Miskolci Egyetem


Gépészmérnöki és Informatikai Kar

# Unix/Linux operációs rendszer üzemeltetése

## (BSc/MSc hallgatóknak)

**(béta változat)**

**Készítette:**

**Dr. Mileff Péter**

**Miskolci Egyetem**

**Általános Informatikai Tanszék**

**2022**

# Tartalomjegyzék

## Előszó

Jelen tananyag folyamatos fejlesztés alatt áll. Ezek alapján lehetséges, hogy bizonyos részek hiányosak, vagy rosszul dokumentáltak, esetleg helyesírási hibákat tartalmazhatnak. Amennyiben a kedves olvasó ilyesmit fedez fel, kérem jelezze a jegyet készítőjének.

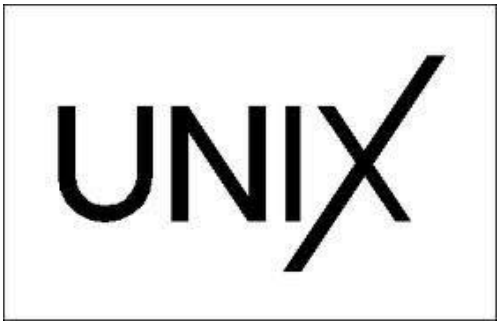
# 1. Bevezetés

A Unix nem egy új operációs rendszer, régóta (informatikai mértékkel mérve nagyon régóta) stabilan és egyre növekvő arányban jelen van a számítástechnikai világban. Hosszú ideig az egyetemi, kutatói szférában volt egyeduralkodó, és mostanában egyre újabb és újabb területeket (banki, vállalati, adatfeldolgozó szféra) hódít meg. Legfőbb ereje dinamikusságában, alkalmazkodóképességében rejlik: képes ugyanazt a környezetet nyújtani mind a multiprocesszoros mainframe, mind az otthoni 386-os PC-je előtt ülő felhasználónak. Manapság, amikor az otthoni számítógépek teljesítménye, illetve a velük szemben támasztott igények, az elvégezendő feladatok már egyre közelebb kerülnek az egykori „nagygépek" szintjéhez, egyre inkább szükség van egy olyan környezetre, amely képes hardvertől, platformtól függetlenül mindenhol ugyanazt nyújtani.

## 1.1 A Unix rövid története

A Unix első változatát 1969-ben készítette **Ken Thomson** és **Dennis Ritchie** az AT&T Bell Laboratóriumában egy PDP-7 típusú számítógépre. A rendszer magját 1973-ban átírták C nyelvre - ennek köszönheti a Unix mind a mai napig legnagyobb előnyét, a könnyű hordozhatóságot. Az AT&T kezdetben ingyen az amerikai egyetemek rendelkezésére bocsátotta a Unix forráskódját, így tíz éven belül százezer fölé emelkedett a működő Unix rendszerek száma. A gyors terjedésnek köszönhetően jelentkeztek a hátrányai is: nem volt egységes ellenőrzése senkinek sem a forráskód, a rendszer egysége felett, így számos (helyi módosításokon alapuló) változat alakult ki, amelyek közül a két legjelentősebb a Berkeley egyetemen kifejlesztett BSD Unix, illetve az AT&T „hivatalos" változata a System V (System Fiv - SVR4), amelyet a Unix System Laboratories fejleszt tovább. (Az USL-t később felvásárolta a Novell). Ezen fő változatok mellett számos kisebb-nagyobb alváltozat van forgalomban még napjainkban is.

Amint a Unix egyre népszerűbbé kezdett válni a kereskedelmi szférában, egyre több cég ismerte fel egy egységes Unix szabvány fontosságát, és több egységesítő, szabványosító bizottság és csoportosulás kezdett dolgozni. Az USL köré tömörülő cégek az SVR4 mögé sorakoztak fel, a BSD irányból érkezők pedig az OSF (Open Systems Foundation) ajánlását, az OSF/1-et támogatják. Időközben független (nem az érdekelt cégek támogatásával létrejött) bizottságok is próbálták valamennyire egységesíteni a BSD és System V ajánlásokat, és az IEEE kidolgozta (az ANSI és az ISO támogatásával) a „**POSIX**" (Portable Operating System Interface (x)) ajánlást, amely igyekszik egyesíteni a két fő szabványt.




Természetesen, mivel a Unix nagyon könnyen hordozható, már elég korán megszülettek az Intel-PC-alapú Unixok is, először csak oktatási célokra (pl. a már 286-oson működő XENIX), majd megjelentek a már komoly munkára is képes PC-s Unix verziók.

## 1.2 Elterjedt UNIX-ok

Ma számos különböző Unix, vagy Unix féle rendszerrel találkozhatunk. Ezekből a legfontosabbak / legismertebbek talán a következők:

**Solaris:** Solaris Operating System eredetileg SUN Microsystems által kifejlesztett számítógépes operációs rendszer. Ma az Oracle fejleszti. Utolsó verzió 11.4 - 2018 augusztus 28. (2022-ben ellenőrizve).

**AIX:** az IBM Unix alapú operációs rendszere. Kizárólag RISC (PowerPC) processzoros verzióban adják ki. Az AIX a UNIX System V rendszeren alapul 4.3BSD-kompatibilis kiegészítésekkel. Utolsó verzió 7.3 - 2021 december 10. (2022-ben ellenőrizve).

**IRIX:** Az IRIX egy UNIX System V-alapú operációs rendszer BSD elemekkel. A rendszer fejlesztője az Silicon Graphics Inc. A fejlesztés 2006-ban leállt.

**HP-UX:** Hewlett-Packard által fejlesztett Unix operációs rendszer, amely UNIX System V rendszeren alapszik. Utolsó verzió 11i v3 - 2022 május 1. (2022-ben ellenőrizve).

**Mac OSX:** Mac operációs rendszer az Apple Computer Inc.-től. A Mac OS egy Unix alapú rendszer, amely főként a OpenStep (korábban NextStep) BSD variánsból származik. De hordoz közös elemeket a Darwin Unix rendszerből is.

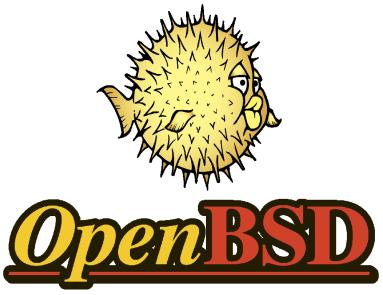
## 1.3 Ismertebb BSD rendszerek

**FreeBSD**: a legnépszerűbb BSD származék. A **FreeBSD** egy szabad Unix-szerű operációs rendszer, amelye az AT&T UNIX-ból keletkezett Berkeley Software Distribution (BSD) egyik leszármazottja. Nem UNIX klón, de a UNIX-hoz hasonlóan működik, UNIX kompatibilis belső felépítéssel és rendszerszintű API-val.


**PCBSD:** A **PC-BSD** egy FreeBSD-re épülő Unix-szerű, asztali felhasználásra fejlesztett operációs rendszer. Grafikus telepítőjén keresztül igyekszik minél könnyebben telepíthetővé rendszerré válni. Célja egy megbízható BSD alapokra épített asztali és szerver operációs rendszer szerepének betöltése. Saját bináris csomag gyűjteménnyel rendelkezik.

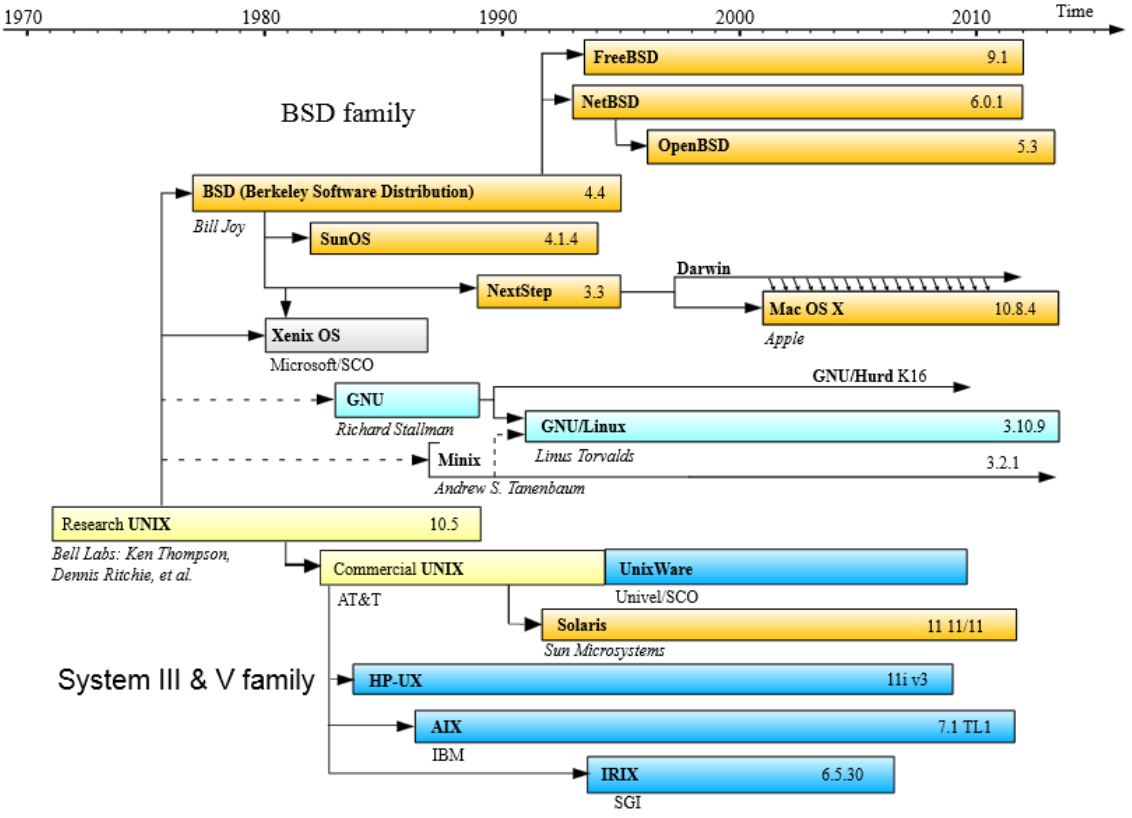

**NetBSD:** népszerű BSD származék a NetBSD Project fejlesztésében. A NetBSD Projekt elsősorban a rendszer fejlesztésében alkalmazott minőségi tervezési megoldásokat, a megbízhatóságot és a teljesítményt tartja szem előtt. A hordozhatósága és barátságos licencelési feltételei miatt a NetBSD leginkább beágyazott rendszereken jelenik meg.


**OpenBSD:** népszerű BSD származék az OpenBSD Project fejlesztésében. 1995 vége felé vált ki a NetBSD fejlesztéséből. A projekt híres fejlesztőinek a forráskód nyíltsága, a minőségi dokumentációja, a szoftverek licencelésével kapcsolatos álláspontja, valamint a forráskód helyességének és biztonságossága melletti rendíthetetlen kitartásáról.




**DragonflyBSD:** a DragonFlyBSD egy szabad Unix-szerű operációs rendszer, amelyet Matthew Dillion a FreeBSD 4.8 kiadásából hozott létre. DragonFly fejlesztése eredetileg a ,,a FreeBSD 4.x vonalának logikai folytatásának" indult, noha idővel jelentős mértékben eltávolodott a FreeBSD fejlesztésétől. Ennek egyik példája a Light Weight Kernel Threads (LWKT) és a ráépülő pehelysúlyú üzenetküldési rendszer implementálása.


Az említett operációs rendszerek idővonalon való ábrázolása világosabbá teszi az egymásra való épülést.




[Forrás: wikipedia]

## 1.4 A GNU projekt, avagy hogy kerül ide a Linux?

Amikor a Unix még csak az egyetemi és akadémiai szférában volt közismert, kialakult körülötte egy hatalmas programkörnyezet: minden egyetem, kutatóintézet elkészítette saját megoldásait felmerülő számítástechnikai problémáira (szövegszerkesztés, mindenféle apró utility, fordítóprogramok), és mivel ezek az intézmények non-profit szervezetek voltak, elkészült szoftvereiket publikussá tették. Az egységes C nyelv

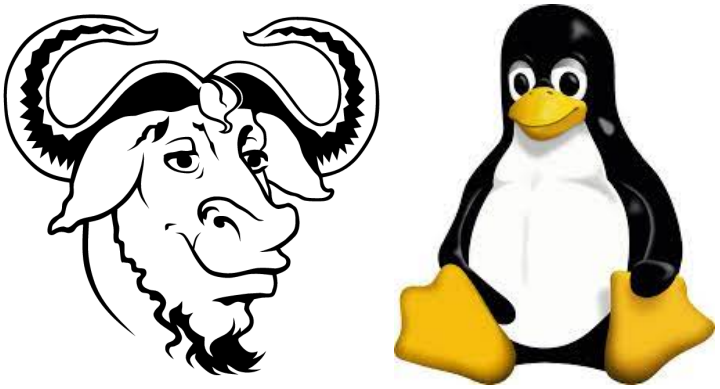
és a környezet miatt minden Unix felhasználó lefordíthatta, használhatta, módosíthatta és továbbfejleszthette őket szinte teljes szabadsággal.

Ennek a folyamatnak az eredményeként alakult meg **Richard Stallman** kezdeményezésére az **FSF** (Free Software Foundation) alapítvány, melynek célja egy szabadon (forráskódban is) ingyen hozzáférhető szoftverkörnyezet biztosítása bárki számára, illetve ennek részeként 1983-ban a GNU project (GNU is Not UNIX), amely pedig egy minél teljesebb Unix rendszert kíván létrehozni és biztosítani.

Ennek jogi megfogalmazása a **GPL (GNU General Public Licence)**. GPL alá eső szoftvert bárki készíthet, amennyiben megfelel bizonyos feltételeknek, és jogi (copyright) probléma esetén számíthat az FSF segítségére. GPL alá eső szoftvert bárki használhat, sőt módosíthatja is azt, amennyiben amikor a szoftvert továbbadja, továbbadja annak teljes forráskódját is, esetleges módosításai feltüntetésével. GPL szoftverért pénzt kérni nem szabad, viszont fel lehet számítani a másolással, terjesztéssel, installálással konfigurálással stb. kapcsolatos költségeket. A szoftver módosításáért sem szabad pénzt kérni - GPL forrás módosítva is GPL forrás marad.

Megvolt tehát a GNU környezet: fordítók, segédprogramok, és a szabadon terjeszthető XFree grafikus felület. Egyedül egy olyan operációs rendszer mag hiányzott csak, amely bizonyítottan szabad (nem tartalmaz copyright alá eső USL vagy BSD kódot). Ennek megírását kezdte el helsinki egyetemista korában **Linus Torvalds**, hogy aztán több száz segítőjével együtt létrehozza azt, amit ma Linuxként ismerünk: egy teljes, szabad operációs rendszert bárki számára.

A Linux jogi értelemben tehát nem mondható UNIX-nak, leghelyesebb volna Unix-klónnak nevezni. Nem követi szigorúan egyik szabványt sem: sok BSD-s és SYSV jellemvonást egyesít magában. Legközelebb a független POSIX-hoz áll, mind a mai napig a Linux tekinthető az egyik legteljesebb POSIX implementációnak.



Bal oldal: GNU logó, Jobb oldal: Linux hivatalos logója

## 1.5 Linux disztribúciók

A disztribúció egy Linux kernelen alapuló teljes (működőképes) Unix rendszer, segédprogramokkal, alkalmazásokkal együtt. Egy disztribúció elkészítése tulajdonképpen a C forrásban meglévő utility-k, programok lefordításából, jegyzékstruktúrába helyezéséből és összekonfigurálásából áll. Sokféle disztribúció létezik, ingyenesek is és kereskedelmiek is (pl.: Slackware Linux, Mint Linux, OpenSuse, Redhat Linux, Manjaro Linux, Ubuntu, Kubuntu, Zenwalk, Frugalware Linux, Uhu Linux, Fedora Linux, Debian, stb).

Az érdeklődők számára célszerű egy kis figyelmet és időt szánni a **www.distrowatch.com** oldalra. A website figyelemmel kíséri a top 100 Linux disztribúciót, ahol információkat kaphatunk az egyed kiadásokról és a tartalmazott főbb csomagok listájáról és verziójáról.

**Feladat:** látogassunk el a **www.distrowatch.com**. Vizsgáljuk meg az aktuális toplista első 10 disztribúcióját.

### 1.5.1 Különbségek a disztribúciók között

Disztribúciókat legtöbbször az különbözteti meg, hogy milyen célközönségnek és milyen feladatra készítik őket. Így léteznek olyanok, melyekkel szinte az összes konfigurálási lehetőséget egy grafikus felületen végezhetjük el és vannak olyanok is, amelyek megkövetelik, hogy a felhasználó mindent a konfigurációs állományok szerkesztésével állítson be. Egyes disztribúciók célja, hogy mindig a lehető legfrissebb szoftvereket szállítsa, míg mások jól kitesztelt, stabil, ám emiatt kissé elavult csomagokat szállítanak.

További fontos különbség, hogy milyen csomagkezelőt használnak az adott terjesztésben. A könyvtárstruktúra általában hasonló módon van felépítve, viszont kisebb különbségek adódhatnak e tekintetben is, extrém esetekben teljesen eltérő felépítést is alkalmaznak a disztribútorok (pl.: GoboLinux). A disztrók egyik fő jellemzője az egyes programcsomagok installálásának, eltávolításának és frissítésének megkönnyítése és támogatása (**Ismert csomagkezelők: apt, rpm, yum,** stb)

Nagy eltérések vannak a disztrók kiadásai között eltelt időnek: egyes disztrók fix ciklust alkalmaznak (például 6 hónaponként egy új kiadás), más disztróknál nincs kötött kiadási ciklus. Léteznek kereskedelmi terjesztések.

Nem mindegyik disztró ugyanazt a kernel verziót használja, továbbá sok disztró saját igényeinek megfelelően módosítja a hivatalosan kiadott, ún. vanilla kernelt.

## 1.6 Linux rendszer telepítése

### 1.6.1 Disztribúció kiválasztása

Mivel nagyon sokfajta Linux létezik, így nem könnyű általános leírást adni arról, hogyan telepítsünk fel egy disztribúciót. Általában minden disztribúció rendelkezik telepítési útmutatóval (pl. **Linux Mint**), így a legfontosabb, hogy annak tanulmányozásával kezdjük. Azért fontos, hogy megértsük a telepítési folyamatot, mert nem megfelelő konfiguráció esetén, akár az egész már létező Windows-os partíció tönkremehet.

Fontos, hogy eldöntsük milyen disztribúció mellett tesszük le a voksunkat. Célszerű olyat választani, amely nagy népszerűségnek örvend és ezáltal nagy támogatottsága. Így szélesebb körű dokumentációval, és jobban letesztelt telepítővel is rendelkezik. Gyakoratilag azt is mondhatjuk, hogy a kezdők válasszanak a **www.distrowatch.com** listájának első 6 helyéről.


Kezdők számára javasolt disztribúciók: **Linux Mint, Debian Linux, Ubuntu Linux, Mageia Linux, Sabayon Linux, Arch Linux, Fedora Linux, PC LinuxOS, Manjaro**

### 1.6.2 Disztribúció kiadásának kiválasztása

Döntésünk után a verzió kiválasztása következik. A gyakorlatban a könnyebb megjegyezhetőség érdekében általában kódnevekkel illetik az egyes kiadásokat.

**Pl. Debian Jessie, Debian Wheezy, Linux Mint Petra, Linux Mint Qiana, stb.**

Természetesen az esetek legnagyobb részében mindig a legfrissebb kiadásokra van szükség, de a gyakorlatban számos kivétel teszi szükségessé a korábbi kiadások letölthetőségét és támogatását.

**Feladat:** Látogassunk el a főbb disztribúciók oldalára, nézelődjünk a változatok között!

### 1.6.3 Megfelelő architektúra kiválasztása

A kiadás kiválasztása után a telepítési képek, úgynevezett image-ek típusának kijelölése szükséges. Mivel a Linux rendszerek több architektúrát támogatnak, azt kell kiválasztanunk, amelyikkel rendelkezünk. Eltekintve a speciális architektúráktól (pl. SAPRC) általában kétféle image típus támogatott: 32 bites és 64 bites. A 32 bites kiadásokat általában **i386**, **i586** vagy **i686** jelölésekkel látják el, amely azt a legalacsonyabb platformot szimbolizálja, amelyiken még működik a rendszer. A 64 bites kiadást pedig **amd64**-el jelölik ha hétköznapi PC-ről van szó. ARM alapú architektúráknál például **arm64** a jelölés.

Linux telepítésére számos megoldás kínálkozik. Általában lehetőség van a megszokott DVD alapú telepítésre, úgynevezett Live CD/DVD alapú telepítésre, és egy mélyebb informatikai tudást igénylő internetes installációra.

**DVD alapú telepítés:** a legegyszerűbb formája a telepítésnek. A DVD-ről bootol a számítógép, és rögtön a telepítőbe érkezünk, ahol a particionáláson felül számos olyan feladatot végezhetünk el, mint a disztribúció testreszabása, vagy bizonyos konfigurációk elvégzése.

**Live CD/DVD alapú telepítés:** azonos a DVD megoldással annyi különbséggel, hogy a Live verzió lehetővé teszi, hogy telepítés nélkül ki is próbálhassuk a rendszert. A telepítést a desktop-on elhelyezett ikonnal végezhetjük el.

**Internet alapú telepíthetőség:** olyan megoldás, amikor egy nagyon kicsit image fájlról bootolunk be, és a rendszer a szükséges részeit egy internetes repository-ból tölti le. Hatékony megoldás stabil és gyors internet kapcsolat mellett.

### 1.6.4 Kiadások

A Linux-ok világában nem értelmezett az a kifejezés, hogy teljes telepítés, hiszen számos különböző ablakkezelő és temérdek mennyiségű csomag áll rendelkezésre. A fejlesztők amiatt, hogy megkönnyítsék a telepítők helyzetét, gyakran különböző kiadásokat készítenek a főbb ablakkezelők szerint csoportosítva. Ezek általában:

- Cinnamon Edition (Cinnamon asztali környezetet használ)
- MATE Edition (MATE asztali környezetet használ)
- KDE Edition (KDE asztali környezetet használ)
- GNOME Edition (GNOME asztali környezetet használ)
- XFCE Edition (XFCE asztali környezetet használ)

### 1.6.5 Tényleges telepítés

Egy Linux telepítéséhez mindig fájl-rendszerbeli változtatásokra van szükség. A legjobb élményt a natív telepítés jelenti, amikor valós operációs rendszerként telepítjük fel akár a teljes hdd-re, akár egy második operációs rendszerként a Windows mellé. Amennyiben valamelyik létező rendszer mellé szeretnénk telepíteni, akkor particionálásra van szükség, mert a Linux saját fájlrendszerrel rendelkezik. A particionálás nem nehéz, de nem kellően odafigyelve és az esetleges áramszünet miatt (létező partíciók mozgatása sok időt vehet igénybe) elveszthetjük a létező rendszerünk összes adatát.

Ettől biztonságosabb megoldás ha **virtuális gép segítségével telepítjük fel a kiválasztott disztribúciót.** Ebben az esetben azonban a telepített rendszer lassabb lesz és nem nyújtja a megfelelő élményt.

# 2. A Linux működésének rövid áttekintése

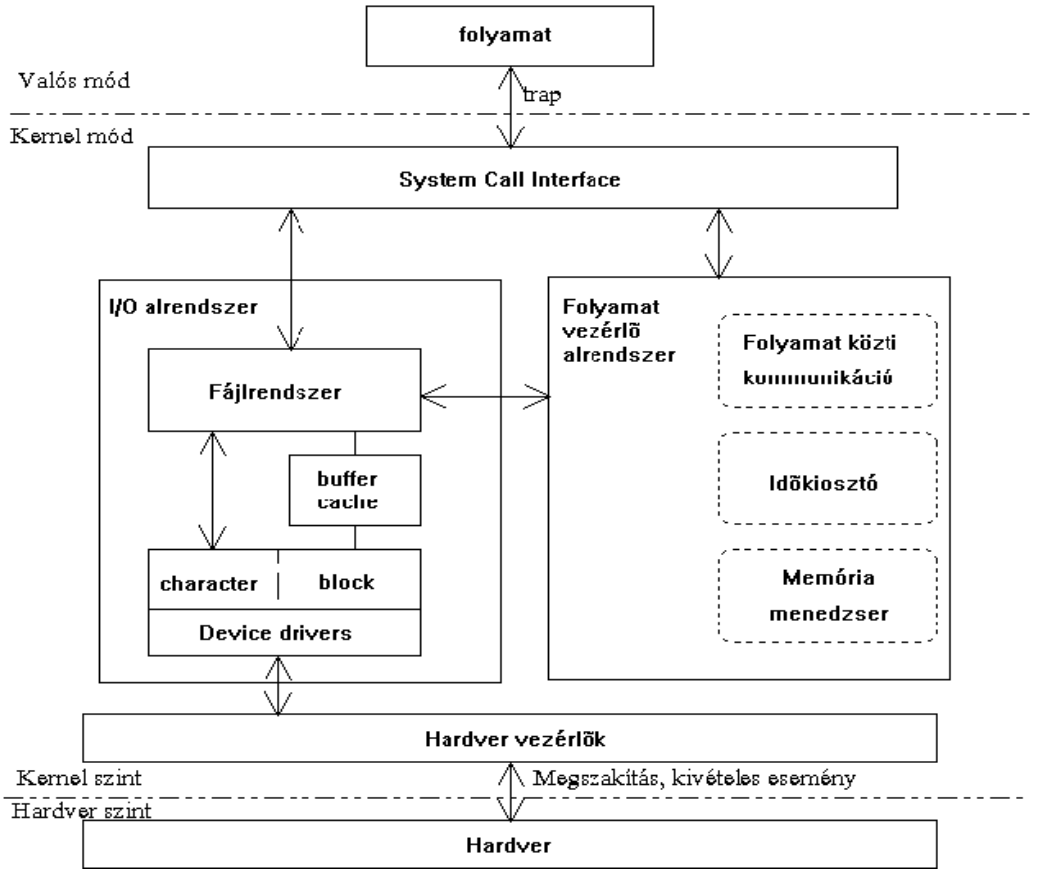
## 2.1 A 386-os csoda

**A Linux egy valódi többfeladatos (multitask) és többfelhasználós (multiuser) operációs rendszer**. Eredetileg az Intel 80386 processzor nyújtotta fejlett tár és taszkkezelési lehetőségeket, valódi időosztásos környezetet kihasználását célozta meg. De mára nagyon sok hardverre adaptálták. A 386-os processzor többféle üzemmódjai közül a Linux a „védett” üzemmódot használja a kernel, azaz az operációs rendszer mag futtatására. Ebben az üzemmódban a kernelnek hozzáférése van a gép összes fizikai erőforrásához, a felhasználói folyamatok (processzek) pedig ún. „user” üzemmódban futnak.

A 386-os processzoron lehetőség van több, egymástól független „user” módú ún. taszk definiálására. amelyek egymástól védettek, nem tudják egymás és a felügyelő kernel memória- területét kiolvasni vagy módosítani, és a gép közvetlen hardver erőforrásaihoz sincs hozzáférésük. Így biztosítható az egyes felhasználói programok egymástól való védelme. Mivel az egyes folyamatoknak a gép fizikai erőforrásaihoz

(pl. winchester, képernyő) sincs közvetlen hozzáférésük, bármilyen perifériaműveletet csak a kernel meghívása útján végezhetnek, így mód nyílik például biztonságos file-rendszer megvalósítására is.

A kernel teljes mértékben, fizikai szinten hozzáfér a gép erőforrásaihoz. Fizikai, a lehető legalacsonyabb szinten kezeli is a hardvert, a legnagyobb teljesítmény elérése érdekében.




**1.ábra.** Unix struktúrája

## 2.2 Memóriakezelés röviden

Memóriakezelésében szintén kihasználja a 386 által nyújtott lehetőségeket: lapozásos virtuális memóriakezelést használ, ahol a fizikai memóriát kiegészíthetjük a winchesterről vett virtuális memóriával (page vagy swap terület). A teljes memóriát lapokra osztja, ezen virtuális lapokat rendeli hozzá az egyes folyamatokhoz, és gondoskodik róla, hogy az éppen szükséges lapok a fizikai memóriában legyenek. A Linux használja a virtuális tárkezelés mindkét (gyakran összekevert) fajtáját, a lapozást (paging) és a tárcserét (swapping) is. Lapozásnál folyamatoktól függetlenül, a rendszer arra ügyel, hogy a szükséges lapok a fizikai memóriában legyenek, ha azok esetleg diszken vannak, akkor gondoskodik memóriába olvasásukról, illetve ha a fizikai memória megtelt, akkor a ritkábban használt lapokat a diszkre írja. Tárcserénél pedig a rendszer figyelemmel kíséri az egyes folyamatok aktivitását is, és ha szabad memóriára van szükség, egy inaktív folyamat egészét háttértárra írja, felszabadítva ezzel a folyamat által használt összes fizikai memóriát.

A Linux a két módszer keverékét használja: amíg rendelkezésre áll elegendő memória, úgy csak egyes lapokat lapoz ki/be, de például ha úgy látja, hogy egy folyamat hosszú ideje inaktív, és nem csak egy-két lapnyi memóriára van szükség, akkor az adott folyamathoz tartozó összes fizikai lapot diszkre menti.

Linux alatt a merevlemez virtuális memóriakezelése dinamikusan, menet közben is változtatható, tehát az operációs rendszer leállítása nélkül lehetőségünk van a virtuális memória méretének megváltoztatására. A **swap** terület használható fájlként, vagy akár külön partíciónként. Akár egyszerre több swap területet is használhatunk.

### 2.2.1 Buffer Cache alapú megoldás

Szorosan kapcsolódik a memória-lapkezelés mechanizmusához a Linux **buffer cache** kezelési módszere. A buffer cache a Unix rendszerek diszk-eléréshez használt gyorsítótárja, amelyet a kernel kezel, mivel minden folyamat csak és kizárólag a kernel meghívásával végezhet diszkműveletet. A gyorsító tár célja minden mai operációs rendszerben gyakorlatilag az I/O hozzáférések gyorsítása, ezzel pedig a „felhasználói élmény" növelése. Néhány alapfogalom:

- A buffer cache mérete dinamikusan, a rendszer-terheléstől függően változik: mindig az éppen szabad fizikai memória egészét erre a célra használja.
- A diszk-írások is a buffer cache-n keresztül történnek: minden írás először a cache memóriába kerül, és vagy egy megadott idő elteltével íródik ki diszkre, vagy pedig akkor, ha a rendszer számára „elegendő" kiírnivaló összegyűlt.
- Ezért fontos az, hogy a megfelelő shutdown procedúra (a rendszer „lelövése", leállítása) végrehajtása nélkül soha ne kapcsoljuk ki a gépet.

Kikapcsolás előtt mindig szükséges a diszk tartalmának szinkronizálása a memóriában lévő állapottal, a nyitott file-ok lezárása - ezen lépések elmulasztása esetén kikapcsoláskor a diszk tartalma helytelen lehet, információk, egész file-ok veszhetnek el. Ugyanez történhet persze áramszünet vagy más hiba esetén is. Ez az az ár, amit a Unix fájlrendszer által nyújtott nagyobb teljesítményért fizetnünk kell. A Unix-ok általában (így a Linux is) rendelkeznek funkciókkal az esetleges információvesztés minimalizálására, illetve a korrekt file-rendszer visszaállítására. Tulajdonképpen egy átlagos Linux rendszerben váratlan rendszerösszeomláskor maximum csak a legutóbbi 30 másodperc munkája veszhet el - nagyon kicsi valószínűségű extrém esetektől eltekintve. De az ilyen adatvesztés veszélye minden, a diszk-írást bufferelő rendszerben fennáll (pl.: Windows).

### 2.2.2 További gyorsító megoldások

**Demand paging:** egy futtatható fájl végrehajtásakor nem az egész fájl töltődik be a memóriába, hanem mindig csak azok a lapjai, amikre a végrehajtás során éppen szükség van. Mivel minden programnak vannak olyan részei melyek csak egyszer (vagy akár egyszer sem) futnak le, ezeket a részeket vagy be sem tölti a rendszer, vagy miután lefutottak felszabadítja az általuk elfoglalt memóriaterületet.

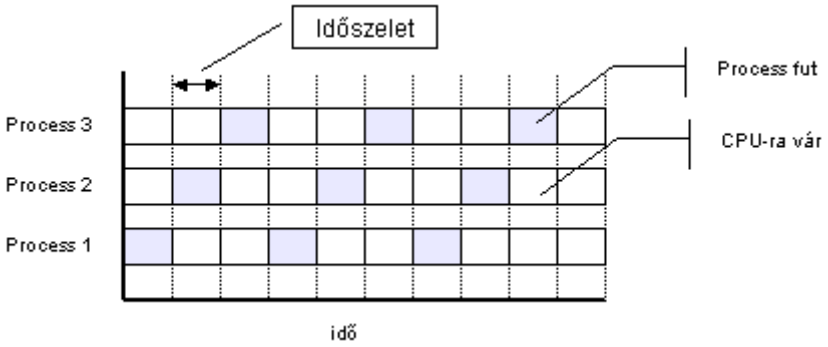
**Osztott kódkönyvtárak használatának alapelve:** a programok nagy része C nyelven íródnak, és valószínűleg sokban van olyan függvény, amely más programokban is előfordul. Ezeket felesleges lenne minden programmal a memóriába tölteni, elég egyszer, és meg kell mondani a programoknak, hogy hol keressék ezeket a függvényeket a memóriában. Ezt csinálja a dinamikus linker, amely a programokba beépített

programrészletnek segítve gondoskodik a függvények megtalálásáról, illetve a memóriába töltésükről, amennyiben még nem lennének betöltve.

**Copy-on-write mechanizmus:** új folyamat létrehozása mindig egy másik folyamat memóriájának lemásolásával történik. Mivel viszont egy memórialapra több folyamat memóriatérképéből tudunk hivatkozni, nem kell azt a lapot lemásolni, csak el kell helyezni a lapra mutató hivatkozásokat a megfelelő helyeken. Innentől kezdve csak arra kell vigyázni, hogy amikor az ugyanarra a lapra hivatkozó folyamatok közül valamelyik módosítani akarja a lapot, akkor le kell másolnunk a számára, és így már módosíthatja, mert az már csak az övé.

## 2.3 A folyamatok ütemezése

Az operációs rendszernek egy processzoron kell konkurensen több feladatot végrehajtania, ezért valamilyen formában meg kell osztania a rendelkezésre álló CPU időt az egyes folyamatok között. A Unix rendszerek (így a Linux is) a **preemptív időosztásos ütemezés** módszerét alkalmazzák, ami azt jelenti, hogy a rendelkezésre álló időt felosztja egyenlő részekre, és ezekből az egyenlő időszeletekből juttat – a folyamat prioritásának megfelelően – többet vagy kevesebbet az adott folyamatnak. Az egyes folyamatok prioritása természetesen állítható. Az ütemezés preemptív volta annyit jelent, hogy amikor az adott folyamat számára kijelölt időszelet letelt, a kernel megszakítja a folyamat futását és más folyamatnak adja át a vezérlést - nincs tehát mód arra, hogy egy folyamat a végtelenségig magánál tartsa a vezérlést, és megakadályozza a többi folyamat futását. Linuxban az ütemezés alapegysége az 1/100 másodperc.




**2.ábra.** Preemtív multitaszk

**A Unix nem valós- idejű (real-time) operációs rendszer:** ami annyit jelent, hogy ha több folyamat fut egyszerre, és az egyiktől elkerül a vezérlés, akkor valamekkora idő múlva vissza is fogja majd kapni – a két aktív (futó) állapot közti időre azonban nincs szigorú felső korlát. Az esetek 99.9999999 százalékában ez az idő (még egy leterhelt rendszeren is) pár tized másodperc – azonban soha nem mondhatjuk, hogy biztosan csak ennyi.

## 2.4 Több felhasználós működés

A Linux, mint korábban említettük, egy több felhasználós operációs rendszer. Ez azt jelenti, hogy egyidejűleg több felhasználó is használhatja ugyanazt a rendszert, és mindegyikük akár több programot is futtathat. Hogyan valósul ez meg? A Unix filozófia minden egyes bejelentkezett felhasználóhoz hozzárendel

egy-egy úgynevezett terminált: egy terminál pedig egy billentyűzet + megjelenítő egység együttesét jelenti. Az adott Unixos géphez legközvetlenebbül csatolt terminált (Linux esetén a gép saját billentyűzetét és monitorát) **konzol terminálnak** nevezzük. Ez abból a szempontból kitüntetett, hogy bizonyos rendszeradminisztrációs feladatok csak innét hajthatók végre. További terminálok csatolhatók még a géphez soros vonalon (ez a legősibb Unixos megoldás).

A hálózaton vagy grafikus felületen keresztül bejelentkezett felhasználókhoz ún. **pszeudo-terminálokat** rendel a rendszer, ahol is a billentyűzet és a képernyő annak a gépnek a billentyűzetéhez és képernyőjéhez rendelődik, amely előtt a felhasználó ül. A terminálok megnevezése a szakzsargonban **tty**, illetve a pszeudo-terminálóké **pty** vagy **ttyp**, a Linux ez utóbbi megnevezést használja. Minden Unix rendszer többféle terminált képes kezelni. Az egyes terminálokat gyártó és típusra utaló megnevezéssel azonosítják: Amennyiben csak konzolról használjuk gépünket, elég annyit tudni, hogy a konzol terminál azonosítója „console”.

### 2.4.1 Felhasználók megkülönböztetése

A Unix/Linux rendszerekben minden felhasználónak van egy **azonosító**ja, és ehhez tartozik egy **jelszó**. A finomabb hozzáférés hierarchia kialakítása érdekében a felhasználókat **csoport**okba oszthatjuk: minden felhasználónak van egy elsődleges csoportja (pl. student2009), és ezen kívül tartozhat még más csoportokhoz is. A csoportneveket konvenció szerint kisbetűvel írják.

A rendszer minden egyes felhasználóhoz egy numerikus felhasználó és (esetleg több) csoport- azonosítót rendel: **UID** – felhasználói azonosító és **GID** – csoportazonosító. Léteznek kitüntetett felhasználónevek is, illetve legalább egy, amelyik minden rendszeren megvan: ez a „**root**” felhasználó, a rendszergazda azonosítója, aki felelős az adott rendszer karbantartásáért és üzemeltetéséért, és aki a rendszerben „mindent megtehet”. Érdekesség, hogy több, a root-tal ekvivalens felhasználót is létrehozhatunk. **Az összes olyan felhasználó, akinek UID-je 0** (felhasználó létrehozásakor ezt megadhatjuk) **root-tal ekvivalens lesz**. Általában nem érdemes ezt tennünk: egy rendszeren éppen elég egy darab teljhatalmú felhasználó.

# 3. A Unix fájlrendszer és alapfogalmai

A mai operációs rendszerek szerves része a *fájlrendszerek* kezelése. A fájlrendszer röviden a háttértáron tárolt fájlok, könyvtárak és egyéb rendszerezésre szolgáló elemek (linkek, named pipe-ok és mások) tárolásának módját, kezelését és elérését értjük. Unix esetében is igaz, hogy a fájlok könyvtárakban (mappákban) helyezkednek el. Ennek oka, hogy a könyvtárak lehetővé teszik a fájlok logikus elrendezését, nem kell minden fájlt ömlesztve, egyetlen helyen tárolni. Ez a fa struktúra jellegű elrendezés növeli a teljesítményt is, hiszen egy kisebb fájlszámú könyvtárban sokkal gyorsabban megtalálható egy keresett fájl. A könyvtárak tartalmát, a tartalomjegyzéket a fájlrendszer nyilvántartja.

**A Unix/Linux-ban (túlzás nélkül) minden fájl**. Definíció szerint a file olyan „kommunikációs végpont”, ahova vagy byte-folyamot (byte stream) tudunk írni, vagy byte-folyamot tudunk onnét olvasni (esetleg mindkettőt). Fájlként kezelhető tehát a billentyűzet (csak olvasható), a szöveges képernyő (csak írható), a nyomtató, a CD írónk, de még a fizikai memória tartalma is. A gépben lévő winchester szektorait (ha nem csak egyes fájlokat akarunk elérni) is egy speciális file olvasásával illetve írásával érhetjük el, ugyanígy az

hálózati eszközöket is a file-kezelés szabályainak megfelelően használhatjuk. Ez utóbbi három esetben persze szükség van a megfelelő jogosultságokra: a fizikai memória tartalmát vagy direktben a winchester szektorait „mezei” felhasználó nem láthatja.

Legalább **háromféle** fájltípusról beszélhetünk: az **egyszerű fájlok**, a **jegyzékfájlok** (röviden jegyzékek) és a **különleges fájlok**.

**Egyszerű fájl:** bármilyen állomány, ami valamilyen adatot, szöveget tartalmaz. Egy program forráskódja, csakúgy, mint a futtatható állomány, egy szöveges dokumentum vagy az ügyféladatbázisunk szintúgy egyszerű fájl.

**Jegyzékfájlok:** lehetővé teszik, hogy a fájljainkat (mindhárom típust) valamilyen logikai rendszerbe szervezzük.

**Különleges fájlok:** ezek rendszerint olyan fájlok, amelyek valamilyen eszközt képviselnek, például a szalagos meghajtót, a terminálunkat vagy a hangkártyánkat.

## 3.2 A fájlrendszer

A Unix operációs rendszer egyetlen fájlrendszerrel dolgozik. Ez alapvetően egy fa szerkezetű struktúra, melynek van **egy gyökere**, a / nevű jegyzék. Ebben, mint minden jegyzékben a korábban tárgyalt háromféle típusú bejegyzések találhatók. Minden egyes jegyzék további bejegyzéseket és így további jegyzékeket tartalmazhat, továbbá biztosan tartalmazza a saját magára mutató **.** és a szülőjére mutató **..** bejegyzést. A gyökérben a .. is saját magára mutat.

Egy jegyzékben minden bejegyzésnek külön névvel kell rendelkeznie, akkor is, ha egyikük sima fájl, másikuk újabb jegyzékbejegyzés. Bejegyzés neve az ASCII 0-ás és a / jel kivételével tetszőleges karaktert tartalmazhat. Nem lehet nullahosszúságú, és hosszának maximumát különböző Unix rendszerek másképp rögzítik. Minden normális Unix-ban legalább 14 karakteres lehet a bejegyzés neve, de többnyire 255 karakter megengedett. Az e fölötti karaktereket szó nélkül levágja, figyelmen kívül hagyja a rendszer.

Néhány további alapfogalom:

**Metaadatok:** a fájlrendszer belső adatstruktúrája, amely biztosítja az adatok megfelelő szervezését és elérését a lemezen. Ezek alapvetően „adatokról szóló adatok”. Szinte minden fájlrendszernek saját metaadat-szerkezete van, ami szerepet játszik abban, hogy a fájlrendszerek eltérő teljesítményjellemzőkkel rendelkeznek. Nagyon fontos, hogy a metaadatok ne sérüljenek, mivel ellenkező esetben a fájlrendszerben tárolt adatok elérhetetlenné válhatnak.

**Inode:** az inode-ok az egyes fájlokkal kapcsolatos információt tartalmazzák: a méretet, a láncok számát, a mutatókat azon lemezblokkokra, amelyek a fájltartalmat ténylegesen tárolják, valamint a létrehozás, módosítás és hozzáférés dátumát és idejét.

**Napló:** a fájlrendszerekkel kapcsolatban naplónak nevezzük azt a belső struktúrát a lemezen, amelyben a fájlrendszer tárolja a fájlrendszer metaadatainak módosításait. A naplózás (journaling) lényegesen csökkenti a Linux-rendszer összeomlását követő helyreállás idejét, mivel feleslegessé teszi a korábbi hosszú keresési folyamatot, amely a rendszerindításnál végigvizsgálta a teljes fájlrendszert. Ehelyett csak a naplóban rögzített események kerülnek újra végrehajtásra.

### 3.2.1 Standard jegyzékszerkezet

A Unix-ok jegyzékszerkezete sok hasonlóságot mutat. Az alábbi jegyzékek több disztribúcióban jellemzőek, ezek alkotják a Unix-os fájlrendszerek gerincvázát.

**/bin:** Itt találhatók a létfontosságú bináris programok.

**/dev:** A speciális eszközök lelőhelye.

**/etc:** Mindenféle vegyes dolgot tartalmaz, például egyes programok konfigurációs fájljait, a jelszóállomány(oka)t, stb.

**/etc/rc.d:** A rendszer indulását és leállását irányító szkriptek vannak itt.

**/etc/skel:** Az itt lévő fájlokat kapja meg minden új ember a home jegyzékébe.

**/home:** Rendszerint itt vannak a felhasználók saját jegyzékei.

**/lib:** A legtöbb program futásához nélkülözhetetlen dinamikusan linkelhető könyvtárak vannak itt.

**/proc:** Linuxokra jellemző, a processzekkel kapcsolatos információkat hordozó virtuális fájlrendszer.

**/root:** A rendszergazda home jegyzéke.

**/sbin:** A rendszergazda számára alapvető fontosságú bináris programok jegyzéke.

**/tmp:** Ideiglenes fájlok tárolására szolgáló jegyzék. Mindenki írhat bele, a rendszergazda pedig egy programocskával mondjuk naponta törli a tíz napnál régebbi fájlokat.

**/usr:** Ez a könyvtár teszi ki a használt lemezterület nagyságrendileg 80-90%-át.

**/usr/bin:** Bináris programok, melyek nélkül végszükség esetén is létezni lehet.

**/usr/doc:** Dokumentációk.

**/usr/games:** Mi is lehet ez?

**/usr/info:** Info oldalak.

**/usr/lib:** Mint a /lib.

**/usr/local:** Az adott szerverre speciálisan jellemző dolgokat tartalmazza. Neki is van bin, lib, man, sbin, src és még sok-sok alkönyvtára.

**/usr/man:** Kézikönyv oldalak.

**/usr/sbin:** Mint /sbin, csak kevésbé fontosak.

**/usr/src:** Forráskódok.

**/var:** Sűrűn változó dolgok otthona.

**/var/catman:** Megformázott kézikönyv oldalak.

**/var/log:** Bizonyos, főleg hálózattal kapcsolatos programok logfájljai (naplói).

**/var/spool:** Várakozási sorok, például elküldendő levelek vagy elvégzendő nyomtatások feljegyzéseit tartalmazza. Az érkező levelek is sok Unixban itt vannak.

#### *3.2.1.1 A /dev jegyzék*

A /dev (devices - eszközök) aljegyzékben találhatók a rendszer- erőforrásokat reprezentáló speciális file-ok. Ezek teremtik meg a kapcsolatot a kernel „device driver”-nek (eszközmeghajtó) nevezett, az egyes fizikai eszközök kezeléséért felelős komponensei és a rendszer egyéb részei között. Kétféle eszközmeghajtót különböztetünk meg: a **karakteres** ("c") és a **blokkos** ("b") típusút, annak megfelelően, hogy az általa reprezentált eszköz milyen szervezésű. Egy winchester, vagy floppy eszközmeghajtója blokkos, míg például egy soros vonalat, vagy terminált reprezentáló file-hoz tartozó eszközmeghajtó karakteres típusú.

Lássuk röviden a főbb file-csoportokat a /dev alatt, illetve egy-két példán keresztül azt, hogyan használhatjuk őket:

**/dev/audio:** Ha valamilyen hangkártya vagy más zajkeltő szerkezet van a kernelbe konfigurálva, akkor a .au formátumú file-okat ide kiírva meghallgathatjuk őket. Példa: "cat x.au >/dev/audio " Ha a hangkártya digitalizálásra is képes, ez a file olvasható is.
**/dev/cdrom:** Ez általában egy link a bonyolultabb nevű, valódi CD-ROM speciális file-ra. Hasonlóan használható, mint egy winchester speciális file.
**/dev/cua*:** A soros vonala(ka)t jelentő speciális file-ok. Írásuk vagy olvasásuk küldést/vételt jelent a megfelelő vonalon.
**/dev/fd*:** A floppy diszkeket reprezentálják. A fd0 kezdetű file-ok az A floppyra vonatkoznak, a fd1 kezdetűek a B-re.
**/dev/hd*:** A rendszerben lévő AT buszos winchesterek: a /dev/hda az első winchestert jelenti, /dev/hdb a másodikat. Ha a gépben két IDE vezérlő vagy egy EIDE vezérlő van, akkor a többi diszkhez a /dev/hd1[ab] néven férhetünk hozzá. Ha ezeket a file-neveket számokkal folytatjuk, az egyes diszkeken lévő partíciókhoz jutunk, például /dev/hda1,/dev/hda2.
**/dev/midi, mixer:** Hangkártyához tartozó file-ok, a /dev/midi értelemszerűen midi file-ok kezelésére.
**/dev/mouse,modem:** Általában ezek linkek valamely soros portra.
**/dev/pty*:** pszeudo-terminál vonalakat reprezentáló speciális file-ok.
**/dev/sd*:** SCSI diszkek.
**/dev/tty*:** A (virtuális) konzol terminálvonalai.
**/dev/ttyS*:** Soros vonali terminálok.
**/dev/null:** Ez egy igen érdekes file: minden beleírt adatot elnyel, és olvasáskor mindig filevége-jelet ad. Akkor hasznos, ha egy parancs kimenetét el akarjuk nyomni. Például, ha nem akarjuk a hibaüzeneteket látni: "parancs 2>/dev/null".
**/dev/zero:** Az előzőhöz hasonló file, azzal a különbséggel, hogy olvasáskor végtelen sok 0 értékű byte-ot ad vissza. A "dd 10kfile bs=1k count=10" utasítással például létrehozhatunk egy 10Kbyte hosszú, 10kfile nevű csupa 0-ból álló file-t.

### 3.2.2 Fájlokról részletesebben

A könyvtárak tartalmát, a tartalomjegyzéket az **ls** parancs használatával lehet (az ls a list szó rövidítése) megtekinteni. Az ls paraméterek nélkül az aktuális könyvtár fájljainak és könyvtárainak nevét mutatja meg:

```
[voxel@orion ~]$ cd /etc
[voxel@orion etc]$ ls
acpi                                mke2fs.conf
adjtime                             mkinitcpio.conf
alsa                                mkinitcpio.conf.pacnew
anacrontab                          mkinitcpio.d
apparmor                            ModemManager
apparmor.d                          modprobe.d
arch-release                        modules-load.d
audit                               mono
avahi                               mpv
bash.bash_logout                    mtab
bash.bashrc                         mtools.conf
bindresvport.blacklist              my.cnf
binfmt.d                            my.cnf.d
bluetooth                           nanorc
…
```

A tartalomjegyzéknek ez a formája nem sok információt ad, ezért a gyakorlatban nem ezt, hanem egy részletesebb megjelenítést szoktunk használni. Ehhez az **ls** parancsot a **-l** kapcsolóval egészítjük ki. A **-l** hatására a parancs az egyes fájlok és könyvtárak részletes adatait is megjeleníti, ún. hosszú listát ad (long, ezért l a kapcsoló neve).

```
(base) [voxel@orion etc]$ ls -al
drwxr-xr-x 121 root root  12288 szept  6 08.13 .
drwxr-xr-x  18 root root   4096 júl    3 16.39 ..
drwxr-xr-x   3 root root   4096 2021 okt    2 acpi
-rw-r--r--   1 root root     46 2021 máj   15 adjtime
drwxr-xr-x   3 root root   4096 2021 máj    6 alsa
-rw-r--r--   1 root root    541 ápr   25 14.53 anacrontab
drwxr-xr-x   2 root root   4096 aug   26 21.01 apparmor
drwxr-xr-x   8 root root   4096 aug   26 21.02 apparmor.d
drwxr-xr-x   3 root root   4096 máj   18 14.44 audit
drwxr-xr-x   3 root root   4096 2022 jan    8 avahi
-rw-r--r--   1 root root     28 2022 jan    9 bash.bash_logout
-rw-r--r--   1 root root    618 2022 jan   14 bash.bashrc
…
```

Nézzük meg részleteiben az oszlopok jelentését:

```
-rw-r--r--   1 root root    618 2022 jan   14 bash.bashrc
-----------   - ----  ----    ---- -------------  ------------
|     |       |   |    |       |       |              |
1     2       3   4    5       6       7              8
```

**Csoport 1:** A sor első karaktere a fenti példában **-** vagy **d**, ez alapján tudható, hogy az adott sor egy fájlt vagy egy könyvtárat ír le. Ha az első karakter **d**, akkor ez egy könyvtár (directory), ha egy **-** karakter áll itt, akkor a sor egy fájlt ír le.
**Csoport 2:** Az első karakter után álló kilenc karakter a bejegyzéshez tartozó jogosultságokat írja le.
**Csoport 2:** A második oszlopban látható szám az ún. linkszám.
**Csoport 3:** A második oszlopban látható szám az ún. linkszám.
**Csoport 4-5:** a fájl tulajdonosa és csoportja.
**Csoport 6:** a bejegyzés mérete. Fájlok esetében könnyen értelmezhető, a fájl hosszát jelenti bájtban. A könyvtárak is foglalnak helyet a diszken, így ezeknek is van méretük, ez olvasható itt le.
**Csoport 7:** dátumot és időt ír le, ez a fájl vagy könyvtár utolsó módosításának dátuma és ideje.
**Csoport 8:** az utolsó oszlop a fájl vagy könyvtár neve.

### 3.2.3 Helyettesítő karakterek

A fájlrendszeren végzett munka során alkalmazhatunk speciális karaktereket. Ezeknek a célja az, hogy megvalósítható legyen több elemre való együttes hivatkozás, amely nagy segítséget nyújt a gyakorlati feladatok megoldása során. Például szeretnénk egy könyvtárból minden jpg fájlt kimásolni, de a png-ket pedig ott hagyni. Természetes igény tehát az, hogy a másolást végző parancsot lehessen úgy paraméterezni, hogy az minden jpg kiterjesztésű fájl másoljon csak.

**Speciális karakterek:**

A * több karaktert helyettesít. Így a foto*.jpg minden olyan bejegyzésnek megfelel, amelynek az első négy karaktere foto, majd bármi jöhet, de végül .jpg kell záródjon a fájl neve. De a * használható egy név

belsejében is: a *.j*gp jelentése. A fenti példánál maradva lehetnek jpeg kiterjesztésű fájlok is, nem csak jpg Ez a minta ezeket is a másolandók közé fogja helyezni.

A ? pontosan egy karaktert helyettesít azon a helyen, ahová azt írjuk. A foto_2022_07_0?.jpg azokat a képeket jelenti, amelyek az első 2022.07 hónap első 9 napjában készültek.

A [] karakterek között több karaktert is felsorolhatunk. Az adott pozíción egyetlen karakter szerepelhet, az, amelyet a zárójelek közé írunk. Így a p[aA]ssword két névre, a password-re és a pAssword-re illeszkedik. Szögletes zárójelpár esetén tagadást is használhatsz a zárójelbe írt ^ karakterrel. A p[^aA]ssword hatása épp az ellenkezője lesz az előzőnek, azt jelenti, hogy a második pozíción nem szerepelhet a betű függetlenül attól, hogy azt kis- vagy nagybetűvel írták.

## 3.2.4 Fontosabb parancsok

### *3.2.4.1 pwd*

A pwd (Print Working Directory) az aktuális könyvtár pontos elhelyezkedését írja ki. Az aktuális könyvtár (vagy munkakönyvtár) az, amelyikben éppen dolgozunk. Minden kiadott parancs is ezt használja. A pwd parancsot nem sokszor használjuk, mert a prompt gyakran úgy van beállítva, hogy mutassa ezt. Jelen példában ez éppen nem így van:

```
[voxel@orion ~]$ pwd
/home/voxel
```

### *3.2.4.2 cd*

A **cd** (Change Directory) paranccsal mozoghatunk a könyvtárrendszerben. Célja az aktuális könyvtár megváltoztatása. A cd parancs egy rendszerbe integrált parancs, azaz nincs szükség külső programra vagy alkalmazásra, mivel azt közvetlenül a Linux Shell hajtja végre. A parancsnak egy kötelező paramétere van, a *könyvtár neve*, amit aktuálissá akarsz tenni (úgy is mondjuk, hogy amelyikbe be akarsz lépni). A könyvtár leírását vagy az aktuális könyvtárhoz képest adjuk meg, vagy a gyökérkönyvtárhoz viszonyítva. Az előbbit *relatív* (hiszen csak az aktuális könyvtárban érvényes), az utóbbit *abszolút* elérési útnak nevezik. Az elérési útban néhány speciális jelölés is használható speciális könyvtárak jelölésére. Lássunk néhány példát!

- A **cd games** parancs az aktuális könyvtárból nyíló *games* könyvtárba lép. Ez a relatív elérési út, mivel az aktuális könyvtárból indul ki.
- A **cd ..** az aktuális könyvtárból egy szinttel feljebb lép. Általában minden operációs rendszerben egy **..** nevű könyvtár, amely a könyvtár keletkezésekor automatikusan létrejött. Ezt a rendszer hozza létre úgy, hogy az a szülő könyvtárra mutat. A **cd ..** hatására ezért lépünk vissza a szülő könyvtárba.
- A **cd /usr/local/lib** parancs a gyökérkönyvtárból nyíló usr -> local -> lib könyvtárat teszi aktuálissá. Ez az abszolút elérési út. Akármilyen könyvtárban is vagyunk, ez működni fog.
- A **cd ~** paranccsal a saját home könyvtárunkba léphetünk bele.
- A **cd ~<felhasználó>** a **felhasználó** home könyvtárába lép. A **cd ~balboa** eredményeként tehát a balboa home könyvtára lesz az aktuális könyvtár, amely általában: /home/balboa
- A fentiek tetszőlegesen kombinálhatók, tehát ennek is van értelme: **cd ~balboa/kepek/teszt**.

A Unix/Linux rendszerekben a gépelés során a Tab használata sokat segít. A Tab által használható név kiegészítő rendszer nagyon jól működik és használható.

### *3.2.4.3* tree

A **tree** parancs nem része minden Unix rendszernek, külön telepíthető fel. A könyvtárakkal történő munka során néha jól jön, mert vizuálisan képes megjeleníteni a könyvtárrendszert az aktuális, vagy meghatározott ponttól. Paraméterei:

- Megadhatod azt a könyvtárat, amelyből kiindulva szeretnéd a könyvtársatruktúrát megjeleníteni pl. így: **tree /etc**.
- A **-d** hatására a **tree** csak a könyvtárakat rajzolja ki, az abban levő fájlokat nem. Alapesetben a fájlok is megjelennek, így a **tree /** hatására a rendszer összes könyvtára és az abban levő összes fájl megjelenítésre kerül. Példa:

```
[voxel@orion wildfly-26.0.1.Final]$ tree -d
```

```
├── appclient
│   └── configuration
├── bin
│   └── client
├── docs
│   ├── contrib
│   │   └── scripts
│   │       ├── init.d
│   │       ├── service
│   │       │   └── amd64
│   │       └── systemd
│   ├── examples
│   │   └── configs
│   ├── licenses
│   └── schema
├── domain
│   ├── configuration
│   └── tmp
│       └── auth
├── modules
│   └── system
│       └── layers
│           └── base
│               ├── asm
```

- A **-f** kapcsoló hatására a mappa tartalmát teljes prefix-el mutatja a program:

```
[voxel@orion wildfly-26.0.1.Final]$ tree -f
.
├── ./appclient
│   └── ./appclient/configuration
│       ├── ./appclient/configuration/appclient.xml
│       └── ./appclient/configuration/logging.properties
├── ./bin
│   ├── ./bin/add-user.bat
│   ├── ./bin/add-user.properties
│   ├── ./bin/add-user.ps1
│   ├── ./bin/add-user.sh
│   ├── ./bin/appclient.bat
│   ├── ./bin/appclient.conf
│   ├── ./bin/appclient.conf.bat
│   ├── ./bin/appclient.conf.ps1
│   ├── ./bin/appclient.ps1
│   ├── ./bin/appclient.sh
│   ├── ./bin/client
│   │   ├── ./bin/client/jboss-cli-client.jar
│   │   ├── ./bin/client/jboss-client.jar
│   │   ├── ./bin/client/README-CLI-JCONSOLE.txt
│   │   └── ./bin/client/README-EJB-JMS.txt
│   ├── ./bin/common.bat
```

```
│   ├── ./bin/common.ps1
│   ├── ./bin/common.sh
```

#### *3.2.4.4* mkdir

Új könyvtár létrehozására az **mkdir** (Make Directory) való. Paraméterként a létrehozandó könyvtár nevét kell megadni. Így az **mkdir Games** elkészít egy új **Games** nevű könyvtárat az aktuális könyvtárban. Az mkdir esetében egymás után akár több mappanevet is megadhatunk. Az **mkdir** Tetris Pong SpaceInvaders kiadásakor mindhárom könyvtár létrejön az aktuális könyvtárban.

Hibaüzenetet kapunk, ha olyan könyvtárat szeretnénk létrehozni, amelynek a szülő könyvtára nem létezik, pl. a **mkdir Games/Tetris** csak akkor működik, ha a **Games** könyvtár már létezik.

#### *3.2.4.5 r*mdir

Egy vagy több könyvtár törlésére szolgál. Paramétere a törlendő mappa neve, melynek üresnek kell lennie, tehát nem tartalmazhat sem fájlt, sem újabb könyvtárat. Ha vannak ilyenek, először azokat kell törölni, vagy az **rm** parancsot kell használni helyette. Pl.:

```
$ rmdir Games/Tetris
```

#### *3.2.4.6 r*m

A törlések elvégzésére egy hatékony parancs az **rm**. Ez eredetileg fájlok törlésére szolgált, de a -r paraméterrel könyvtárak rekurzív törlésére használható akkor is, ha az nem üres. Pl.:

```
$ rm -r Games/Tetris
```

A Tetris mappából minden fájl és könyvtár eltűnik. Az utasítás rekurzívan működik.

#### *3.2.4.7 mv*

Fájlok, mappák mozgatására, áthelyezésére szolgál. De az mv parancs használandó akkor is, ha egy állományt szeretnénk átnevezni. Az, hogy melyik műveletet szeretnénk végrehajtani, a paraméterek megadásától függ. Példa:

```
$ mv Tetris MyTetris
```

A parancsnak kétféle kimenetele lehetséges. Amennyiben a MyTetris mappa nem létezik, akkor a Tetris mappa átneveződik MyTetris-re. Amennyiben létezik, úgy Tetris mappa áthelyezésre kerül a MyTetris mappán belülre.

#### *3.2.4.5 cp*

Állományok másolására szolgál. Mappákat is másolhatunk vele, ekkor paraméterként meg kell adni a forrás és a célkönyvtárat is. Példák:

```
$ cp ./alma.txt korte.txt
```

Ezzel az alma.txt új neve korte.txt lett. Mappák esetében:

```
$ cp -r ./Tetris MyTetris
```

A parancs hatására a Tetris mappa tartalma átmásolódik a MyTetris mappába. A **-r** kapcsolóra azért van szükség, hogy a **cp** számára jelezzük, a target teljes tartalmát, annak minden könyvtárát és fájlját is másolni kell.

### 3.2.5 Linux fő fájlrendszerei

Korábbi Unix/Linux rendszerek esetében a fájlrendszerek támogatottsága nem volt túl nagy. A 2.4 és újabb Linux kernelek esetén azonban már egész sokféle fájlrendszer közül lehet választani. Fontos megjegyezni, hogy nincs olyan fájlrendszer, amely tökéletesen megfelelne mindenféle alkalmazáshoz. Minden fájlrendszernek vannak erősségei és gyengéi, amelyeket figyelembe kell venni.

Alapvetően kétféle fájlrendszerről beszélhetünk: **naplózó** (**journaling**) és **nem naplózó** fájlrendszer.

#### *3.2.5.1 ReiserFS*

A ReiserFS-t Hans Reiser és a Namesys fejlesztőcsapat tervezte. Legfontosabb előnyei:

**Jobb lemezterület-kihasználás:** a ReiserFS fájlrendszerben az összes adat egy kiegyensúlyozott B*-fastruktúrába van szervezve. A fastruktúra jobban ki tudja használni a lemezterületet, mivel a kis fájlok közvetlenül a B*-fa levélcsomópontjaiban kerülnek tárolásra, nem pedig egy másik helyen és csak egy mutató mutat a tényleges tárolási helyre. Ezen felül a tárterület nem 1 vagy 4 kilobájtos egységekben kerül lefoglalásra, hanem az adatok pontosan a szükséges méretet foglalják el. Másik előnye az inode-ok dinamikus lefoglalása. Így a rendszer rendkívül rugalmas, szemben például az Ext2-vel, ahol az inode-ok sűrűségét a fájlrendszer létrehozásakor kell megadnunk.

**Jobb lemezhozzáférési teljesítmény:** kis fájlok esetén az adatok és a „stat_data” (inode) információ általában egymás mellett kerül tárolásra. Ez az információ egyetlen lemez I/O-művelettel kiolvasható, azaz csak egy lemezhozzáférés szükséges a kívánt információ lekéréséhez.

**Gyorsabb helyreállás összeomlás után:** a legutolsó metaadat-módosításokat nyomkövető napló segítségével a fájlrendszer ellenőrzése nagyon nagy fájlrendszerek esetén is csak néhány másodpercet vesz igénybe.

#### *3.2.5.2 Ext2*

Az Ext2 eredete a Linux történetének első napjaira nyúlik vissza. Az eredeti **Extended File System** 1992 áprilisában készült el és lett beépítve a Linux 0.96c-be. Azóta számos módosításon ment keresztül és Ext2 néven évekig a legnépszerűbb Linux-fájlrendszer volt. A rendkívül rövid helyreállítási idejű naplózó fájlrendszerek megszületése miatt azonban az Ext2 mára sokat vesztett fontosságából.

**Megbízhatóság:** az Ext2 számos javításon és komoly tesztelésen ment keresztül. Ez lehet annak az oka, amiért az emberek gyakran sziklaszilárdnak nevezik. Rendszerkimaradás után, amikor a fájlrendszer nem szabályosan lett lecsatolva, az e2fsck elkezdi elemezni a fájlrendszer adatait. A metaadatok konzisztens állapotba kerülnek és a függőben lévő fájlok vagy adatblokkok egy kijelölt jegyzékbe íródnak (ennek neve lost+found). A naplózó fájlrendszerrel ellentétben az e2fsck a teljes fájlrendszert végigvizsgálja, nemcsak az utoljára módosított metaadatbiteket. Ez lényegesen tovább tart, mint a naplózó fájlrendszer naplóadatainak ellenőrzése. Éppen ezért magas rendelkezésre állást igénylő kiszolgálóhoz nem ajánlatos Ext2 fájlrendszert

választani. Mivel azonban az Ext2 nem tart karban naplót és lényegesen kevesebb memóriát használ, néha gyorsabb a többi fájlrendszernél.

#### *3.2.5.3 Ext3*

Szemben a többi új generációs fájlrendszerrel, az Ext3 nem vadonatúj tervezési elvekre épül, hanem az Ext2-re. A két fájlrendszer szorosan kapcsolódik egymáshoz. Az Ext3 fájlrendszer egyszerűen ráépíthető egy Ext2 fájlrendszerre. A legfontosabb különbség az Ext2 és Ext3 között, hogy az Ext3 támogatja a naplózást. Ext3-nak több nagy előnye van:

**Egyszerű és nagyon megbízható frissítés Ext2-fájlrendszerekről:** Mivel az Ext3 az Ext2 kódjára épül, valamint a lemezen lévő formátum és a metaadatok formátuma is egységes, az Ext2-ről Ext3-ra frissítés hihetetlenül egyszerű. A más naplózó fájlrendszerekre – például ReiserFS, JFS vagy XFS fájlrendszerre – való áttéréssel szemben, ami ugyancsak körülményes lehet, az Ext3-ra áttérés néhány perc alatt végrehajtható. Biztonságos is, mivel elképzelhető, hogy a teljes fájlrendszer újbóli létrehozása nem működik hibátlanul.

**Megbízhatóság és teljesítmény:** több más naplózó fájlrendszer „csak metaadatokat" naplóz. Ez azt jelenti, hogy a metaadatok mindig konzisztens állapotban maradnak, de a fájlrendszerben tárolt adatokra ugyanez nem feltétlenül igaz. Az Ext3 a metaadatokra és az adatokra is vigyáz. E „törődés" mértéke pedig szabályozható (**data=journal, data=ordered, data=writeback módok**). Ha a rendszergazda nem állítja át, akkor az Ext3 alapértelmezés szerint data=ordered módban működik.

#### *3.2.5.4 XFS*

Az eredetileg az IRIX operációs rendszerhez tervezett XFS fejlesztését az SGI az 1990-es évek elején kezdte meg. Az XFS mögötti elképzelés egy olyan nagy teljesítményű 64 bites naplózó fájlrendszer létrehozása volt, amely a mai extrém feldolgozási igényeknek is megfelel. Az XFS kiválóan kezeli a nagy fájlokat és jól működik csúcsminőségű hardveren is. Azonban még az XFS-nek is van hátránya. A ReiserFS-hez hasonlóan az XFS is nagy gondot fordít a metaadatok integritására, de az adatok integritására már kevesebbet.

**Kiváló méretezhetőség allokációs csoportok használatával:** az XFS fájlrendszer létrehozásakor a fájlrendszer alapjául szolgáló blokkeszköz nyolc vagy több egyenlő méretű lineáris részre van osztva. Ezeket allokációs csoportoknak hívjuk. Minden allokációs csoport maga kezeli a saját inode-jait és szabad lemezterületét. Mivel az allokációs csoportok egymástól függetlenek, a kernel egyszerre többet is megcímezhet. Ez a funkció a lelke az XFS jó méretezhetőségének.

**Nagy teljesítmény a lemezterület hatékony kezelésével:** a szabad területet és inode-okat az allokációs csoportokon belül B+-fák kezelik. A B+-fák használata nagyban hozzájárul az XFS jó teljesítményéhez és méretezhetőségéhez. Az XFS késleltetett lefoglalást használ.

#### *3.2.5.4 Egyéb, széles körben támogatott fájlrendszerek*

| Fájlrendszer | Jellemző |
|---|---|
| **cramfs** | Tömörített ROM fájlrendszer: Tömörített, csak olvasható fájlrendszer ROM-okhoz. |
| **hpfs** | Nagy teljesítményű fájlrendszer: Az IBM OS/2 szabványos fájlrendszere – csak írásvédett módban támogatott. |
| **iso9660** | A CD-ROM-okon használt szabványos fájlrendszer. |

| | |
|---|---|
| **minix** | Ez a fájlrendszer elvileg egyetemi operációsrendszer-projektekből származik, és ez volt a Linuxon használt első fájlrendszer. Manapság hajlékonylemezek fájlrendszereként használják. |
| **msdos** | A fat fájlrendszert eredetileg DOS alatt használták. Ma már más operációs rendszerek is alkalmazzák. |
| **ncpfs** | A Novell-kötetek hálózaton keresztüli felkapcsolására szolgáló fájlrendszer. |
| **nfs** | Hálózati fájlrendszer (Network File System): Az adatok a hálózat bármely gépén tárolhatók és a hálózaton keresztül elérhetővé tehetők. |
| **smbfs** | A Server Message Block protokollt elsősorban a Windows különböző verziói használják a hálózaton keresztüli fájlelérés megvalósításához. |
| **sysv** | SCO UNIX, Xenix és Coherent (kereskedelmi UNIX- rendszerek PC-khez) alatt használt fájlrendszer. |
| **ufs** | BSD, SunOS és NeXTstep alatt használt fájlrendszer. Csak írásvédett módban támogatott. |
| **umsdos** | UNIX MSDOS-os: (UNIX on MSDOS): A szokványos fat fájlrendszert bővíti ki: speciális fájlok létrehozásával eléri a UNIX funkcionalitását (jogosultságok, láncok, hosszú fájlnevek). |
| **vfat** | Virtuális FAT (Virtual FAT): A fat fájlrendszer kiterjesztése (támogatja a hosszú fájlneveket). |
| **ntfs** | Windows NT fájlrendszer, csak olvasható |

### 3.2.6 Linkek

Sok operációs rendszerben egy az egyes összerendelés van az állományok és az állománynevek között. Minden állománynak egy neve van és minden állománynév egy állományt jelöl. A Unix a nagyobb rugalmasság érdekében szakított ezzel a koncepcióval. Az állomány egyedi azonosítója az inode-ja. Ez tartalmaz minden információt az állományról (jogok, méret, hivatkozások száma). A katalógusok csak egy fájlnév – inode összerendelést tartalmaznak, ezért lehetséges, hogy egy katalógusból több néven, vagy különböző katalógusokból hivatkozzunk ugyanarra az inode-ra. Ezt a Unix-ban linknek nevezik.

Amikor különböző névvel hivatkozunk az inode-ra, **azt hard link**nek nevezik. Ebben az esetben az állomány addig létezik, ameddig van rá mutató bejegyzés. Hard linket csak egy köteten (partíció, eszköz) belül hozhatunk létre.

Azonban lehetséges úgynevezett **szimbolikus link**ek létrehozása is. Ilyenkor különböző nevekkel hivatkozhatunk egy katalógus bejegyzésre. Ilyen módon különböző kötetek között is létrehozhatunk linkeket. Mindkét linket az *ln* paranccsal lehet létrehozni.

**Soft link:**

```
#  ln -s ./alma ./körte
```

Ekkor a bármely olyan kérés, amely az „*alma*” jegyzékre vonatkozik, az a „*körte*” jegyzékre fog vonatkozni.

**Hard link:**

```
#  ln  ./alma.txt ./körte.txt
```

### 3.2.7 A /proc fájlrendszer

A proc állományrendszer **nem valódi állományrendszer**, a Linux rendszermag által szimulált jegyzékbejegyzéseket és adatokat tartalmazza. Mindazon állományok és jegyzékek tehát, amelyek a proc állományrendszer beillesztésével válnak elérhetővé, nem a merevlemezen vagy a számítógépbe beépített egyéb háttértáron tárolódnak, hanem csak a memóriában léteznek, és csak akkor, amikor éppen használjuk őket. Jól mutatja ezt az is, a jegyzék tartalmát egy ls -l paranccsal kilistázzuk, akkor nulla lesz a mérete.

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

Mik találhatók a /proc fájlrendszeren? Egy példa:

```
# cd /proc
# ls
1 10102 10105 10311 10315 10702 10710 10712 10714 10715 10716 10717 10718 10719 10720 10721 10722 10723
10755 10766 1145 1146 1147 1148 1149 1150 1151 1152 2 3 330 345 346 355 369 4 420 429 443 446 447 449 450
461 475 490 5 508 528 583 6 612 626 675 715 716 717 718 719 720 723 745 752 753 757 771 773 775 776 7922 808
811 8784 8786 8788 8789 8936 9008 9088 911 914 9957 apm bus cmdline cpuinfo devices dma fb file sy st ems fs ide
interrupts ioports kcore kmsg ksyms loadavg locks mdstat meminfo mise modul es mounts net partitions pci rtc scsi
self slabinfo sound stat swaps sys tty uptime version
```

Láthatjuk, hogy mindenféle információt találunk itt a rendszerünkről. A processzorról, a memóriákról, a megszakításokról, stb. Például a ***version*** fájl tartalma egy teszt gépen:

```
Linux version 2.6.19-1.2895.fc6 (brewbuilder@hs20-bc1-5.build.redhat.com) (gcc version 4.1.1 20070105 (Red Hat
4.1.1-51)) #1 SMP Wed Jan 10 18:32:37 EST 2007
```

Az egyik fájl (**kcore**) mérete azonban nem nulla, hanem egészen nagy. Ez a rendszermag egyfajta tükre. Látható, hogy a fájl nagyjából akkora, amekkora fizikai memóriával rendelkezünk. Ez nem véletlen egybeesés: a kcore a rendszer memóriája, a RAM, minden megtalálható itt, ami az adott pillanatban a fizikai memóriában van, méghozzá folyamatosan frissülve.

### 3.2.8 Fájlrendszer létrehozása

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

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

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

A munka során különféle parancsokkal módosíthatjuk a merevlemez lemezrészeinek szerkezetét, és ha úgy gondoljuk, hogy elértük a célunkat, kiírhatjuk az eredményt a merevlemezre. Az fdisk csak a mentéskor változtatja a merevlemez tartalmát. Ha valamit elrontottunk, bármikor kiléphetünk a változtatások elvetésével, megőrizve az eredeti elrendezést. A lemezrészek elrendezésének mentése után nincs szükség a számítógép újraindítására, a módosítások azonnal életbe lépnek, az új lemezrészek használhatók.

**Mikre képes az fdisk?**

Új partíció létrehozása, létező törlése, partíciók listázása, partíciós tábla ellenőrzése, stb. Amennyiben csak információt szeretnénk kérni arról, hogy milyen partícióink vannak, úgy azt a következőképpen tehetjük meg:

```
# fdisk -l
Device     Boot  Start   End     Blocks     Id  System
/dev/hda1  *        1    940    1895008+   83  Linux
/dev/hda2           941  1023    167328     5  Extended
/dev/hda5           941  1001    122944+   82  Linux swap
/dev/hda6          1002  1023     44320+   83  Linux
```

**Fájlrendszer tényleges létrehozása**

Miután a partíciók terén rendbetettünk mindent, a fájlrendszer létre is kell hoznunk. A Linux alatt az ***mkfs*** parancs használatos erre a célra:

```
mkfs –t <fájlrendszer típus> <eszköz>
```

Példa egy ext3 fájlrendszer létrehozására:

```
#  mkfs –t ext3 /dev/sda7
```

## 3.3 A csereterület

A csereterület *(swap space)* nem valódi állományrendszer, állományok tárolására nem alkalmas. A csereterület a számítógépbe épített operatív memória tehermentesítésére, kiterjesztésére használható. A Linux/Unix rendszermag képes arra, hogy a memóriának az éppen nem használt területeit a csereterületre mentse, és így memóriát szabadítson fel. A rendszermag természetesen automatikusan visszatölti a mentett memóriatartalmat, amint arra szükség van. Mivel a mentést és visszatöltést a Linux automatikusan elvégzi, semmiféle feladat nem hárul a rendszeren futó programokra vagy a rendszert használó felhasználókra.

Szokás látszólagos memóriának (*virtual memory)* is nevezni. Ez a memória nagyságrendekkel lassabban működik, mint a számítógép operatív memóriája. Amíg ritkán kell az adatokat a csereterület és a memória közt mozgatni, a sebességkülönbség nem lassítja jelentősen a számítógép működését. Ha azonban a csereterületre másolás és az onnan való visszatöltés sűrűn jelentkezik, a számítógép működése jelentősen lassul.

A csereterület mérete dinamikusan állítható. Akár ki is kapcsolható. Méretére nincs egyértelmű szabály. Nem szerencsés, ha a csereterület nagysága meghaladja az operatív memória nagyságának háromszorosát. A legtöbb rendszeren az operatív memória nagyságának kétszeresénél nem nagyobb csereterületet használunk.

Ha meg akarjuk tudni, hogy a Linux mennyi csereterületet használ, a /proc/swaps látszólagos állományból kiolvashatjuk. Példa:

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

A swap területet létrehozhatjuk külön partíción vagy külön fájlban is. Erről az operációs rendszer telepítésekor általában választhatunk. Van lehetőség azonban később is létrehozni új területet, akár méretet növelni vagy cs ökkenteni.

### 3.3.1 Csereterület létrehozása

**Külön fájlként**

Ahhoz, hogy swap területet fájlon használjunk, meg kell határoznunk a méretét (max. 2 GB x86 rendszeren). Majd a dd parancs nyújt nekünk segítséget, amellyel könnyen és gyorsan létrehozhatunk tetszőleges nagyságú üres virtuális lemez image-et.

```
dd bs=1024 count=1M if=/dev/zero of=/path/to/swapfile.n
```

Ez a parancs egy 1GB-os fájlt (1 MB * 1024 bytes) hoz létre a "/path/to/swapfile.n"-on. Ezt meg lehet csinálni többször is, ha több swap fájl akarunk használni (maxium 32 darab). A swap fájlok nem lehetnek elszórtan, az egészet allokálni kell a használat előtt. Ha ez sikerült, akkor meg kell formázni (inicializálni), „**swap header**” információval kell ellátni:

```
mkswap /work/swapfile.1
```

**Külön partíción**

Ez egy kis előzetes tervezést igényel még a merevlemez felosztása (partícionálás) során. A swap terület, amely partíción foglal helyet, be van jegyezve a /etc/fstab fájlba. Például:

```
/dev/hda2 none swap defaults 0 0
```

Miután a swap terület allokálva van, meg kell formázni (inicializálni). Ezt csak egyszer kell elvégezni:

```
# mkswap /dev/hda2
```

A partíción levő swap terület a boot folyamat init részében (lásd később) automatikusan aktiválódik. Az a swap területe, amely fájlrendszeren helyezkedik el, szükséges engedélyezni. Erre minden bootolás során szükség van, így célszerű egy init szkriptet írni (ha a disztribúció azon kis hányadhoz tartozik, ami nem kezeli automatikusan az ilyen állományokat). Swap terület bekapcsolása a **swapon** paranccsal történik:

```
# swapon /work/swapfile.1
```

A swap területet ki is lehet kapcsolni, ha nincs rá szükség. Ezt a **swapoff** parancsal tehetjük meg. Pl.:

```
# swapoff /work/swapfile.1
```

## 3.4 Az *fstab* fájl

Az /etc/fstab fájl szerepe UNIX/Linux rendszereken, hogy az itt lefektetett szabályok megkönnyítik az állományrendszerek használatát. Megléte rendkívül fontos, mivel az fsck (lásd később) és a mount (tehát a boot során az automata mount is) ebből tájékozódik az elérhető állományrendszerekről.

A szöveges állomány tartalma statikus (elvileg programok csak olvashatják) és kézzel kell elkészíteni. Jelenleg már több disztribúció képes automatikusan kezelni az fstab tartalmát. Ennek ellenére nem árt megismerkedni a felépítésével, mivel így jobban kontrollálható és egyedi elvárások is megvalósíthatóak.

Egy példa fstab fájl tartalmára:

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/swapfile       none            swap    sw              0       0
/dev/hda1       /               jfs     defaults,errors=remount-ro      0       1
/dev/hda5       /mnt/data       ext3    defaults        0       2
/dev/hdc        /media/cdrom0   iso9660 ro,users,noauto,utf8    0       0
/dev/fd0        /media/floppy0  auto    rw,users,noauto 0       0
/dev/hdb1       /mnt/adat       vfat    defaults,rw,users,uid=0,gid=6,umask=0003,codepage=852,utf8
/dev/hdb2       /mnt/winnt      ntfs    defaults,ro,users,uid=0,gid=6,umask=0003,utf8,noatime   0
/dev/sda1       /mnt/usbcam/    vfat    defaults,noauto,users,noatime   0       0
```

**Minden mount point-ot (csatlakozási helyet) egy külön sor képvisel**, minden opció tetszőleges mennyiségű szóközzel (vagy tabulátorral) van elválasztva. Természetesen lehet megjegyzéseket is beleszúrni, ha egy sor elejére tett # jellel.

**Az első oszlop:** megmutatja, hogy fizikailag hol helyezkedik el a kezelendő állományrendszer. Alapesetben ez egy device (/dev-ben), de lehet még NFS kötet (lásd később). Kevésbé statikus megoldás (pl.: mobil rack vagy pendrive esetén nagyon hasznos), hogy a mount pontra lehet hivatkozni a volume label (kötetnév) vagy az UUID alapján (pl.: LABEL=Boot vagy „UUID=3e6be9de-8139-11d1-9106-a43f08d823a6”), ezek lekérdezésére használható a disktype (disktype /dev/hdxy) vagy a hdparm (hdparm -I /dev/hdxy).

**A második oszlop**: megadja, hogy hová kerüljön mountolásra az állományrendszer (ez a mount point). A rootfs (felsőszintű állományrendszer) mindig a / -be mountolódik. Speciális a swap, aminél ez "none" értékkel rendelkezik. A merevlemezeket a /mnt jegyzékbe szokás mountolni, a hordozható lemezeket a /media-ba.

**Harmadik oszlop:** megadja az állományrendszer típusát. Rengeteg típus támogatott, legelterjedtebbek: ext2, ext3, iso9660, jfs, reiserfs, vfat, xfs, de csak olyan típusokat írhatunk be, amiket támogat az aktuális futó kernel. Ezt a /proc/filesystems alatt lehet ellenőrizni. Erről is informál a disktype. A floppyt és más hordozható lemezeket (pl.: pendrive) auto-ra szokás állítani a jobb kompatibilitás érdekében. (Megj.: vesszővel elválasztva megadhatunk több fajta állományrendszert is, pl.: jfs, ext3.)

**Negyedik oszlop**: az állományrendszer specifikus mountolási beállítások sorát tárolja, itt van lehetőség egy kis tuningra. A lista elemeit vesszővel kell elválasztani, a lehetőségről a mount manpage tudósít. Néhány példa:

- **ro/rw:** csak olvasható / olvasható és írható az állományrendszer
- **exec/noexec:** a bináris állományok futtathatóságát engedi / tiltja
- **defaults**: az általános működés beállításai: rw suid dev exec auto nouser async

## 3.5 Fájlrendszerek épségének ellenőrzése

A fájlrendszerek viszonylag bonyolult felépítésűek, ezért hajlamosak a meghibásodásra. Egy ellenőrzés során a cél mindig azonos: **a fájlrendszer tüzetes átvizsgálásával megtalálni az esetleges problémákat, inkonzisztenciákat**. Egy fájlrendszer helyessége és érvényessége UNIX/Linux rendszerekben az ***fsck*** paranccsal ellenőrizhető.

Ez a program képes a megtalált kisebb problémákat kijavítani, és figyelmeztetni a nem javítható hibákra. Szerencsére a fájlrendszert megvalósító kódot nagyon hatékonyan hiba mentesítették, ezért nagyon ritkán akad probléma, és az is többnyire áramkimaradás, hardverhiba vagy kezelési hiba (nem megfelelő rendszerleállítás) miatt keletkezik.

A legtöbb rendszer úgy van beállítva, hogy rendszerindításkor automatikusan futtatja az fsck programot azon fájlrendszerek esetén, amelyek automatikusan kerülnek felcsatolásra, így remélhetően minden hiba kiderül és javításra kerül a rendszer használata előtt. Az fsck parancsot kézzel kell indítanunk a többi fájlrendszer (például hajlékonylemezek) esetén.

Sérült fájlrendszer használata esetén a hibák csak szaporodnak. Ha a metaadatokban van esetleg a probléma, a fájlrendszer használata még több adatvesztéshez vezethet.

Az fsck futtatása fájlrendszer típustól erősen függően eltart egy ideig. Bár az ellenőrzés szinte mindig felesleges, ha a rendszerleállás szabályos volt, ezért néhány trükkel meg lehet gátolni az ellenőrzés állandó automatikus végrehajtását.

Az első trükk, hogy ha létezik az /etc/fastboot fájl, nem történik ellenőrzés. A második, hogy az ext2-es fájlrendszereknek van egy speciális jelzése a szuperblokkban, mely jelzi, hogy a fájlrendszer szabályosan lett-e lecsatolva az utolsó felcsatolás óta.

**Nagyon fontos!**

Az fsck programot csak lecsatolt fájlrendszeren szabad alkalmazni! (Egyetlen kivétel a csak olvasható gyökér fájlrendszer a rendszerindítás során.) Ez azért van, mert a program a nyers lemezblokkokkal dolgozik, és ezért úgy módosíthatja a fájlrendszert, hogy azt az operációs rendszer nem veszi észre, ami biztosan jelentős hibákhoz vezet.

Az fsck parancs csak egy front-end. Minden fájlrendszernek megvannak a maga fsck parancsi. Példa egy ext2 fájlrendszer átvizsgálására:

```
# fsck.ext2   /dev/hda7     vagy    # fsck –t ext2   /dev/hda7
```

ext3 fájlrendszer esetén:

```
# fsck.ext3   /dev/hda7     vagy    # fsck –t ext3   /dev/hda7
```

Amennyiben bővebb információkat szeretnénk kapni egy fájlrendszerről, úgy a ***dumpe2fs*** parancsot kell meghívnunk. Példa (részlet):

```
# A dumpe2fs
dumpe2fs 1.40.2 (12-Jul-2007)
Filesystem volume name:
Last mounted on:
```

```
Filesystem UUID: ec7a16c5-c680-45ac-9f1b-856ab960d759
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: has_journal resize_inode dir_index filetype needs_recovery sparse_super large_file
Filesystem flags: signed directory hash
Default mount options: (none)
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 997472
Block count: 1994060
Reserved block count: 99703
…
stb…
```

Ha egy létező fájlrendszeren szeretnénk módosításokat, további hangolásokat elvégezni akkor erre a ***tunefs/tune2fs*** nevű parancs nyújt segítséget, amely a legtöbb Unix/Linux rendszer szerves része. Például egy ext2 fájlrendszer ext3-ra való alakítása kapcsán.

## 3.6 Chroot

Időnként szembekerülhetünk olyan problémával, hogy átmenetileg meg kellene változtatnunk a root jegyzékét egy program lefuttatásának idejére.

Ennek egyik tipikus esete, amikor egy rendszer visszaállítás során felcsatolunk egy root partíciót valamely pontra a jegyzékstruktúrában. Majd ezen a köteten szeretnénk lefuttatni egy programot úgy, hogy az root jegyzéknek a kötet eredeti root jegyzékét lássa.

Ilyenkor a megoldás a ***chroot*** parancs használata, amely indít egy olyan shellt, ami a megadott jegyzéket látja root jegyzéknek.

```
# chroot <új gyökér könyvtár>
```

A másik ok, amiért a gyökér jegyzéket átállíthatjuk egy program működési területének lekorlátozása. Így ha a szolgáltatást fel is törnék az úgy gyökérnek megadott aljegyzéknél nem juthatnak feljebb a jegyzékstruktúrában, ezáltal nem szerezhetnek hozzáférést a rendszer kényesebb részeihez.

Ezzel egy kisebb rendszert hozhatunk létre a nagyobbik rendszeren belül. Gyakran jelen van Linux telepítőkészletekben, és ún. LiveCD-k használatakor.

# 4. Fájlok megosztása és átvitele

Manapság a számítógépek már nagy háttértárolókkal rendelkeznek, azonban még mindig szükség van a fájl szerverekre. Az okok lehetnek az alábbiak:

- Fontos információk központi tárolása, mások számára elérhetővé tétele.

- Az egyes számítógépek különböző veszélyeknek vannak kitéve, vírusok, meghibásodások, emberi hibák. Ezért biztonsági másolatokat helyezhetünk el a központi szerveren.
- A munkaállomások könnyebb adminisztrációja, az installációs állományok tárolása, vagy a programok központi szerverről való futtatása révén.

A következőkben néhány ismertebb fájlmegosztási és fájlátviteli módszerek kerülnek ismertetésre.

## 4.1 FTP

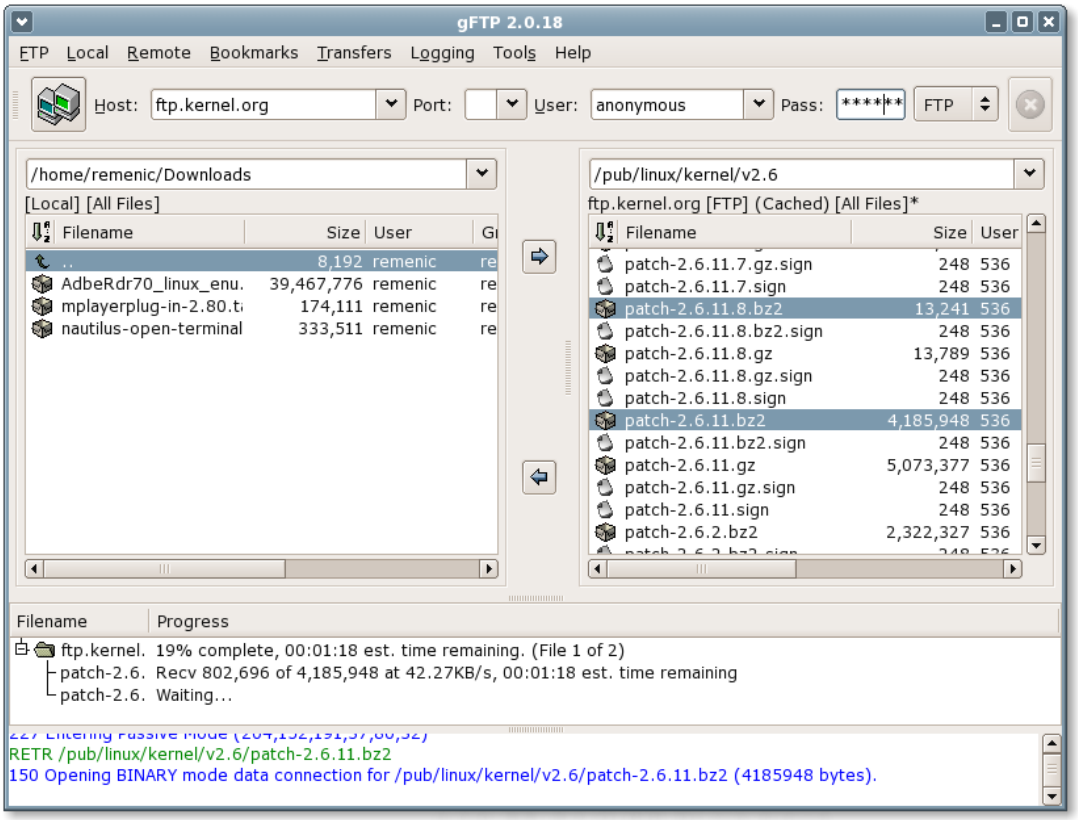
Az állományok megosztásának egyik legrégibb, és manapság is elterjedten használt módja az FTP *(File Transfer Protocol)* szerverek használata. Az ftp protokollt 1985-ben rögzítették az **RFC 959** szabványban. Megkülönböztetünk ftp szolgáltatást nyújtó programot (ftp szerver), és az ftp szolgáltatást igénybe vevő programot (ftp kliens). A fájl átvitelt a kliens oldalon kell kezdeményezni. A klienstől a szerver felé történő átvitelt feltöltésnek (**upload**), míg a szervertől a kliens felé történő fájlátvitelt letöltésnek (**download**) nevezzük.

A Linux rendszereken, akárcsak máshol, az FTP szerverek több változata is elterjedt. Az egyik legismertebb a vsftpd *(Very Secure FTP Daemon)* program. Működésük *daemon* jellegű, de léteznek *inetd (lásd később)* alapú FTP szerver implementációk is. A vsftpd konfigurációs állománya az /etc/vsftpd/vsftpd.conf.

Az FTP szerverek működése alapján két változatot különböztetünk meg. A legáltalánosabban használt, amikor a rendszer felhasználói a nevükkel és jelszavukkal bejelentkezve, a felhasználói jogaikkal látják az állományrendszert. Ilyenkor kezelhetik a saját állományaikat. A másik működési mód, amikor publikálunk állományokat. Ebben az esetben a kliens programok azonosítás nélkül, un. "anonymous"-ként jelentkeznek be. Ekkor az *ftp* felhasználó jogaival tevékenykednek, és mozgási területük az *ftp* felhasználó *home* jegyzékére korlátozódik. A vsftpd programban is bekapcsolhatjuk az *anonymous* működési módot.

Az ***ftpd*** az ftp szolgáltatásért felelős daemon. Ő kezeli az ftp kliensektől érkező csatlakozási kéréseket, és bonyolítja a fájl átvitelt a kliens kérése és a saját beállításai szerint. Az ftpd a 21-es portot figyeli, minden ide érkező kérést az ftp szerverhez irányít.

A mai linux disztribúciók szinte mindegyike rendelkezik a gftp nevű programmal, amely grafikusan is lehetővé teszi a fájlok átvitelét több protokoll támogatásával is:




## 4.2 NFS

Az NFS *(Network File System)* protokoll a Unix rendszerek elterjedt megoldása az állományok megosztására. A Linux rendszerek üzemelhetnek NFS szerverként is, de kliensként fel is csatolhatjuk más szerverek megosztott könyvtárait.

Linux alatt az NFS szerver implementációja két részre oszlik. Egyrészt egy kernel modulra, másrészt egy felhasználói módú programra. A felhasználói módban futó folyamat fogadja a kapcsolódási kérelmeket, és indítja a kernel szálakat.

Az NFS szerver elindulása és működése részben különbözik a *daemon*, illetve *inetd* alapú működéstől.

**Mi is az az inetd?**

Az *inetd* démont gyakran csak „internet szuperszerverként” nevezik, mert a helyi szolgáltatások kapcsolatainak kezeléséért felelős. Amikor az inetd fogad egy csatlakozási kérelmet, akkor eldönti róla, hogy ez melyik programhoz tartozik és elindít egy példányt belőle, majd átadja neki a socketet (az így meghívott program a szabvány bemenetéhez, kimenetéhez és hibajelzési csatornájához kapja meg a socket leíróit). Az inetd használatával úgy tudjuk csökkenteni a rendszerünk terhelését, hogy a csak alkalmanként meghívott szolgáltatásokat nem futtatjuk teljesen független önálló módban.

Az inetd démont elsősorban más démonok elindítására használjuk. A konfigurációs állománya az /etc/inetd.conf.

Ennek oka, hogy **RPC** (Remote Procedure Call) alapú kommunikációra épül. Ezért igényli a működéséhez az RPC kommunikációs rendszert is, amelyhez kapcsolódik a folyamatosan futó daemon jellegű szerver. Az RPC kommunikációs rendszert a ***portmap*** szolgáltatással indíthatjuk el. Ezek után már futtathatjuk az nfs és az nfslock szolgáltatásokat.

Különböző disztribúciókban egy nfs szerver kialakításához szükséges csomagok nevei eltérhetnek egymástól, de általában a következő csomagok megtalálhatók az ismertebb disztribúciókban: nfs-utils, nfs-common, portmap, rpcbind.

Ubuntu típusú disztribúcióban a szükséges csomagok installálása a következőképpen történhet:

```
# sudo apt-get install nfs-kernel-server nfs-common portmap
```

A jegyzék megosztásokat az **/etc/exports** állományban állíthatjuk be. Minden sor elején a megosztott jegyzék szerepel, majd ezt követi a megosztást elérhető kliensek listája (üres helyekkel elválasztva). Minden kliensnek külön megadhatjuk a hozzáférési jogait zárójelben.
Kliensnek megadhatunk egyes gépeket névvel vagy IP címmel. De megadhatjuk gépek egy csoportját úgy, hogy a név megadásánál használjuk a "*" és "?" karaktereket, vagy IP címtartományt adunk meg.

A hozzáféréseket paraméterekkel szabályozhatjuk. Néhány fontosabb paraméter:

| Paraméter | Leírás |
|---|---|
| **ro** | Csak olvasható a megosztás. (Alapértelmezett) |
| **rw** | Olvasható-írható a megosztás. |
| **secure** | A kliens kérelmének 1024 alatti portról kell érkeznie, vagyis csak *root* lehet a felhasználó. (Alapértelmezett) |
| **asvnc** | Aszinkron módon kezeli a szerver a meghajtót, vagyis hamarabb válaszol, mint hogy letárolta a módosításokat. |
| **all_squash** | Minden felhasználót anonymous-ra képez le. |

**Példa egy /etc/export fájl tartalomra:**

```
/files *(ro,sync)              # Csak olvasás mindenkinek

/files 192.168.0.100(rw,sync)  # Írás olvasás a 192.168.0.100 kliensnek

/files 192.168.1.1/24(rw,sync)  #  Írás, olvasás minden 192.168.1.1 – 192.168.1.255 kliensnek
```

**Példa:**

```
#  /pub  *.uni-miskolc.hu(rw,all_squash)
```

Itt a /pub jegyzéket tettük elérhetővé a uni-miskolc.hu *domain* minden gépe számára olvasás-írás joggal, és a felhasználók *anonymous*-ra képezésével.

**Megjegyzés:** Figyeljünk arra, hogy a gép név és az opciók között ne legyen üres hely!

Egy szerver NFS megosztásait a következő paranccsal listázhatjuk ki:

```
#  showmount –e <gépnév>
```

A fejlett disztribúciókban grafikus felületet is biztosítanak az NFS megosztások menedzselésére.

**NFS megosztás használata**

Valamely NFS szerver megosztásait kliensként a mount paranccsal használhatjuk a következőképpen:

```
#  mount szerver_IP_címe:/megosztani/szánt/jegyzék /ahova/csatolni/szeretnénk/mappa
```

## 4.3 Samba

A Samba egy olyan eszközgyűjtemény, amelyek segítségével a hálózaton olyan erőforrások oszthatók meg, mint a nyomtatók és a fájlok. A Samba a Microsoft és az IBM által is elfogadott **Server Message Block** (SMB) protokollt használja arra, hogy TCP/IP hálózaton keresztül alacsony szinten adatokat cseréljen Windows ügyfelek és Unix kiszolgálók között. A Samba szabad szoftver így szinte minden platformon megtalálható, de léteznek kereskedelmi forgalomban kapható változatai is.

A Samba teljes megoldást kínál a helyi hálózatok számára – álljanak azok akár két otthoni számítógépből, akár egy nagyvállalat több száz csomópontjából. A Samba könnyen telepíthető és felügyelhető, és olyan átlátszó hálózati környezetet teremt, amelyben a felhasználók a munkájukhoz szükséges összes erőforráshoz hozzáférhetnek.

Mit kínál a Samba?

- Unix fájlokat kínálhatunk fel Windows, OS/2 és más operációs rendszereket használó ügyfeleknek.
- Unixos ügyfelek számára elérhetővé tehetjük a Windows-os fájlokat.
- Elérhetővé tehetjük a hálózati nyomtatókat a Windows ügyfelek számára.
- Névkiszolgálást kínálhatunk (broadcast és WINS).
- Engedélyezhetjük, hogy Windows ügyfelek böngésszék a hálózati erőforrásokat.
- Windows munkacsoportokat vagy tartományokat hozhatunk létre.
- Előírhatjuk az ügyfelek felhasználónevének és jelszavának a hitelesítését.

Az SMB egy kapcsolat orientált protokoll. Minden kapcsolatra létrehoz egy kapcsolat ID-t („service user ID” (UID)), ezen belül pedig minden megosztáshoz létrehoz egy TID és ezen belül minden fájlnak egy FID-t. TID az erőforrás ID-je, amihez kapcsolódni akar a kliens (pl. egy megosztás) a FID pedig azon az erőforráson egy file-hoz kapcsolt ID.

Jelenleg a Samba csomag működése két Unix démon körül forog, amelyek megosztott erőforrásokat – vagy más szóval *megosztásokat* – kínálnak a hálózatba kapcsolódó SMB ügyfeleknek. A két démon:

- ***smbd:*** Ez a démon lehetővé teszi fájlok és nyomtatók megosztását SMB hálózatban, és az SMB ügyfelek azonosságának és jogosultságának vizsgálatát. Minden beérkező kérésre a új smbd processz indul, amely újraolvassa a konfigurációs fájlokat és kiszolgálja a kérelmet.
- ***nmbd:*** Ez a démon a WINS (Windows Internet Name Service, Windows internet névkiszolgáló) kezeléséről gondoskodik, és segítséget nyújt a tallózásban.

### 4.3.1 Samba konfiguráció

A Samba szerver konfigurációs állományai az **/etc/samba/** jegyzékben találhatóak.

- Az smb.conf fájl tartalmazza a szerver beállításait. Megosztásokat is ebben az állományban hozhatunk létre.
- Az lmhosts a hosts fájlhoz hasonlít, a Név - IP cím leképezéseket tartalmazza.
- Az smbusers az NT felhasználók Linux felhasználó listába való leképezését tartalmazza.
- Az smbpasswd állomány a Samba felhasználói jelszókat tartalmazza kódolt formában.

A rendszer további fontos jegyzéke a /var/log/samba könyvtár, ahol a log állományok találhatók.

### 4.3.2 Egy alap szerver konfiguráció

A fő konfigurációs opciók tehát az /etc/samba/smb.conf állományban találhatóak. Itt kell kezdeni a szerver konfigurálását. Első lépésként beállíthatjuk a szerver workgroup paraméterét, a leíró szövegét, és esetleg a NetBIOS nevét is.

(Hálózatban résztvevő minden gépnek van egy netbios neve. Egy NetBIOS név 16 karakterből áll, de ebből csak 15 használható a név megadására, a tizenhatodik byte az erőforrás (pl. megosztás) típusát adja meg)

Tehát:

```
netbios name = név
workgroup = CSOPORT
server string = Ez egy samba szerver
```

Következő lépésként lekorlátozhatjuk a hozzáférést a szerverhez egyes IP cím tartományokra a cím részleges megadásával:

```
host allow = 192.168.
```

### 4.3.3 Authentikációs beállítások

Az SMB esetében alapvetően kétféle authentikációs metódust különböztetünk meg. A régi Windows 9x metódust (share), amely a megosztásokra ad meg jelszavakat, illetve a felhasználók azonosításán alapuló metódust. Utóbbi esetben 4 különböző megoldás közül is választhatunk az információk ellenőrzésénél:

• **user:** a Samba szerver fogadja a usernév/jelszó párost és az adatbázisa alapján leellenőrzi.
• **domain:** ebben az esetben a szerver egy NT domain tagja, rendelkezik egy gép accounttal a domain controllerben. Minden authentikációs információt a domain controller felé továbbít.
• **ADS:** a Samba szerver egy Active Directory domainhez való csatlakozási lehetősége.
• **server:** elavult opció. Régebben még a Samba nem volt képes a domain authentikációra. Lényege, hogy a szerver átveszi a usemevet/jelszót a klienstől majd ezzel megpróbál a domain controllerre belépni. Amennyiben sikerül, akkor elfogadja az authentikációt.

Az egyszerűség kedvéért válasszuk példaként a user authentikációs metódus, titkosított jelszókezeléssel. Ezt a következő sorral állíthatjuk be:

```
security = user
```

Azonban ahhoz, hogy a jelszavak a hálózaton biztonságban legyenek, szükséges hogy titkosítva továbbítsuk. Ennek beállítása:

```
encrypt passwords = yes
```

Ez felvet egy problémát. Az MS Windows rendszeren és a Unix/Linux rendszerek jelszó kódolási algoritmusa különbözik egymástól. Ugyanakkor a kódolás egyirányú, ezért a titkos jelszóból az eredeti jelszó nem nyerhető vissza, ebből következően a Linux jelszavakat nem tudjuk felhasználni a samba authentikációhóz. Vagyis kódolt jelszavak esetén a samba jelszavakat külön meg kell adnunk és tárolnunk. Az állomány, amely a jelszókat tárolja a következő sorral adható meg:

```
smb passwd file = /etc/samba/smbpasswd
```

A jelszót az **smbpasswd** programmal tudjuk beállítani. De ehhez először fel kell vennünk a samba felhasználókat. így az első alkalommal a -a paraméterrel kell meghívnunk:

```
smbpasswd -a <felhasználó>
```

Ha a Samba felhasználók neve nem egyezik meg a Linux accountokkal, akkor leképezést is megadhatunk a kettő között. Ezt a korábban említett smbusers állománya tartalmazza, és az alábbi sorral adhatjuk meg:

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

### 4.3.3 Megosztások kezelése

A főbb paraméterek beállítása után már létrehozhatunk megosztásokat is. Ha egy publikus, de csak olvasható megosztást szeretnénk létrehozni, akkor annak alakja a következő lehet:

```
[public]
    comment = Publikus
    path = /home/samba
    public = yes
    writable = no
    printable = no
```

Itt a [] jelek közötti szöveg adja meg a megosztás nevét, a comment a megosztás leírását. A *path* paraméterrel állíthatjuk be a megosztott jegyzéket. A *public* jellemző beállításával mindenki számára elérhetővé tesszük. Viszont mivel a *writable* paramétert *no*-ra állítottuk, így csak olvasható lesz.

Ha azt szeretnénk, hogy a felhasználók egy csoportja írhassa, akkor az alábbiak szerint módosítsuk:

```
[public]
        comment = Publikus
        path = /home/samba
        public = yes
        writable = yes
        printable = no
        write list = felhasználói felhasználó2 …
```

A felhasználók felsorolása helyett megadhatunk csoportot is. Ilyenkor a csoport neve elé be kell írnunk a "@" jelet. Azonban ha csak egyes felhasználók számára szeretnénk valami teljes eléréssel megosztani, akkor a következő konfigurációt használhatjuk:

```
[joe]
      comment = Joe könyvtára
      path = /home/joe
      valid users = joe
      public = no
      writable = yes
      printable = no
```

Ha a szerver minden felhasználójának szeretnénk megosztani a saját *home* jegyzékét, akkor szerencsére nincs szükségünk arra, hogy mindegyikhez írjunk egy ilyen konfigurációs részt. Ehhez a Samba támogat egy speciális megosztást, amelynek a neve "homes".

```
[homes]
    comment = Home Directories
    browseable = no
    writable = yes
    create mode = 0664
    directory mode = 0775
```

### 4.3.4 Megosztások felcsatolása

Eddigiekben bemutattuk, hogy hogyan konfigurálható egy SAMBA szerver. A továbbiakban már a megosztások használata, felcsatolása marad. Erre többféle megoldás létezik. Több disztribúció külön segédprogramot nyújt a SAMBA megosztások felcsatolására. Ezek az úgynevezett **smbclient, smbmount** alkalmazások. Az smbclient használata:

```
#  smbclient //gépnév/megosztás -U <felhasználó>
```

Egy további egyszerű megoldása a megosztások felcsatolásának a **mount** parancs használata:

```
# mount -t smbfs -o username=skywalker //gépnév/megosztás /mnt/név
```

# 5. Rendszerindítás

## 5.1 Általános áttekintés

Azt a folyamatot, amely a számítógép bekapcsolása után az operációs rendszer betöltését végzi, rendszerindításnak vagy boot-nak nevezzük. A ''boot'' (csizma) név egy olyan képből származik, mely szerint a számítógép ilyenkor felhúzza a csizmáját (**bootstrap**).

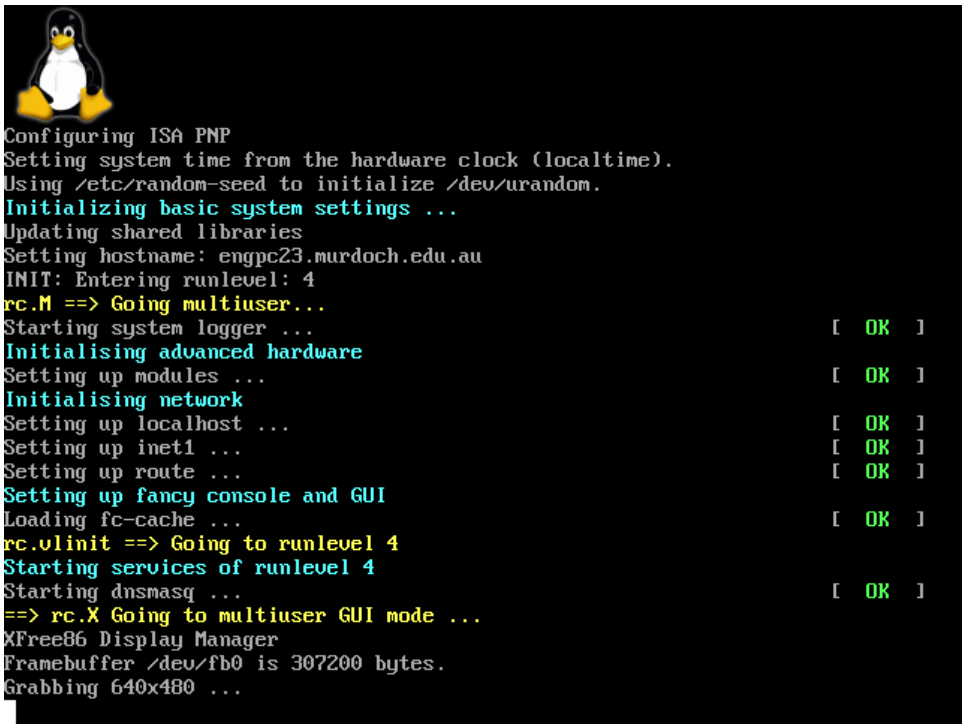
Amikor egy PC elindul, a BIOS különféle teszteket végez annak ellenőrzésére, hogy minden rendben van-e. (Ezt szokás power on self test-nek vagy röviden POST-nak nevezni.) Ezután indul a tényleges rendszerindítás. Először egy lemezmeghajtó kerül kiválasztásra, az ebben levő lemez legelső szektorát, a boot szektort olvassa be a rendszer. Merevlemezeknél a **master boot record (MBR)** kerül beolvasásra, ugyanis egy merevlemez több partíciót is tartalmazhat, mindegyiken saját boot szektorral.

A boot szektor egy kis (egy szektorba elférő) programot tartalmaz (ún. ***bootstrap loader***), melynek a feladata az aktuális operációs rendszer beolvasása és elindítása. Amikor merevlemezről boot-olunk, a master boot record-beli (MBR) kód megvizsgálja a partíciós táblát, hogy azonosítsa az aktív partíciót (azaz amelyik boot-olhatóvá lett téve), beolvassa annak boot szektorát, és elindítja az itteni kódot. A partíció boot szektorában található kód beolvassa a kernelt és elindítja. A részletek ugyan egy kicsit változatosak, mivel általában nem célszerű egy külön partíciót fenntartani a kernel képmásának (kernel image), ezért a boot szektorban található kód nem olvashatja egyszerűen sorban a lemez blokkjait, hanem meg kell találni azokat a blokkokat, ahova a fájlrendszer lerakta a kernel képmását. Több megoldás is létezik erre a problémára, de a leggyakoribb a **boot menedzserek** használata (LILO, GRUB).

Miután a Linux kernel bekerült a memóriába valamilyen értelemben, és valóban elindult, általánosságban a következők történnek:

- A Linux kernel a lemezen tömörítve van, ezért először kicsomagolja önmagát. A kernel képmás eleje egy kis programot tartalmaz e célból.
- Ezután a kernel ellenőrzi, milyen hardver elemek (merevlemezek, floppy meghajtók, hálózati kártyák, stb.) léteznek a gépben, és megpróbálja ezek eszközmeghajtóit megfelelően konfigurálni. A hardverelemek megtalálásáról üzenetet is ad.
- Ezután a kernel csatlakoztatja (mount) a gyökér fájlrendszert. A fájlrendszer típusát automatikus felismeri a rendszer. Ha a gyökér fájlrendszer csatlakoztatása nem sikerül, mert pl. elfelejtették a megfelelő fájlrendszert belefordítani a kernelbe, a kernel *''pánikba esik''* (**kernel panic**), és megállítja a boot-olás folyamatát.
- Ha minden rendben ment, akkor a kernel elindítja az ***init*** programot (/sbin/init) a háttérben. (Értelemszerűen az init processz-azonosítója (PID) mindenképpen 1 lesz.)
- Az init ezután több felhasználós üzemmódba kapcsol, és elindítja a **getty** programot a virtuális konzolokra és a soros vonalakra, így lehetővé teszi a bejelentkezést a felhasználók számára. Az init elindíthat még más programokat is a konfigurációjától függően.

Ezzel a rendszerindítás kész van, és a rendszer normálisan fut tovább. Kép egy Linux boot-olásáról:




## 5.2 Az *init*, mint első folyamat

Az **init** egyike azon programoknak, melyek alapvetőek egy Linux rendszer működése szempontjából, ám többnyire nem veszünk róla tudomást. Amikor a kernel elindítja önmagát (azaz betöltődött a memóriába, elindult, inicializálta az eszközvezérlőket és az adatstruktúrákat), a rendszerindítás folyamatának rá eső részét egy felhasználói szintű program, az init indításával fejezi be. Ezért az init mindig a legelső processz, azaz processz-azonosítója mindig 1. A kernel az init-et néhány olyan helyen keresi, ahol a rendszer korábbi változataiban lenni szokott, de egy Linux rendszeren az igazi helye az /sbin/init. Ha a kernel sehol sem találja az init-et, megpróbálja futtatni a /bin/sh-t, és ha azt sem találja, a rendszer indítása sikertelenül fejeződik be.

Amikor az init elindul, először a rendszerindítási folyamatot fejezi be néhány adminisztratív feladat elvégzésével. Ilyen pl. a fájlrendszerek ellenőrzése, a /tmp kitakarítása, különféle szolgáltatások elindítása és a getty indítása minden terminálra illetve virtuális konzolra, ahonnét a felhasználók bejelentkezhetnek.

Az init örökbe fogadja (adopt) az árva (orphan) processzeket: amikor egy processz gyermekprocesszt indít, és gyermeke előtt meghal, a gyermek azonnal az init gyermekévé válik. Ez többféle technikai okból fontos, és jó tudni róla, mivel könnyebben érthetővé teszi a processzek listáját és fadiagramjait.

Az init-nek néhány változata érhető el. A legtöbb Linux terjesztés a **sysvinit**-et használja, és amely a System V rendszer init-jén alapul. A UNIX BSD változatainak más az init programja. Az alapvető különbség a futásszintekben van: a System V-nek vannak futásszintjei, a hagyományos BSD-nek nincsenek.

Az init konfiguráció állománya az etc/inittab (részletezése később).

## 5.3 Futási szintek

A Unix/Linux rendszerek különféle üzemmódokat, futási szinteket *(run level)* különböztetnek meg. A futási szintek az operációs rendszer állapotai, amelyek mindegyik egy-egy külön célt szolgál, és a rendszergazda választja ki, hogy éppen melyik felel meg a feladat elvégzésére.

Azt, hogy az adott GNU/Linux rendszeren melyik futási szint milyen viselkedést jelent, a beállító állományok határozzák meg. Minden GNU/Linux gyűjtemény a rá jellemző beállító állományokkal a rá jellemző futási szinteket határozza meg. A Red Hat Linux alapú terjesztések például általában a következő futási szinteket különböztetik meg:

**0 – Leállítás**: ez a futási szint a számítógép leállítását, kikapcsolását végzi. Amelyik számítógép felveszi ezt a futási szintet, az rövid időn belül leáll.
**1 – Egy felhasználós üzemmód:** amelyet szokás S betűvel is jelölni. Ez az üzemmód szervizcélokra, helyreállításra használható. Ha egy számítógép felveszi ezt a futási szintet, csak a rendszergazda számára engedi a rendszer használatát.
**3 – Több felhasználós üzemmód**: Ez a futási szint a számítógép normális használat közben használt futási szintje.
**5 – Több felhasználós üzemmód grafikus felülettel**: Ezt a futási szintet használva a felhasználók számára grafikus bejelentkezési felületet biztosíthatunk.
**6 – Újraindítás:** Az a számítógép, amely felveszi ezt a futási szintet, rövidesen újraindul.

Az, hogy rendszerünk éppen milyen futási szintben van a ***runlevel*** paranccsal kérdezhetjük le.

### 5.3.1 Futási szintek konfigurációja

A futási szintek konfigurációja az úgynevezett indítótáblázatban található. Ennek helye: /etc/inittab, amely egy egyszerű szöveges állomány. Ezt az állományt az init processz olvassa be. Különlegesen fontos, mivel az init által a különféle futási szinteken indítandó szolgáltatások leírását tartalmazza. Ha a fájl sérült, vagy nem létezik, akkor a számítógép csak egyfelhasználós üzemmódban indítható.

A /etc/inittab sorai négy, kettősponttal határolt mezőre oszlanak:

**id:runlevels:action:process**

A mezőket pedig a fájlban lejjebb részletezzük ki. A megjegyzéseket a #-val kezdődő sorok jelentik. És akkor a jelentések:

**id:** Ez azonosítja a fájlon belüli sort. A getty vonalakra ez mondja meg a terminált, amelyen fut, pontosabban a /dev/tty utáni karaktereket az eszközfájl nevében. Más sorokra tartalma lényegtelen (de nem lehet túl hosszú), és egyedinek kell lenni.

**runlevels:** A futásszintek, melyekre a sor vonatkozik. A szinteket egyetlen számként kell megadni elválasztójel nélkül.

**action:** Mit kell tenni a megadott sorral. Pl. mindig újraindítani (respawn), vagy csak egyszer futtatni (once).

**process:** A futtatandó parancs.

Példa:

16:6:wait:/etc/init.d/rc 6

**Jelentése:** Az első mező egy tetszőleges címke, a második azt jelenti, hogy ez a sor csak a 6. futásszintre vonatkozik. A harmadik mező azt jelenti, hogy az init-nek csak egyszer, a futásszintbe való belépéskor kell futtatni a negyedik mezőben megadott parancsot, és meg kell várni, hogy az befejeződjön.

A rendszer futása közben a futási szint a **telinit** paranccsal módosítható, a kívánt szint számát paraméterként megadva. Erre csak a rendszergazda jogosult. Váltás 3-as futási szintre:

```
# telinit 3
```

## 5.4 Alrendszerek indítása és leállítása

Nem minden szolgáltatást indít az init közvetlenül. Nem volna szerencsés, ha egy ilyen fontos program indítaná azokat a szolgáltatásokat, amelyek a rendszer működése szempontjából nem elengedhetetlenek, hiszen akkor a kevésbé fontos szolgáltatások telepítése, törlése, beállítása veszélyeztetné a működőképesség szempontjából létfontosságú alapszolgáltatások meglétét.

Nem volna szerencsés, ha az egyes magas szintű szolgáltatásokat megvalósító programcsomagok telepítése során az /etc/inittab állományt módosítanunk kellene.

A Linux rendszerekben az init csak az elengedhetetlenül fontos alapszolgáltatásokat indítja közvetlenül a /etc/inittab bejegyzései alapján. A magasabb szintű szolgáltatások - amelyeket alrendszereknek *(subsystem)* nevezünk - saját indítóprogramokkal rendelkeznek. Az alrendszerek indíthatók és leállíthatók a saját - általában BASH programként megvalósított - indítóprogramjuk segítségével.

GNU/Linux rendszereken az init az alrendszerek indítását és leállítását közvetve végzi. Minden futásiszint-váltásnál elindítja a /etc/rc.d/rc programot, amely eldönti, hogy mely alrendszereket kell leállítani és elindítani az új futási szinten, és a megfelelő indítóprogramok segítségével el is végzi ezt a munkát.

Az egyes futási szintekhez tartozó elindítandó „programok” az etc/inittab fájlban megtalálhatók. Példa:

```
> cr0:0:wait:/etc/rc.d/rc 0
> cr1:1:wait:/etc/rc.d/rc 1
> cr2:2:wait:/etc/rc.d/rc 2

> cr3:3:wait:/etc/rc.d/rc 3
> cr4:4:wait:/etc/rc.d/rc 4
> cr5:5:wait:/etc/rc.d/rc 5
> cr6:6:wait:/etc/rc.d/rc 6
```

Az rc 0, rc 1, stb valójában egy-egy jegyzék az etc jegyzékben (tehát pl. /etc/rc 1), és ezekben találhatók meg azon alrendszerek indító scriptjei, amelyeket az adott futási szinten indítani szeretnénk. Ezt a folyamatot az /etc/rc.d/rc BASH program végzi el.

# 6. Felhasználók nyilvántartása

## 6.1 A felhasználói azonosítók

Minden többfelhasználós operációs rendszernek rendelkeznie kell jogosultsági rendszerrel. Ahhoz, hogy egy jogosultsági rendszer működhessen, felhasználói "adatbázisra" van szükség. A felhasználói adatbázis az operációs rendszer által kezelt nyilvántartás, amelyben minden felhasználói azonosítóhoz **felhasználói nevet** (username) és más, fontos kiegészítő információkat rendelünk.

A Unix rendszerek alatt minden felhasználó rendelkezik egy **egyedi azonosító** számmal (uid = user identifier), és **egy vagy több csoport azonosító** számmal (gid = group identifier). A különféle műveletek elvégzése előtt az operációs rendszer mindig ellenőrzi, hogy az adott folyamatnak van-e joga a műveletet végrehajtani.

A felhasználók azonosítása a rendszerben az uid szám alapján történik. Bejelentkezéskor a rendszer a név alapján kikeresi az **/etc/passwd** állományból a felhasználó azonosítóját és a későbbiekben ezt használja a rendszerben folytatott tevékenységünk során.

A csoportokba sorolás segítségével a felhasználók egy csoportját összefoghatjuk és a rendszer egyes részeihez több jogot biztosíthatunk, mint a többieknek. Egy felhasználó legalább egy (elsődleges csoport), de több csoport tagja is lehet egyszerre. Ebből az egyik csoport szerepe megkülönböztetett. Ez az alapértelmezett, amelyet az állományok létrehozásakor használ a rendszer.

**A felhasználó azonosítója és az alapértelmezett csoport azonosítója az /etc/passwd állományban van eltárolva.** A többi csoport azonosító az /etc/group állomány bejegyzései alapján rendelődik hozzá a felhasználóhoz.

Létezik egy kitüntetett felhasználó, a rendszergazda (super-user, a login neve "root"). A rendszergazdára nem vonatkoznak a felhasználókra beállított hozzáférési jogok, a rendszer még az állományok tulajdonosnál is bővebb lehetőségeket engedélyez számára. A rendszer annak alapján ismeri fel, hogy az uid-je 0. Jogai gyakorlatilag korlátlanok.

## 6.2 Felhasználók létrehozása

Unix rendszereken az új felhasználó létrehozásához elegendő, ha egy rá vonatkozó bejegyzést írunk az /etc/passwd állományba. Továbbá létre kell hoznunk neki egy ***home*** jegyzéket, amelyet a tulajdonába adunk, és bejegyzünk szintén a **passwd** állományba. Azonban léteznek segédprogramok is, amelyek ezeket a műveleteket elvégzik nekünk. Parancssoros programok közül ilyen a mindenhol megtalálható **useradd**.

Alapvetően két működési módját különböztetjük meg. Az egyik feladata, hogy új felhasználókat hozzon létre. A másik módja a konfigurációs mód, amikor az alapértelmezett értékeket állítjuk be.

A legegyszerűbb, és leggyakrabban használt formája a következő:

useradd -c <valódi név> <login név>

Alapértelmezett esetben bejegyzi a felhasználót a passwd állományba, majd létrehoz egy home könyvtárat, amelybe bemásolja az /etc/skel könyvtárban található állományokat. Mivel a létrehozott felhasználónak nem állítottunk be jelszót, ezért blokkolja a hozzáférését a rendszerhez. A jelszót utólag rendszergazdaként a passwd <login név> paranccsal adhatunk.

Azonban nem csak valódi felhasználókat hozhatunk létre, hanem úgy nevezett rendszerfelhasználókat is. Ezek jelszó nélküli, blokkolt felhasználók home jegyzék nélkül. Csak azért szükségesek, hogy egyes szolgáltatások ezeknek a virtuális felhasználóknak a nevében fussanak, és ez által a rendszer a hozzáférési jogosultságaikat szabályozhassa. Ezek a rendszer-felhasználók abban is különböznek a normál felhasználóktól, hogy az azonosítójuk általában kisebb. A useradd programmal az -r opcióval hozhatjuk létre őket.

Az alapértelmezett paramétereket módosíthatjuk is a -D opció használatával:

useradd -D <opció és alapértelmezett érték>

A rendszer az alapértelmezett értékeket az /etc/default/useradd állományban tárolja.

## 6.3 A felhasználó letiltása

A felhasználó hozzáférését számos okból letilthatjuk. Egyik szokásos eset, amikor csak egy bizonyos időszakra adjuk a hozzáférést. Ilyenkor már a felhasználó létrehozásánál a **useradd** parancs -e opciójával megadhatjuk a lejárati dátumot, amikortól nem férhet hozzá a rendszerhez.

A másik szokásos eset az automatikus letiltásra, amikor kötelező érvényűvé tesszük, hogy a felhasználók bizonyos idő elteltével cseréljék a jelszavukat. Ezt a passwd parancs -x opciójával adhatjuk meg napokban. A -i opcióval pedig azt állíthatjuk be, hogy ha elmulasztaná a felhasználó a jelszó megváltoztatását, akkor hány nap múlva tiltódjon le a hozzáférése.

Adódhat olyan eset is, amikor manuálisan kell letiltanunk egy hozzáférést valamilyen oknál fogva. Ezt szintén a passwd paranccsal tehetjük meg. A -1 opció blokkolja, a -u opció pedig újra engedélyezi a hozzáférést.

## 6.4 A felhasználó törlése

A felhasználó törlése történhet kézzel. Ilyenkor törölnünk kell a felhasználó bejegyzését az /etc/passwd állományból, továbbá shadow jelszó használata esetén az /etc/shadow állományból is. Amennyiben az /etc/group állomány tartalmaz rá vonatkozó bejegyzéseket, akkor azt is el kell távolítanunk. Ezzel a felhasználó megszűnt létezni, azonban még további állományai lehetnek a rendszerben.

Ez alapértelmezett esetben a home jegyzék tartalma, és a levelesládája (általában /var/spool/mail/<login név> ).

Persze ezeket a műveleteket elvégezhetjük egyszerűbben is a **userdel** paranccsal:

userdel -r <login név>

A -r paraméter használatával érhetjük el, hogy a felhasználó home jegyzéke, és a levelesládája is törlődjön.

## 6.5 Felhasználó váltás

A Linux rendszer többfelhasználós tulajdonsága révén lehetőséget ad a felhasználók közötti váltásra is. Ezt az **su** paranccsal tehetjük meg:

**su - (felhasználónév)**

Példa: # su – superman

Amennyiben a rendszergazda felhasználóra szeretnénk váltani, akkor elég az su parancsot önmagában alkalmazni paraméter nélkül. Természetesen mindkét esetben a rendszer kéri a váltani kívánt user kódját.

## 6.6 Felhasználók tulajdonságainak megváltoztatása

A rendszergazda bármely felhasználó tulajdonságait megváltoztathatja a felhasználói adatbázisban, de maga a felhasználó is módosíthatja a róla nyilvántartott adatok egy részét.

**A héj megváltoztatása**

A felhasználó bármikor beállíthatja, hogy a belépés után melyik héj induljon el a számára. Mindezt a **chsh** *(change shell*) programmal tudja megtenni. Körültekintőnek kell lennünk azonban a héjprogram megválasztásakor. Amennyiben olyan héjat választunk, amely nem létezik, vagy esetleg nem működik megfelelően, akkor kizárhatjuk magunkat a rendszerből. A *chsh* programot sem tudjuk futtatni, hogy a használt héjat visszaállítsuk. Annak érdekében, hogy ne „zárjuk ki magunkat" a chsh program segít nekünk. Mielőtt megváltoztatná a beállítást, a chsh ellenőrzi, hogy létezik-e a program, amelyet héjként használni akarunk, és azt is megvizsgálja, hogy szerepel-e a **/etc/shells** állományban.

A felhasználók csak olyan programokat állíthatnak be héjprogramként, amelyek szerepelnek a /etc/shells állományban.

A következő példa bemutatja, hogyan változtathatja meg egy felhasználó a héjprogramot, ami elindul, amikor bejelentkezik.

```
# chsh -s /bin/csh
Changing  shell for luke.
Password:
Shell changed.
```

*A program jelszót kért a felhasználótól, mielőtt elvégezte volna a módosítást. Természetesen ez az ellenőrzés kikapcsolható.*

A rendszergazda bármely felhasználó héjprogramját beállíthatja. Ez lehetőséget teremt a számára arra is, hogy az interaktív hozzáférést megtiltsa a felhasználó számára. Ha a rendszergazda olyan héjprogramot állít be a felhasználó számára, ami nem működik héjként, a felhasználó nem tud parancsokat kiadni, nem tud a számítógépre bejelentkezni.

**Az interaktív bejelentkezés tiltására általában a /sbin/nologin programot szokás használni**. Ha a rendszergazda ezt a programot állítja be a felhasználó számára, az nem tud bejelentkezni, de egyéb szolgáltatásokat elérhet, így például letöltheti állományait az ftp program segítségével, olvashatja elektronikus postafiókját távoli számítógépről és így tovább.

Példa az interaktív bejelentkezés letiltására:

```
# chsh -s /bin/nologin
Changing  shell for luke.
Shell changed.
```

**Személyes adatok megváltoztatása**

A **chfn** program segítségével a felhasználó megváltoztathatja a róla tárolt személyes adatokat (név, munkahelyi szobaszám, munkahelyi telefonszám és otthoni telefonszám). A rendszergazda bármely felhasználó személyes adatait megváltoztathatja.

chfn [kapcs.] [felhnév]

| *Kapcsoló* | *Jelentés* |
|---|---|
| *-í név* | A felhasználó nevének megváltói |
| -o *iroda* | A felhasználó irodájának megváltoztatása. |
| -p *telefon* | A felhasználó irodai telefonszámának megváltoztatása. |
| -h *telefon* | A felhasználó otthoni telefonszámának megváltoztatása. |

**Egyéb adatok megváltoztatása**

További változtatásokat a **usermod** program segítségével végezhetünk a felhasználói adatbázison. Ezt a programot használhatja a rendszergazda a felhasználó saját jegyzékének, elsődleges csoportjának és néhány egyéb adatának megváltoztatására.

usermod [kapcs.] felhnév

A rendszergazda a felhasználók minden adatát más programokkal is megváltoztathatja, hiszen írási joga van a felhasználói adatbázis állományaira. A legegyszerűbb módja az állományok megváltoztatásának a szövegszerkesztővel való módosítás. A szöveges adatállományokat megnyithatjuk és átírhatjuk bármelyik szövegszerkesztővel, így az adatbázist tetszésünk szerint módosíthatjuk.

Példa: # usermod -g (kókusörs) luke

## 6.7 Csoportok adminisztrációja

A felhasználókhoz hasonlóan a csoportok adminisztrálására is találhatunk segédprogramokat. Ilyen a **groupadd** és a **groupdel**, ami mint a felhasználóknál, itt is az új csoportok létrehozására, illetve törlésére szolgál.

A **groupadd** program is rendelkezik egy -r kapcsolóval, amellyel rendszer-csoportokat hozhatunk létre, amely ebben az esetben csak azt jelenti, hogy a group azonosító kisebb lesz a normál azonosítóknál.

## 6.8 /ectc/password fájl

A Unix/Linux rendszerek alapvető felhasználói adatbázisa a **/etc/passwd** szöveges fájl, amit jelszófájlnak (password file) nevezünk. Ez felsorolja az összes érvényes felhasználónevet és a hozzájuk kapcsolódó információkat. Amikor a rendszergazda felvesz egy új felhasználót, (többek közt) ebben a fájlban egy új sor keletkezik, a törlésekor a rá vonatkozó sor törlésre kerül.

Nézzük az alábbi minta sorokat egy korábbi típusú a passwd állományból:

```
root:HhzIK643GFhujMM:0:0:Rendszergazda:/root:/bin/bash
luke:K3xcO1Qnx8LFN:2332:1999:Luke Skywalker:/home/luke:/usr/local/bin/bash
—---|------------|----|----|-------------|----------|------------------
  1        2         3    4        5             6               7
```

A fájlban minden felhasználói névhez egy sor tartozik, és 7, kettősponttal elválasztott mezőre oszlik. Ezek:

**1. Felhasználónév (username)/ bejelentkezési név**

**2. Titkosított jelszó:** korábbi rendszerekben itt a kódolt jelszó szerepelt. Ma már nem itt tárolódik a kódolt jelszó, helyette csak egy x látható.

**3. Felhasználói azonosító szám (uid):** A felhasználót a rendszeren belül valójában nem a neve azonosítja, hanem ez a szám. Bár különböző rendszerekben eltérő szabályok vannak arra, hogy milyen tartományokban milyen jellegű felhasználókat definiálnak, egy általános "hétköznapi" Linuxban, a normál felhasználók számozása 1000-től kezdődik, az előre definiált felhasználóké 1-99 közt, a rendszer felhasználóké pedig 100-999 közt van. Figyeljük meg, hogy a fenti példában **a root felhasználó uid-je 0**!

**4. Csoportazonosító szám (gid):** a felhasználó csoportazonosítója

**5. Teljes név, vagy egyéb leírás:** Ebben a mezőben vesszővel elválasztva több adat is található (az irodalom gyakran GECOS-nak hívja ezt). Az elsőt mindig meg szoktuk adni, ez a felhasználó teljes neve. Ezután meg lehet adni a felhasználó szobaszámát, telefonszámát stb. Ezekre az információkra a **finger** nevű paranccsal lehet rákeresni egy Linux/Unix rendszerben.

**6. Home jegyzék**: a felhasználó saját mappája. Minden hagyományos (nem rendszer) felhasználónak általában van ilyenje, ebben szabadon tárolhat fájlokat, hozhat létre könyvtárakat.

**7. Bejelentkezési burok (login shell)**: Ez a program indul el a sikeres bejelentkezést követően. Ez gyakran egy parancsértelmező (shell), de nem feltétlenül kell annak lennie, bármilyen program meghatározható itt. Amikor a felhasználó ebből kilép, a bejelentkezési folyamat is véget ér.

A rendszer minden felhasználója olvashatja a jelszófájlt, így pl. megtanulhatják a többi felhasználó nevét. Ez korábban azt jelentette, hogy még a jelszó (a második mezőben) is mindenki számára elérhető volt. Igaz, itt csak egy titkosított változata van. Azonban a titkosítás feltörhető, különösen gyenge jelszavak esetén (pl. ha az túl rövid, vagy benne van egy szótárban). Ezért nem jó ötlet, hogy itt vannak a titkosított jelszavak.Az újabb rendzserekben már a **shadow** fájlt használják erre.

Ma már tehát a **/etc/shadow** fájl tartalmazza a felhasználók kódolt jelszavait, és egyéb kiegészítő adatokat. Ennek tartalmát csak a root olvashatja. Ekkor a /etc/passwd fájl csak egy speciális jelet("x") tartalmaz a második mezőben. Minden program, melynek egy felhasználó azonosításával kell törődnie, el kell érje az árnyék jelszófájlt. A szokásos programok elérhetnek a jelszón kívül minden információt az eredeti jelszófájlból, de magát a jelszót nem. Az /etc/passwd file-hoz hasonlóan az is egy egyszerű szöveges állomány, ahol mindegyik felhasználó egy sort foglal el. Itt is kettőspont választja el az adatmezőket.

Árnyék fájl esetén a korábbi passwd bejegyzés a következőre módosul:

```
root:x:0:0:Rendszergazda:/root:/bin/bash
luke:x:2332:1999:Luke Skywalker:/home/luke:/usr/local/bin/bash
```

Az /etc/shadow tartalma ekkor:

```
root:$1$q59x0VBc9KL$J:14435:0:Rendszergazda:::::
luke:$6$2j6Geq78PU$Hdj1FVC:14437:1999:0::::37:
————|———————————————|—————|————|—|—|—|—|
  1            2             3    4   5 6 7 8
```

**1. Login név:** A felhasználó bejelentkezési neve. Ez köti össze a /etc/shadow adott sorát az /etc/passwd megfelelő sorával.

**2. Kódolt jelszó:** A kódolt jelszó

**3. Utolsó csere:** A jelszó cseréje óta az itt látható számú nap telt el 1970. január 1-je óta.

**4. Minimum:** Két jelszócsere közt minimálisan ennyi napnak kell eltelnie.

**5. Maximum:** A jelszó maximum ennyi napig érvényes, utána ki kell cserélni. A 99999 jelentése 273 évente, a gyakorlatban soha nem kell cserélni a jelszót.

6. Figyelmeztetés: A jelszó lejárta előtt ennyi nappal küld a rendszer figyelmeztetést.

**7. Inaktivitás:** Ha a felhasználó ennyi napon át nem jelentkezik be, a hozzáférését a rendszer letiltja. Az üres érték azt jelenti, hogy nincs ilyen lejárat beállítva.

**8. Lejárat:** A hozzáférés eddig érvényes, ezt is az 1970. január 1-je óta eltelt napok számával kell megadni. Az üres érték azt jelenti, hogy nincs ilyen lejárat beállítva.

Amennyiben nem rendszergazdaként szeretnénk elérni a fájlt:

```
voxel@Pandora:~$ cat > /etc/shadow
bash: /etc/shadow: Permission denied
```

Ha megnézzük a passwd fájlt közelebbről, akkor felfedezhetjük, hogy a rendszerfelhasználók bejelentkezési shell-re általában /usr/bin/nologin. Ez nem véletlen, azt jelenti, hogy ezeknél a felhasználóknál nem lehetséges a távolról való bejelentkezés. Nincs értelme, valamint biztonsági kockázatot is jelentene, hiszen ezen felhasználók létezésének célja az, hogy a rendszeren belüli folyamatokat tudjanak elindítani.

Az alábbi felhasználó egy tipikus példa erre:

```
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

A **www-data** felhasználó általában a webes kiszolgálók és az ezekhez kapcsolódó programok futtatási jogkörért felelős. Tehát az apache webszerver, vagy a PHP ezen a felhasználón “keresztül” fog működni.

### 6.8.1 Jelszó felépítése

A Unix/Linux rendszerek lehetővé teszik, hogy a jelszó kódolási algoritmust megváltoztassák, így rendszerenként eltérő megoldással találkozhatunk. Általánosan azonban elmondható, hogy a kódolt jelszó több részből áll melyet **$** karakterek választanak el egymástól három vagy négy mezőre mezőre (függ az algoritmustól).

Tekintsük példának az alábbi shadow fájl bejegyzést és jelszót:

```
voxel:$y$j9T$ntSMxZ.iBQg2U6AUCt/Du.$.wOW3QB/JdHQEdZkuo/UoVYkgJYDCA7NKsoLVh3aT7.:19243:0:9
9999:7:::
```

Ebből a jelszó:

```
$y$j9T$ntSMxZ.iBQg2U6AUCt/Du.$.wOW3QB/JdHQEdZkuo/UoVYkgJYDCA7NKsoLVh3aT7.
--|---|----------------------|-----------------------------------------
 1  2            3                                4
```

Ebben a példában tehát 4 mezőre oszlik fel a jelszó. A jelentések az alábbiak:

**1. Prefix:** Azaz az alkalmazott algoritmus. Jelen példában ez példánkban "`$y$`", azaz **`yescrypt`** algoritmus került alkalmazásra (Mint Linux).

**2. Options:** algoritmus specifikus mező, nem mindig létezik. Jelen példában `($j9T$)` a yescrypt valamilyen opcióját jelenti.

**3. Salt:** A salt (só) a jelszóbiztonság növelésére szolgál. A rendszer ezt a felhasználó létrehozásakor véletlenszerűen generálja, és a jelszavához illeszti. A bejelentkezéskor szintén ugyanez történik, a felhasználó megadja a jelszavát, ehhez a rendszer újra hozzáilleszti a jelszót védő salt-ot, majd ezt kódolva kell a harmadik mezőben levő kódolt formát (lenyomatot) kapnunk.

**4. Hash:** A kódolt jelszó hash-elt lenyomata.

# 7. Bejelentkezés a rendszerbe

A továbbiakban röviden megvizsgáljuk a felhasználói bejelentkezés folyamatát, hogy mi is történik a háttérben.

Egy felhasználók többféleképpen is bejelentkezhetnek egy Unix/Linuxot futtató számítógépre:

- A legegyszerűbb esetben a számítógép előtt ülve valamelyik karakteres munkafelületen gépelik be a felhasználói nevüket és a jelszavukat.
- A felhasználók bejelentkezhetnek grafikus felületen is.
- A bejelentkezés történhet számítógép-hálózaton keresztül is.

## 6.2 A klasszikus bejelentkezés folyamata

A Unix rendszerek bejelentkezési folyamata többé-kevésbé megegyezik, azaz a bejelentkezés általában hasonlóképpen megy végbe minden UNIX rendszeren.

### Üzenetek a bejelentkezéskor

A bejelentkezés előtt a Unix rendszerek a **/etc/issue** állomány tartalmát írják ki a helyi képernyőre. (Távoli bejelentkezés esetén a felhasználó képernyőjén a **/etc/issue.net** állomány tartalma jelenik meg.) Ezt az állományt a rendszergazda a bejelentkezéshez szükséges információk kiíratására használhatja.

A /etc/issue.net állományban gyakran helyeznek el a számítógépre, a rajta található operációs rendszerre vonatkozó információkat. Ez biztonsági szempontból nem szerencsés, mert így az esetleges támadó minden erőfeszítés nélkül információkat szerezhet a számítógépen futtatott szoftverrendszerről, annak gyenge pontjairól.

Néhány Linux disztribúció esetében a /etc/issue és a /etc/issue.net rendszerindításkor felülíródik. Ha az állományokat állandó tartalommal kívánjuk feltölteni a /etc/rc .d/rc. local vagy a /etc/init. d/rc. local állományok megváltoztatásával gondoskodnunk kell arról, hogy az állományokat a rendszer a rendszerindításkor ne írja felül.

Továbblépve, **a /etc/motd** állomány tartalma a sikeres bejelentkezés után jelenik meg, akkor, amikor a felhasználó a hiteles bejelentkezési nevét és jelszavát beírta, és annak ellenőrzése megtörtént. A rendszerre jellemző információkat inkább ebbe az állományba tegyük, hiszen ezt csak azok látják, akik érvényes névvel és jelszóval rendelkeznek. A misztikus MOTD rövidítés mögött a „napi hír" *(message of the day)* kifejezés áll.

**Na és hogyan történik a beléptetés?**

A felhasználó beléptetését Unix rendszereken a **login program** végzi. Az, hogy milyen program indítja a login programot, az egyes Unix rendszereken, sőt az egyes Linux terjesztésekben is meglehetősen változó.

A login fő feladata a felhasználó azonosítása és a felhasználó alapértelmezett héjprogramjának elindítása az azonosítás után. A login ezt a feladatot a következő lépések elvégzésével látja el:

1. Ha a login program indulásakor még nem ismert a felhasználó felhasználói neve, a login megkérdezi.
2. Ha az adott felhasználó bejelentkezése jelszóhoz kötött, a login megkérdezi a jelszót. A jelszó begépelésekor a képernyőn *semmi* nem jelenik meg, hogy ne lehessen kikémlelni a jelszót és azt sem, hogy az hány karakterből áll.
3. Ha a bejelentkezés minden feltétele adott, a login elindítja az azonosított felhasználó alapértelmezett héjprogramját. A héjprogram elindítását a program úgy végzi, hogy az már az azonosított felhasználó nevében fusson, az azonosított felhasználó jogaival rendelkezzen.

4. Ezek után a login vár, amíg a héj fut. Amikor a felhasználó kilép a héjból, a login is kilép, hogy az őt indító program tudja, új felhasználó jelentkezhet be.

A login a Unix rendszereken néhány apró feladatot is elvégez: ha a **/etc/nologin** állomány létezik, akkor csak a rendszergazda léphet be. Minden más felhasználó belépési kérelmét elutasítja a login, az elutasítást a /etc/nologin tartalmának kiírásával jelezve. Ha tehát a felhasználók bejelentkezését valamilyen okból kifolyólag meg akarjuk akadályozni, az okot egyszerűen be kell írni.

Példa: # echo "Karbantartás miatt a szerver nem üzemel!" > etc/nologin

A login bejelentkezéskor megvizsgálja, hogy a **/var/spool/mail/** jegyzékben van-e a felhasználó felhasználói nevével megegyező nevű állomány és az 0 hosszúságú-e. Ha nem 0 a hossza az állománynak, akkor kiírja a képernyőre, hogy a felhasználónak olvasatlan levele van.

Nyilvánvaló, hogy a levéltovábbító alrendszer a beérkezett elektronikus leveleket a /var/spool/mail/ könyvtárban tárolja, minden felhasználó számára egy állományt tartva fenn. Az is magától értetődik, hogy a

felhasználói levelezőprogram az olvasott elektronikus leveleket eltávolítja innen, így ha az állomány nem üres, a felhasználónak olvasatlan levele van.

Ha a felhasználó a **.hushlogin** *(hush,* csendes, nyugodt) rejtett állományt elhelyezi a saját könyvtárában, akkor a belépéskor a login nem ellenőrzi a levelesládát, és nem írja ki az utolsó belépés időpontját.

A login következő feladata a bejelentkezés naplózása. **A Unix rendszerek a wtmp és utmp állományokban rögzítik a felhasználók bejelentkezéseit, így a login is ezekben az állományokban rögzíti a bejelentkezéseket és a kilépéseket.** A **w** és a **who** programok is ezt a nyilvántartást használják a pillanatnyilag belépett felhasználók listájának kiírására.

A login szerencsés esetben túljutva ezeken az adminisztratív feladatokon elindítja a felhasználó héj programját, azaz kiolvassa a felhasználói nyilvántartásból, hogy milyen héjat használ a felhasználó, és elindítja azt. Azt gondolhatnánk, hogy ez már nem lehet túlságosan bonyolult, sajnos azonban a héj indítása korántsem egyszerű folyamat.

Tovább bonyolítja a helyzetet az, hogy UNIX rendszereken sokféle héj létezik és ezek viselkedése már induláskor különbözhet.

## 6.3 A héj indulása

A Linux rendszerekben a legelterjedtebb héj a BASH héj. Igen sok GNU/Linux-felhasználónak a BASH az alapértelmezett héjprogramja. A BASH alapjában véve **három üzemmód**dal rendelkezik, háromféleképpen indulhat el:

- **Beléptető héj üzemmód *(login shell mode):*** ez az üzemmód kimondottan a belépéskor való indításra készült erőforrás-takarékossági okokból; akkor szokás használni, amikor a felhasználó belépésekor alapértelmezett héjként indul a BASH.
- **Interaktív héj üzemmód *(interactive shell mode)*:** az interaktív kifejezés arra utal, hogy a BASH akkor használja ezt az üzemmódot, amikor közvetlenül a felhasználóval áll kapcsolatban, a felhasználótól kapja a parancsokat. Ez természetesen nem jelenti azt, hogy a *beléptető üzemmód* nem interaktív jellegű! Az *interaktív üzemmód* egyik jellemzője az, hogy interaktív kapcsolattartást valósít meg, de más üzemmódban is használható interaktív kapcsolattartással a program.
- **Nem interaktív héj üzemmód *{non-interactive shell mode)*:** ez az üzemmód szöveges állományban elhelyezett programok futtatására szolgál. Ilyenkor a BASH nem egy felhasználóval tartja a kapcsolatot, hanem egy állományból veszi az utasításokat, és ennek megfelelően kissé másképp viselkedik.

A BASH beléptető üzemmódban indul, ha indításkor a nevének az első betűje a „-" karakter vagy ha a -login kapcsolót kapja. Ha a két feltétel közül legalább az egyik fennáll, a BASH mindenképpen beléptető héj üzemmódban indul el. A login program a BASH héjat GNU/Linux rendszereken **-bash** néven indítja, ahogyan ezt a következő példa mutatja:

```
# w
08:43:47 up   2:03,   4 users,   load average:  0,01, 0,03, 0,06
USER   TTY   FR0H     LOGIN@  IDLE    JCPU    PCPU    WHAT
luke   tty1   -        07:15   l:07m   0.27s   0.27s   -bash
```

Ha a BASH nem kapott -login kapcsolót, és indításkor a neve nem „-" karakterrel kezdődött, a héj nem beléptető üzemmódban indul el. Ilyenkor a program megvizsgálja, hogy a bemenete és kimenete terminálhoz kapcsolódik-e. Ha a bemenet és kimenet terminálhoz kapcsolódik, a program feltételezi, hogy a terminált egy felhasználó használja, és interaktív üzemmódban indul el.

Ha a BASH nem beléptető üzemmódban és nem is interaktív üzemmódban indul, akkor az indulás nem interaktív üzemmódban történik.

A BASH induláskor különféle állományokat keres, olvas és hajt végre attól függően, hogy milyen üzemmódban indult el. A rendszergazdának ismernie kell ezeket az állományokat, hiszen az induláskor betöltött és végrehajtott állományok segítségével rugalmasan testre szabhatja a felhasználók munkakörnyezetét.

Amikor a BASH beléptető üzemmódban indul el, a következő állományokat olvassa és hajtja végre induláskor:

**/etc/profile:** ha a /etc/profile állomány létezik, a BASH először ezt tölti be. Ezt az állományt a rendszergazda általában a mindenkire érvényes beállítások érvényesítésére használja.

**$HOME/.bash_profile:** ha létezik a felhasználó saját könyvtárában a .bash_profile állomány, akkor második lépésben ezt az állományt tölti be és hajtja végre a BASH. A felhasználók ebben az állományban testreszabhatják munkakörnyezetüket.

**$HOME/.bash_login:** ha a BASH nem találja a .bash_profile állományt a felhasználó saját jegyzékében, akkor megkísérli ugyaninnen betölteni a .bash_login állományt.

**$H0ME/.profile:** ha a .bash_login állomány sem létezik, akkor a BASH megkísérli betölteni a .profile állományt.

**$HOME/.bashrc:** a BASH ezek után mindenképpen megkísérli, hogy betöltse és végrehajtsa a felhasználó saját könyvtárából a . bashrc állományt.

**$HOME/.bash_logout:** a BASH kilépéskor megkísérli betölteni és lefuttatni ezt az állományt. Kitűnően használható arra, hogy kilépéskor ideiglenes állományokat töröljünk, letöröljük a képernyőt, stb.

# 8. Az X grafikus felület

## 7.1 Az X Window System röviden

Manapság ha egy operációs rendszer versenyképes szeretne lenni a piacon, akkor kétség kívül szüksége van egy grafikus kezelői felületre (GUI). A GUI felület számos feladatra egyértelműen barátságosabb és könnyebben kezelhető, mint egy szöveges terminál.

A Unix/Linux rendszerek grafikus felülete nagyon eltér a Windows rendszerekben megismert grafikus felülettől. Maga a rendszer egy nyitott szabványon alapul, amelyet **X Window System** (vagy egyszerűen **X**) –nek neveznek, amely hasonlóan sok más Unix-os fejlesztéshez egy akadémiai projekt (MIT – Athena projekt) eredménye.

Az X szabványt az **X Consortium** (http://www.x.org) vezeti, nagyon sok operációs rendszerre írtak X megvalósítást, sok programcsomag látott napvilágot ezen szabvány alapján. Ezen programok a szabvány alapján hiba nélkül képesen egymással kommunikálni, ezáltal a grafikus programok kimenete

platform-független lett. A Linux egyik leginkább elterjedt X megvalósítását az **Xfree86** szervezet (http://www.xfree86.org) készítette

Az X egy kliens-szerver alapú architektúrájú rendszer, ahol a kliens és a szerver tud hálózaton keresztül is kommunikálni, úgynevezett **X protokoll**on keresztül. A grafikus képernyőt egy szerverprogram kezeli, és minden program, amely írni vagy rajzolni szeretne a képernyőre, ezzel a szerverrel kommunikál. Ezek a programok a kliensek. A szerver mindig a lokális gépen fut. Az X protokollt TCP/IP vagy DECnet fölött lehet használni.

Ennek a megoldásnak az előnye egyértelmű: a kliensnek nem kell az adott eszköz jellemzőivel foglalkoznia, csak a szerver kommunikációt kell implementálnunk. Ez pedig nem különbözik attól, mintha csak egy grafikus könyvtárat használnánk.

### 8.1.1 Az X belső működése

**Az X protokoll (Xlib implementációja) aszinkron.** Az Xlib a kéréseket egy belső adatbufferben időlegesen eltárolja, és nem küldi egyből a szervernek (ezzel is csökkentve a hálózati forgalmat). A kérések a következő esetek valamelyikében kerülnek továbbításra a szerver felé:

- A kliens egy adott típusú esemény bekövetkeztére vár, de a várt típusú esemény még nem következett be, nem érkezett meg a szervertől róla információ. Ekkor minden kérés azonnal továbbítódik a szerver felé, hátha ezek között a kérések között van az, amelyik a várt eseményt kiváltja.
- Egyes szervertől jövő üzenetek azonnali választ igényelnek a kliens programtól. Egy ilyen üzenet vételekor a kliens Xlib része azonnal válaszol a szervernek, de az üzenetek helyes sorrendjének betartása érdekében az összes addigi (de még a bufferben levő) üzenetek is el lesznek küldve.
- Van több olyan Xlib függvény, amely ab üríti (ilyen például az XFlush() és az XSynch()). Ezeket a függvényeket akkor kell használni, ha a program hosszú ideig rajzol úgy, hogy közben nem vár semmiféle eseményt. (Ha ezt elfelejtenénk, akkor lehet, hogy hosszú ideig nem látszana semmi változás a szerver képernyőjén).

Az aszinkronitásnak egy további, hátrányos következménye, hogy nagyon nehéz az X Window rendszert használó programokat nyomon követni, mert lehet, hogy valamely hibaüzenet több Xlib függvény meghívásával később jut el a klienshez. Tehát nem biztos, hogy a hibát az az Xlib függvény okozta, amely után az jelentkezett.

A rendszer hálózat-orientált, és ezen azt értjük, hogy a programokban használt grafikai rutinok nem közvetlenül a képernyő memóriát manipulálják, hanem a hálózaton, hálózati csomagok formájában lesznek elküldve annak a gépnek, amelynek meg kell jelenítenie az eredményt.

**A szabvány nem tartalmaz ablakkezelő stratégiát**, ezek ellátása egy speciális kliens, az ablakkezelő (window-manager) feladata. Egy átlagos kliens létrehoz egy ablakot a képernyőn, azon belül létrehozhat ablakokat (önmagán belüli területeket megváltoztathat), de az önmagán kívüli területeket nem változtathatja meg. Az ablakkezelő ezenkívül megteheti azt, hogy az ablakot egy meghatározott kerettel veszi körbe, és a képernyő egész részét birtokolhatja. Egy X szerverhez bármennyi X kliens csatlakozhat.

Mivel az egyes hardverek különbözőek lehetnek, a szabványnak ennek kezelésére is ki kell terjednie. Támogatja a mono és a színes monitorokat, 8-16-24-32 bps színmélységig. A felbontásnak csak a hardvereszközök szabnak határt. A mutatóeszköz lehet egér, toll vagy érintőképernyő. Az egerek 5 gombig támogatottak. Billentyűzetből is többfélét támogat, a billentyűzetkiosztás szoftverből állítható. A munkaterület több monitoron is elhelyezkedhet egyszerre.

### 8.1.2 Kliensalkalmazások és ablakkezelők

Az X-es alkalmazások készítésénél a programozók úgynevezett **widget** (Egy widget lényegében egy előre gyártott felhasználói interfész alkotóelem) könyvtárakat használnak. Ezek a programkönyvtárak tartalmazzák az ablakok felületén megjelenő különböző vezérlő elemeket (pl. gombok, edit mező, lista mező, stb.). Mivel ezek működése eltérően lehet implementálva az egyes programkönyvtárakban, ezért a programok viselkedése nagyban függ a fejlesztő által választott widget könyvtártól.

Az első *Widget* könyvtár az Athena projekt keretében kifejlesztett Athena könyvtár volt. Csak a legalapvetőbb elemeket tartalmazza, és a kontroll elemek kezelése eltér a manapság használatosaktól. Ezt követő legismertebb könyvtár az Open Software Foundation (OSF) **Motif** csomagja volt. 1980-tól a korai 1990-es évekig volt igen elterjedt, de a mai napig sok helyen használják. A legkomolyabb hibája, hogy súlyos összegekbe kerül a fejlesztői licence. Manapság már vannak jobb alternatívák árban, sebességben, szolgáltatásokban. Ilyen például a **Gtk,** amely a GIMP projekthez készült. Aránylag kicsi, sok szolgáltatással, bővíthető, és teljesen ingyenes, illetve egy másik népszerű *toolkit,* a Qt, amely a KDE projekt óta ismert igazán, mivel a KDE alapját szolgáltatja. A Qt forráskódja nem, de a használata ingyenes. További alternatíva a **LessTif,** amely egy ingyenes helyettesítője a Motif-nak.

Azonban mivel minden X kliens alkalmazás alapvetően az Xlib-en alapul, ezért egyes opciókat egységesen kezelnek.

### 8.1.3 A Desktop környezet

Felhasználóként tehát választhatunk ablakkezelőt kedvünk szerint, és a programok írói is választhatnak *widget* könyvtárakat. Azonban ennek a nagy szabadságnak megvannak a hátrányai:

- A kliens program írója határozza meg, hogy milyen *widget* könyvtárat használ. Azonban mivel e könyvtárak kontroll elemei nagyban különbözhetnek, előfordulhat, hogy ahány programot használunk, annyiféle módon kell kezelni.
- Ezek a *widget* könyvtárak általában dinamikusan linkelődnek a programokhoz, azonban ha a kliens programjaink különbözőket használnak, akkor az plusz memóriát igényel.
- Az ablakkezelők is sokfélék, különböző kezelési koncepciókkal.
- Az egész felület ezeknek következtében nem áll össze egy egységes egésszé, például nem lehet egyben konfigurálni a kinézetét, kezelését.

Ez azért alakult így, mert az X kialakítása során a flexibilitásra, és a felhasználók szabadságára helyezték a hangsúlyt, ellentétben egyéb operációs rendszerek (például Windows, MacOS) megkötöttségeivel, zártságával. Mindez azonban a fent említett helyzethez vezetett, ezért előtérbe került a komplex felület létrehozása a szabadsággal szemben. Ez a helyezet vezetett az **Asztali Környezetek** (Desktop Environment) kialakulásához.

Az Asztali Környezet tartalmaz szolgáltatásokat és módszertanokat, amellyel a felület egységesíthető és a problémák leküzdhetőek. Ugyanakkor több fajtája is létezik, és a választás szabadsága ez által megmarad. Néhány ismertebb: CDE (Common Desktop Environment), KDE (K Desktop Environment), GNOME, XFCE, Enlightenment, Fluxbox, OpenBox, BlackBox, stb.

## 8.2 Az X elindítása

Az X *session* elindítására két metódus közül választhatunk. Amennyiben a gépünk nem futtat folyamatosan X szervert, ami egy szerver gép esetén gyakori, akkor a **startx** paranccsal indíthatjuk el és hozhatunk létre egy *session*-t.

A másik lehetőség, amikor egy **xdm *(X Display Manager)*** nevű szolgáltatás (vagy ennek egy variánsa) folyamatosan fut a gépünkön (daemon process). Ez a munkaállomások eseten gyakori. Az xdm menedzseli az X felületet. Egy authentikációs szolgáltatást nyújt, ahol bejelentkezhetünk és elindítja számunka a *session-X.* A modern ablakkezelőknek köszönhetően többen megalkották a maguk „xdm"-jét. Például a KDE grafikus környezet esetén a **kdm**, és a GNOME esetében pedig a **gdm** veszi át ezt a feladatot.

### 8.2.1 A startx működése

A startx valójában *egy front-end* az **xinit** program szamara. Egy kicsit barátságosabb felületet nyújt számunkra, mint az xinit parancs formátuma. Az xinit futtatja a $HOME/.xinitrc *script-et.* Ha ez nem létezik, akkor a rendszerszintű xinitrc script indul el (/usr/lib/x11/xinit/xinitrc vagy /etc/x11/xinit/xinitrc). Ez az állomány elindíthat X kliens applikációkat és a *window manager*-t, de jelenlegi konfigurációkban ezt az *Xclients* script teszi.

Az .xinitrc futatja a $HOME/.Xclients *script-et,* vagy ha ez nem létezik, akkor a rendszer szintű változatát (/etc/x11/xinit/Xclients). Ennek feladata az X kliensek és a választott *window manager* indítása.

### 8.2.2 Az X session indulása

Az xdm használata esetén belépéskor hasonló folyamatok játszódnak le, mint a startx esetén. Az xdm létrehoz egy új session-t. Lefuttatja rendszerszintű Xsession script-et (/usr/lib/X11/xdm/Xsession vagy /etc/X11/xdm/Xsession). Ez a script futtatja a felhasználó $HOME/.xsession script-jét. Ha nem létezik az .xsession script, akkor a $HOME/.Xclients vagy az /etc/X11/xinit/Xclients script hajtódik végre. Ezekben a script-ekben indíthatjuk el az X kliens applikációkat és a window manager-t.

## 8.3 X használata távoli kliensekkel

Korábban említésre került, hogy az X architektúrájában lehetőség van rá, hogy az X szerver és a kliens program két különböző gépen fusson, és a hálózaton keresztül kommunikáljanak. Ez az X protokoll-on keresztül zajlik. Lassú kapcsolatok esetén a kommunikációt tömöríteni is lehet. Amennyiben használni szeretnénk ezt a funkcionalitást, akkor a következő képen kell eljárnunk:

1. A lokális géppel (X szerver) közölnünk kell, hogy fogadja a távoli gép kapcsolódását.
2. A távoli géppel (kliens program) közölnünk kell, hogy a kimenetét a lokális gépünkre irányítsa.
3. Lehetséges, hogy egy gépen több X szervert használjunk, vagy több felhasználó által futtatott programok egy X szerveren jelenjenek meg. Ezeket is a távoli elérésnél használatos módszerekkel szabályozhatjuk.

Az X szerver nem fogad kapcsolódást bárhonnan. Ha nem így lenne, az biztonságilag komoly problémákat vetne fel. Bárki bármit odarakhatna a képernyőnkre, és monitorozhatná a billentyű leütéseinket, egér-eseményeket, stb. Az X Window kétféle authentikációs módszert támogat:

- a host listát használó mechanizmus (**xhost**),
- és a *magic cookie*-t használó **xauth**

**Xhost esetén**

Ebben az esetben az X szerver egy listát kezel a kapcsolódásra jogosult gépek neveiről. Ehhez a listához hozzáadhatunk és elvehetünk gép neveket:

```
Xserver-host># xhost +xclient.valahol.hu
Xserver-host># xhost -xclient.valahol.hu
```

De akár teljesen ki is kapcsolhatjuk (mindenkit engedélyezünk), de ez természetesen nem javasolt, mert a fentebb említett biztonsági problémákat veti fel.

```
Xserver-host># xhost +
```

Visszaállítása:

```
Xserver-host># xhost -
```

Az xhost, mint látható, nem biztonságos megoldás. Egyrészt nem különbözteti meg az adott gépen lévő felhasználókat, másrészt gépnév alapján dolgozik, ami félrevezethető. Csak olyan hálózaton célszerű használni, ahol megbízunk a felhasználó-társainkban.

**Xauth esetén**

Az xauth azon klienseknek engedi a kapcsolódást, amelyek ismerik a titkos jelszót (anthorization record vagy magic cookie). A *cookie*-k az ~/.Xauthority állományban tárolódnak. (Vigyázni kell a jogok beállításánál, hogy csak a tulajdonos olvashassa.) Ezt az állományt az xauth program kezeli.

Az xauth protokoll használatához az X szervert az -auth <authfile> paraméterrel kell indítani. Kezelése nehézkesebb, mint az xhost-os megoldásé. Ugyanakkor az authentikációs kommunikáció továbbra is kódolatlanul zajlik a hálózaton, ezért a biztonság szempontjából érdemes lehet egy kódolt csatornával kiegészíteni: ssh.

### 8.3.1 Az X kliens kimenetének átirányítása

A DISPLAY környezeti változón keresztül adhatjuk meg az X applikációknak, hogy melyik X szerverhez kapcsolódjanak.

```
DISPLAY=:0
DISPLAY=localhost:1
DISPLAY=x.valahol.hu:0
```

A DISPLAY változó egy host névvel kezdődik, ami opcionális. Követi egy ":". Utána az X szerver sequence száma következik. Ha egy gépnek több display-e van, vagy csak több X szervert futtatunk, akkor ez az indexe az azonosításhoz.

A kliens programok induláskor megvizsgálják a DISPLAY környezeti változót, hogy hova kell kapcsolódniuk. Azonban ezt felülírhatjuk a -display paraméterrel, ami után ugyanúgy kell megadnunk a display leírását, mint korábban a DISPLAY változónál.

Egy rövid példa:

```
Xclient>#   export DISPLAY=x.valahol.hu:0
Xclient>#   xterm &

Xclient>#   DISPLAY=x.valahol.hu:0 xterm &
Xclient>#   xterm -display x.valahol.hu:0 &
```

Használhatjuk az **rsh** ***(remote shell)*** parancsot, vagy a biztonságosabb **ssh** ***(secure shell)*** programot.

```
# rsh  xclient.valahol.hu  xapplikáció  -display   szerver:0
```

### 8.3.2 X átirányítása ssh segítségével

A hagyományos megoldásokhoz képest a legegyszerűbb és legbiztonságosabb megoldást az ssh program *X átirányítás (forward)* funkciója nyújtja. Ez lehetővé teszi, hogy amennyiben a gépünkön X-et használunk, és az ssh programmal belépünk egy távoli gépre, akkor a *shell*-ből indított programok kimenete automatikusan az X-es felületünkre irányítódik át egy titkosított csatornán keresztül. Ehhez a felhasználónak semmilyen beállítást nem kell elvégeznie

A kapcsolat felépítéséhez az ssh automatikusan létrehoz egy **Xauthory** állományt egy véletlen jelszóval. Később ezzel azonosítja az átirányított kapcsolatokat.

**Megjegyzés:** Az ssh ezen funkciójának használatához az *ssh*-szervernek támogatnia kell az X kapcsolatok átirányítását. A konfigurációs állományban (~./ssh/config) a *FonwardX11* opciónak engedélyezve kell lennie.

## 9. Linux hálózatba kapcsolása

Ahhoz, hogy számítógépünket valamely hálózatba szeretnénk kapcsolni, szükség van a hálózati eszköz kernel támogatottságára. Ha az aktuális kernelünk nem támogatja a hálózati eszközünket, akkor nincs lehetőségünk hálózatra kapcsolódni. Ebben az esetben egy kernel frissítés megoldhatja a problémát. (lásd később)

Jelenleg a Unix/Linux kernel több, mint 50 féle hálózati kártyához nyújt támogatást. Az elterjedtebb kártyákat támogatja a Linux, az újabb eszközök támogatása attól függ, hogy a hardver gyártója kiadja-e az adott hardver specifikációját.

A Unix rendszerek alatt a hálózati eszközök leírói gyakran a /dev jegyzékben vannak. Azonban a Linux nem tartozik ezek közé. Linux alatt a hálózati eszközök dinamikusan a kernelben jönnek létre, nem tartozik hozzájuk eszköz állomány. Az esetek többségében automatikusan hozzák létre az eszközvezérlőket. A Linux szabványos neveket használ a hálózati interfészek megkülönböztetésére. Mivel többet is használhatunk a

hálózati eszközökből, ezért számozza őket a rendszer, az első eszköz száma 0-s (például eth0). Ezek a nevek a kernelben vannak tehát definiálva, és nem eszközfájlok a /dev könyvtárban.

Az interfészek elnevezése általában az eszköz típusától függ (Linux):

- **lo: a visszahurkoló eszköz (loopback device).** Bármely ide küldött datagramm azonnal visszakerül a gép hálózati rétegéhez. Hálózati alkalmazások kifejlesztésekor, ellenőrzésekor nagyon hasznos, de sok hálózati alkalmazás is használja. Például az összes RPC-n alapuló alkalmazás ezt használja induláskor, hogy regisztrálja magát a ***portmapper*** démonnál.
- **eth0, eth1,…,ethx: hálózati (Ethernet) kártyák.** A kernel csak egy hálózati kártyát próbál meg érzékelni, ezért ha egynél többet szeretnénk használni, akkor azt explicit módon a kernel tudomására kell hoznunk.
- **wlan0, wlan1,...,wlanx: vezeték nélküli hálózati(Wireless) kártyák.**
- **sl0, sl1,…,slx: SLIP interfészek.** A soros portokon keresztüli hálózati kapcsolatokat teszik lehetővé.
- **plip0, plip1,…,plipx: PLIP interfészek.** A párhuzamos portokon keresztüli hálózati kapcsolatokat teszik lehetővé.
- **ppp0, ppp1,…,pppx: PPP interfészek.** A soros portokon keresztüli hálózati kapcsolatokat teszik lehetővé.
- **tr0, tr1,…,trx:** Token Ring hálózati kártyák.

Összehasonlításként célszerű megemlíteni a BSD rendszerek interfész kezelését is. A Linux esetén alkalmazott általános ethX alakú azonosítók helyett például a FreeBSD az adott hálózati kártya meghajtójának nevével és utána egy sorszámmal hivatkozik.

Például két Intel® Pro 1000 hálózati kártya esetén **em0** és **em1**.

Ha valamely elterjedt kártyát használunk, akkor nem kell külön drivert keresni hozzá, mert az ismertebb hálózati kártyák meghajtói ugyanis alapból benne vannak a FreeBSD GENERIC rendszermagjában. Az elnevezés ilyenkor: **dc0**, **dc1**, stb, **plilp0:** a párhuzamos port felülete (amennyiben található párhuzamos port a számítógépben), **lo0:** a loopback eszköz.

Ahhoz, hogy használni tudjuk ezeket az eszközöket, a kernelnek tudnia kell azt, hogy hogyan kommunikáljon az adott eszközzel. Ezt biztosítják az úgynevezett illesztőprogramok. Ezek tudják, hogy hogyan kell utasításokat és adatokat küldeni az eszközre, az eszköz pedig ennek a programnak továbbítja a hálózatról kapott adatokat.

,,PC-kben ez a kommunikáció az I/O memória területén keresztül valósul meg, amely kártyán lévő regiszterekre és hasonlókra van leképezve. A kernel által a kártyára küldött minden parancsnak és adatnak át kell mennie ezeken a regisztereken. Az I/O memóriát általában a kiindulási vagy báziscímével adják meg.'' [11]

A kernel a hálózati kártya moduljának betöltésekor (vagy a rendszer indulásakor, ha a kártya nem modulba van fordítva) megpróbálja automatikusan megállapítani a kártya által használt memóriaterületet. Ezt úgy csinálja, hogy kiolvas néhány jellemző memóriahelyet, és a kapott adatokat összehasonlítja azzal, amit látnia kellene, ha ott lenne a hálózati kártya. Ez azonban nem minden esetben sikerül, ilyenkor meg kell mondanunk a kernelnek, hogy milyen címen található a kártya.

A másik paraméter, amit esetleg meg kell mondanunk a kernelnek, a kártya által használt IRQ.

A mai világban számos különböző gyártó gyárt hardver elemeket. Ilyenkor valójában a Unix/Linux rendszert telepítő személynek tudnia kell, hogy a telepítendő operációs rendszer támogatja-e a rendelkezésre álló hardvert. Ezt természetesen senki sem tudhatja pontosan fejből, ezért a közösség több internetes adatbázist hozott létre ennek támogatására. (Pl.: FSF - http://www.fsf.org/resources/hw, Linux-Hardwaredatenbank – http://www.linux-hardware.org, http://www.linux-drivers.org/)

## 9.1 A hálózati interfészek konfigurálása

A hálózati interfészek konfigurálását az átlag felhasználók főleg interaktív felületek segítségével végzik. Azonban sok esetben szükség arra, hogy tudjuk, mi zajlik a háttérben. A hálózati interfészek kernel rezidens részeinek kezelésére az **ifconfig** program szolgál. Segítségével aktiválhatjuk, vagy deaktiválhatjuk az interfészeket, illetve beállíthatjuk az interfészek jellemzőit: IP cím, netmask, stb. Paraméterek nélkül meghívva kilistázza az aktív interfészeket. Tanulmányozzuk a manuált a kapcsolók megismerése érdekében.

**Példa Linux esetén:**

```
root@Orion:/home/skywalker/Egyetem#> ifconfig

eth0   Link encap:Ethernet  HWaddr 00:1D:92:4D:9F:C7
       inet addr:169.254.215.144  Bcast:169.254.255.255  Mask:255.255.0.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0
       TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
       Memory:febc0000-febe0000

lo     Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:4 errors:0 dropped:0 overruns:0 frame:0
       TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:240 (240.0 b)  TX bytes:240 (240.0 b)

wlan0   Link encap:Ethernet  HWaddr 00:1D:E0:BD:AF:19
        inet addr:192.168.1.102  Bcast:255.255.255.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:576  Metric:1
        RX packets:712654 errors:0 dropped:0 overruns:0 frame:0
        TX packets:741057 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:328241629 (313.0 Mb)  TX bytes:260141398 (248.0 Mb)
```

( Kérdés: Mi olvasható ki ebből a példából? )

**Példa FreeBSD esetén:**

```
% ifconfig
```

```
dc0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80008<VLAN_MTU,LINKSTATE>
        ether 00:a0:cc:da:da:da
        inet 192.168.1.3 netmask 0xffffff00 broadcast 192.168.1.255
        media: Ethernet autoselect (100baseTX <full-duplex>)
        status: active
dc1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=80008<VLAN_MTU,LINKSTATE>
        ether 00:a0:cc:da:da:db
        inet 10.0.0.1 netmask 0xffffff00 broadcast 10.0.0.255
        media: Ethernet 10baseT/UTP
        status: no carrier
plip0: flags=8810<POINTOPOINT,SIMPLEX,MULTICAST> mtu 1500
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        options=3<RXCSUM,TXCSUM>
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x4
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
        nd6 options=8010<POINTOPOINT,MULTICAST> mtu 1500
```

Valamely interfész deaktiválása. Példa az *eth0* lekapcsolására:

```
# ifconfig eth0 down
```

Az *eth0* interfész újraindítása:

```
# ifconfig eth0 up
```

A hálózati csatolónak a működéséhez mindenféleképpen rendelkeznie kell IP címmel és alhálózati maszkkal, amelyet könnyedén beállíthatunk az ifconfig-al. Például állítsuk be egy alhálózati címet 255.255.255.254 maszkkal.

```
# ifconfig eth0 netmask 255.255.255.254 193.6.5.71
```

**Eredmény:**

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1D:92:4D:9F:C7
          inet addr:193.6.5.71  Bcast:193.6.5.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Memory:febc0000-febe0000
```

Amennyiben be is szeretnénk üzemelni, úgy az *# ifconfig eth0 up* paranccsal tehetjük meg.

### 9.1.1 Perzisztens IP beállítások

Az ifconfig-al elvégzett beállítások nem tárolódnak el automatikusan, a rendszer újraindításakor elvesznek. Célszerű ezért olyan megoldást keresni, amely perzisztensen képes tárolni a hálózati konfigurációkat.

**Végleges beállításhoz több lehetőség adódik:**

- az adott Linux disztribúció grafikus beállító felületén keresztül adjuk meg az IP címet, a netmaszkot és a broadcast címet
- a webmin hálózatbeállító modulját alkalmazzuk
- megkeressük a hálózati szolgáltatások konfigurációs fájljait és abban végezzük el a beállításokat (javasolt)

A hálózati konfigurációs fájlok helyei és nevei disztribúciófüggőek, így sajnos minden disztribúció esetén ezt fel kell kutatni. A fájlok felépítése és megértése azonban általában magától értetődő. Az alábbiakban a RedHat és Debian és FreeBSD alapú konfigurációs állományokra látunk példát:

**Redhat:** a hálózatot konfiguráló fájlok az /etc/sysconfig/network-scripts könyvtárban találhatóak ifcfg-<hálózati eszköz> formában. Az első Ethernet interfész fájlja az ifcfg-eth0, ezt vagy létre kell hozni vagy szerkeszteni.

**Példa:**

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.1.1
BROADCAST=192.168.1.255
NETMASK=255.255.255.0
NETWORK=192.168.1.0
GATEWAY=192.168.1.254
```

**Debian:** /etc/network/interfaces konfigurálja valamennyi interfészt.

**Példa:**

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 192.168.1.1
network 192.168.1.0
netmask 255.255.255.0
broadcast 192.168.1.255
gateway 192.168.1.254
```

**FreeBSD:** /etc/rc.conf állományba kell felvennünk a hálózati kártyák érvényes beállításait.

**Példa:**

```
ifconfig_dc0="inet 192.168.1.3 netmask 255.255.255.0"
ifconfig_dc1="inet 10.0.0.1 netmask 255.255.255.0 media 10baseT/UTP"
```

## 9.2 A vezeték nélküli hálózati interfészek konfigurálása

A vezeték nélküli interfészek konfigurálása nem sokban tér el a hagyományos Ethernet konfigurálásától. Az ifconfig parancsnál megfigyelhettük, hogy ha paraméter nélkül hívjuk meg, akkor kilistázza a rendelkezésre álló hálózati interfészeket. Jól megfigyelve láthattuk, hogy az utolsó interfész a tesztkörnyezetben a *wlan0* volt, amely egy vezeték nélküli hálózati interfész rendelkezésre állását jelenti. Az ifconfig utolsó bekezdése tehát:

```
# ifconfig
……
……

wlan0   Link encap:Ethernet  HWaddr 00:1D:E0:BD:AF:19
        inet addr: 192.168.1.102  Bcast:255.255.255.255  Mask: 255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:576  Metric:1
        RX packets:712654 errors:0 dropped:0 overruns:0 frame:0
        TX packets:741057 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:328241629 (313.0 Mb)  TX bytes:260141398 (248.0 Mb)
```

A példa jelenleg egy működő vezeték nélküli alhálózatot mutat be. A vezeték nélküli hálózat konfigurálására azonban külön parancsokat hoztak létre. Az ifconfig mellett az **iwconfig** parancsot alkalmazhatjuk a hálózat beállítására. Kiadva az utasítást, gyakorlatilag az ifconfig-hoz hasonló eredményt láthatunk. Példa:

```
root@Orion:/home/luke#>  iwconfig
lo       no wireless extensions.

eth0     no wireless extensions.

wmaster0  no wireless extensions.

wlan0    IEEE 802.11abgn  ESSID: "OTTHON_WLAN"
          Mode:Managed  Frequency:2.437 GHz  Access Point: 00:19:E0:A4:DB:98
          Bit Rate=54 Mb/s   Tx-Power=15 dBm
          Retry min limit:7   RTS thr:off   Fragment thr:off
          Encryption key:646F-7269-73   Security mode:open
          Power Management:off
          Link Quality=55/70  Signal level=-55 dBm  Noise level=-93 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Itt minden olyan fontos információt látunk, amely a kapcsolódott vezeték nélküli hálózathoz kapcsolódik.

### 9.2.1 Kapcsolódás egy ismert vezeték nélküli hálózathoz

Vezeték nélküli hálózathoz való kapcsolódás egy mindennapi kérdés. A következőkben lépésről lépésre megvizsgáljuk, hogy hogyan tudunk kapcsolódni egy WEP alapú tetszőleges vezeték nélküli hálózathoz. A mai modern disztribúciók már lehetőséget nyújtanak arra, hogy valamilyen egyszerű grafikus felület segítségével állítsuk be a hálózatot. Ilyen például a Fedora, Debian, Ubuntu, stb. Azonban nagyon fontos, hogy megismerjük, hogy mi is zajlik valójában a háttérben ekkor. Ennek több oka is van. Egyrészt hibaelhárítás szempontjából célszerű ismerni a konzolból való csatlakozás alternatíváját is. A grafikus felületű programok is ezeket használják. Valamint távoli számítógép adminisztráció során általában nincs GUI, amin keresztül könnyedén beállíthatnánk a hálózatot. Mindezek mellett pedig megnyugtatja a rendszer használóját az, hogy ismeri a hálózat alacsony szintű működését, konfigurációs lehetőségeit.

Az első és legfontosabb kérdés az, hogy milyen hálózathoz akarunk kapcsolódni. Mert lehetséges, hogy egy adott helyen több elérhető hálózat is elérhető. Erre az úgynevezett **iwlist** parancsot használhatjuk, amellyel listázható a hatótávolságon belül elérhető vezeték nélküli hálózatok és azok legfontosabb információi, amelyek a kapcsolódáshoz szükségesek majd. A parancsnak több kapcsolója van (iwlist --help), azonban itt csak a szkennelési opcióját vizsgáljuk.

Példa a hatótávolságon belül elérhető vezeték nélküli hálózatok listázására:

```
        #   iwlist wlan0 scanning

lo          Interface doesn't support scanning.
eth0        Interface doesn't support scanning.
wlan0       Scan completed :
      Cell 01 - Address: 00:19:E0:A4:AB:98
            Channel: 6
            Frequency: 2.437 GHz (Channel 6)
            Quality=65/70  Signal level=-45 dBm
            Encryption key:on
            ESSID: "OTTHON_WLAN"
            Bit Rates: 1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                  12 Mb/s; 24 Mb/s; 36 Mb/s
            Bit Rates:9 Mb/s; 18 Mb/s; 48 Mb/s; 54 Mb/s
            Mode: Master
            Extra: tsf=00000005c46c1181
            Extra: Last beacon: 1ms ago
            IE: Unknown: 00036D7369
            IE: Unknown: 010882848B960C183048
            IE: Unknown: 030106
            IE: Unknown: 0706485520010D14
            IE: Unknown: 2A0100
            IE: Unknown: 32041224606C
            IE: Unknown: DD0900037F0101001DFF7F
            IE: Unknown: DD0C00037F020101040002A34000
              IE: Unknown: DD1A00037F03010000000019E0A4DB980219E0A4DB9864002C011D08
```

A példában csak 1 darab hálózat szerepel, mégpedig az OTTHON_WLAN nevű. Ezt mutatja az „ESSID” név. Másik fontos információ, hogy kódolt csatornán történik-e a kommunikáció. Ezt az „Encryption” mező nyújtja. Jelen példában kódolt csatornáról van szó. További fontos információ lehet még az eszköz MAC címe. Ezt a „Cell  01 – Address:” mutatja. Az itt megszerzett információk elengedhetetlenek ahhoz, hogy továbblépjünk a csatlakozáshoz.

A csatlakozás elvégzéséhez tanulmányozzuk az iwconfig manual-ját, ahonnan sok fontos információt tudhatunk meg a különböző beállításokról. A továbbiakban egy-egy egyszerű csatlakozási példát vizsgálunk meg röviden.

A kapcsolódás első lépése, miután kiválasztottuk, hogy mely elérhető hálózathoz akarunk kapcsolódni, hogy tudatjuk ezt az iwconfig-al is a következőképpen:

```
# iwconfig wlan0 essid OTTHON_WLAN
```

Az alábbi sor beállítja a használni kívánt hálózat nevét, az úgynevezett ESSID-t, amely a hálózat azonosítására szolgál. A következő lépés a hálózat működési módjának beállítása:

```
# iwconfig wlan0 mode Managed
```

Jelen példában egy menedzselt típusú hálózati módot állítunk be, amely megfelel egy otthoni átlagos hálózatnak (vezeték nélküli eszközeink nem közvetlen egymással kapcsolódnak, hanem egy ún. Access Point (hozzáférési pont) segítségével). A hálózat topológiájának megfelelően természetesen több különféle mód beállítható (Ad-hoc, Master, Repeater, stb).

Amennyiben WEP alapú titkosítással rendelkezünk, úgy a key kapcsoló segítségével adhatjuk meg a hálózatunk kódját. Jelen segédletben más titkosítással nem foglalkozunk.

Kulcs megadása hexadecimális számként:

```
# iwconfig wlan0 key 7380567799
```

Kulcs megadása szövegesen megadott kódként:

```
# iwconfig wlan0 key s: alma7
```

Természetesen, ha kódolatlan hálózathoz akarunk csatlakozni, akkor ez a lépés nem szükséges. Mindezek után a hozzáférési pontot kell megadnunk. Ezt megtehetjük címmel, amelyet az iwlist parancs kiadása során kapunk (Address: 00:19:E0:A4:AB:98), vagy egyéb módon:

Címmel megadva: `# iwconfig wlan0 ap 00:19:E0:A4:AB:98`
Egyszerűbb úton: `# iwconfig wlan0 ap any ( vagy esetleg (auto) )`

Ezzel a kapcsolóval a kapcsolat viselkedését szabályozhatjuk, hogy mindenféleképpen egy AP-hez ragaszkodjon, vagy esetleg a jel csökkenése esetén más AP-khez is csatlakozhatna. Bővebb információt a manual-ból kaphatunk. Ezután már csak egy IP címre van szükség a kapcsolathoz. Általában a vezeték nélküli hálózatok dinamikus IP cím kiosztással dolgoznak, azaz DHCP segítségével kell IP-t igényelni az újonnan beállított hálózatunkhoz. Ezt pedig nagyon egyszerűen tehetjük meg:

```
# dhcpcd wlan0
```

Amennyiben minden jól sikerült, máris van hálózatunk. Legalábbis belső hálózat. Ezt úgy ellenőrizhetjük, hogy a router ping-eljük:

```
# ping 192.168.1.1
```

## 9.3 Az útvonalválasztó táblázat

A továbbiakban érdemes egy kicsit a hálózat működésének mélyére nézni. Amikor a Linux rendszermagnak egy csomagot kell elküldenie a hálózaton, ki kell választania, hogy azt milyen irányban bocsássa útjára. Ez fontos és bonyolult döntés akkor is, ha a számítógép csak egyetlen ponton kapcsolódik a hálózathoz.

Az útválasztás kérdését a rendszermag az **útválasztó táblázat *(route table)*** alapján válaszolja meg. A rendszergazda feladata az, hogy a megfelelő parancsok kiadásával fel kell építenünk egy útválasztó táblázatot, amelyet a rendszermag a memóriában tárol, és az összes csomag elküldésekor felhasznál.

Az útválasztó táblázat kezelésére a **route** parancsot használhatjuk. A program paraméter nélkül indítva kiírja az éppen érvényben lévő útválasztó táblázatot a szabványos kimenetre, de mi egyelőre kizárólag a -n kapcsolójával együtt használjuk a programot. A route -n kapcsolója arra ad utasítást, hogy az útválasztó táblázat bejegyzéseit IP címekkel és nem a hozzájuk tartozó számítógépnevekkel írjuk ki. Ha viszont nem adjuk meg a programnak a -n kapcsolót, az megpróbálja kideríteni az IP címekhez tartozó neveket is a hálózat segítségével.

Vizsgáljuk meg egy tökéletesesen működő számítógép útválasztó táblázatát:

```
# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags  Metric  Ref Use  Iface
10.0.0.0        0.0.0.0         255.0.0.0       U       0       0   0   ethO
127.0.0.0       0.0.0.0         255.0.0.0       U       0       0   0   lo
0.0.0.0        10.255.255.254  0.0.0.0         UG      0       0   0   ethO
```

A route által kiírt táblázat a következő oszlopokat tartalmazza:

**Destination:** a küldendő csomag címzettjének címe. Ha a küldendő csomag címzettje illeszkedik az itt található címre, akkor az adott sor jelöli ki, hogy a küldendő csomagot milyen úton kell elküldeni.
Ennek az oszlopnak az értelmezéséhez fel kell használnunk a harmadik oszlopban található értéket (Genmask), amely megadja az alhálózati maszkot. A küldendő csomag címzettjének az első és a harmadik oszlop által meghatározott címtartományba kell esnie ahhoz, hogy a sor érvényesüljön.
Különlegesen kell kezelnünk az 0.0.0.0 címet, ha az első oszlopban van! A 0.0.0.0 cím minden címre illeszkedik, az a bejegyzés tehát, amely ezt a címet tartalmazza, bármely címzett számára megfelelő.
Azt mondjuk, hogy az a bejegyzés, amely a 0.0.0.0 értéket tartalmazza az első oszlopban, az alapértelmezett átjáró (default gateway), ahova azokat a csomagokat küldjük, amelyek célcíme nem volt illeszthető egyetlen más bejegyzésre sem.

**Gateway:** az átjáró címe. Ha itt nem 0.0.0.0 található, hanem egy valós IP cím, akkor a csomagot nem közvetlenül kell küldeni, hanem az itt megadott átjárónak kell továbbítani továbbküldésre.

**Genmask:** Az alhálózati maszk, amely megadja, hogy az első oszlopban található cím miképpen illesztendő a címzett IP címére.

**Flags:** az adott bejegyzésre érvényes módosító kapcsolók. A példában alkalmazott kapcsolók:

- **U:** A bejegyzés érvényes, használandó, be van kapcsolva.
- **G:** A bejegyzés átjárón keresztüli utat jelöl, a csomagot az átjárónak kell küldeni továbbítás céljából.

**Iface:** a hálózati csatoló, amelyen keresztül a csomagnak távoznia kell.

A táblázat értelmezése a következő:

1. Ha a csomag címzettje a 10.0.0.0/255.0.0.0 címtartományban található, a csomag közvetlenül küldendő a címzettnek az *eth0* hálózati csatolón keresztül.
2. Ha ellenben a csomag címzettje a 127.0.0.0/255.0.0.0 címtartományban található, a csomagot az *lo* hálózati csatolón keresztül kell közvetlenül küldenünk.
3. Minden más esetben továbbítani kell a csomagot a 10.255.255.254 IP című számítógépnek, amely pontosan tudja, merre kell küldeni a csomagokat helyettünk.

### 9.3.1 Az útvonalválasztó táblázat módosítása

Az útvonalválasztó táblázatot a **route** paranccsal módosíthatjuk. E paramétereként megadhatjuk a del *(delete,* törlés) és az add (hozzáadás) kulcsszavakat, így sorokat törölhetünk és szúrhatunk be az útválasztó táblázatba. A sorokat a következő kapcsolókkal írhatjuk le:

- **net cím:** az első oszlopban megadott cím, ha a bejegyzés alhálózatra vonatkozik.
- **netmask:** a harmadik oszlopban található alhálózati cím megadása.
- **host cím:** az első oszlopban megadott cím, ha a bejegyzés nem alhálóra, hanem egyetlen címre vonatkozik.
- **gw cím:** a második oszlopban megadott átjáró. Ha az adott bejegyzés nem használ átjárót, nem kell megadni.
- **def ault gw cím:** az alapértelmezett átjáróra vonatkozó bejegyzés.
- **dev csatoló:** a hálózati csatoló - utolsó oszlop - nevének megadása.

**Példa:** az eth0 hálózati csatoló a 10.0.0.0/255.0.0.0 alhálózatba tartozik. Helyezzünk el egy bejegyzést, amelynek segítségével a rendszermag képes kapcsolódni azokhoz a számítógépekhez, amelyek ugyanebbe az alhálózatba tartoznak!

```
# route add -net 10.0.0.0 netmask 255.0.0.0 dev ethO
# route -n
Kernel IP routing table
Destination Gateway       Genmask  Flags Metric Ref Use Iface
10.0.0.0    0.0.0.0        255.0.0.0   U    0      0  0   ethO
```

Ez után már csak az alapértelmezett átjáró számára kell egy bejegyzést készítenünk, hogy az Internet többi számítógépét is elérhessük.

```
# route add default gw 10.255.255.254
# route -n
Kernel IP routing table
Destination   Gateway          Genmask     Flags  Metric  Ref  Use   Iface
10.0.0.0       0.0.0.0          255.0.0.0    U       0      0    0     ethO
0.0.0.0        10.255.255.254     0.0.0.0    UG      0      0    0     ethO
```

No és akkor mi volt az a **lo** hálózati csatoló a korábbi példában? Minden egyes számítógép felhasználja a 127.0.0.0/255.0.0.0 IP tartományt arra, hogy saját magának küldjön csomagokat, a Linux rendszermag pedig erre a célra a **lo** hálózati csatolót használja. Azok a csomagok, amelyek a **lo** szimulált hálózati eszközön távoznak, ugyanitt gyorsan vissza is jönnek. Azért van rá szükség, mert lehetővé teszi az egyazon gépen futó alkalmazások közti szabványos adatáramlást.

## 9.4 Főbb konfigurációs állományok

A hálózati beállítások konfigurációs állományainak helye mindig az adott disztribúcióhoz köthető, azonban nagy azonosságok mutatkoznak e téren. Néhány főbb konfigurációs állomány rövid ismertetése:

**/etc/host.conf:** fő hálózati tulajdonságok helye.
**/etc/hostname:** a saját rendszerünk nevét tartalmazza.
**/etc/hosts.allow:** azoknak a gépeknek a listáját tartalmazza, akiknek engedélyezett a hozzáférés a rendszerünkhöz.
**/etc/hosts.deny:** azoknak a gépeknek a listáját tartalmazza, akiknek tiltott a hozzáférés a rendszerünkhöz.
**/etc/hosts.equiv:** ebben a fájlban tilthatjuk és engedélyezhetjük gépeknek és felhasználóknak, hogy használják az r* parancsokat (rsh, rlogin, rcp) jelszó nélkül.
**/etc/hosts.lpd:** azoknak a gépeknek a listáját tartalmazza, akiknek engedélyezett a nyomtatási szolgáltatás igénybevétele a rendszerünkön.
**/etc/hosts:** a saját gépünk nevét és címét tartalmazza. Formája: IP cím, a gép teljes neve, a gép rövid (alias) neve.
**/etc/inetd.conf:** az Internet szerver konfigurációs adatbázisa. Gyakran csak "internet szuperszerverként" nevezik, mivel a helyi szolgáltatások kapcsolatainak kezeléséért felelős. Amikor az inetd fogad egy csatlakozási kérelmet, akkor eldönti róla, hogy ez melyik programhoz tartozik és elindít egy példányt belőle, majd átadja neki a socketet (az így meghívott program a szabvány bemenetéhez, kimenetéhez és hibajelzési csatornájához kapja meg a socket leíróit).

# 10. Folyamatok

A folyamatok létrehozása szigorúan ellenőrzött és meghatározott folyamat. Szabályai vannak, amelyek minden programra érvényesek. Egy program indulásakor egy folyamat (processz) jön létre, amely tartalmazza a program kódját, adatait és a környezeti beállításokat is (munkakönyvtár, környezeti változók, stb.). Minden folyamathoz az operációs rendszer egy egyedi folyamatazonosítót (Process ID) és egy folyamat csoportazonosítót (Process Group ID) rendel.

Egy folyamatot (kivéve a legelső folyamatot, az úgynevezett *init-et)* mindig egy másik folyamat hoz létre. A létrehozó folyamatot szülő folyamatnak (Parent Process), a létrehozott folyamatot gyermek folyamatnak (Child Process) nevezzük. Az *init* folyamatot kivéve minden folyamatnak van szülője. A gyermek a szülő másolata, csak az azonosítója különbözik. Így többek közt örökli a megnyitott állományok leíróit, az umask, ulimit értékeit, szignálok kezelésére vonatkozó beállításokat.

A szülő-gyermek reláció alapján egy fa struktúrát is felrajzolhatunk. Ezt a **pstree** paranccsal nézhetjük meg. Amennyiben egy szülő folyamat előbb szűnik meg, mint annak gyermek folyamatai, a gyermek folyamatok árvákká (orphans) válnak, és szülőjük automatikusan az *init* folyamat lesz.

## 10.1 A daemon folyamatok

A daemon egy olyan program, amelyik a Linux/Unix rendszerben folyamatos végrehajtás alatt áll. (Hasonló, mint pl. Windows NT rendszerben a szolgáltatás). A daemon várakozik egy eseményre, melynek

bekövetkezését figyeli, majd annak megfelelően valamit végrehajt. A Linux (és a UNIX) rendszerek a daemonokat feladatok automatikus végrehajtására használják.

Néhány fontosabb daemon:

**inetd:** Az inetd a hálózati szuperszerver. Ennek a daemonnak a feladata, hogy több portot figyelve a hálózatról érkező datagramot fogadja és a megfelelő porthoz tartozó daemont felébresztve, átadja annak a datagram feldolgozását. Az inetd működésének célja, hogy a különböző daemonok ne terheljék fölöslegesen a memóriát, de ha éppen szükség van rájuk, akkor aktuálisan betöltődjenek.

**ftpd:** Az ftpd az ftp szolgáltatásért felelős daemon. Ő kezeli az ftp kliensektől érkező csatlakozási kéréseket, és bonyolítja a fájl átvitelt a kliens kérése és a saját beállításai szerint. Az ftpd a 21-es portot figyeli.

**telnetd:** A telnetd a telnet szolgáltatásért felelős daemon. Ő kezeli a telnet kliensektől érkező csatlakozási kéréseket. A telentd a 23-as portot figyeli.

**httpd:** A httpd a Web-szerver szolgáltatásért felelős daemon. Ha pl. Apache Web-szervert működtetünk, akkor ez a daemon (vagy ennek több példánya is) fut, és figyeli a gépünk felé irányuló Web-oldalra vonatkozó kéréseket. Ilyen kéréseket HTTP kliensektől – azaz böngésző szoftverektől kaphatunk (pl. Internet Explorer, Netscape Navigator, Opera, stb.). A httpd általában a 80-as portot figyeli.

**nfsd:** A nfsd a NFS szolgáltatásért felelős daemon. Ennek a daemonnak a segítségével a Linux operációs rendszer más Linux (vagy UNIX) rendszerek felé meg tudja osztani erőforrásait. (Hasonló a Windows-os „share - megosztás" opcióhoz.) Linux-os és Windows-os gépek között hasonló szolgáltatást nyújt a SAMBA – SMBFS.

**lpd:** Az lpd daemon a Linux nyomtatásáért felelős daemon. Feladata, hogy a spool könyvtárat figyelje. Ha a könyvtárba fájl érkezik, akkor azt előkészíti nyomtatáshoz.

**pop3d:** A pop3d a pop3 levelezési szolgáltatásért felelős daemon. Ha ez a daemon fut, akkor lehetőség van távoli gépről pop3 szolgáltatás igénybevételére. A pop3d a 110-es portot figyeli.

## 10.2 A folyamatok monitorozása

Azt, hogy az operációs rendszer milyen folyamatokat futtat éppen, a ***ps*** paranccsal listázhatjuk ki. Alapértelmezésben a *ps* parancs csak azokat a folyamatokat listázza ki. amelyeket az adott terminálról indítottunk. A listában csak a folyamat azonosítója, a **terminál,** a státusz az eltelt idő, és a parancs neve szerepel. Amennyiben bővebb információt szeretnénk kapni, úgy a kapcsolók használata elengedhetetlen.

Néhány fontosabb kapcsoló:

- **A:** Hatására az összes folyamatról információt kaphatunk.
- **a:** A terminálokhoz kapcsolódó összes folyamat listáját kapjuk, csak a shell-ek nem szerepelnek benne.
- **r:** csak a futó folyamatokat írja ki.
- **-u user:** egy felhasználóhoz tartozó folyamatok listázása.

A top parancs segítségével folyamatosan is monitorozhatjuk a folyamatokat. Alapértelmezésben a CPU használatának sorrendjében listázza ki az egyes processzeket, azonban ezt könnyen átrendezhetjük.

## 10.3 Háttérfolyamatok

Korábban említettük, hogy a Unix/Linux rendszerek multi-taszk tulajdonsággal rendelkeznek. Ez azt jelenti, hogy a rendszerben egyszerre párhuzamosan több alkalmazás is futhat, azaz lehetőség nyílik úgynevezett háttérfolyamatok létrehozására. Erre több megoldás is létezik, a legegyszerűbb:

```
# parancs &
```

Azonban ezt a módszert csak rövid folyamatok esetén használjuk, mert ha kilépünk a shell-ből, akkor a folyamat is megszűnik. További fontos jellemzője, hogy a program kimenete ilyenkor a terminál. Miért érdemes használni a háttérfolyamatokat ilyen formában? Mert mialatt a rendszer a háttérben végrehajtja a folyamatunkat, addig mi dolgozhatunk tovább a shell-ben megszakítás nélkül.

**Hosszú folyamatok** esetén a ***nohup*** parancsot használhatjuk:

```
# nohup parancs &
```

A *nohup* parancs megmondja a folyamatnak, hogy ne vegye figyelembe a 01 és 03 szignált (HANGUP és QUIT). Ezáltal a folyamat tovább fut akkor is, ha a felhasználó kilép. (A szignálokat később tekintjük át)

A *nohup* által indított folyamat kimenete nem kerülhet a terminálra, mert lehet, hogy a felhasználó rövidesen kijelentkezik. Ha a parancssorban nem irányítjuk át a kimenetet, akkor a *nohup* parancs automatikusan átirányítja a *nohup.out* állományba. Ha már létezik a *nohup.out* fájl, akkor hozzáfűzi a végéhez.

Mivel minden folyamatnak kell, hogy legyen szülő folyamata, ezért amikor kilépünk a rendszerből, a *nohup* által indított folyamatnak az *init* folyamat lesz a szülője.

## 10.4 Kommunikáció a folyamatokkal, megszüntetés

A ***kill*** parancs segítségével szignálokat küldhetünk a folyamatnak, így kommunikálhatunk velük. A *kill* parancs szintakszisa:

```
kill [-szignál] folyamatszám ...
```

Ha nem adunk meg szignált, akkor az alapértelmezés a TERM (15) szignál. Ezzel a szignállal leállíthatjuk a folyamatot, ha nincsen maszkolva. Ha ez nem sikerül, akkor keményebb eszközhöz kell nyúlnunk, ez a KILL (9) szignál. Ezt már nem maszkolhatják a folyamatok. Azonban óvatosan bánjunk vele, mert nyitott állományokat, vagy *lock-okat* hagyhat maga mögött.

A szignálokat a *kill -1* paranccsal listázhatjuk ki. Néhány fontosabb:

| Szignál | Leírás |
|---|---|
| **SIGHUP(1)** | Akkor generálódik, ha kilépünk, miközben a folyamat még fut. |

| | |
|---|---|
| **SIGINT (2)** | interrupt karakter (ctrl-c) |
| **SIGQUIT(3)** | quit karakter (ctrl-Y) |
| **SIGKILL (9)** | A legerősebb szignál, ezt nem hagyhatja figyelmen kívül a program. |
| **SIGTERM(15)** | Folyamatok leállítása. Az alapértelmezett érték. |

A hangup (SIGHUP (1)) szignált akkor kapja a folyamat, amikor a szülője leállt. Például ha a shell-ből indítottunk egy háttérfolyamatot, majd kilépünk.

Az interrupt szignált (SIGINT (2)) akkor generálódik, amikor az interrupt gombot (ctr-c) használjuk.
A quit szignál (SIGQUIT (3)) a quit kombináció esetén (ctrl-Y) jön létre.

### 10.4.1 Folyamat vezérlése a Bash shell-ben

Néhány további műveletre is lehetőségünk nyílik a *bash shell* használata esetén. Az előtér folyamatot felfüggeszthetjük a <ctrl-z> kombinációval. Ez nem szünteti meg a folyamatot, csak megállítja a pillanatnyi állapotában. Amikor megállítjuk a folyamatot, a *shell* megjelenít egy *job* számot, amivel később hivatkozhatunk rá. Ha háttér folyamatként szeretnénk folytatni, akkor az a ***bg*** paranccsal tehetjük meg:

```
#  bg  <job>
```

Ha megint előtérbe akarjuk hozni, akkor azt az ***fg*** paranccsal tehetjük meg.

```
#  fg  <job>
```

A jobs paranccsal kilistázhatjuk a felfüggesztett- és háttérfolyamatokat. *A job* számot használhatjuk a *kill* parancs paraméterezésére is, így nem kell a processz azonosítót megjegyezni.

```
#  kill  [-szignál]  %job
```

### 10.4.2 Folyamatok prioritása

Lehetőségünk van arra, hogy az egyes folyamatok prioritását módosíthassuk, akár futási ideje alatt is. Ezáltal megmondhatjuk a kernel processz ütemezőjének, hogy mely taszkokat részesítsen előnyben, és mely taszkok nem fontosak. A prioritást a (***re)nice*** paranccsal állíthatjuk be. Amennyiben még nincs elindítva a folyamat, de szeretnénk több vagy kevesebb prioritást rendelni hozzá, úgy a ***nice*** parancs segítségével kell elindítani az alkalmazásunkat. A nice szintaxisa:

```
#  nice –n prioritás parancs
```

A prioritások fontossági értéke rendszergazdaként -20-tól 19 értékig hangolhatjuk, míg normál felhasználóként ez az intervallum 0-tól 19-ig terjed csak. A 0 prioritás a folyamatok alapértelmezett prioritása. Ehhez képest a pozitív értékek egyre nagyobb előzékenységet jelentenek. Ha egy folyamat *nice* értéke 19, az azt jelenti, hogy csak akkor fut, amikor más folyamatnak nincs szüksége a processzorra. Természetesen ezzel óvatosan kell bánnunk, hogy ne szívjuk el az erőforrásokat más processzek elől. Példa:

```
#  nice -n -20   unrar x ./starwars.rar
```

Amennyiben egy futó alkalmazásnak szeretnénk módosítani a fontossági értékét, úgy a ***renice*** parancs van a segítségünkre. Szintaktikája:

```
#  renice <prioritás> [ [-p] pid ...] [[-g] pgrp …] [ [-u] user ...]
```

Egy vagy több folyamatot is megadhatunk egyszerre. Felsorolhatjuk a folyamatok azonosítóját (-p pid), hivatkozhatunk folyamat csoportokra is (-g pgrp), vagy egy felhasználó összes folyamatának állíthatjuk a prioritását (-u user), vagy kombinálhatjuk is ezeket. A leggyakrabban használt alak, amikor csak egy folyamat prioritását módosítjuk:

```
#  renice 16 -p 13245
```

Jelen példában az 13245-as azonosítóval rendelkező folyamat *nice* szintjét 16-ra állítjuk.

## 10.5 Automatizált programindítás

A Linux rendszerben nem csak a boot-olás során indíthatunk programokat, hanem más előre meghatározott időben, vagy időközönként is. így megtehetjük, hogy egyes erőforrás igényesebb feladatokat éjszakára időzítünk, amikor a szervert kevesen használják. De felhasználhatjuk arra is. hogy bizonyos időközönként automatikus tisztogató, információgyűjtő, archiváló programokat futtassunk.

Azonban nem csak időpontokhoz köthetjük a folyamatok indítását, hanem a rendszer terheléséhez is. Beállíthatjuk, hogy bizonyos terheltségi szint alatt futtasson a rendszer programokat.

A ***cron daemon*** segítségével ismétlődő feladatokat hajthatunk végre. Ennek meghatározására idő és dátum részekből állítunk össze egy feltételt. A cron feltételezi, hogy a rendszerünk folyamatosan működik. Vagyis a beállításoknak megfelelő időpontokban elindítja a programokat. Azonban ha ezt az időpontot lekési, mert a rendszer nem működött, akkor az indítás elmarad.

A cron daemon-t a boot folyamat során el kell indítanunk hasonlóan a többi folyamatosan futó szolgáltatáshoz. Az indító script: /etc/rc.d/init.d/crond. A cron szolgáltatás fő konfigurációs állománya az /etc/crontab:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts

01   *  * * *   root run-parts   /etc/cron.hourly
02  4  * * *   root run-parts   /etc/cron.daily
22  4  * * 0   root run-parts   /etc/cron.weekly
42  4 1 * *    root run-parts   /etc/cron.monthly
```

Az első sorok a környezeti változók beállításait tartalmazzák. A MAILTO változó éneke azt a felhasználót adja meg. ahova a cron a végrehajtott feladatok kimenetét küldi egy levélben. Ha értéke "", akkor nem készül jelentés a feladat végrehajtásáról. A # jellel kezdődő sorok megjegyzésnek számítanak.

A további sorok a cron egy-egy feladatát jelentik a következő formában: <idő megadása> <parancs>. Az idő az alábbi mezőkből áll:

| | |
|---|---|
| perc | 0-59 |
| óra | 0-23 |
| hónap napja | 1-31 |
| hónap | 1-12 |
| hét napja | 0-7 (a 0 és a 7 vasárnap) |

A mező értékek a következők lehetnek:

- Használhatunk *-t. amely az adott mezőre minden értéket jelent.
- Megadhatunk tartományt: 1-3
- Megadhatunk listát: 13.5
- Lépték megadására is lehetőségünk van. Ezt a / jellel tehetjük. Ha például a hónap mezőbe a következőt írjuk: */2 akkor minden második hónapban hívódik meg a feladat. De tartományokkal is kombinálhatjuk. Például az óra mezőben a 7-12/2 értéket adjuk meg. akkor 7 és 10 között minden második órában teljesül a feltétel.

A példában az egyes sorok a run-parts scriptet futatják, amely meghívja a paraméterként megadott jegyzékben található scripteket.

*Feladat: értelmezzük a fenti crontab sorokat!*

### 10.5.1 Felhasználók lehetőségei

Az /etc/crontab állományt csak a rendszergazda módosíthatja, de a felhasználóknak is lehetősége van rá, hogy használják a cron daemon-t. Ezt a ***crontab*** paranccsal tehetik meg. A parancs meghívja az alapértelmezett szövegszerkesztőt, amellyel létrehozhatunk egy saját crontab állományt. Ezt a program a /var/spool/cron/ jegyzékben helyezi el a felhasználó nevével.

**A hozzáférés szabályozása**

Az /etc/cron.allow és /etc/cron.deny állományokban felsorolhatjuk azokat a felhasználókat, akiknek engedélyezzük, vagy tiltjuk a cron szolgáltatás használatát. Szintaktikailag egy névlistát kell megadnunk, egy nevet egy sorba.
Ha egyik állomány sem létezik, akkor minden felhasználó számára engedélyezett. Ha létrehozzuk a cron.deny állományt, akkor csak az abban nem megadott felhasználók számára lesz engedélyezett a szolgáltatás használata. A cron.allow állomány létezése esetén csak az abban felsorolt személyek használhatják a cron-t.

# Forrásmunkák:


[1] PERE LÁSZLÓ, **GNU/LINUX rendszerek üzemeltetése I-II**. Kiskapu Kiadó. 2005.

[2] **A Unix fájlrendszere:** http://home.fazekas.hu/~egmont/unix/filesystem.html. 2010.

[3] **Linux fájlrendszeri alapfogalmak:** https://secure-www.novell.com. 2010.

[4] **Linux swap terület:** http://wiki.hup.hu/index.php/Swap_terület_mini-HOWTO. 2010.

[5] **fstab:** http://wiki.hup.hu/index.php/Fstab.

[6] **Linux boot folyamat:** http://www.ibm.com/developerworks/linux/library/l-linuxboot/. 2010.

[7] **Kernel boot process:** http://duartes.org/gustavo/blog/post/kernel-boot-process. 2010.

[8] **X Window System:** http://prog.hu/cikkek/473/Az+X-Window+System.html. 2010.

[9] **Xorg:** http://www.x.org. 2010.

[10] Csizmazia Balázs, **Programozás az OSF/Motif könyvtárral**: http://www.cab.u-szeged.hu/local/linux/motif/motif.html. 2010.

[11] KIRCH, OLAF: **LINUX Hálózati adminisztrátorok kézikönyve**. Budapest, Kossuth Kiadó, 1998.

[12] RÓDE PÉTER: **Amit a Linuxról tudni érdemes**. Budapest, Műszaki Könyvkiadó, 1997.


# Szótár

**minix:** amelyet egy régebbi Unix implementációtól örökölt fájlrendszer.

**ext:** (kiterjesztett) fájlrendszer formátum, amely az előző továbbfejlesztése

**ext2:** second extended, amely szintén az előző továbbfejlesztése, jelenleg ez a legnagyobb teljesítményű Linuxos fájlrendszer formátum, szinte minden Linux rendszeren ezt használják

**xiafs:** amely egy, Frank Xia által kifejlesztett fájlrendszer: mára az "ext2" már teljesen kiszorította

**msdos:** MS-DOS-os, FAT formátumú partíciók, floppy-k kezelésére.

**umsdos:** speciális fájlrendszer, segítségével DOS-os, FAT fájlrendszeren hozhatunk létre "igazi" Unix-os fájlrendszer. (Vagyis biztosítja a file-okhoz tartozó Unix-os plusz információ kezelését.)

**NFS:** Network File System: TCP/IP hálózati környezetben elosztott hálózati fájlrendszert hozhatunk létre segítségével.

**ISO9660:** a CD-ROM-ok file formátuma.

**HPFS:** (egyelőre csak olvasható) OS/2 HPFS fájlrendszer.

**SYSV/Coherent:** egy újabb Unix-os fájlrendszer.

**UID:** a felhasználó azonosítója (a rendszerben minden egyes felhasználónak egy ilyen egyedi azonosítója van).

**GID:** csoportazonosító. A UNIX rendszerben minden felhasználó be van osztva egy csoportba. A gid annak a csoportnak az azonosítója, amelybe a felhasználó tartozik.

**PID:** Folyamat-azonosító.

# Gyakori Unix parancsok

**arena:** grafikus internet böngésző program X Window alá.

**at, atq, atrm, batch:** adott feladatot valamikor később hajt végre, amikor esetleg nem is ülünk a gép előtt. (Ha akkor épp ki lenne kapcsolva a gép, akkor bekapcsolása után esik neki a feladatnak.) Lásd még crontab.

**basename:** a paraméterként megadott stringből, azt fájlnévként kezelve eltávolítja az útvonalnevet és az opcionálisan megadott kiterjesztést is. Lehetne rugalmasabb is paraméterezhetősége, több shellben (pl. bash) beépített módon jóval összetettebb, basename jellegű programocskák készíthetők.

**bash:** bourne Again Shell. Az egyik legelterjedtebb parancsértelmező program.

**bc:** Tetszőleges pontosságú, könnyen használható számológép. Változókat hozhatunk létre, és akár összetettebb függvényeket is leprogramozhatunk benne. Kényelmesen lehet számrendszerek között is konvertálni vele. Némelyik változata mindössze egy preprocesszor (előfeldolgozó) a dc programhoz.

**cal:** Adott hónap vagy év naptárát nyomtatja ki. Az évszámot négyjegyűként kell megadni.

**cat:** Az adott fájlokat vagy a standard inputot a standard outputra írja, így például fájlok összefűzésére is alkalmas. Minimális konverziót is képes végezni.

**cc:** C fordító. Manapság talán a GNU C fordító, a gcc a népszerűbb. Millió kapcsolója létezik. Kapcsoló nélkül egy darab valami.c fájlt kér argumentumként, és a.out néven hozza létre a futtatható programot.

**chfn:** A finger által kiírt információ (név, szobaszám, telefonszám satöbbi) változtatható meg vele.

**chgrp:** Ha több csoportba is beletartozunk, fájljaink csoportazonosítóját tudjuk ezzel átállítani.

**chmod:** Fájl hozzáférési jogait változtatja meg, akár rekurzív módon is.

**chown:** Fájl tulajdonosát tudja a rendszergazda megváltoztatni ezzel.

**chroot:** Linux rendszergazda nézze meg a fájlrendszeres fejezetben.

**chsh:** Bejelentkezési shell megváltoztatására szolgál.

**cksum:** CRC ellenőrző összeget számít a fájlokra. Lásd még sum.

**clear:** Törli a képernyőt.

**cmp:** Két fájlt vagy fájlt a standard inputtal hasonlít össze, jelzi az első eltérést.

**compress, uncompress:** Tömörítő program. A tömör fájl .Z kiterjesztésű.

**cp:** Fájl másol, akár speciális eszközre is.

**crontab:** Adott időközönként elvégzendő feladatokat fogalmazhatunk meg egy megfelelő formátumú fájl előállításával. Lásd még at.

**csh:** A C programozási nyelv szintaktikájára hasonlító parancsértelmező. Szerepét lényegében átvette a tcsh.

**cut:** Adott fájl vagy standard input minden sorából a megadott oszlopokat vagy a megadott mezőket (a mezőhatároló karakter is megadható) írja ki a standard outputra.

**date:** Megadott formátum-string szerint írja ki a dátumot és időt.

**dc:** Tetszőleges pontosságú számológép. A bc-vel ellentétben posztfix avagy fordított lengyel módszert használ. Használata a vi jellegű programok kedvelőinek ajánlott.

**dd:** Fájlt másol, bizonyos konverziókat is képes végrehajtani. Nagy előnye: megadható, hogy az input fájlt hányadiktól hányadik bájtjáig másolja át. Főleg speciális fájlba (eszközre) másolás során használt.

**df:** Kiírja a felmountolt partíciókat és kihasználtságukat.

**diff:** Két állományt összehasonlítva leírja, mik a különbségek. Az így készült fájl alapján a patch program hajlandó megfoltozni az eredeti fájlt, megkapva annak az új változatát.

**du:** Az adott jegyzékre rekurzív módon kiszámítja a felhasznált lemezterületet.

**dumpe2fs**: információkkal szolgál egy ext2 fájlrendszerről, főleg a szuperblokk alapján.

**echo:** Kiírja kapott argumentumait. Többnyire beépített shell parancs. -n kapcsoló hatására nem tesz soremelést a végére, -e kapcsoló esetén pedig backslash után bizonyos karaktereknek speciális jelentést ad.

**ed:** A legősibb unixos szövegszerkesztő.

**elm:** Levelező program. Egy fokkal felhasználó barátiabb, mint a mail.

**expr:** Egyszerű matematikai műveleteket végez el, melyeket parancssorban kell neki átadni, külön argumentumként az összes számot ill. műveleti jelet.

**fdisk:** hasznos segédprogram partíciók fájlrendszerek létrehozására, módosítására és törlésére.

**file:** Megpróbálja (úgynevezett mágikus számok alapján) eldönteni, hogy mi áll a megadott fájlban (végrehajtható fájl, C forráskód, valamilyen kép, tömörített fájl satöbbi). Természetesen a fájl neve nem érdekli őt, nem az alapján dönt.

**find:** Adott jegyzéktől kezdve rekurzívan keres olyan fájlokat, melyekre a kapcsolók közt megadott dolgok (pl. jogok, idő, tulaj) stimmelnek, és hogy mit csináljon ezekkel, az is megadható kapcsolóként.

**finger:** Információt közöl a megadott gép adott felhasználójáról, vagy a pillanatnyilag bentlévő emberekről. Ha egy felhasználóra kérdezünk rá, megjeleníti annak ~/.plan illetve ~/.project fájlját is. Ha nem adunk meg felhasználót, a tty oszlopban * jelzi, ha az adott felhasználó letiltotta az üzeneteket, lásd mesg, talk és write.

**ftp:** Távoli géppel fájl átvitele céljából létesít kapcsolatot.

**fsck:** fájlrendszer integritásának ellenőrzése.

**tunefs:** fájlrendszer beállítások módosítása

**gcc:** GNU C fordító. Lásd cc.

**grep, egrep, fgrep:** Szavak, sorok, reguláris kifejezések fájlban történő keresésére alkalmas. Kapcsolóval megadható, hogy a minta illeszkedése esetén a sort vagy a fájl nevét vagy a sorszámot stb. írja ki, így szkriptekben igen gyakran használt, hatékony program.

**gzip, gunzip:** Tömörítő program, többnyire szűrőként használjuk: standard inputon olvas és standard outputra ír. Kitömörít, ha gunzip a neve vagy kap -d kapcsolót.

**head:** A fájl első valahány sorát írja ki.

**id:** A tulajdonos- és csoportazonosítót írja ki.

**info:** Leírást jelenít meg az adott programról. A man-nel ellentétben ez nem lineáris jellegű, hanem menüs szerkezetű.

**irc:** Csevegés az interneten.

**jed:**Linuxok népszerű szövegszerkesztője. Lásd még xjed.

**joe:** Kényelmes, könnyen használható szövegszerkesztő.

**kill:** Szignált küld a megadott processznek. Többnyire beépített shell parancs.

**ksh:** Korn shell. Bourne shell jellegű parancsértelmező.

**less:** A more továbbfejlesztett változata.

**ln:** Linket hoz létre, alapértelmezésben hard linket, -s kapcsoló esetén szimbolikus linket.

**locate:** Kiírja az összes fájlt, akinek nevében a megadott karaktersorozat előfordul. A root által rendszeresen lefuttatott updatedb építi fel azt az adatbázist, amelyet ez a program néz. A kapott * és ? karaktereket a locate is értelmezi. Általában csak Linuxon van fenn.

**ls:** A parancssorban megadott bejegyzéseket, illetve ha jegyzéket adunk meg, akkor annak tartalmát kilistázza. Rengeteg kapcsolója van, unix-onként picit eltérhetnek. Rekurzív listázásra is képes.

**lynx:** Szöveges, terminálra készült internet böngésző.

**last:** a korábbi bejelentkezések történetének megtekintése

**mail:** Levelező program. Kényelmetlen a pine-hoz képest, előnye, hogy lehet nem-interaktívan is használni.

**man:** Kézikönyv oldalt hoz le a kért címszóval. Egyes változatai ismernek kapcsolót több leírás egymás utáni megjelenítésére, illetve értelmezik maguk is a * és ? karaktereket. Másmilyen leírást kapunk az info programmal.

**mc:** Vizuális shell.

**mesg:** Engedélyezi vagy megtiltja az adott terminálra üzenet küldését, vagy kiírja az aktuális állapotot.

**mkdir:** Jegyzéket hoz létre, megfelelő kapcsoló esetén rekurzív módon létrehozva az esetleges hiányzó további jegyzékeket.

**mkfs:** fájlrendszer létrehozása blokkorientált eszközön.

**mknod:** Blokk-orientált, karakter-orientált speciális fájl vagy fifo (pipe) hozható vele létre. Előbbi kettőt csak a root tud készíteni.

**mkpasswd:** Ad egy tippet új jelszóra.

**more:** Szövegfájlt oldalanként jelenít meg. Lásd még less.

**mosaic:** Internet böngésző X Window alá.

**mount:** Eszközön lévő fájlrendszert beilleszt a logikai fájlrendszer valamely jegyzék alá. Egyszerű felhasználó csak kivételes esetekben mountolhat, viszont az argumentum nélküli indítás kiírja, hogy pillanatnyilag mi, hova és hogyan van mountolva. Lásd még umount.

**mtools:** Ez a programcsalád olyan programokat tartalmaz, mint például mcd, mdir, mcopy, melyek dos fájlrendszerű lemezek (tipikusan floppyk) kezelését teszik lehetővé felmountolásuk nélkül.

**mv:** Fájlt átnevez, más jegyzékbe tesz.

**netscape:** Internet böngésző program X Window-hoz.

**nice:** Gyengébb prioritással indít új processzt. Lásd még renice.

**passwd, yppasswd:** Jelszó megváltoztatására szolgál. Lásd még mkpasswd.

**patch:** Fájlt foltoz, tipikusan program újabb változatát állítja elő a régi megléte esetén. Ez azért hasznos, mert az új változat teljes egészében többnyire jóval nagyobb, mint az előző változathoz képesti újdonságokat leíró patch-file. Lásd még a diff parancsot.

**pgp:** Titkosító program, nyilvános kulcsú titkosítást használ.

**pico:** A pine meglehetősen szűk parancskészletű, de azért használható szövegszerkesztője.

**pine:** Igen széles körben elterjedt levelező, hírolvasó-olvasó stb. program.

**printf:** A C programozási nyelv printf függvénye. Az első argumentumban megadott formátumstring szerint írja ki a további argumentumokat.

**ps:** A futó processzekről ír ki információkat.

**pwd:** Az aktuális munkajegyzéket írja ki. Sok shellnek beépített parancsa, és ezek eltérő eredményt adhatnak, például ha tartalmaz szimbolikus linket az útvonal. A /bin/pwd symlinkek nélküli útvonalat ír.

**renice:** Processz nice-szintjét utólag módosítja. Lásd nice.

**rm:** Fájlt töröl. Ha symlinket kap, csak a linket törli, nem magát a fájlt. Ha van az adott fájlra hard link, akkor is csak a bejegyzés törlődik a jegyzékből, maga a fájl más néven még elérhető marad.

**rmdir:** Üres jegyzékeket kitöröl.

**sed:** Az ed-hez hasonló parancskészletű, nem-interaktív szerkesztési feladatok elvégzésére alkalmas program.

**sh:** A hagyományos, jó öreg Unix shell. Linuxban bash-t használnak helyette, vagyis a /bin/sh egy symlink a bash-re, többek között azért, mert az sh nem ingyenes szoftver, míg a bash ingyenes és felülről kompatibilis vele. Komolyabb Unixokon viszont az igazi sh is megtalálható.

**sleep:** A megadott ideig várakozik.

**smbclient:** SAMBA megosztások felcsatolását segítő parancs

**sort:** Ábécérendbe tesz.

**split:** ájlt kisebb darabokra tördel.

**su**: Más tulajdonossal indít shellt. Persze jelszót kér előtte.

**sum:** Ellenőrző összeget számít a fájlokra. Lásd még cksum.

**tac:** Olvasd visszafelé a program nevét.

**tail:** Fájl utolsó néhány sorát írja ki.

**tar:** Tape archiver. Összefűz több fájlt, az új fájlban (melynek .tar kiterjesztést illik adni) emellett leadminisztrál olyan dolgokat is, mint tulaj, idő, szimlinkek stb. Az eredményt a -z kapcsolóval többnyire átnyomattatjuk a gzip szűrőn, ilyenkor .tar.gz vagy .tgz kiterjesztést használunk.

**talk:** Beszélgetést kezdeményez a megadott felhasználóval. A beszélgetésből ^C-vel lehet kilépni. Lásd ytalk, mesg.

**tcsh:** A C shell (csh) továbbfejlesztett változata.

**tee:** A standard inputot a standard outputra másolja és a megadott fájlokban is eltárolja.

**telnet:** Távoli gépre bejelentkezés.

**top:** Rendszeresen kiírja a gép idejének legnagyobb részét elvivő processzeket, a szabad memóriát, a terhelési átlagot stb.

**touch:** Fájl idejét a mostani, vagy a megadott időpontra állítja át.

**tr:** Karakterkonverziót végez az inputján.

**tree:** könyvtárrendszer megjelenítése vizuálisan

**true:** Azonnal visszatér igaz, vagyis 0-ás értékkel.

**umount:** Egy felcastolt fájlrendszert (lásd mount) lecsatol, ha senki nem dolgozik rajta. Rendszerint csak root teheti meg.

**uniq:** Az input szomszédos ismétlődő sorait csak egyszer írja ki.

**uptime:** A rendszer fennállásának idejét, a bent lévő emberek számát és a rendszer terheltségi átlagát írja ki.

**uuencode, uudecode:** Látható ASCII bájtokkal kódolja le a fájlt, így az gond nélkül átküldhető hétbites, CR/LF átalakítást végző protokollon is.

**vi:** Az első képernyő-orientált unixos szövegszerkesztő.

**view:** Link a vi-re, alapértelmezésben nem enged írni a fájlba.

**w:** Kiírja, hogy ki van bejelentkezve és hogy éppen mit csinál.

**wc:** A fájlban lévő sorok, szavak és karakterek számát mondja meg.

**who:** Kiírja, ki van bent pillanatnyilag a rendszerben. Pontosan két argumentum esetén csak az adott terminálra vonatkozó sort írja ki (who mom likes).

**whoami:** Olyasmi, mint a who am i, meg mint az id.

**write:** Üzenetet írhatunk a gépen dolgozó valamelyik másik embernek, ha ő nem tiltotta meg ezt a mesg paranccsal. A terminálon írt üzenetet a CTRL + D-vel fejezzük be.

**xjed:** A jed szövegszerkesztő X-es változata, amely külön ablakot nyit magának.

**xterm:** A legfontosabb X-es program. Ablakában UNIX terminált emulál, visszatérve a jó öreg parancssoros környezethez.

**xv:** Képnézegető, konvertáló és csekély keretek között szerkesztő program.

**yes:** Végtelen sok, egy y-ból avagy a megadott szóból álló sort ír ki. Fölöslegesen kérdezősködő program standard bemenetére lehet pipe-olni.

**yppasswd:** Lásd passwd.

**ytalk:** Többrésztvevős talk, X Window alatt saját ablakokat is megjelenít. Lásd talk és mesg.