifejezés</i>	<i>jelentése</i>
*	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 kellene):

```
\([0-9a-zA-Z]{2}\)[-:]\\{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 írta:

```
.*:
```

A fenti kifejezés azt jelenti, hogy bármennyi karakter egy „:”-ig. Az elképzelés majdnem jó, de sajnos a regexpek 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 script 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 scriptek, 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 valószínűleg meg, ha nem lenne kezünkben ez az eszköz?

3.8 Egyszerű shell script példák

Néhány egyszerű példa szemlélteti, hogy a shell scripttel 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 script 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 script minden egyes fájlt megpróbál átnevezni, mivel a nem *.htm* végződésű fájlokra a `sed` nem végez konverziót, az `mv` nem nevezi át ugyanarra a névre a fájlt, így hibaüzenetet ír ki. Mint már volt róla szó a „,” karakterek használata a paraméter kifejtését jelenti. 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 "őóóúúúéái "
"ooouueai_"
ekezetes_szoveg_amiben_szokoz_is_van
laptop:~#
```


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

```
dev:/tmp/test# ls -l
árvíztűrőtükörfúró
próba
proba file
dev:/tmp/test# ../test.sh
dev:/tmp/test# ls -l
arvizturotukorfuro
proba
proba_file
dev:/tmp/test# cat ../test.sh
#!/bin/bash
export IFS=$' \t\n'

for i in *; do
mire=`echo $i| tr "őóüúűéái " "ooouueai_"`
mv "$i" $mire
done
dev:/tmp/test#
```

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

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

3.9 shell programok

Mint már említettem, a scripteket 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 változóba 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(2) vagy túl sok(4) argumentumot adott meg!";
4     exit 1;
5     echo "ennyi argumentumunk van: $#";
6 else
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 „vagy” vizsgálat. A fentiek ismeretében tehát, ha az argumentumok száma kevesebb, mint 1, 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 könyvtárba.

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

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

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

A ha már a konfigurálásnál tartunk. 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 része lesz, az „n” billentyű hatására, eltávolítható a kernelből, amennyiben más kernelrész nem igényli, hogy a kernel része legyen (semmi nem *dependel* rá, a dependencia Debian alatt egyébként is egy fontos kifejezés. Magyarul függőségnek lehet fordítani. Egy csomag vagy kernel rész függ, egy másik csomagtól vagy kernel résztől.) jelölhetjük ki őket. 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.

Az kernelbe fordított meghajtó programkódok előnye, hogy mindig szerepel a rendszerben, így nem kell azt külön betölteni, 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 rendszerünkől. Tipikusan ilyenek az USB-s eszközök (UHCI, OHCI). A lefordított modulokat az `insmod` vagy a `modprobe` parancs segítségével tölthetjük be, az `rmod` 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 (2.4.21-es verzió alapján készült)

Az első menüpontban kiválasztható, hogy a tesztelés fázisában lévő (még nem végleges) meghajtó programokat is feltüntesse a menüben, vagy nem (*Code maturing level options Prompt for development and/or incomplete code/drivers*). Itt érdemes ezt a funkciót engedélyezni, mert az esetleges új eszközeinknek lehet, hogy ilyenek a meghajtó programjai.

Tovább haladva a betölthető modulok támogatását tudjuk engedélyezni (*Loadable module support Enable loadable module support*). A 2.0 Linux megjelenése óta lehet meghajtó programokat modulként fordítani. Erre az `insmod` és a `modprobe` parancsok használhatók.

Harmadik menüpontban a processzorunk fajtáját tudjuk kiválasztani (Processor type and features). Itt különösen fontos, ha maximális teljesítményt szeretnénk a rendszerünknek, akkor a processzorunknak megfelelőt válasszuk ki, de kiindulva abból, hogy a mai IBM PC-k processzorai rendre i386 kompatibilisek, elegendő lehet számunkra 386 kompatibilisre fordítani. Sok más funkció található itt például a sokprocesszoros rendszerek (*Symmetric multi-processing support*), vagy a Co-processzor emuláció (*Math emulation*) támogatása.

A negyedik menüpont alatt az általános beállításokat (*General setup*) találhatjuk. Számítógép-hálózatok támogatása (*Networking support*), PCI, ISA busz, EISA, MCA, PCMCIA/Cardbus, stb. eszközök támogatása. Kiválaszthatjuk kernelünk milyen bináris állományként forduljon le (ELF, vagy a.out). Megtalálható az a.out (GCC-vel fordított bináris állományok) futtatásának, illetve ELF és MISC bináris állományok futtatásának támogatása, az APM (*Advanced power management*) támogatása, amely segítségével a Linux is képes leállítani az ATX tápellátású számítógépeket.

Ezek voltak a fő beállítási opciók, most pedig csak a számunkra fontos menüpontokat vesszük át:

Párhuzamos port támogatás (*Parallel port support*) Érdemes modulként lefordítani

Plug and Play konfiguráció (*Plug and play support*) Ez mehet a kernelbe

Block devices: - Érdemes a floppy támogatást kernelbe fordítani, ha szükségünk van néhány meghajtóra (pl.: párhuzamos portra kapcsolható cd-rom támogatás) azt modulként fordítsuk le

Multi-device support Ezen menüpont alatt kapcsolhatjuk be pl. a RAID támogatást

Hálózati opciók (*Networking options*), ezt nézzük kicsit részletesebben:

Packet socket

Network packet filtering Itt állítható be, hogy csomagszűrést akarunk használni. Lenyíló menüjét lejjebb találhatjuk meg.

Socket filtering Szükséges az ARP (MAC szintű protokollok) számára pl.: dhcpd

UNIX Domain sockets

TCP/IP networking TCP/IP protokollcsalád támogatása. A lenyíló menüben található: *IP: multicasting* = multicast címzés az IP-n, *IP: advanced router*, *IP kernel level autoconfig*, *IP tunneling* = IP alagút támogatás, *IP: GRE tunnels over IP*, *IP: multicasting* = multicast routolás támogatása.

IP csomagszűrő konfiguráció (*IP: Netfilter Configuration*)

Connection tracking Csatlakozás segítő az alkalmazás szintű protokollok számára (Pl.: ftp, irc, tftp ...) NAT-hoz (9. fejezet)

Userspace queueing via NETLINK A későbbiekben bemutatott shaperd forgalomkorlátozáshoz szükséges modul, vigyázat még experimental, azaz néha előidézhethet bolondságokat!!!

IP tables support A későbbiekben (a 9.fejezetben) bemutatott hálózati forgalomirányításhoz és csomagszűréshez szükséges programkódok. Itt érdemes az almenüben lenyíló összes (nem experimental) funkciót modulként lefordítani, mert későbbiekben szükség lesz rá, és ha használni szeretnénk az iptables automatikusan betölti azokat.

ARP tables support, *ARP packet filtering* szintén fontos a MAC szintű kezelőprogramok számára.

Visszalépve egy menüpontot ismét a hálózati opcióknál találjuk magunkat. Itt a következő pont az IPv6 protokoll (*The IPv6 protocol*). Ez is még fejlesztés stádiumában álló funkció, sok program (pl.: apache web szerver) tartalmaz biztonsági réseket az IPv6 alkalmazásánál.

IPv6 esetében is használhatjuk a Linux kernelünket csomagszűrésre (*IPv6: Netfilter Configuration*). Ezt a funkciót az iptables6 programmal tudjuk használni, azonban ez még fejlesztési stádiumban lévő funkció, nem meglepő esetleges kernel pánik okozás.

További hasznos funkciók: *802.1Q VLAN support*. A VLAN (lásd: Kommunikációs rendszerek programozása) használatára beállíthatjuk esetleges szoftveres hálózati hidunkat vagy routerünket.

The IPX protocol: IPX hálózati protokoll támogatása

802.1d Ethernet bridging: Ethernet hálózati híd támogatása

További menükben megtalálhatjuk az IDE/ATA, SCSI, karakteres és grafikus megjelenítők, hang- és videokártyák, hálózati csatolók meghajtó programjait. Ezt mindenki saját „*masinájához*” konfigurálhatja, azonban amit lehet érdemes modulként lefordítani, mert ha szükség van rá, akkor csak be kell tölteni azokat...

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 a 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, úgy készíti el,
A távoli gépen az install:

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

Fontos csomagok, parancsok:

yaird

initrd-tools

kernel-package

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 gyakorlat 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 jelentkezik, 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 ellenőrzöttén kell végezni. Ezt csak az ún. multitaskingos, azaz több feladatos operációs rendszerek tudják teljesíteni. Ilyennek tekinthető pl. az összes UNIX klón, OS/2 stb. Tipikusan nem több feladatos, 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:

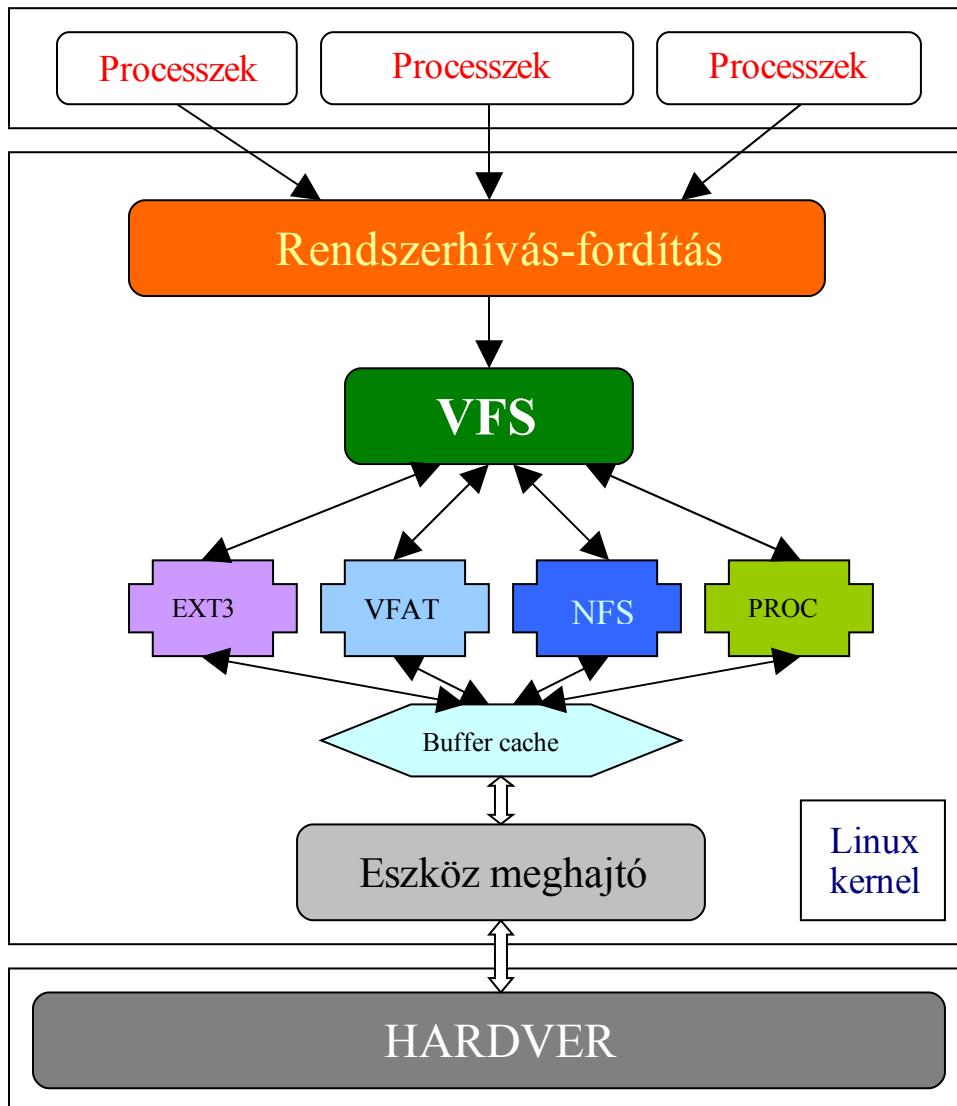
- Egyszerre több program futhat
- A felhasználók egyedi azonosítóval rendelkeznek UserID (authenticáció)
- Az erőforrás elosztás szabályzott
- A rendszer működése védett mind a felhasználókkal, mind a hibás működésű programokkal szemben
- A rendszer ellenőrzött hozzáférést biztosít az egyes hardver elemekhez 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 művelet a partíción naplózva van, így rendszerleállítás esetén nem kell a partíciónkat hosszasan végigellenőrizni elég csak egy kis szeletét. 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.

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.



5.1. ábra: A VFS 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 (gyöker 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/mstab`). Így érhetjük el, hogy egy másik számítógép megosztását valamilyen hálózati 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 fájlt helyezünk el egy 4096 byte-os minimális blokkméretű partíción, akkor az valójában 12Kbyte-ot fog elfoglalni, mert legalább 3 darab egyenként 4 Kbyte-os blokkot fog lefoglalni. Ennek a 3 darab blokknak a partíción belüli címét írjuk bele az *inode* megfelelő adatmezéjébe. 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 mutatókat tartalmaz, most már adatokat tartalmazó diszk blokkokra. Mivel egy blokkba 256 mutató fér el, így a fenti egyszeres indirekt címmel 256K 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 adatot lehet megcímezni. A következő, egyben utolsó mutató egy háromszoros indirekt blokk mutató, 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 adatot lehet címezni. Tehát a fent vázolt struktúrával összesen:

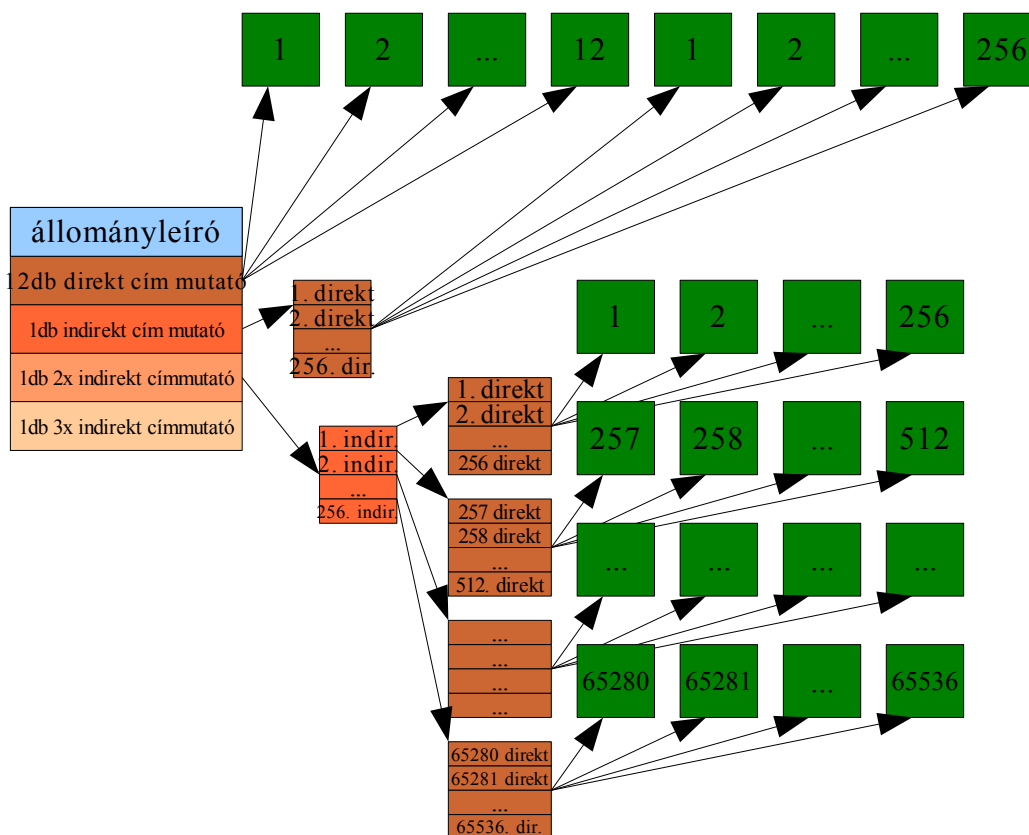
12 db direkt blokk 12K

1 db egyszeres indirekt blokk 256K

1 db kétszeresen indirekt blokk 64M

1 db háromszorosan indirekt blokk 16G

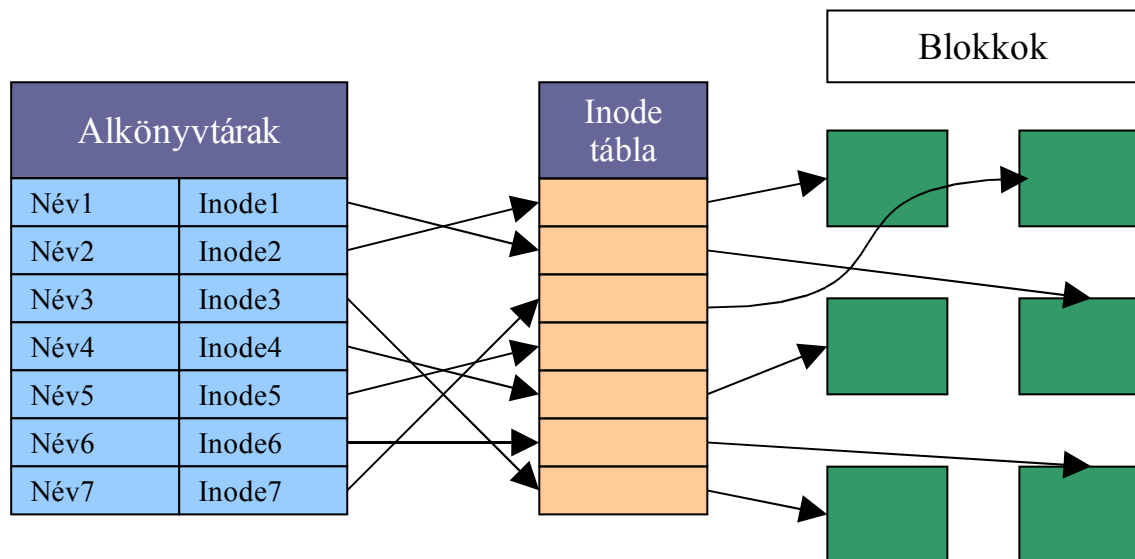
adatot lehet tárolni. A valóságban persze ennél kevesebbet, mert az *inode* állományhossz mezője is 4 byte, így az korlátozza a maximális állományméretet.



5.1. ábra: Inode többszörös címmutatókkal

5.8 Alkönyvtárak

Az alkönyvtárak tulajdonképpen speciális fájlok, ahol a fájl tartalma egy lista, amely azoknak a fájloknak az *inode*-jára mutat, amelyek az alkönyvtáron belül vannak. Az alkönyvtárakat is ugyanolyan *inode*-ok írják le, mint a fájlokat, csak a fájl típus mezőben alkönyvtár-jelzés van („d” bejegyzés). Természetesen egy alkönyvtár bejegyzése mutathat másik alkönyvtár bejegyzésre is. Az alkönyvtár bejegyzéseket a 4. ábra szemlélteti.



5.2. ábra: Alkönyvtárak bejegyzései.

5.9 Linkek

A UNIX fájlrendszerben bevezették a link fogalmát, amely azt jelenti, hogy egy fájl 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 törölünk egy *hard link*-et 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 <fájlnev> <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 hardver elemet 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 olvasható 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 néz ki egy UNIX filerendszer*

Mint minden 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/>)

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 típusai a következők: reguláris, könyvtár, FIFO, socket, karakteres vagy blokkos berendezés.

A fájl 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 nyújt, gcc pl ragaszkodik a .c kiterjesztéshez) 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 bájtt, 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 jön 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 biteit 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 decimálisan `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
```

Másképpen:

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

Legegyszerűbben egy parancsból lehet megadni a felhasználót és csoportot:

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

Meg kell jegyeztem, egyre inkább terjed az ún. POSIX ACL, ami arra szolgál, hogy sokkal „finomabb” beállításokat csináljunk 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
drmmomo:x:1000:100:Molnár Zoltán,laboros,L1-7,tel.szám,:/home/drmmomo:/bin/bash
```

...

Nézzük meg részletesen lépésről lépésre (a mező elválasztó 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-t`) a `/etc/passwd` fájlt. Értelemszerűen be kell jegyezni az új felhasználó adatait. Ezután szinkronizálni kell a `passwd` fájlt a `shadow` fájlal. Erre szolgáló parancs: `pwconv`

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

Erre szolgál (egyszerűbb módszer) a Debian GNU/Linux-ban egy `adduser` nevű script, ezt értelemszerűen kitöltve ugyanazt érhetjük el, mint a fent említett módszerrel ami, viszont minden UNIX rendszerben ugyanúgy működik. (Az `adduser` script 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:

```
# pico /etc/passwd      editáljuk a passwd fájlt
```

Bejegyezzük:

```
buksi:x:1077:100:Kamu Bela,o csak fake user,,:/home/buksi:/bin/bash
```

CTRL+X, YES, ENTER

```
# pwconv      szinkronizálás
# mkdir /home/buksi      home létrehozás
# chown buksi:users /home/buksi      tulaj, csop. megadás
# passwd buksi      jelszó megadás
```

Changing local password for buksi.

```
New password:      ide írjuk be a jelszót
Retype new password:      megerősítjük a jelszót
```

7 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.

7.1 Quota

Mivel egy adott rendszerben a háttértárak kapacitása véges, korlátozni, illetve maximalizálni kell a felhasználók helyfoglalását. Erre 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 egy külön partíción helyezkedik 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.*` 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 módosítuk:

```
/dev/hda4 /home ext2 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 tartalmazza.

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.

7.2 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 fájl méret
- `-n`: – nyitott fájlok maximális száma
- `-s`: – maximális verem méret
- `-t`: – maximális processzoridő másodpercben
- `-u` – az adott felhasználó által futtatott processzek maximális száma
- `-v` – a shell által használható maximális virtuális memória mérete

Bővebb információ: `man bash`, `/ulimit` alatt.

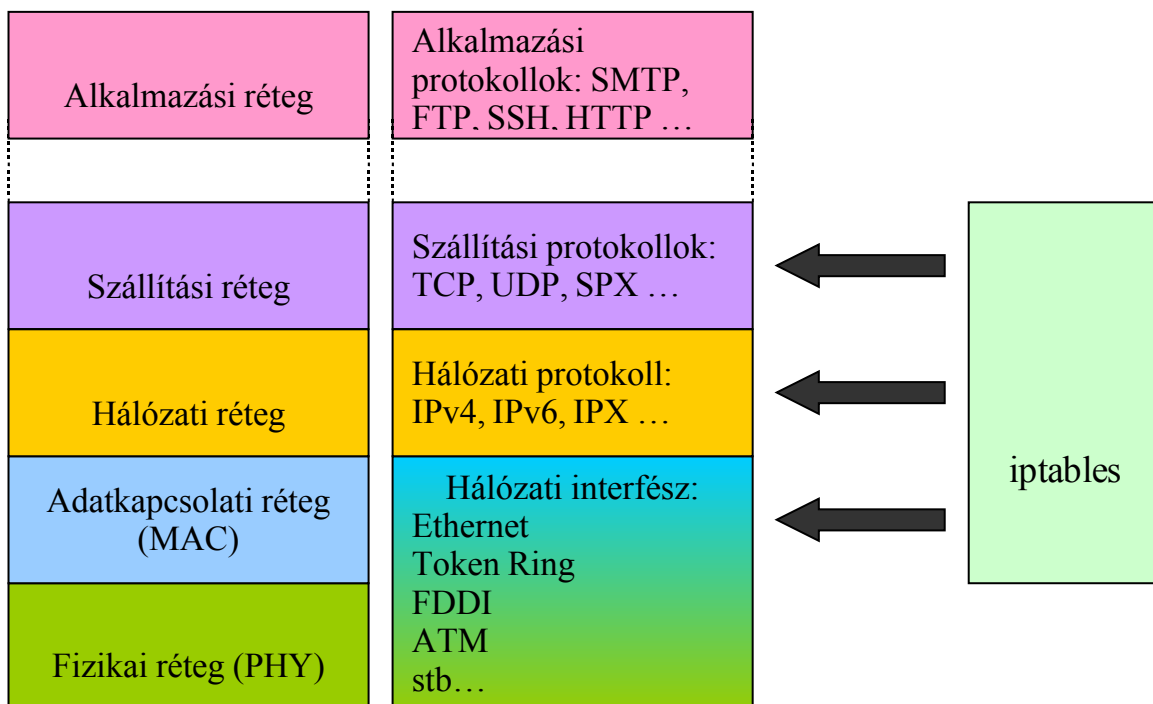
8 Netfilter

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`.

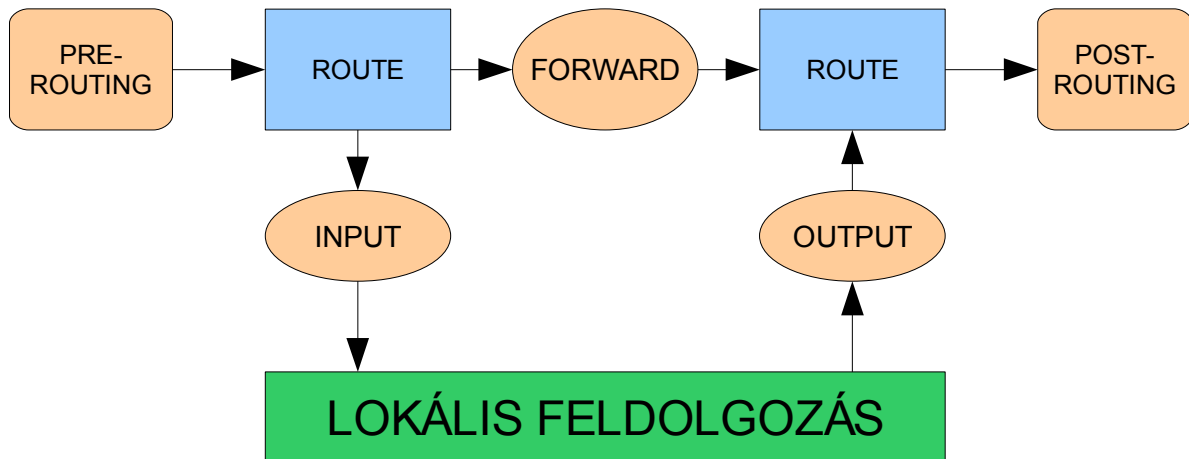
Vizsgáljuk meg, hogy az `iptables` OSI modell szerint milyen rétegekben tevékenykedik.



8.1 ábra: Az `iptables` hatásköre az OSI modell szerint

Mint azt később látni fogjuk az `iptables` egészen az adatkapcsolat rétegtől szállítási rétegig tudja befolyásolni a csomagok útját. Ahhoz, hogy a gép tudjon

forwardolni (adatokat átengedni), 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.



8.2 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.: forgalom korlátozó) kiveszik, és döntési mechanizmusuk szerint továbbítják egy részét, a többit eldobják.

Az `iptables` egy sor különböző funkciókkal rendelkezik (10.2 ábra). Három beépített láncal 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)
- A 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 lenni, 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” láncához 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 törölni.

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 netmaszket 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 interfészüik `“-o”`, ezért az *INPUT* láncan ilyen szabály semmilyen csomagra nem fog illeszkedni (nem is fogadja el az *iptables*). És hasonló képen az *OUTPUT* láncan kimenő csomagok bemeneti interfésszel nem rendelkeznek `“-i”` ezért ezen a láncan a `“-i”` kapcsolókkal meghatározott szabályokra nem fog illeszkedni a csomag (nem is fogadja el az *iptables*). Csak a *FORWARD* láncan átfutó csomagok rendelkeznek mind kimeneti, mind bemeneti interfésszel.

Ha jelenleg nem működő interfészre határozunk meg szabályt, az helyes szabály lesz, 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 szabály, amely az összes Ethernet interfészre illeszkedni fog a `“-i eth+”` kapcsolóval használhatjuk. Az interfész elé „!” –t téve az összes csomagra illeszkedni fog, melyek nem az adott interfészre érkeznek, vagy nem az adott interfésztől 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.

NAT (Network Address Translation, azaz hálózat cím 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 többi – 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 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.

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. Ez lehetővé teszi, 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 elválasztva, vagy ha a porttal egyenlő vagy annál nagyobb portokat akarjuk megváltoztatni a port neve „:”-al utána.

`--sport` ugyan az, mint a `--source-port`

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

`--dport` ugyan az, 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 működéséhez mindkét irányban adat forgalmat 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 kéréelmet.

```
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.

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 üzemelt hálózatban vannak olyan gépek is, melyek számára nem akarunk routolni. Ezt ha 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) eredetileg nem változnak, 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 POSTROUTING -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 címekre NAT-ol a routerünk.

DNAT, azaz a Destination-NAT

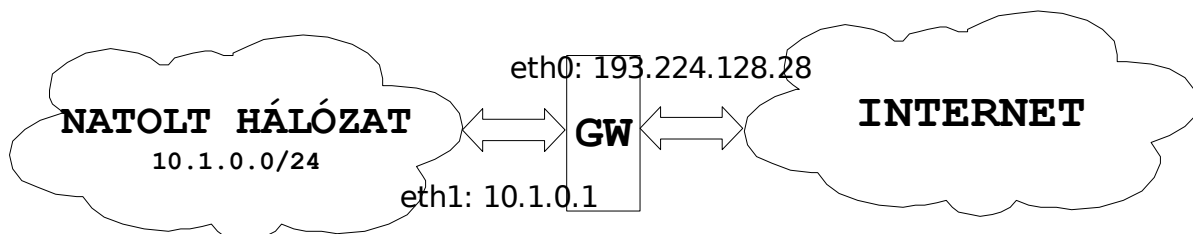
- Továbbítsa a router a 10.0.0.11 IP címre a datagramot: `iptables -t nat -A PREROUTING -j DNAT --to-destination 10.0.0.11`
- Küldjük a WEB kéréseket a 10.0.0.99 IP cím 8080 TCP portjára: `iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 10.0.0.99:8080`

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/2.4.21/kernel/net/ipv4/netfilter` könyvtárban, és `modprobe` paranccsal betölthetők. Ilyen például az FTP ahol két port funkcionál: 21-es port a kommunikációra, a 20-as port 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`
- Alap esetben minden bejövő kapcsolatot szűrjük ki: `iptables -A INPUT -j DROP`



A fenti ábránkon, egy maximum 255 gépes hálózatot szeretnénk, NAT-olni. 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 szabálylanc alapértelmezett szabályát kell DROP-ra beállítani:

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

A helyes szintaktikára figyeljünk oda!

A fenti parancsok ha távolról adjuk ki igen veszélyesek, hiszen az INPUT láncre kiadott DROP parancssal azonnal kizárjuk magunkat a gépről melyet 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 gépet amelyen dolgozik, ezért célszerű a 127.0.0.1 címről érkező összes csomagot engedélyeznünk, de ezt szeretnénk a második helyre beszúrnunk:

```
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.

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 rendszernaplózők 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 nem csak UDP, hanem TCP protokollon keresztül is kommunikál, ami azért fontos mert a TCP csomagokat `stunnel` segítségével lehet titkosítani, az UDP csomagokat alkalmazás szinten kellene titkosítani de erre a rendszernaplózők (tudomásom szerint – utána járni) nem képesek.

9.1 *facility*

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` - autentikációs

`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 és windows szerverek használják általában)

`syslog` - a `syslog` saját üzenetei.

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

Szemben a facility-vel, 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 ; hibadetektálás

Értelemszerűen, egy jól működő rendszer esetében, emerg nem fordul elő vagy csak ritkán, debug szint meg 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 esetén adatvesztést okozhat. Tipikusan nem gyorsító tárazzák a warning-tól kisebb prioritású szinteket. A "mit" kissé bonyolultabb. Szerepelhet benne a "*" wildchar, ha "," -vel választom el őket egész más 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 és 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 és alerttől nagyobb prioritású üzenetet logol.

Néhány példa a konfigurációs fájlból:

```
*.=debug;\n    auth,authpriv.none;\n    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;\n    auth,authpriv.none;\n    cron,daemon.none;\n    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 szerver a -r kapcsolóval lett elindítva, ezt Debian alatt, a /etc/init.d/syslogd scriptben 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. Nem csak a facility és level szerint képes logolni, hanem ezeken belül regexpekkel is képes szűrni a log tartalmat. A távoli naplózást alapértelmezetten kevés maximális kapcsolattal adjak meg (10). Ezt a logszerveren kell megemelni attól függően, hány gép logol rá. Az állomány 3 fő részre osztható (de nem szigorúan), a sorrendek felcserélődhetnek, csak az átláthatóság miatt 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)  
        max-connections(100)  
    );  
};
```

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

A filterek szintaktikája:

filter hivatkozási_név { facility(facility_lista); };

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

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

források, filterek, fájlnevek összerendelése

Miután felépítettük a filter és fájldeklarációinkat, hogy működni is tudjanak, logfájlunként 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 (gondoljuk el, 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.

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, 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 mikor rotálja a logokat, a `tail`-ünk meg mindig a „rég” `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

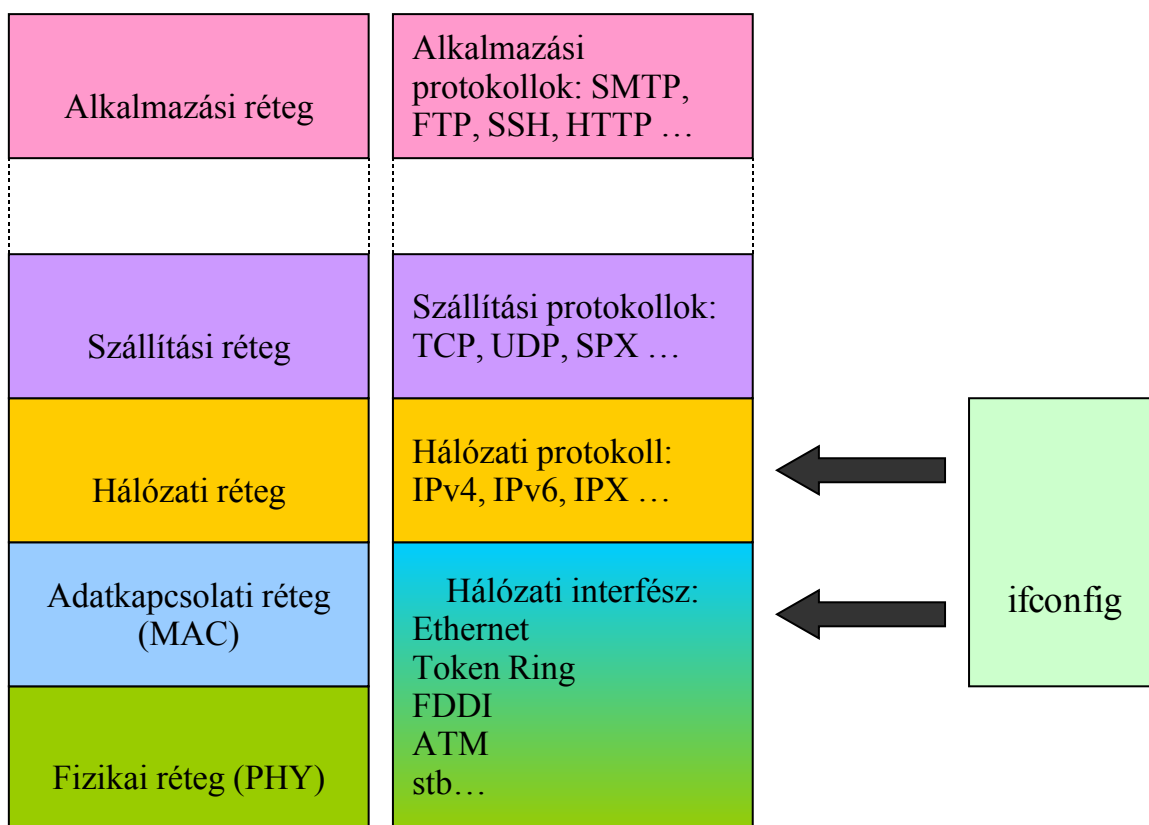
A logok mérete rohamosan növekedhet. Ezért, hogy ezt kiküszöböljük és azért, hogy egyszerűbb legyen keresni a logokban (pl. ha tudjuk mikor történt az esemény nem kell

2 évet visszakeresni), bizonyos időközönként rotáljuk es 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 interfészek konfigurációja

10.1 Interfészek paraméterezése

A hálózati beállításokat (ahogy gyakoroltuk is számítógéphálózatokból) 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).



10.1. 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.

10.2 Adatkapcsolati réteg paraméterezése.

A UNIX rendszerekben a hálózati interfészekre hivatkozni kell. A Linux esetében az Ethernet típusú hálózati csatolókarták hivatkozása: eth0, eth1, stb. függően attól, hány darab Ethernet hálózati kártya van felkonfigurálva a rendszerünkben. A ponttól-pontig kapcsolható 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}
modem	ppp{0,1,2,...,n}
atheros kártya	ath{0,1,2,...,n}
nameif név MACADDR	név

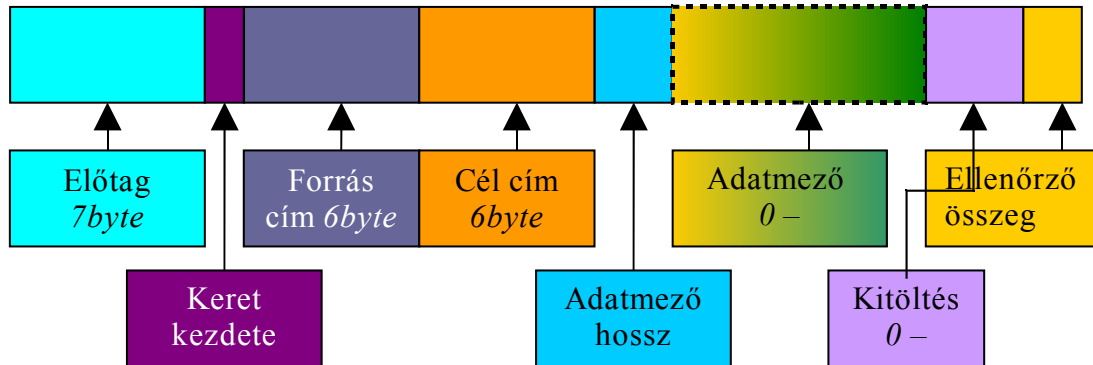
Vizsgáljuk meg Ethernet esetén az adatkapcsolati réteg paramétereit!

Mivel 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 (6. ábra).



10.2 ábra: Ethernet (IEEE802.3) keretszerkezete

Az MTU azaz Maximum Transfer Unity (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 hálózathoz érkező csomagok 1500as MTU-ja problémát okozhat).

Az `ifconfig`-gal állíthatjuk a hálózati csatlólkártyánk MAC címét is (*ha a kártya meghajtó modulja támogatja ezt a funkciót*). A jelenlegi MAC címünk 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.

Érdekes 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 svájcbicska.

Az `ifconfig` (helyes paraméterezése során) a hálózati címet 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 hálózati címet és az alapértelmezett átjárót a `route6` paranccsal adhatjuk meg.

IPX felkonfigurálás: `ifconfig <interface> ipx <hálózati cím> <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 indítóscriptek 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:~#
```


10.3 Az arp, arping, és a rarp

Egy kis elmélet: Az arp (Address Resolution Protocol = Cím felbontó protokoll) OSI 2. rétegbeli protokoll. A Routerok a hálózati szegmensben úgy juttatják el a kereteket a címzettnek, hogy arp-t használva felderítik a szegmensben lévő hálózati interfészek MAC címét. 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 Routerok 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).*

11 Hálózati szolgáltatások UNIX alatt

11.1 Szolgáltatások indítása *inetd* segítségével

Ahhoz, hogy egy adott szolgáltatást nyújtsunk az Internet felé UNIX segítségével, futtatnunk kell egy olyan programot, mely az adott protokollal rendelkező kérésre válaszolni tud. Például, ha egy WEB szerver (HTTP kiszolgálót) szeretnénk üzemeltetni, állandóan futnia kell egy olyan programnak, mely figyeli a 80/TCP porton érkező kéréseket, és képes feltölteni a kliens által kért fájlokat. Régebben nem voltak olyan nagy teljesítményűek a számítógépek, mint napjainkban, ezért UNIX esetében is igyekeztek erőforrás 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 figyel, 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 a például a POP3 protokollt ezután úgy neveznénk, hogy „kutya”, akkor az *inetd.conf*-ba is kutyát kell írunk! 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*.

`/etc/hosts.equiv` – Itt megmondhatjuk, melyek azok a gépek, amelyek bizonyos szolgáltatások (pl: nyomtatás) szempontjából egyenrangúnak számítanak gépünkkel.

`/etc/hosts.lpd` – Itt megadhatjuk, kik nyomtathatnak a szerverünkre telepített nyomtatóra.

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 **obsoleted (nemhasználatos) kategória**. Napjainkban csak nagyon kevés szolgáltatás veszi igénybe. Debian GNU/Linux alatt bejegyzéseket a `/etc/inetd.conf` fájlba az `update-inetd` paranccsal tehetünk.

11.2 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]
```

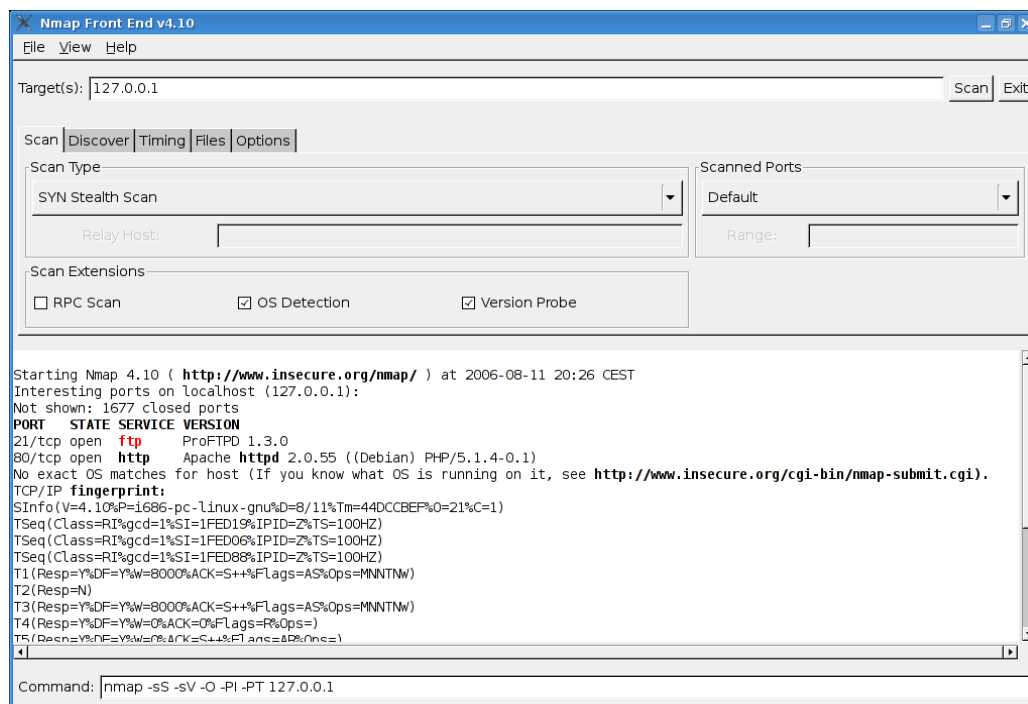
```
6745 ? S 0:00 /usr/sbin/rpc.mountd --no-nfs-version 3
# /etc/init.d/nfs-kernel-server stop
Stopping NFS kernel daemon: rquotad nfsd mountd lockd statd
Unexporting directories for NFS kernel daemon... done
```

11.3 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, például mit sikerült okoznunk az `inetd` átkonfigurálásával:



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 protollokat tekinti a szoftver különösen veszélyesnek: ftp. Hogy miért éppen ez a protokoll? Mert **kódolatlanul küldi a jelszavakat a hálózaton!**

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

```
# fuser -vn tcp 80
```

	USER	PID	ACCESS	COMMAND
80/tcp	root	22539	f....	apache2
	root	22540	f....	apache2

Szintén hasznos parancs a netstat amivel 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

```
#
```

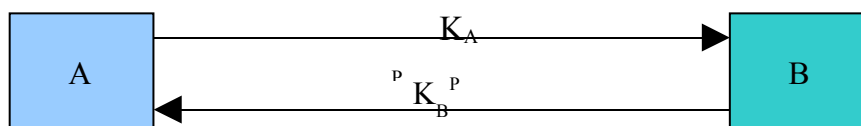
11.4 Az ssh

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.



19, ábra: Session key

Majd mindkét fél generál 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 $K_{\text{SESSION}} = f(r_a, r_b) = f(r_b, r_a)$. Ezután következik az autentikáció. Az autentikálás két módon futhat le. Az a, esetben erős (nyilvános kulcsú) titkosítással egy már előzetesen egyeztetett és megfelelő helyre beillesztett kódok alapján, illetve b, esetben a hagyományos Unix-os jelszó megadása során.

a, eset: 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}` – függően attól, 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 felejtjük el az `id_{rsa,dsa}` fájlunkra 600 jogot adni(!), bár ez rendszerint automatikusan jól jön létre.

11.4.1 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 felhasználó@gepnev
```

Ez akkor is helyesen teszi a kulcsot a megfelelő fájlba, ha a fájl nem létezett vagy már tartalmazott más kulcsokat.

11.4.2 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@gépnev bash -i
```

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

11.4.3 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` parancsal.

```
root@teacher:~/# scp kep.jpg lencse@tilb.sze.hu:~/pic001.jpg
```

11.4.4 Az `sshd` konfigurációja.

Az `sshd` konfigurációs fájlja az `/etc/ssh/sshd_config`. Az alábbiakban bemutatunk egy példa konfigurációs fájlt, majd a fontosabb beállítási lehetőségeket kiemelem:

```
$ more /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

# Élettartama és mérete az egyes verziójú szerverkulcsnak
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 forwardolása 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 vesszük 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 autentikáció. Ha ezt no-ra állítjuk, akkor a rendszer csak kulcs
alapú autentikációval enged be.
UsePAM yes
```

A `/etc/init.d/ssh` `[start|stop|reload|force-reload|restart]` opciókkal irányíthatjuk az sshd futását.

11.5 FTP szerver: proftpd és konfigurációja

A proftpd egy eléggé 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.

A proftpd program `/usr/sbin` könyvtárban található. Ha tudjuk, hogy a rendszeren van proftpd, és nem találjuk meg a `/usr/sbin`-ben akkor a `which proftpd` parancs kiadásával kereshetjük meg.

Az anonymous hozzáféréshez létre kell hozni egy felhasználót, jelen esetben `ftp` nevűt, és adni kell neki egy home könyvtárat. Értelmszerűen csak ezt a home könyvtárat lehet majd anonymous –ként érni.

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

11.5.1 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 szedtek 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      # minden könyvtárban megjelenő üzenet amit
először nyit meg az anonymous felhasználó (ez a file az anonymous home
könyvtárában kell, hogy elhelyezkedjen).

<Limit WRITE>      # írás jog korlátozás anonymous -ként a gyöker 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 eljárunk(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 létre illetve töröljön!  
  
</Anonymous>  
  
# Itt ért véget az anonymous hozzáférés konfigurációja.
```

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

```
RequireValidShell On|Off
```

A fenti engedi az olyan felhasználók belépését akik nem rendelkeznek shelllel. Ez nagyon hasznos amikor a felhasználó shellje 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
```

A felhasználók, ha nincs ez az 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 következőképp használjuk:

```
DefaultRoot ~
```

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

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

11.6 DHCP szerver: *dhcpcd* és konfigurációja

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 (hálózat cím, 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 *dhcpcd* program konfigurációs fájlja a `/etc/dhcp3/dhpcd.conf` alatt található. A maximális frissítési (bérleti) idő megadása lehetséges a *max-lease-time*, az alap beállítás 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/defaults/dhcpd` vagy `/etc/defaults/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;
}
}

```


11.7 DNS szerver: named és konfigurációja

11.7.1 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...)

11.7.2 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)

11.7.3 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 (régábban 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 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 nevekkal 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.

11.7.4 A névfeloldás folyamata.

Vegyünk egy egyszerű esetet. Otthoni internetes kapcsolatunkkal felkeressük a `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. Ezután a már hosszú utat bejárt névfeloldó 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 névfeloldó szerver (62.112.192.4) vissza is adja nekünk a választ, így mi el tudjuk érni a labor szervert. Mi van, ha másodszor is lekérdezzük ezt a nevet? Semmi baj, 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.

11.7.5 Caching-only name server.

Ha egy host nevet beírunk például a WEB böngészőbe, elég sokat kell várnunk míg a kapcsolat felépül. Ennek oka, hogy a DNS feloldás lassú, és esetleg több domain-utótag feloldását is ki kell várni, ami fel van sorolva a `/etc/resolv.conf`-ban. A caching-only name server ezen úgy segít, 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`)

11.7.6 A root DNS szerverek.

Ahhoz, hogy a DNS szerverünk tudja, egy adott TLD feloldásához hová kell fordulnia, meg kell adnunk neki az ún. root DNS szerverek IP címeit. A root DNS szerverek nagy teljesítményű gépeken futó DNS szerverek, melyek az Internet fő gerincvonalain ülve a Top Level Domain-ek feloldását végzik. 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 konfigurációs fájlban majd egy hint típusú zónának kell megadni.

11.7.7 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 fájlkat, esetünkben ebből egy van, ez pedig a `/var/named/mydomain/127.0.0` nevű fájl.

11.7.8 A zónaleíró fájl.

Nézzük meg a 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. A SOA rekordról később még lesz szó.

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 (127.0.0).1 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
```

11.7.9 Egy egyszerű domain 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.

Ezután szükségünk van még egy IP tartományra, amit a hálózati címmel és a netmaszkkal adunk meg: 193.224.130.160/27 (másképp 193.224.130.160/255.255.255.224). Így már mindent tudunk, hogy megírjuk a named.conf fájlt.

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

Az „in-addr.arpa” egy speciális domain név, melynek segítségével egy gép nevét kérdezhetjük meg a DNS rendszertől, az IP cím ismeretében. A „zone” kulcsszó megadja, hogy mi lesz a zóna fájlunk esetében a „gyökér” domain, azaz innen ered a hierarchiánkban majd minden név (ezután ezt nehezen körülírható fogalmat eredetnek fogjuk hívni.)

Ez tehát megmutatja, hogy a 127.0.0 domaint mi fogjuk kezelni, persze csak a saját hatáskörünkben, ez tulajdonképp a /etc/hosts fájl tilb.sze.hu bejegyzésének helyettesítésére szolgál. Lássuk a /var/named/tilb/127.0.0 fájl tartalmát:

```
$TTL 3D
@      IN      SOA      tilb.sze.hu. root.tilb.sze.hu. (
                                2003090102
                                8H      ; Refresh
                                2H      ; Retry
                                4W      ; Expire
                                1D)     ; Minimum TTL
                                NS      tilb.sze.hu.
1      PTR     localhost.
```

Egy zóna fájl három darab úgynevezett erőforrás rekordot tartalmaz, ezek: A SOA, az NS és a PTR rekordok. A SOA a Start Of Authority rövidítése, ezzel indítjuk a DNS zónafájlt:

```
@      IN      SOA      tilb.sze.hu. root.tilb.sze.hu. (
```

A „@” az eredet egy rövidítése, tehát a fenti sor tulajdonképp így néz ki:

```
0.0.127.in-addr.arpa      IN      SOA      tilb.sze.hu. root.tilb.sze.hu. (
```

A Refresh, Retry, Expire, Minimum TTL 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 őrizze meg a másodlagos névszerver a zónafájlokat. A minimum TTL egy elutasított kérés elévülését jelenti.

Az NS rekord megmutatja, hogy a tilb.sze.hu domain-hez tartozó name szerver a tilb.sze.hu gép. A „@” rövidítés hatás itt is érvényesül, már nem kell kiírni:

0.0.127.in-addr.arpa	IN	NS	tilb.sze.hu.
----------------------	----	----	--------------

Végül a PTR (Domain Name Pointer) rekord megmutatja, hogy az 1-re végződő (kezdődő) IP címhez tartozó gépnév a hálózatban (1.0.0.127.in-addr.arpa, azaz 127.0.0.1) a localhost.

A SOA rekorddal kell kezdődnie a DNS szerverben minden zóna fájlban, ebben vannak az alapvető információk a DNS zónáról. Minden zóna fájlban csak egy darab lehet belőle.

Most pedig adjunk egy új zóna bejegyzést a /var/named/named.conf fájlhoz.

```
zone "tilb.sze.hu" {
    type master;
    file "tilb/tilb.sze.hu";
}
```

Ezzel létrehoztunk egy új zónát, mely a tilb.sze.hu névre hallgat, és a hozzá tartozó zóna file a /var/named/tilb/tilb.sze.hu, melynek a tartalma a következő:

```
$TTL 3D
@      IN      SOA      ns.tilb.sze.hu.  root.ns.tilb.sze.hu. (
                                2003090102
                                8H
                                2H
                                4W
                                1D)
                                NS      ns.tilb.sze.hu.
                                MX      10 www.tilb.sze.hu.
localhost      A          127.0.0.1
ns.tilb.sze.hu.      A          193.224.128.28
apc              IN      A          193.224.130.164
camserv          IN      A          193.224.130.165
```

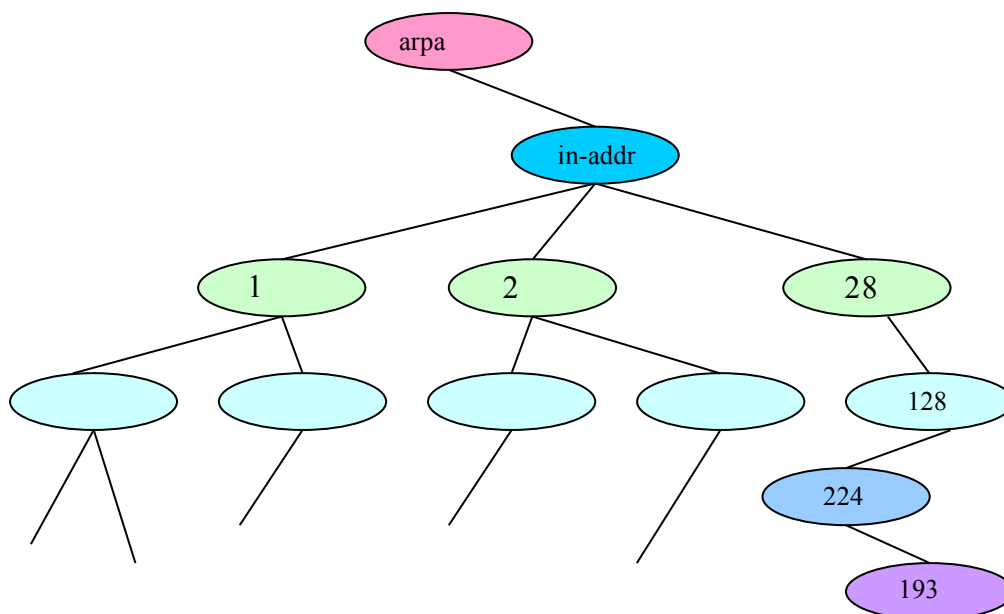
cam1	IN	A	193.224.130.166
ta65		CNAME	www.tilb.sze.hu.
ta13		CNAME	www.tilb.sze.hu.

Az MX rekord a alapértelmezett levelező szerveret adja meg a hálózatunkon. Az „A” rekordok mindegyike egy IP címhez hozzárendelt domaint jelöl meg. Így érhetőek el domain név alapján a zónafájlbán 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. Viszont ha a böngészőbe az. Látható tehát, hogy a virtuális domain-ek mindegyike (amit a „CNAME” rekorddal hoztunk létre) a www.tilb.sze.hu-ra mutat.

11.7.10 A reverse DNS beállítása.

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

A reverse DNS feloldását az arpa. TLD végzi. Az itt található in-addr szerver decimális számok bejegyzéseit tartalmazza. Ezek a decimális számok az IP címek 8 bites számait jelentik. Ezek fastruktúra szerűen szétágazódnak (lásd 20. ábra). A feloldás menete letről felfelé történik. Pl.: a 193.224.128.0/24 tartományhoz tartozó zónafájl megfordítva kell bejegyezni a named.conf-ba: 0/24.128.224.193.in-addr.arpa.



11.1 ábra: reverse DNS feloldás

11.7.11 Reverse DNS konfiguráció

A példánkban a `named.conf` fájlba bejegyeztük a `tilb.sze.hu.rev` zónafájlt. Ez a zónafájl felel a reverse DNS beállításokért. A `tilb.sze.hu.rev` fájl a következő elemekből áll:

```
$TTL 3D
@      IN      SOA      tilb.sze.hu. root.tilb.sze.hu. (
                                2003090201
                                28800
                                7200
                                604800
                                86400)
                                NS      tilb.sze.hu.
166    PTR     cam1.tilb.sze.hu.
167    PTR     cam2.tilb.sze.hu.
168    PTR     agytroszt.tilb.sze.hu.
```

Itt látható, hogy a reverse DNS zónafájl nem sokban különbözik a DNS zóna fájlától. 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>.

11.8 HTTP szerver: apache2 és konfigurációja

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

Az apache jelenleg a legelterjedtebb webszerver a világon, nem sokkal a Microsoft IIS-e előtt. A régi NCSA webszerverből származik az apache, 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 (ezt a bekezdést nem javítom: gecko).

11.8.1 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.

11.8.2 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 apachenak figyelnie kell. Ez az egyik legnagyobb újdonás, hogy itt nincs külön `apache-ssl` és `apache` hanem egy program oldja meg a 2 szolgáltatást.

`conf.d` – egyéb betöltendő konfigurációs állományok helye

`mods-*` - 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-*` - hasonló a `mods-*` 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`. 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 `.pem` szöveges állomány. Ez tartalmazza a kulcsot és a tanúsítványt.

11.8.3 Az apache részletes konfigurációja – a httpd.conf felépítése.

A `httpd.conf` fájlt megnyitva láthatjuk, hogy alapvetően három fő részből áll:

- A globális opciók, melyek a szerverprogram működését adják meg, pl.: hány child processz fusson, stb...
- A szerverre vonatkozó opciók, melyek a nem-virtuális webserverek működését adják meg. (Ezt később részletesen is elmagyarázzuk)

- A virtuális webserverek 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.

11.8.4 Globális opciók

```
ServerType standalone|inetd
```

Itt megadhatjuk, hogy a httpd szerver állandó processzként fusson-e, vagy az inetd szuperszerver indítsa el. Ha az előbbi szeretnénk ide „standalone”-t, ha az utóbbit „inetd”-t kell írni. Ha inetd-vel akarjuk indítani, akkor az inetd.conf-ba 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 száz-as nagyságrend).

```
KeepAliveTimeout 15
```

Ennyi másodpercet várunk egy nyitott TCP kapcsolatban a következő kérésre.

```
MinSpareServers 5  
MaxSpareServers 10
```

A „tartalék” server processzek számának minimális és maximális értéke. Kis szervereknél érdemes a minimumértéket 1-en tartani, ez indítás után 2 processzr fog eredményezni: 1db szülő és 1db tartalék (gyerek). Ha a WEB szerver terhelése növekszik a szülő httpd „ellik” még gyereket. 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 WEB szerver 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.

```
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 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 bővítmódul 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.

11.8.5 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 tilb.sze.hu
```

Ez a szerver neve, amire hallgat, emellett ennek nyilván 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.: Szimbolikus linkek követése.

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

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 adatok 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 webservert 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. Pl.: Az „Indexes” kulcsszó engedélyezi egy alkönyvtár tartalmának listázását. A különféle opciók jelentését lásd az Apache dokumentációban (`apt-get install apache-doc apache2-doc`).

11.8.6 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` domain zóna fájlban a következő sornak kell állnia:

ta65	CNAME	tilb.sze.hu.
------	-------	--------------

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

```

NameVirtualHost *
<VirtualHost *>
    ServerAdmin webmaster@mrtg.tilb.sze.hu
    DocumentRoot /var/www/mrtg/
    ServerName mrtg.tilb.sze.hu
    ErrorLog /var/log/apache/mrtg.error
    CustomLog /var/log/apache/mrtg.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 főszerveret. A log-ok külön vannak választva, itt is érvényes, hogy a log-ot tartalmazó alkönyvtárnak léteznie kell.

11.8.7 Az apache v2 konfigurálása:

Az alábbiakban megmutatjuk hogyan kell telepíteni, beállítani egy apache2 webszerveret, ami kezeli az ssl kapcsolatokat, redirectel ha egy bizonyos kiterjesztése van az url-nek, valamint virtuális hostot is támogat. A célun eléréséhez egy publikus IP-címmel rendelkező gépet használunk amire van egy CNAME bejegyzés is a nevszerveren:

Feltelepítjük az apache-unkat:

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

installálás után létrejön a következő 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
```

Létrehozuk 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+ZebfRq6TNC0dt9njgkqLQcbkeij0ahsYIGifC9ZpAOsZ5HHqFT+Xa
wxOUrynpHyRdhNZQUqKAKSVXPLAvgwV/CHB/Tf+4sg8ArhIZjGt7Y37VQ92XXzbN
3Z+kzhfayyedLpTg3sLzAQvOCDxlFvBWko5DuCVnKiddE20pAkeA8KWWXg3rI+tl
nXIMy4x+xBE2R1KrD2l1XRrmRrjTqOVwgmWHfDQuTY7JzyDOUhc9eOForHbU/7c
SDsD0n//9QJBAOrKmpQuzE9y02B6P0zpmNqxOObyXWormvd1k2AsDTlXA3fIC+1s
IYVW7gtLpA7H5Pq56V06gEIxrfqSgs0FcacCQEPmtTL8wmh008+dKrcUgYBZSvN9
A/9pQm0NWH7u80xxJASIJ+9yz7Iy9yXvESfMn05SUJbDkJt2wjkvTbjLnTECQEXP
ROamW30r6QmAj52wvxAXLZINchUFThhRzzYDIxNvecp9on4/bmAVDsYxAWCbssuR
```

```
x38+y6/i0ZX1RgrrmXUCQQDZKQQimGJDwMv2sGBeQEVZqKykXtUb0Mnyg6YqiMoV
xSQQhrnf8QA8fG3FEkIfA0B8Y2Nin/b7LeLl2OxLCqvJ
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICHZCCAFACCQDjoK4I8OulIjANBgkqhkiG9w0BAQUFADCBBhZELMAkGA1UEBhMC
SFUxDTALEBgNVBAgTBEdZTVMxDTALBgNVBAcTBEdZT1IxDDAKBgNVBAoTA1NaRTEN
MAsgA1UECxmEVElMQjEYMBYGA1UEAxMPZGV2LnRpbGIuc3plLmhlMSMwIQYJKoZI
hvcNAQkBFBhRnZWNrb0BzaWQuc3RoLnN6ZS5odTAeFw0wNjA4MTUyMzQxMjlaFw0w
NjA5MTQyMzQxMjlaMIGHMQswCQYDVQQGEwJlVENMAsgA1UECBMER1lNUzENMAsg
A1UEBxMER1lPUjEMMAoGA1UEChMDU1pFMQ0wCwYDVQQLEwRUSUxCMRgwFgYDVQQD
Ew9kZXludGlsYi5zemUuaHUxIzAhBgkqhkiG9w0BCQEWFGdlY2tvQHNPZC5zdGgu
c3plLmhlMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDctW2mQgcHH02L+Yl0
dr2XNfg7V4QBkhdHETMjhhyCjj/0K609/FLE6M4XGTTzBbRrjaIV9wt26BJD4BFj
CJGg7v84QxM8oAGiB30NjRJKFzy3Ya/B/JiLlTcjqt62YOXWGi3e51I94y0VScg3
eYdVvjKJppzWzgQ3CoBKAKqd0wIDAQABMA0GCSqGSIb3DQEBAQUAA4GBAAO3tG8w
Vt+GEDB8z2CLz7whOu8YuAmI6AgZm49zJa4kSnWabNP1ObIU+2HSYqeN5oLB2s3s
K8BsrjftTlCUJMtrYeMOpSv7Mj1UlgSj9WN/DtdLPkZvnN7v/VjSr/gYb6cDf3bG
Kr9T2S4lmxkU6HGFkrDS0K6yIr4X5HzXJpx5
-----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 csináltuk: 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 sok vizet zavarni számunkra ez a konfigfájl, amikor is a virtualhostokat állítjuk be.

Kövezkezőkben 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 blokkképzés azzal a különbséggel, hogy itt a záró jelet nem kell megvédeni, a „\$” pedig, hogy ez álljon a sor végén. A második kifejezés 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.

11.9 Microsoft networks kezelése Linuxszal: Samba és konfigurációja

Ha a Linuxot Microsoft Windows gépekkel szeretnénk 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 következő képen lehetséges: `apt-get install samba samba-common`.

A konfiguráció megkezdéséhez a `/etc/samba/smb.conf-sample` fájlt másoljuk át `/etc/samba/smb.conf` fájl 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 tartalmazza 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.) Ezen kívül 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 töltsse 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.

```
guest account = pcguest
```

Ha szeretnénk pcguest felhasználót, ezt engedélyezni kell, és felvenni a pcguest-et a `/etc/passwd` fájlba, egyébként 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 password szerver nevét adhatjuk meg.

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

A felhasználónevek és jelszavak maximális hosszát szabályozhatjuk.


```
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 = *New*UNIX*passwd* %n\n *ReType*new*UNIX*passwd %n\n *passwd:*all*authenticall
```

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
```

Tallózaskezelő 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 tallózaskezelők, 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 elekción kör (master browser election) kezdeményez, és nagyobb esélyünk van, hogy 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 = host lmhosts bcast
```

A Netbios neveket IP címekre kell fordítani, és a "Name Resolve Order" segítségével megadhatjuk a feloldás sorrendjét. Az alap sorrend "hosts lmhosts bcast". A "hosts" 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 így használjuk, egy-két megosztás nem működik!

11.9.1 Megosztások kezelése.

Ha szeretnénk a felhasználók home könyvtára automatikusan megosztásra kerülni 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.

11.9.2 Standard megosztások.

```
[bigspace] /o q
    comment = Home
    path = /bigspace      #A megosztott könyvtár elérési útja
    valid users = lencse tbalazs drmomom      #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?
```

11.9.3 Nyomtató megosztása.

```
[HP_LaserJet]
    comment = HP LaserJet 2200DN
    printer name = HP LaserJet 2200DN      #Nyomtató neve a Windowsban
    path = /var/spool/lpd/ljet4-a4-auto-mono#Nyomtató spool elérési útja
    browseable = yes      #Tallózható
    printable = yes       #Lehet rá nyomtatni
    writeable = yes       #Lehet rá írni
    guest ok = no         #Guest user használhatja-e?
    print command = echo '/bin/date' %U nyomtatja a %s fajlt. >> \
/tmp/print.log; lpr %s; rm %s      #Nyomtató parancssor, a UNIX hajtja végre %U
user, %s fájl
```

Ez a három beállítás az NT kompatibilitás miatt kell.

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

11.9.4 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>` paranccsal változtathatjuk meg a jelszavát.

11.9.5 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.

11.9.6 A Samba parancsai.

A Samba használatához tartoznak különböző utasítások.

- o `smbmount <//Gépnév/Megosztás> <Könyvtár>` – Windowsos megosztások mountolása (Kernelnek tudnia kell az smbfs-t kezelni)
- o `smbclient <opció> <//Gépnév/Megosztás>` – Windowsos megosztásokra csatlakozás. Opció: `-L` egy gép megosztásainak kilistázása.
- o `smbmnt` – Samba megosztáskezelő programja.
- o `nmblookup <gépnév>` – Samba névfeloldó parancsa.

11.10 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 konfiguráció nagyon sok beállítási lehetőség van, mi csak a legalapvetőbbet tárgyaljuk. Ezeket a sorokat muszáj megváltoztatni, hogy a squid működjön.

```
#cache peer
```

Ezt az opciót akkor használjuk, ha 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 tényleges 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, 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 a cache-peer-ként működő gépek hozzáférését, senkinek nem engedjük, hogy rajtunk keresztül cache-eljen.

```
alc labor src 193.224.130.160/27
```

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, csináljunk 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 tudnia kell írni a squid-nak.

11.10.1 A squid indítása.

Mielőtt részabradítanánk a squid-ot és a felhasználókat még oly békés rendszerünkbe, 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-al meg kell csináltatni a könyvtárstruktúrát, ahová cachelni fog. Ezt csak most kell csinálni, 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, (jó sokáig tart), akkor a következő paranccsal indítjuk a squid-ot:

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

A `ps aux` paranccsal ellenőrizhetjük a squid futását.

11.11 Sendmail és konfigurációja

11.11.1 Az MTA működése.

Protokollok és szoftverek I. című tárgyunkból emlékezhetünk arra, hogy hogyan működik a levelezés, mégis egy rövid áttekintésre szükség lehet hogy megértsünk egyes folyamatokat. A következő folyamat játszódik le míg egy levél eljut a címzethez:

- A felhasználó megírja a levelet az e-mail kliens programmal (MUA-val pl.: pine)
- Elküldés után az e-mail kliensben beállított SMTP szerverrel (outgoing mail szerver), vagy egyes kliensek esetében közvetlenül a címzett szerverével kapcsolatot létesít a felhasználó gépe, és az SMTP szerver TCP 25-ös portjára bejelentkezve elküldi a levelet.
- Az SMTP szerver a cél e-mail címhez a DNS-ből lekéri a cél domain leveleit kezelő gép címét, és a TCP 25-ös portjára bejelentkezve elküldi azt.
- A céldomain-hez tartozó SMTP szerver továbbküldi, vagy a megfelelő felhasználói fiókban elhelyezi a levelet.

Ahhoz, hogy tudjuk, hogy egy adott domain (pl.: tilb.sze.hu) leveleit ki kezeli, a domain DNS szerverében, a megfelelő konfigurációs fájlban kell lenni egy bejegyzésnek, amely megmutatja, hogy a domain leveleit melyik gépnek kell elküldeni, hogy eljussanak a címzethez. A következő sorok a tilb.sze.hu domain DNS rekordját mutatják, a levelező szervert az "MX" azonosítót tartalmazó sor adja meg:

```
$TTL 3D
@           IN      SOA      ns.tilb.sze.hu.    root.tilb.sze.hu. (
                                2003090102
                                8H
                                2H
                                4W
                                1D)
                                NS      ns.tilb.sze.hu.
                                MX      10 users.tilb.sze.hu.
localhost   A        127.0.0.1
ns.tilb.sze.hu.      A        193.224.128.28
```

Ahhoz, hogy ez a gép leveleket tudjon fogadni, futnia kell rajta egy SMTP szervernek (MTA). Esetünkben ez a `sendmail` program lesz, ennek beállításával fogunk foglalkozni.

A `sendmail`-t egyszerűen feltelepíthetjük az `apt-get install sendmail` parancs kiadásával.

A `sendmail`-hez tartozó fájlok: (a `sendmail` konfigurációs fájljai a `/etc/mail` alatt találhatók)

`sendmail.cf` – Itt találhatók a `sendmail` beállításai.

`aliases` – Itt tudunk megadni a felhasználóhoz plussz neveket.

`aliases.db` – Ebben tárolódnak az előző pontban említett plusz nevek, az ún. aliasok.

`helpfile` – Ebben az SMTP protokollal kapcsolatos segítő szövegeket találjuk.

A küldésre váró kimenő leveleket a `sendmail` a `/var/spool/mqueue` könyvtárból veszi ki, az érkezett leveleket a `/var/spool/mail` könyvtárban helyezi el a felhasználó nevével megegyező nevű fájlban, innen tudja a levelező kliensprogram kiolvasni.

11.11.2 Mail alias-ok létrehozása.

Tegyük fel, hogy van egy a számítógépen, pl.: `bela.toth` néven. Ekkor az ő e-mail címe lehet pl. `bela.toth@tilb.sze.hu`. Szeretnénk, ha ezután lenne egy `beci@tilb.sze.hu` e-mail címe is, de természetesen ehhez nem akarunk új felhasználót felvenni. Az új e-mail címet, más néven alias-t a `/etc/mail/aliases` fájlban kell megadnunk:

```
beci: bela.toth
```

Miután beírtuk az alias-t a megfelelő helyre, le kell futtatni a `newaliases` nevű programot. Ez újrategyűjti az `aliases.db` nevű fájlban lévő adatbázist, így a `sendmail` gyorsabban hozzáfér az aliasokhoz, mintha állandóan az `aliases` szövegfájlból kellene kiolvasnia. Ezek után, ha egy e-mail érkezik a `beci@tilb.sze.hu` címre, azt a `bela.toth` nevű felhasználó fogja megkapni, azaz hozzáfűződik a `/var/spool/mail/bela.toth` fájlhoz.

11.11.3 E-mail relay beállítása.

Ha szeretnénk, hogy a `sendmail` közvetlenül elfogadjon leveleket másik domain-ben lévő számítógéptől is, akkor meg kell adnunk, hogy melyik domainből engedélyezzük a levelek továbbítását. Egy gyakorlatias példán keresztül könnyebben megérthető ez a tulajdonság: a `tilb.sze.hu` domain leveleit kezelő `tilb.sze.hu` számítógép el kell, hogy fogadjon leveleket a `sze.hu` és a `szif.hu` domain-ekből is, ezen felül van Grúziában egy felhasználó barátunk, aki a `tilb`-et szeretné SMTP szervernek beállítani. A `/etc/mail/sendmail.cf` fájl első részében a következőt kell megadnunk:

```
FR-o /etc/mail/relay-domains
```

Ez annyit jelent, hogy lesz egy `/etc/mail/relay-domains` nevű fájlunk, és itt kell felsorolnunk azokat a domain-eket, amelyekből szeretnénk leveleket elfogadni. A tartalma tehát a következő:

```
sze.hu  
szif.hu  
sanet.ge
```

Ezzel elértük, hogy pl.: a `sanet.ge` domain-ben lévő számítógép is bejelentkezhessen `tilb.sze.hu` TCP 25-ös portjára. Ahhoz, hogy a változások érvénybe lépjenek, még rá kell vennünk a `sendmail`-t, hogy olvassa be újra a konfigurációs fájlt.

```
# killall -HUP sendmail
```

11.11.4 Több domain levelének kezelése.

Előfordulhat, hogy szeretnénk egyszerre több domain leveleit is kezelni, pl. a `tilb.sze.hu` mellett pl.: az `eskola.hu` domain leveleit is elfogadjuk. Emellett szeretnénk, hogy a gépünk ne csak a `tilb.sze.hu` hanem a `mail.tilb.sze.hu` névre is „hallgasson”.

A `/etc/mail/sendmail.cf` fájl első részében a következőt kell megadnunk:

```
Fw/etc/mail/lclhstnames
```

Ez azt jelenti, hogy a `/etc/mail/lclhstnames` nevű fájl fogja tartalmazni a kezelt domain-ek listáját:

```
# cat /etc/mail/lclhstnames  
tilb.sze.hu
```

```
mail.tilb.sze.hu
eskola.hu
mail.eskola.hu
```

Ha ezután valaki a `drmomo@eskola.hu` címre fog levelet küldeni, a felhasználó ugyanúgy megkapja, mint ha az a `drmomo@tilb.sze.hu` címre jött volna. Természetesen ehhez is szükséges, hogy az `eskola.hu` domain DNS szervertől a `tilb.sze.hu` legyen megadva az MX rekordban.

11.11.5 E-mail fiókok letöltése távolról.

Amikor a szerverten számos felhasználó van felvéve, és e-mailjeik ide érkeznek, lehetőséget kell biztosítani arra, hogy távolról is elérjék e-mailjeiket, azaz egy számítógép és egy POP3 vagy IMAP levelező kliens segítségével letöltsék leveleiket.

Az IMAP és POP3 kliensek kiszolgálására szervertől megfelelő programokat kell futtatnunk, melyek lekezelik a TCP 110-es POP3, illetve a TCP 143-as IMAP portokat.

UNIX alatt az Internet felé irányuló szolgáltatásokért az `inetd` szuperszerver felelős, amely figyel a `/etc/inetd.conf` fájlban beállított portokat, és kérés esetén elindítja a megfelelő protokoll daemonját.

A következőkben látjuk az `inetd.conf` megfelelő sorait:

<code>pop3</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>/usr/sbin/tcpd</code>	<code>popa3d</code>
<code>imap4</code>	<code>stream</code>	<code>tcp</code>	<code>nowait</code>	<code>root</code>	<code>/usr/sbin/tcpd</code>	<code>popa3d</code>

Az első oszlopban a szolgáltatás nevét, majd a hálózati csomagok típusát és szállítási protokollját látjuk. Ezután a várakozást beállító flag-ek, a daemont futtató felhasználót, majd a parancsot láthatjuk, amellyel a daemon elindul. Látjuk, hogy minden esetben a `/usr/sbin/tcpd` nevű daemon indít el egy másikat, amely esetünkben mind a POP3, mind az IMAP4 protokollokat lekezele. Ez a `popa3d` nevű daemonja, mely a Debian csomagban megtalálható. (`apt-get install popa3d`)

11.12 MTA POSTFIX

A postfix jelenleg az egyik legelterjedtebb MTA a linuxos világban. A sendmail-el szemben a konfigurációja lényegesen egyszerűbb és nem hasonlít az ementáli sajtra sem. Könnyedén kezeli a maildir formátumot, nagyon minimális konfigurációval is tökéletesen, megbízhatóan működik. Tud autentikálni adatbázisból, kezeli a virtuális felhasználókat és virtuális hosztokat. Könnyedén konfigurálhatjuk spam és vírus szűrésre 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 kényszerít arra, hogy a 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 a FQDN-t:

```
193.224.130.172 users.tilb.sze.hu users
```

Szintén fontos a `myorigin` beállítása miatt, hogy a `/etc/mailname`-be is a 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 futtanunk.

11.12.1 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)

#ezt nem tudom mi
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 szervertől 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 a alapértelmezetten
# 10M körül van
# message_size_limit = 102400000

# ???
recipient_delimiter = +

# az itt beállított 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 usefűl 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 a 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 autentikaciohoz 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
```

11.12.2 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 tárgyából 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 jórésze 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 mitcsinálnak 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 hibaüzenetek és hogy mi idézte azokat elő:

```
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, 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 a `reject_invalid_hostname` kapcsolóval, ha FQDN-nek látszó de nem A, PTR rekordokkal rendelkező hostnevet írunk,

Röviden a kapcsolók és azok amik 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 nak 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 nak ez alapértelmezett
    reject_unauth_destination
# dobja el ha az helo/ehlo után nem FQDN van
    reject_non_fqdn_hostname
# dobja el ha az 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
# azok akik a fenti követelménynek megfeleltek engedélyezett a levélküldés.
```

permit

Az `smtpd_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.

11.12.3 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 tették. 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 is 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. Fájlrendszer hibája 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.

11.13 Courier POP3, IMAP szervercsalád

A courier egy nagyon sokrétű szervercsalád. Van smtp, pop3, pop3s, imap, imaps szervere. Ezek tudnak autentikálni adatbázisból. Egyszerű a telepítésük 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 el 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 fájlja, ebből generáljuk a pem kiterjesztésű fájlokat a /usr/lib/courier/mk[imapd|pop3d|stb]cert scriptekkel. Ez

Irodalomjegyzék

Ez a jegyzet eredetileg Kallai Péter UNIX című Hálózati operációs rendszerek I. óravázlat- segédanyag felhasználásával készült.

[7.2, 7.3, 9.9, 10.4] fejezetek Dr. Lencse Gábor előadás-vázlatából készültek.

[10.6] fejezet Büki Péter előadásanyagából készült.

További felhasznált segédanyagok forrásai:

[1] <http://www.szabilinux.hu>

[2] <http://www.freeswan.org>

[3] <http://www.netfilter.org>

[4] <http://www.debian.org>

[5] <http://www.kernel.org>

[6] <http://www.linux.org>

[7] Debian GNU/linux Manual

<https://tanari3.piar.hu/suliportal/sugo/unix.html>

<http://progkor.inf.elte.hu/200405.2/eloadas.htm>

<http://vehok.vein.hu/~brios/jegyzet/Oper%20rendszer%20Ici%20F3s%20rendszer%20Ici%20F3s%20rendszer.htm>

A jegyzetet készítette: Molnár Zoltán Vilmos a Széchenyi István Egyetem – Informatikai és Villamosmérnöki Intézet – Távközlési Tanszék megbízásából.

Ábrajegyzék

ábra: A Linux 1.0.0 hivatalos emblémája (TUX).....	4
ábra: A VFS felépítése.....	19
ábra: Inode többszörös címmutatókkal.....	22
ábra: Alkönyvtárak bejegyzései.....	23
ábra: Az ifconfig hatásköre az OSI modell szerint.....	72
ábra: Ethernet (IEEE802.3) keretszerkezete.....	74
ábra: Az iptables hatásköre az OSI modell szerint.....	78
ábra: IP routing a 2.4.x kernelekben.....	79
ábra: reverse DNS feloldás.....	111

Tartalomjegyzék

1 BEVEZETÉS.....	2
1.1 A UNIX története.....	2
1.2 A UNIX és a LINUX kapcsolata, a Linux története.....	2
1.3 Linux disztribúciók.....	4
1.3.1 Debian GNU/Linux.....	5
1.3.2 Debian GNU/Linux telepítésének előkészítése.....	5
1.4 A Linux rendszer elindulása.....	6
1.4.1 POST.....	6
1.4.2 Boot metódusok: PXE (Preboot Execution Environment).....	6
1.4.3 Boot metódusok: LILO.....	7
1.4.4 Boot metódusok: GRUB.....	7
1.4.5 Initrd vagy nem initrd?.....	7
1.4.6 Init.....	8
1.4.7 /etc/inittab.....	8
1.4.8 Futási szintek (runlevelek).....	9
2 ALAPVETŐ PARANCSONK A UNIX-BAN.....	11
2.1 Alapvető parancsonk:.....	11
2.2 Az dpkg, apt, aptitude, dselect a Debian GNU/Linux csomagkezelői.....	15
2.3 A VI kezelése.....	18
2.4 VIM (Vi Improved).....	19
3 SHELL SCRIPTEK.....	19
3.1 Shell-ek fajtái, kezelésük.....	19
3.1.1 A bash.....	19
3.1.2 A ksh.	21
3.1.3 A csh	21
3.2 Shell scriptek írása.....	21
3.2.1 A kapcsos zárójel kifejtése (Brace expansion).....	22

3.2.2 A tilde kifejtése (Tilde expansion).....	23
3.2.3 Paraméterek és változók kifejtése (Parameter expansion).....	23
3.2.4 Eredmények helyettesítése (Command substitution).....	24
3.2.5 Matematikai kifejezések kiértékelése (Arithmetic expansion).....	24
3.2.6 Szavakra bontás (Word splitting).....	25
3.2.7 Elérési utak kifejtése (Path name expansion).....	25
3.2.8 Idézőjelek kifejtése (Quote removal).....	26
3.2.9 A standard ki és bemenetek irányítása.....	27
3.2.10 bash specifikus fájlok.....	29
3.2.11 A bash néhány fontosabb környezeti változója.....	30
3.3 SED.....	30
3.4 AWK.....	32
3.5 FIND.....	34
3.6 GREP.....	37
3.7 Reguláris kifejezések.....	38
3.8 Egyszerű shell script példák.....	40
3.9 shell programok.....	41
4 LINUX KERNEL.....	42
4.1 Konfiguráció elkészítése.....	42
4.2 A kernel konfiguráló menü felépítése (2.4.21-es verzió alapján készült).....	44
4.3 A kernel fordítása.....	46
5 A UNIX FELÉPÍTÉSE.....	48
5.1 Több felhasználós rendszer lényege.....	48
5.2 Fájlrendszer típusok.....	49
5.3 A VFS, virtuális fájlrendszer.....	49
5.4 Mount.....	51
5.5 A Proc fájlrendszer.....	51

5.6 Fájrendszer, inode-ok, linkek.....	51
5.7 Inode-ok.....	52
5.8 Alkönyvtárak.....	54
5.9 Linkek.....	54
5.10 Eszközfájlok.....	55
5.11 Hogy néz ki egy UNIX filerendszer.....	56
5.12 Fájlok és jogaik a UNIX-ban.....	57
6 FELHASZNÁLÓK KEZELÉSE A UNIX-BAN.....	59
7 FELHASZNÁLÓI KORLÁTOZÁSOK.....	61
7.1 Quota.....	61
7.2 Ulimit.....	63
8 NETFILTER.....	64
8.1 Csomagszűrés működése és megvalósítása.....	65
8.2 Műveletek egy egyszerű szabályon.....	66
8.3 Forráscím és célcím meghatározása.....	67
8.4 Protokoll meghatározása.....	68
8.5 Interfész meghatározása.....	68
8.6 Töredékek meghatározása.....	68
8.7 Kiterjesztések az iptables-hez: új illesztékek.....	70
8.8 MAC cím alapján való vizsgálat.....	71
8.9 Belső hálózatok route-olása, NAT megvalósítása.....	72
8.10 Protokoll segédek.....	73
8.11 Tűzfal megvalósítások iptables segítségével.....	73

9 RENDSZERNAPLÓZÁS, LOGOLÁS.....	76
9.1 facilityk.....	76
9.2 Logolási szintek (loglevelek, severity, severities, priority).....	76
9.3 A sysklogd.....	77
9.4 A syslog-ng.....	79
9.5 A logger.....	81
9.6 A logrotate.....	81
10 HÁLÓZATI INTERFÉSZEK KONFIGURÁCIÓJA.....	83
10.1 Interfészek paraméterezése.....	83
10.2 Adatkapcsolati réteg paraméterezése.....	84
10.3 Az arp, arping, és a rarp.....	89
11 HÁLÓZATI SZOLGÁLTATÁSOK UNIX ALATT.....	90
11.1 Szolgáltatások indítása inetd segítségével.....	90
11.2 Szolgáltatások egyedi indítása.....	91
11.3 Szolgáltatások felderítése, rendszerbiztonság.....	92
11.4 Az ssh.....	93
11.4.1 Kulcsgenerálás.....	94
11.4.2 SSH.....	94
11.4.3 SCP.....	95
11.4.4 Az sshd konfigurációja.	95
11.5 FTP szerver: proftpd és konfigurációja.....	98
11.5.1 Az /etc/proftpd.conf felépítése:.....	98
11.6 DHCP szerver: dhcpd és konfigurációja.....	101
11.7 DNS szerver: named és konfigurációja.....	105
11.7.1 A DNS alapjai.	105
11.7.2 Domain Zóna.	105

11.7.3 Helyi feloldás.	106
11.7.4 A névfeloldás folyamata.	106
11.7.5 Caching-only name server.	107
11.7.6 A root DNS szerverek.	107
11.7.7 A named.conf fájl.	107
11.7.8 A zónaleíró fájl.	109
11.7.9 Egy egyszerű domain megvalósítása.	109
11.7.10 A reverse DNS beállítása.	112
11.7.11 Reverse DNS konfigurálás.	113
11.8 HTTP szerver: apache2 és konfigurációja.....	114
11.8.1 Az apache v1.....	114
11.8.2 Az apache v2.....	115
11.8.3 Az apache részletes konfigurációja – a httpd.conf felépítése.	115
11.8.4 Globális opciók.....	116
11.8.5 A szerver konfigurációja.	118
11.8.6 Virtuális HTTP szerverek konfigurációja.	121
11.8.7 Az apache v2 konfigurálása:.....	122
11.9 Microsoft networks kezelése Linuxszal: Samba és konfigurációja.....	127
11.9.1 Megosztások kezelése.....	132
11.9.2 Standard megosztások.....	133
11.9.3 Nyomtató megosztása.....	133
11.9.4 Felhasználók kezelése.....	134
11.9.5 A Samba indítása.	134
11.9.6 A Samba parancsai.	134
11.10 Proxy szerver: SQUID és konfigurációja.....	135
11.10.1 A squid indítása.....	137
11.11 Sendmail és konfigurációja.....	138
11.11.1 Az MTA működése.....	138
11.11.2 Mail alias-ok létrehozása.....	139
11.11.3 E-mail relay beállítása.....	139
11.11.4 Több domain levelének kezelése.....	140
11.11.5 E-mail fiókok letöltése távolról.....	141
11.12 MTA POSTFIX.....	142
11.12.1 A main.cf.....	142
11.12.2 A postfix korlátozásai.....	144
11.12.3 Maildir vs mbox.....	148

11.13 Courier POP3, IMAP szervercsalád.....	149
--	------------