Operációs rendszerek gyakorlat

Dr. Adamkó, Attila

Operációs rendszerek gyakorlat

Dr. Adamkó, Attila

Publication date 2010

A tananyag a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-magyarországi Informatika Tananyag Tárház projekt keretében készült. A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával valósult meg.

Nemzeti Fejlesztési Ügynökség http://ujszechenyiterv.gov.hu/ *06 40 638-638*

Tartalom

A táblázatok listája

I. rész - A gép, mint munkaeszköz!

Tartalom

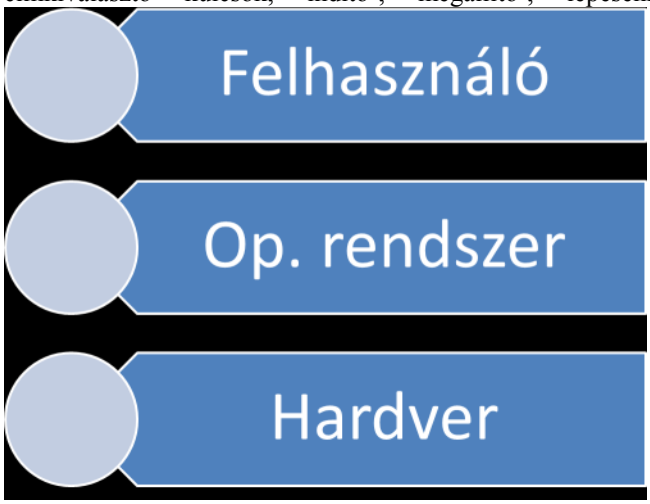
1. fejezet - Bevezetés

1. Az operációs rendszer fogalma

A számítógép utasítások nélkül olyan, mint egy dobozba zárt vaslerakat. Önmagától a legegyszerűbb dolgokat sem tudja végrehajtani, semmilyen műveletet sem képes elvégezni. Ahhoz, hogy a számítógép munkát végezhessen speciális programra van szükség.

Az ISO nemzetközi szabványosítási szervezet definíciója szerint az **operációs rendszer**: "Olyan programrendszer, amely a számítógépes rendszerben a programok végrehajtását vezérli: így például ütemezi a programok végrehajtását, elosztja az erőforrásokat, biztosítja a felhasználó és a számítógépes rendszer közötti kommunikációt."

A hangsúly itt a kommunikáción van. A korai számítógépekkel a kommunikáció bonyolult kapcsolókból és kijelzőkből álló konzolon (főpanelon) keresztül történt. Az első és második generációs gépeknél igazából nem volt operációs rendszer, pontosabban leginkább csak hardver elemekből állt (különféle kapcsolók, címkiválasztó kulcsok, indító-, megállító-, lépésenkénti végrehajtást kiváltó gombok stb.).



A *harmadik generációban* (1965 körül) alakult ki a - már túlnyomó részben - szoftverrel megvalósított operációs rendszer, amit ma már mindenki operációs rendszernek tart. Később vezérlő nyelvek készültek az utasítások átadására, valamint programozási nyelvek a problémák kódolására. A terminálok tették lehetővé, hogy e két műveletet egy hardvereszközzel tudjuk megoldani, azonban *a kapcsolattartás tipikusan „szöveges"* jellege megmaradt. Ezek közül a legkorszerűbbnek annak idején a *menü vezérelt kapcsolat* számított.

A számunkra ma személyi számítógépként ismert doboz az IBM PC-kkel indult, amin leginkább a DOS (Disk Operation System) hódított teret, ami karakteres képernyőn parancsok beírásával, üzenetek kiírásával kommunikált a felhasználóval. Egyszerre csak egy felhasználót tudott kiszolgálni, illetve egy programot tudott futtatni. Az előző generációhoz képest jóval felhasználó barátabb volt, de a hosszú parancsokat minden esetben be kellett gépelni (és fejben kellett tartani a szintaktikát). Félreírás esetén pedig vagy nem hajtódott végre a parancs, vagy ha a gép számára értelmezhető volt, akkor nem az történt, amit a felhasználó várt volna. Érdemes megemlíteni, hogy a DOS nevet nem szabad egyértelműen a Microsofthoz kötni, mert számos különböző cég által fejlesztett DOS létezett és mai napig is több verzió él. Azonban szerepük megkopott.

Mára a karakteres terminálok helyét a a grafikus megjelenítés vette át. A grafika lehetőségeinek növekedésével terjed a *kép-orientált kapcsolattartás*, amelyet *grafikus interfésznek (GUI - Graphical User Interface)* hívnak. Ennél a menüelemek nemcsak szövegesen, hanem képecskék, ún. ikonok formájában, maguk a menük pedig vagy legördülő táblákon, vagy külön, a képernyő más részein átlapoló ablakokban jelennek meg.

Összességében elmondhatjuk, hogy operációs rendszernek (Operating System) nevezzük mindazon programok összességét, amelyek felügyelik és vezérlik a felhasználói programok végrehajtását, ellenőrzik és összehangolják a számítógép különböző erőforrásainak működését. A programozók szempontjából például egy magasszintű programnyelv használata során nem kell azzal foglalkoznia egy feladat programozásakor, hogy az alkalmazott változók és a program a tárban milyen címre kerülnek. Hogyan és pontosan hol helyezkednek el a

háttértárolón az adatok, milyen sorrendben hajtódnak végre a különböző programok, hogyan biztosítja a rendszer az adatok védelmét illetéktelen hozzáférésektől. Ilyen és ehhez hasonló számtalan feladatot oldanak meg az operációs rendszer programjai. A számítógéprendszer összehangolt működését legmagasabb szinten biztosító operációs rendszer feladatát több alrendszer jellegű program aktivizálásával oldja meg. Ezek a különböző számítógép környezetekben esetenként kissé más csoportosításban jelennek meg, bizonyos határfeladatokat más alprogramok látnak el, ezért nehéz egységes, általános érvényű szoftvercsoportosítást meghatározni. Az operációs rendszer nem teljesen egyértelmű fogalom. Tartalma függ a hagyományoktól, a hardverréteg jellemzőitől és a különböző szolgáltatásoknak a teljes rendszeren belül való elosztásától. Az operációs rendszert tehát környezetéből kiragadva (géptől függetlenül) nem lehet meghatározni.

Összegezve az operációs rendszer, egy a hardvert működtető rendszer, amelynek alapvető feladata, hogy kapcsolatot teremt és tart fenn a számítógép és a számítógép kezelője, az operátor között. Az operációs rendszer és a felhasználói programok között az a határozott különbség, hogy amíg az operációs rendszer, a hardvert működtető rendszer, addig a felhasználói programok általános igényeket kielégítő, széles körben használható, adott feladatok megoldására, ill. egyéb adatok feldolgozására kifejlesztett, az operációs rendszerre épülő programok, programrendszerek. Használatukhoz alapfokú számítástechnikai ismeret szükséges.

1.1. Az operációs rendszerek csoportosítása

Az operációs rendszerek leggyakoribb csoportosítási módjai a következőek:

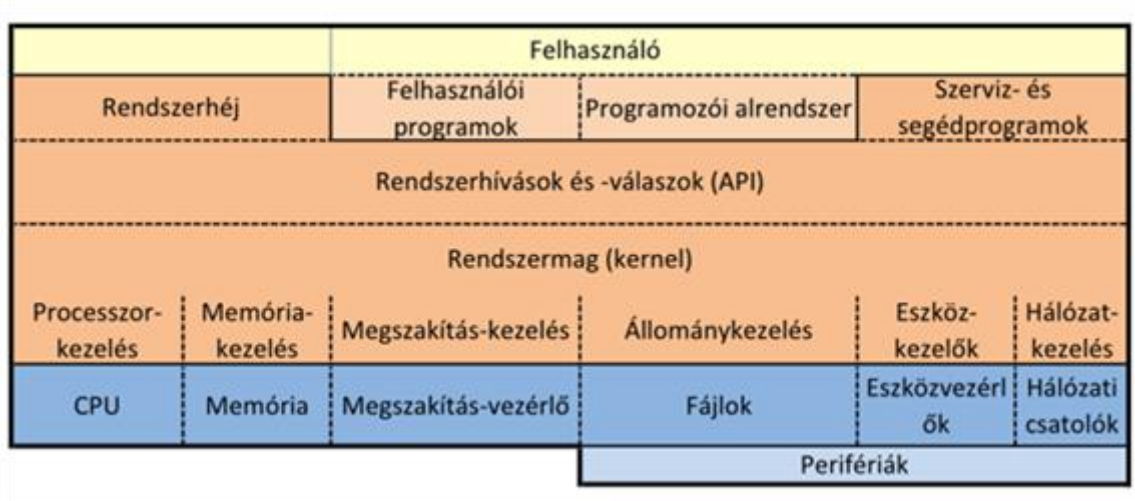
1. Felhasználók szempontjából
   - Egyfelhasználós (single user) : DOS
   - Többfelhasználós (multiuser) : UNIX, Novell, Windows XP
2. Egyidőben futtatható programok száma
   - Egyfeladatos (single task): egyszerre csak egy program futhat a gépen : DOS
   - Többfeladatos (multitask): egyszerre több akár különböző alkalmazás képes futni a gépen (pl. míg a háttérben nyomtat addig írom a következő fejezetet) : UNIX, Windows
3. Megvalósítás szerint
   - Interaktív: üzenet alapú
   - Nem interaktív: végrehajtás alapú
4. A gép méretétől függő
   - Mikrogépes
   - Kisgépes
   - Nagygépes (mainframe)
5. Terjesztési licenc alapján
   - kereskedelmi (DOS, Windows és a UNIX bizonyos kiadásai)
   - szabad (UNIX egyes változatai)

1.2. Az operációs rendszer komponensei

Az operációs rendszer komponensei:

- rendszermag (kernel): Feladata a hardver optimális kihasználásának irányítása, a kért programok futtatása, alkalmazói kéréseknek kiszolgálása.

- API (Application Programing Interface): Olyan szabálygyűjtemény, mely leírja hogyan kell kérni a szolgáltatásokat a kerneltől, és a kernel válaszát hogyan kapjuk meg.
- Rendszerhéj (shell): Feladata a parancsértelmezés. Lehet a shell parancssoros (CLI - Command Line Interface - mint, pl. DOS), vagy grafikus - GUI - felületű (pl. Windows)
- Szerviz- és segédprogramok (utility): a felhasználói "élményt" fokozó kiegészítő programok (pl szövegszerkesztők, fordítóprogramok), amelyek nem képzik a rendszer elválaszthatatlan részét.



Az operációs rendszerek által ellátandó **legfontosabb feladatok**:

- összehangolt működés biztosítása a hardverrel, input/output szinkronizálás (I/O bemeneti/kimeneti műveletek)
- megszakítási rendszer kezelése
- több felhasználó igényeinek egyidejű kielégítése (multiprogramozás)
- központi memória kezelése (Virtuális memóriakezelés)
- erőforrások felhasználásának ellenőrzése
- a processzort, a perifériás erőforrásokat jól kihasználó és a különböző igényeket a lehető legjobban kielégítő működés
- felhasználóval folytatott kommunikáció, információcsere megvalósítása
- állománykezelés
- sokféle, más alkalmazásokhoz kapcsolódó interfészek kezelése.

Szavakkal kifejezve, felületet kell biztosítanunk a felhasználók valamint a futó programok (folyamatok) számára. A memóriakezelés az operációs rendszerekben kritikus rész, mivel egyszerre több programot szeretnénk a memóriába tölteni. Nem megfelelő memóriakezelés nélkül a programok egymás memóriaterületeire írhatnak a rendszer összeomlását okozva (ld. Windows BSOD [Blue Screen Of Death - Kék Halál]). A futtatandó programok általában a merevlemezen helyezkednek el, elindításukkor a memóriába töltjük, azaz futó programról vagy folyamatról beszélünk. A számítógéphez csatlakozhat többféle be- vagy kiviteli eszköz amelyeket perifériáknak nevezünk. Az operációs rendszernek ki kell szolgálnia ezeket a hardvereket, adatokat kell átadni és átvenni azoktól. Az adatokat és a programokat valamilyen módon rendszerbe kell foglalni, elérhetővé kell tenni. Ez az állománykezelési feladat. A hardver, illetve valamely szoftver szokatlan vagy nem kívánatos működése esetén az operációs rendszer feladata az adott helyzet kezelése anélkül, hogy a rendszer leállna. Egy működő számítógépen meg kell védenünk az adatainkat, a programjainkat, folyamatainkat, eszközeinket más rosszindulatú vagy óvatlan felhasználóktól, amelyet szintén az operációs rendszer lát el. A hiba kezelése és naplózása nagyon fontos, hogy felderíthető és elhárítható legyen a rendszergaza által. Belépések, folyamatok indítása, leállítása, újraindítása egy számítógépen, egy hálózaton

mind fontos információ lehet hibakövetés vagy betörésvédelem során. Az ilyen eseményeket az operációs rendszer ezért feljegyzi, vagy másként mondva naplózza.

2. Jelentősebb operációs rendszerek történeti fejlődése

2.1. Disc Operating System (DOS)

A DOS operációs rendszerből számos különböző gyártó számos különböző verziója létezik. (pl. PC-DOS, DR-DOS, Novell DOS, OpenDOS, FreeDos, FreeDOS 32, GNU/DOS, PTS-DOS) Az IBM-kompatibilis PC platformon a Microsoft lemezes operációs rendszere, az **MS-DOS***(Microsoft Disc Operating System)* volt a legszélesebb körben használt operációs rendszer (mára már kiváltották az asztali gépek terén a Windows különféle változatai). Első verziója 1981-ben jelent meg, és 8 fő változat készült belőle, mielőtt a Microsoft 2000-ben befejezte a fejlesztését. A bevételének és piaci részesedésének köszönhetően ez volt a Microsoft kulcsfontosságú terméke a programozási nyelveket gyártó cégből kiterjedt szoftverfejlesztő vállalattá való növekedés során.

Mind az IBM, mind a Microsoft adott ki saját DOS változatokat. Eredetileg az IBM csak ellenőrizte és becsomagolta a Microsoft termékeit, így az IBM verziói röviddel a Microsoftéi után jelentek meg. Az MS-DOS 4.0 azonban valójában az IBM PC-DOS 4.0 verzión alapult, mert akkoriban a Microsoft az OS/2 fejlesztésére koncentrált. A Microsoft „MS-DOS”, az IBM „PC DOS” néven forgalmazta DOS változatait.

Az időrend:

- PC DOS 1.0 – 1981. augusztus – *Az első kiadás az IBM PC-vel*
- PC DOS 1.1 – 1982. május
- MS-DOS 1.25 – 1982. május – *Az első nem IBM hardverhez kapcsolt változat*
- MS-DOS 2.0 – 1983. március – *Unix-ból ismert funkciók bevezetése, mint az alkönyvtárak, fájlleíró (handle) alapú fájl műveletek, bemenet/kimenet átirányítás, csővezetékek (pipe-ok) használata. Elérési út elemeit elválasztó karakterként a Microsoft a Unixban használt / karakter helyett a \ karaktert választotta, mivel a perjelet a legtöbb DOS és CP/M program kapcsolóként használta a parancssorban.*
- PC DOS 2.1 – 1983. október
- MS-DOS 2.11 – 1984. március
- MS-DOS 3.0 – 1984. augusztus
- MS-DOS 3.1 – 1984. november
- MS-DOS 3.2 – 1986. január – *Két merevlemez partíció támogatása 32 MB-ig, egy elsődleges és egy "logikai meghajtó" a "kiterjesztett partíción"*
- PC DOS 3.3 – 1987. április
- MS-DOS 3.3 – 1987. augusztus – *Több logikai meghajtó támogatása*
- MS-DOS 4.0 – 1988. június – *az IBM kódból származtatott verzió*
- PC DOS 4.0 – 1988. július – *A DOS Shell héjprogram megjelenése, grafikus menü kiválasztó, 32 MB-nál nagyobb merevlemezek támogatása a Compaq DOS 3.31 formátumában. Számos hibát tartalmaz!*
- MS-DOS 4.01 – 1988. november – *hibajavításokat tartalmazó kiadás*
- MS-DOS 5.0 – 1991. június – *A DR-DOS 5.0-s verziójára reagálva hasonló tulajdonságok megjelenése: memóriakezelés, teljes képernyős szövegszerkesztő, QBASIC programozási nyelv, súgó, feladatváltó a DOS Shellben*

- MS-DOS 6.0 – 1993. március – *A DR-DOS 6.0-ra reagálva: DoubleSpace lemez tömörítés (a Stackerről másolva) és más funkciók*
- MS-DOS 6.2 – 1993. november – *hibajavításokat tartalmazó kiadás*
- MS-DOS 6.21 – 1994. február – *A Stacker pere folytán eltávolítják a DoubleSpace tömörítést*
- PC DOS 6.3 – 1994. április
- MS-DOS 6.22 – 1994. június – *Utolsó önálló verzió. A DoubleSpace technológiát kicserélik a jogilag tiszta, de kompatibilis DriveSpace-re*
- PC DOS 7.0 – 1995. április – *A DriveSpace helyett a Stackerrel csomagolva*
- MS-DOS 7.0 – 1995. augusztus – *A Windows 95-ben megtalálható DOS változat*
- MS-DOS 7.1 – 1996. augusztus – *A Windows 95 OSR2 (Windows 95B) és Windows 98 változatokkal csomagolt DOS. Támogatja az újonnan megjelenő FAT32 fájlrendszert*
- MS-DOS 8.0 – 2000. szeptember 14. – *A Windows Me DOS változata, az MS-DOS utolsó verziója. Nem szerepel benne a SYS parancs, nem lehet elindítani csak parancssorosan és más képességek is hiányoznak*
- PC DOS 2000 – *Y2K kompatibilis verzió több kisebb jelentőségű újdonsággal. Az MS-DOS család utolsó tagja*

Az MS-DOS-t nem többfelhasználós vagy többfeladatos operációs rendszernek tervezték, kernele monolitikus. Ezt a hátrányt megpróbálták leküzdeni az évek során. A Terminate and Stay Resident (TSR) rendszerhívás eredeti célja a betölthető eszközmeghajtók kezelése volt, valamint más, többnyire nem dokumentált függvényeket használtak felugró alkalmazások készítésére. Számos gyártó viszont ezt felhasználva próbálta meg a többfeladatos környezetet megvalósítani.

A DOS alapvetően parancssoros felhasználói felületet biztosít, kötegelt szkript (batch script) futtatási lehetőséggel. A parancsértelmezője (alapértelmezetten a command.com) úgy lett tervezve, hogy könnyen cserélhető legyen. Grafikus felület nem készült hozzá, de tágabb értelemben a Windows első ága (amely független az NT vonaltól) tekinthető ennek az 1.0-s verziótól kezdve egészen a Windows ME (Millenium Edition) kiadásáig, amely már teljesen feledtette a DOS alapvető karakterisztikáját, mert hatékonyabban tudta kihasználni a 80386-os processzor architektúra előnyeit és biztosítani a virtuális memóriakezelést. A DOS-ból indultak, de kiterjesztették azt, hogy védett módban fussanak. A későbbi Microsoft Windows verziók már a DOS-tól függetlenül futottak, de megtartották a korábbi kód nagy részét, így a DOS az új operációs rendszerben virtuális gépként tudott futni. (Linux alapú gépek esetén is hasonló a helyzet, ott is virtualizálva futtathatóvá tehető, bár szerepe nem túl jelentős.)

2.2. Windows

A Microsoft Windows a Microsoft Corporation gyártotta operációs rendszerek, illetve az ezekbe épített többfeladatos grafikus felhasználói felületek, valamint bizonyos mobiltechnológiák családja. A "Windows" szó és logó a Microsoft cég védjegye. A Windows operációs rendszerek alapfilozófiája a popularitás, a könnyű kezelhetőség, valamint a „minden egyben”-filozófia (a számos beépített böngésző, médialejátszó stb. alkalmazás), amely leveszi a felhasználó válláról a telepítés, a kezelés terheit, és biztosítja a számítógép széles körű használatát az informatikában járatlan felhasználók számára is.

Kezdetét tekintve a cég Új-Mexikóban lett alapítva 1975-ben Bill Gates és Paul Allen által **Micro-soft** név alatt. Fő profiljuk BASIC-fordítóprogramok fejlesztése és árusítása. Azonban a nagy előrelépést számukra az jelentette, hogy az IBM cégnek eladták Tim Paterson Seattle Computer Products nevű cégétől licencelt QDOS rendszert MS-DOS néven. Jó ütemben, jól léptek, mert a PC-k rohamos elterjedésével nagy részesedést szereztek, mindazon ellenére, hogy nem a legjobb minőségű terméket szállították (ellenben olcsón kínálták).

A Microsoft azonban nem csak operációs rendszereket szállított, számos szoftverterméket készít: irodai szoftverek, fordító programokat, beágyazott rendszereket.

A szoftverek használhatósága igen fontos volt a Microsoftnak, és ez is hozzásegítette őket korai sikereikhez. Ennek néhány főbb aspektusa:

- Egységes felhasználói felület: az összes Microsoft-alkalmazás ugyanazokat a menüparancsokat, billentyűkombinációkat és folyamatokat használta a hasonló feladatokhoz. Így jóval kevesebb idő volt megtanulni egy másik szoftver használatát.

- Visszafele kompatibilitás: a Microsoft biztosította, hogy a régebbi kódok és adatok az újabb rendszereken is működni fognak.

- Összekapcsolhatóság: általában, de főleg a Microsoft Office esetében az egyik Microsoft-alkalmazásban létrehozott adat más Microsoft-alkalmazásokba is átmozgatható.

A Windows első verziója 1985-ben jelent meg, amely az Apple cég operációs rendszerének alapötletét felhasználva kezdte el kifejleszteni az ablaktechnikán alapuló rendszerét. A rendszer a DOS után forradalmian újnak számított, sikere pedig a konkurenciához képest alacsony árában volt keresendő. Az első jelentősebb verziója a 3.0-s lett, amely 1993-ban jelent meg. A kezdeti 16 bites verziók (1.0-3.1) a kényelmet és a teljesítményt tartották szem előtt a biztonsággal szemben. Ezeknek a verzióknak nem volt összetett jogosultsági rendszere. A korai és DOS alapú Windows verziók a „mindent szabad, kivéve néhány dolgot” elv alapján épültek fel, ellentétben a Unix-szerű rendszerek „minden tilos, kivéve a megengedett dolgokat” elvével. Ennek eredményeképp a felhasználói programok gyakorlatilag a teljes számítógépet és a rendszert képesek voltak elérni és módosítani. ennek minden előnyével és hátrányával együtt.

2.2.1. A Windows 9x vonal

Az egyik legjelentősebb és leglátványosabb újítást a Windows családban a Windows 95 1995-ös megjelenése jelentette. Hálózati támogatás, széles multimédia lehetőségek és True Type betűtípusok jellemzik, de még mindig a DOS-on alapszik. Sok dolgot a 95-ből örökölt meg a Windows 98, melynek a második kiadása, a Windows 98SE volt igazán sikeres, és nem egy gépen a mai napig fellelhető. A Windows Me (Millennium Edition) rendszerek csak rövid időre tűntek fel. A Windows ME már képes volt a Windows-rendszerfájlok védelmére és illetéktelen módosítás után a visszaállításukra, ennek ellenére rengeteg kritika érte megbízhatatlansága miatt, ezt még a Microsoft is elismerte. Ezután 2000-ben jelentkezett a Windows 2000, melynek célja az volt, hogy összefonja az NT és 9x vonalat.

2.2.2. Windows NT, XP és napjaink

A Windows NT vonal működését, rendszermagját úgy tervezték újra, hogy a fenti biztonsági hiányosságokat kiküszöbölhetővé váljanak, a kezdetektől fogva teljesen függetlenül folyt a fejlesztése a DOS alapú Windowsokétól. Ennek megfelelően nemcsak hogy fájl szintű Hozzáférés Vezérlő Lista (HVL – Access Control List – ACL) alapú jogosultságkezelést kaptak az NT vonal tagjai, hanem a "minden objektum" elvet felhasználva HVL adható tetszőleges objektumhoz, legyen az egy folyamat, egy szál, egy eszközmeghajtó, egy szinkronizációs objektum vagy akár egy osztott memória régió.

Az XP a rengeteg - elsősorban biztonsági jellegű - probléma ellenére a Microsoft egyik legsikeresebb operációs rendszerének tekinthető (a problémák nagy része a Service Pack 1-3 javítócsomagokkal, valamint az Internet Explorer használatának mellőzésével különben is orvosolható volt). Erre 6 év elteltével próbált meg válaszolni a Microsoft a Vista kiadásával, ami stabil(abb)nak és biztonságos(abb)nak mondható az eddigieknél. Ellenben az XP-hez képest meglehetősen leromlott rendszerigény/teljesítmény arány jellemzi és emiatt az egyik legkiábrándítóbb operációs rendszerük lett.

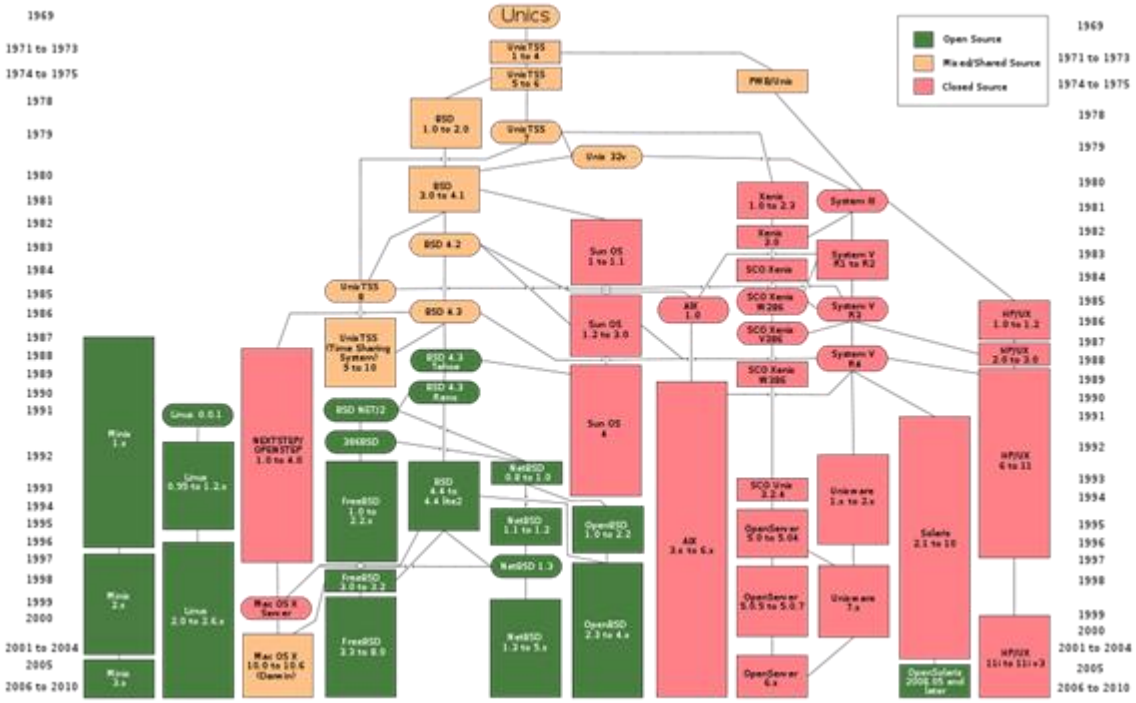
A Windows Vista után megjelent a Windows 7 2009-ben. A rendszer rengeteg újítással, új GUI-val (Grafikus felhasználói felület) büszkélkedhet, és teljesítményét tekintve is előnyös az otthoni számítógép használók szempontjából. A biztonság terén tovább vitte a Vista által bevezetett újításokat (pl. User Access Control - UAC) a vírusok és kártékony programok megfékezésére, de ennek ellenére még mindig a felhasználó a leggyengébb láncszem, mert a számos védelmi modul közül sokat kapcsolnak ki annak érdekében, hogy a rendszer kevesebbszer "zaklassa" őket magasabb szintű engedélyek kérésével.

A jövő pedig még megjósolhatatlan. A Microsoft már egy teljesen új alapokon álló Windows-utódon is dolgozik 2010 óta, amely a már be is mutatott Singularity kísérleti rendszerre épül, s pillanatnyilag *Midori* néven ismert.

2.3. UNIX és a Linux

Unix (hivatalosan védjegyezve **UNIX**) egy többfeladatos, többfelhasználós számítógépes operációs rendszer, amely eredetileg 1969-ben az AT&T cég Bell Laboratóriumában került kifejlesztésre, melynek munkatársai

között volt Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy és Joe Ossanna. A Unix operációs rendszert először assembly nyelven írták, de 1973-ra már szinte teljesen átírták a C programozási nyelvre, nagyban megkönnyítve a további fejlődés lehetőségét. Az AT&T kezdetben ingyen az amerikai egyetemek rendelkezésére bocsátotta a Unix forráskódját, így éveken belül százezer fölé emelkedett a működő Unix rendszerek száma. A gyors terjedésnek azonban a hátulütői is jelentkeztek: 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 Five), amelyet a Unix System Laboratories (USL) fejlesztett tovább. Ezen fő változatok mellett számos kisebb-nagyobb alváltozat van forgalomban még napjainkban is, a Unix-szerű rendszerek pedig széttagozódtak, sokszínűvé váltak. Ezt láthatjuk a Unix rendszerek evolúcióját bemutató ábrán.



2.3.1. Standardok, ajánlások, formátumok

Időközben független bizottságok is próbálták valamennyire egységesíteni a BSD és System V ajánlásokat, és az IEEE kidolgozta (az ANSI és az ISO támogatásával) a "POSIX" (**P**ortable **O**perating **S**ystem **I**nterface [for Uni**x**]) ajánlást, amely igyekszik egyesíteni a két fő ágat. Mellette, az 1990-es évek elején elindult egy hasonló kezdeményezés a Common Open Software Environment (COSE) konzorcium keretein belül, amely létrehozta a Single UNIX Specification ajánlást. Az ezredforduló előtt, 1998-ban az IEEE és a COSE létrehozta az Austin Group nevű csoportot, hogy a POSIX és fenti specifikációt között kialakuljon egy közös mag. Ennek jelenleg a legújabb eredménye a 2009-ben megjelent **POSIX:2008**.

A kompatibilitás mellett nem csak a rendszer alapvető jellemzőit rögzítették, hanem a **bináris és tárgy kód szabványos formátum**át is elfogadták. Ennek alapját a System V R4 ág Executable and Linkable format (ELF) formátuma alkotta. Ez a közös formátum nagymértékű kompatibilitást biztosít az azonos CPU architektúrán futó UNIX rendszerek között.

A szabványosítás legutolsó - harmadik - lépcsője pedig a **könyvtár hierarchia rendszerbe foglalása** volt, amely meghatározta az egyes fájlok helyét és mappájának nevét (ld FHS később).

Ezek a lépések hatást gyakoroltak más operációs rendszerekre is. Jelentős lépés, hogy az operációs rendszer kódja már magas szintű programozási nyelven íródik, a binárisok portolhatóak (más platformú operációs rendszerre is átültethető) és a könyvtárszerkezet is jól meghatározott. A futtatható binárisok terén számos más operációs rendszer is biztosít POSIX kompatibilis réteget. A legmeglepőbb talán a Microsoft lépése, miután a Windows 2000 már beépített POSIX réteggel rendelkezik, a Windows XP pedig a "Windows services for UNIX" szolgáltatás telepítését követően lesz kompatibilis. Windows Vista és Windows 7 esetén az Enterprise és

Ultimate kiadások rendelkeznek beépített UNIX alrendszerrel (más kiadásaihoz viszont nem telepíthető). Természetesen ennek ellenére megoldható a POSIX felület elérése, ehhez az egyik legjelentősebb szállító, a Red Hat által megvásárolt Cygnus Solutions cég Cygwin termékét kell telepítenünk. (De alternatíva lehet ennek az 1.3.3 ágából származó MinGW [minimalist GNU for Windows] is.) Mindazonáltal számunkra a legfontosabb a továbbiakban a POSIX szabványhoz elég közel álló Linux.

2.3.2. Linux

Amikor a Linux szót használjuk, akkor a jelentése nagyban függ a szövegkörnyezettől is. A „Linux” elnevezés szigorú értelemben véve a Linux-rendszermagot jelenti, amelyet Linus Torvalds kezdett el fejleszteni 1991-ben. A köznyelvben mégis gyakran a teljes Unix-szerű operációs rendszerre utalnak vele, amely a Linux-rendszermagra és az 1983-ban, Richard Matthew Stallman vezetésével indult GNU projekt keretében született alapprogramokra épül. A Linux pontosabb neve ebben az értelemben GNU/Linux.

A „Linux” kifejezést használják Linux-disztribúciókra (terjesztések) is, ám ilyenkor általában a disztribúció nevét is hozzáteszik. Egy-egy disztribúció olyan összeállítás, amely az alaprendszeren túl bizonyos szempontok alapján összeválogatott és testre szabott programokat tartalmaz. Disztribúciókat legtöbbször az különbözteti meg, hogy milyen célközönségnek és milyen feladatra készítik őket, így mindenki megtalálhatja a neki leginkább megfelelőt. Így léteznek olyanok, melyek lehetőséget nyújtanak arra, hogy szinte az összes konfigurálási lehetőséget egy grafikus felületen végezzük el és vannak olyanok is, amelyek megkövetelik, hogy a felhasználó mindent a konfigurációs állományok szerkesztésével állítson be a saját ízlésének megfelelően.

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 disztribúciók egyik fő jellemzője az egyes programcsomagok installálásának, eltávolításának és frissítésének megkönnyítése és támogatása (ehhez a két leginkább elterjedt csomagkezelő rendszer az APT és az RPM). A disztribúciók nagy részének készítői komolyan veszik a biztonsági problémákat, és az ismert hibák javításait rövid időn belül elérhetővé teszik disztribúciójuk csomagfrissítési módszerének segítségével.

Magyar fejlesztésű disztribúciók:

- blackPanther OS
- UHU-Linux
- Frugalware
- Kiwi
- Suli

Főbb nemzetközi fejlesztésű disztribúciók:

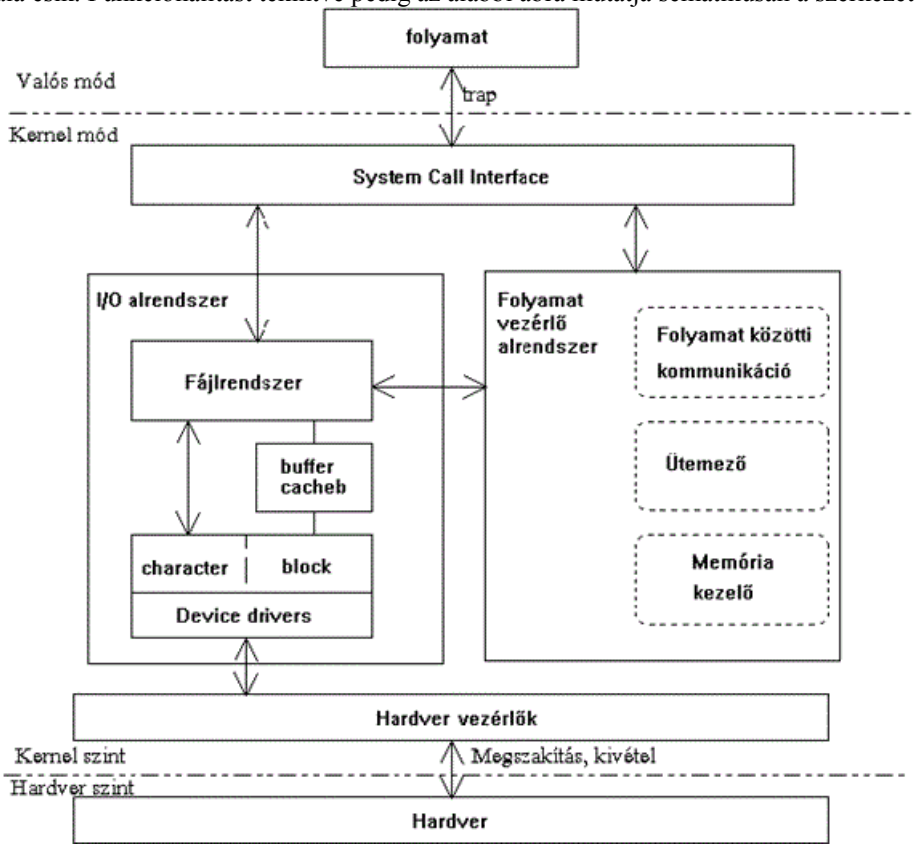
- Debian
- Fedora, Red Hat Linux
- Gentoo
- Arch Linux
- Mandriva (régebben Mandrake)
- Slackware
- SuSE
- Ubuntu Linux

Történetét tekintve a szabad forráskód csere jelentette a legnagyobb löketet. A hálózat segítségével könnyen elérhető volt, a C nyelv és az egységes környezet miatt pedig 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 a GNU projekt (GNU is Not UNIX), amely pedig egy minél teljesebb Unix rendszert kívánt létrehozni és biztosítani. Ennek jogi megfogalmazása a GPL (GNU General Public License). 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.

Azonban a GNU projektben hiányzott egy olyan mag, amely bizonyítottan szabad (nem tartalmaz copyright alá eső USL vagy BSD kódot). A GNU elkezdte a kernel fejlesztését Hurd kódnéven 1984-ben, de a megjelenésére még éveket kellett volna várni a tervezet alapján. A fejlesztést 1986-ban abba is hagyták, majd 1990-ben indult újra.

Ebben az űrben jelent meg Linus Torvalds finn egyetemista, aki kezdetben csak hobbi szinten szeretett volna kernelt írni. Kiindulási alapja a Minix (tanulásra szánt) operációs rendszer volt. Viszont a kezdeti lépések annyira jól sikerültek, hogy a kezdeti megjelenését követően nagyon gyorsan terjedt és fejlődött. Emiatt is került be a GNU projektbe, mint a rendszer lelke, azaz a kernel. A Linux folyamatosan fejlődő rendszer. Gyors fejlődésének és terjedésének egyik oka az, hogy a fejlesztők már a munkaverziókat elérhetővé tették (és teszik jelenleg is) bárki számára, akárki kipróbálhatta (kipróbálhatja) a fejlesztés bármely stádiumában. Teljesen tipikus eset - mert a teljes forráskód mindig hozzáférhető -, hogy az önkéntes (önjelölt) tesztelők a megtalált hibákat már a javítással együtt küldték vissza a fejlesztőknek az Interneten. A Linux jogi értelemben nem UNIX tehát, leghelyesebb volna Unix-klónnak nevezni, és nem is 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 a legteljesebb POSIX implementációnak. Maga a Linux, illetve a Linuxon futó szoftverek legnagyobb része a GPL alá esik. Funkcionalitást tekintve pedig az alábbi ábra mutatja sematikusan a szerkezetét:

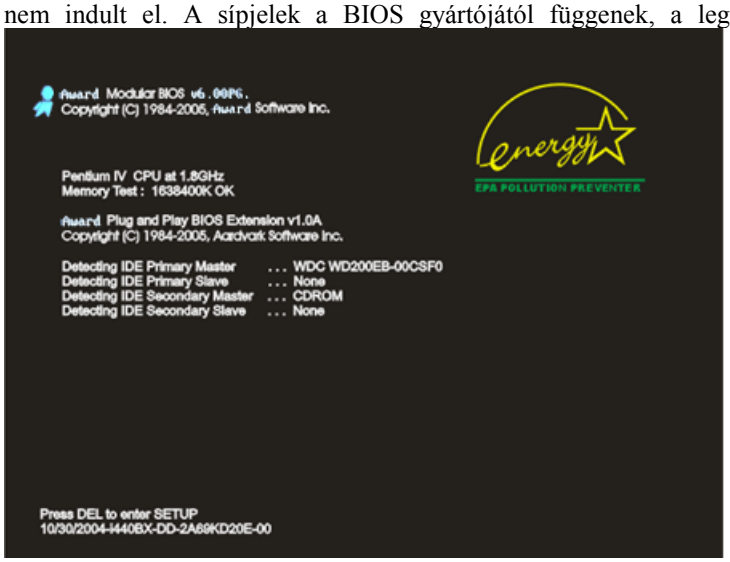


2. fejezet - Betöltődés

Az operációs rendszer betöltődési folyamatát idegen kifejezéssel boot-olásnak is szoktuk nevezni. A betöltendő operációs rendszer nem szükségszerűen kell, hogy a számítógépen legyen, megoldható az indítási folyamat másik eszközökről vagy egy hálózaton lévő másik számítógépről is (hálózati boot, network computer).

1. A betöltődés előkészítése

Amikor egy PC elindul, a BIOS különféle teszteket végez annak ellenőrzésére, hogy minden rendben van-e a hardverekkel. (Ezt szokás Power On Self Test-nek vagy röviden POST-nak nevezni.) A POST egy diagnosztikai program, mely leellenőrzi a számítógép összes hardver eszközét, hogy jelen vannak-e és megfelelően működnek-e. Csak a teszt hiba nélküli lefutásakor kezdődhet meg a számítógép indítási folyamatának elindítása. Az indulás során a későbbiekben is végrehajthat tesztfolyamatokat, ahogy a rendszerindulási folyamat megkívánja. Ha mindez rendben lezajlott, akkor az egy csippanással jelzi, hiba esetén pedig a hiba kódját rövidebb-hosszabb sípolással szakaszosan ismétli. Azért kell sípjeleket adnia, mert ekkor a videóvezérlő még nem indult el. A sípjelek a BIOS gyártójától függenek, a legnagyobb 2 cég az AMI és az AWARD.



A tényleges rendszerindítás ezt követően kezdődik. Általánosságban elmondható, hogy először egy lemezmeghajtó kerül kiválasztásra, és az ebben levő lemez legelső szektorát, a boot szektort olvassa be a rendszer. (Az, hogy mely eszközökön vagy lemezmeghajtókon, illetve milyen sorrendben keresi a gép a megfelelő boot szektort, a számítógép beállítása mondja meg. Tipikusan először az első floppymeghajtóval, majd az első merevlemezzel próbálkozik a BIOS.)

Ezen beállítások egy speciális chipen tárolódnak, egy SRAM-on (Static Random Access Memory), amit az építési elve alapján **CMOS**-nak (*C*omplementary *M*etal-*O*xide *S*emiconductor) neveznek. Kis áramfelvétele miatt ideális a telepes táplálású (hordozható) elektronikai eszközökben, valamint a hosszú idejű tárolási feladatokra. Ezért találkozhatunk egy apró elemmel az alaplapon.

1.1. A betöltés előkészítése merevlemezről

Merevlemezeknél mindezt pontosítani kell: a Master Boot Record (MBR) kerül beolvasásra, ugyanis egy merevlemez több partíciót is tartalmazhat, mindegyiken saját boot szektorral. A **partíció** a merevlemez egy önálló logikai egysége, amely fájlrendszer tárolására alkalmas. Ahhoz, hogy egy merevlemezt használni tudjunk, legalább egy formázott partíciót kell tartalmaznia.

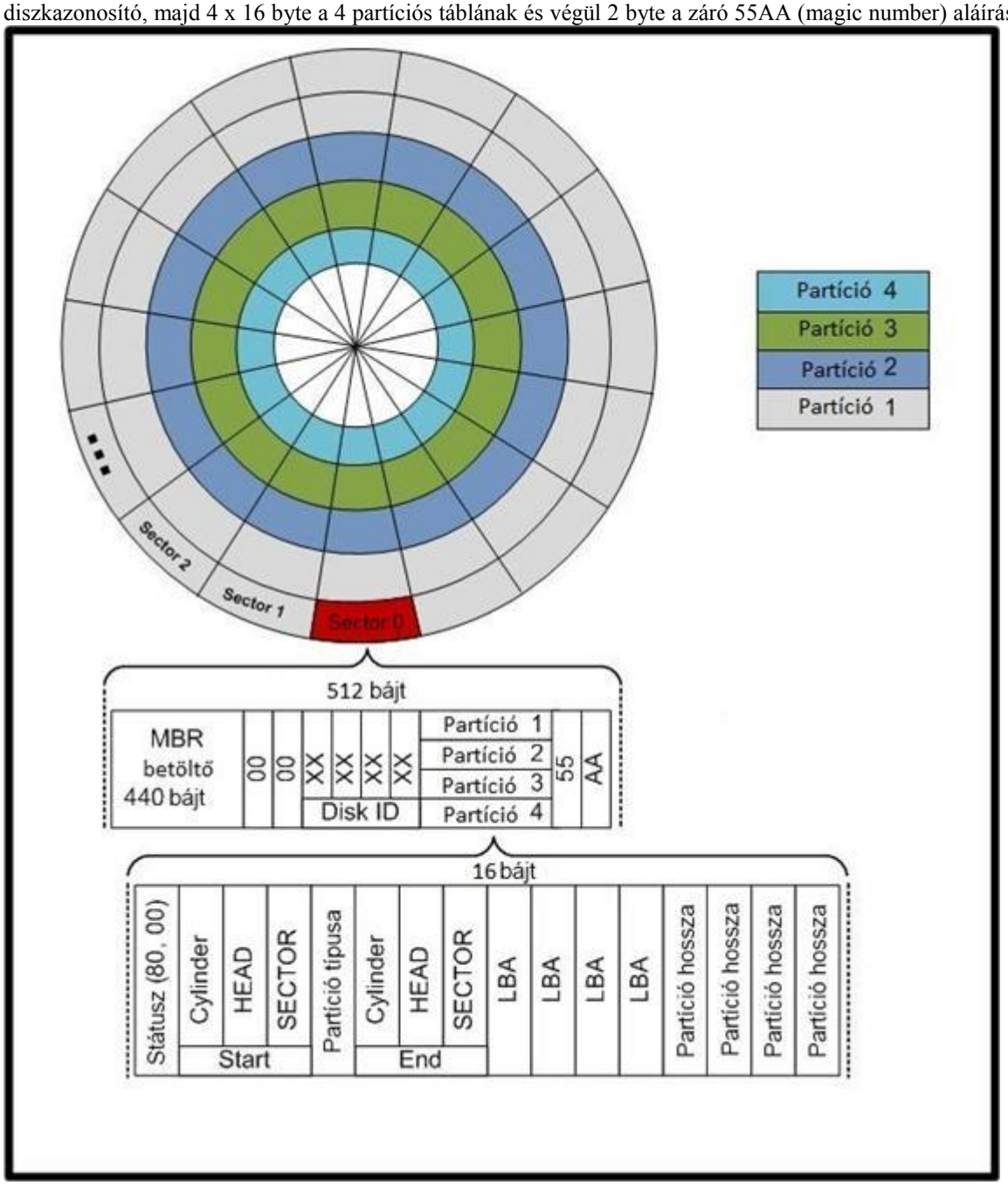
Egy partíció egyetlen fájlrendszer adatait képes tárolni, ezért ha több fájlrendszert szeretnénk, mindenképpen particionálnunk kell a merevlemezt. Több oka is lehet különböző fájlrendszerek használatának: egyrészt különböző operációs rendszerek használata, másrészt ha kisebb logikai egységekre akarjuk bontani a merevlemezünket. Ez utóbbinak az a veszélye, hogy mivel egy partíció méretét előre meg kell adni, előfordulhat hogy egy partíciónk betelik, míg némelyik szinte teljesen kihasználatlan.

1.1.1. Az MBR partíciós tábla

Ahogy már említettük, a betöltési folyamat során merevlemez esetén a BIOS az MBR tartalmát tölti be, innen folytatódik az operációs rendszer betöltése. A fő rendszertöltő rekordban **legfeljebb négy partíció** adatainak tárolására van hely, ezért a merevlemez legfeljebb négy valódi partíciót tartalmazhat. Particionáláskor célszerű lehet megadni az aktív (boot) partíciót, ami MS-DOS és Windows rendszereken rendszerint az első elsődleges partíció, hogy a rendszer bootolásra képes legyen.

Az első boot szektor (a szektorméret 512 byte), azaz az MBR egy kis programot tartalmaz, amelynek a feladata az aktuális operációs rendszer beolvasása és elindítása. Az 512 byte első 440 byte-ja a betöltőkód, 6 byte

diszkazonosító, majd 4 x 16 byte a 4 partíciós táblának és végül 2 byte a záró 55AA (magic number) aláírásnak.



A partíciós táblákhoz szinte minden operációs rendszer alatt hozzáférhetünk az `fdisk` program adott platformon létező verziójával (Windows XP óta ez `diskpart`-ra módosult), de talán az Ubuntu Linux `cfdisk` programja mutatja a leglátványosabban a fentiekkel egyező információt.

```
# cfdisk -P ts /dev/sda
Partition Table for /dev/sda

                First       Last
 # Type        Sector      Sector   Offset    Length   Filesystem Type (ID) Flag
-- ------- ----------- ----------- ------ ----------- -------------------- ----
 1 Primary           0    51857819     63    51857820 HPFS/NTFS (07)       Boot
 2 Primary    52130925    62380394      0    10249470 Minix / old Lin (81) None
 3 Primary    62380395   103346144      0    40965750 FreeBSD (A5)         None
 4 Primary   103346145   160071659      0    56725515 Linux (83)           None
Partition Table for /dev/sda

          ---Starting---       ----Ending----     Start     Number of
 # Flags Head Sect Cyl   ID  Head Sect Cyl      Sector    Sectors
-- ----- ---- ---- ---- ---- ---- ---- ---- ----------- -----------
 1  0x80    1    1    0 0x07  254   63 1023          63    51857757
 2  0x00  254   63 1023 0x81  254   63 1023    52130925    10249470
 3  0x00  254   63 1023 0xA5  254   63 1023    62380395    40965750
 4  0x00  254   63 1023 0x83  254   63 1023   103346145    56725515
```

Természetesen ezt megtekinthetnénk hexában is, de az nem nyújtana túl sok lényegi információt számunkra, mert az ott található gépi kódot igen nehezen tudjuk mi magunk értelmezni. Viszont a feladata egyszerűen úgy foglalható össze, hogy betölti a boot szektort a memória egy [a BIOS által előre rögzített (`0x00007C00`)] részére, majd kicseréli önmagát a választott operációs rendszer betöltéséhez szükséges boot szektorra.

Ellenben hasznos lehet erről mentést készíteni, hogy ha bármikor is történik valami a merevlemez eme szektorával, akkor gyorsan meg tudjuk gyógyítani a rendszert. A feladat adott, az MBR mentéséhez az első 512 bájtra van szükségünk, egyéb partíció esetén az adott partíció első 512 bájtos szeletére, mint boot szektor. Ez akkor lehet fokozottan érdekes, ha a meglévő (pl. Windows-os) rendszerünk mellé szeretnénk egy másik (pl. Linux alapú) operációs rendszert feltenni és ragaszkodunk a már megszokott betöltő programhoz a Linuxos változatokkal szemben. A megoldás egyszerű lesz!

A Linuxos disztribúciók telepítő programja mindig megkérdezi - szemben a Windows-ok rossz szokásával -, hogy hol szeretnénk elhelyezni a betöltőt - az MBR-ben, vagy a partíció elején. A második választás esetén a partíció elején lévő 512 bájtot kell lementenünk és elérhetővé tennünk a már meglévő Windows betöltője számára. Ehhez használhatjuk a nagyon egyszerű lélekzetű - a későbbiekben még tárgyalt - `dd` parancsot az alábbi módon:

```
# dd if=/dev/sda of=mbr.bin bs=512 count=1
```

A mentés eredményességéről a Linux-os gépeken elérhető `file` parancs segítségével lehet meggyőződni, ami a záró 2 bájt (magic number) alapján tudja azonosítani az állomány típusát (boot szektornak) és ennek megfelelően a fentebb látható kimenethez hasonlóan jeleníti meg a fájlról elérhető információkat.

1.1.2. A partíciók típusai

Amint már tudjuk, maximum négy partíció adatait tudja tárolni az MBR. Jogosan merülhet fel a kérdés, hogy ez a darabszám vajon minden esetre elegendő-e? Természetesen nem, mert életszerűtlen lenne ez a limit. Ezért teremtették meg a lehetőséget, hogy logikailag tovább bonthassuk a rendelkezésre álló tárterületet. Alapvetően a partíciók fájlrendszerek tárolására alkalmasak, de ahhoz, hogy elérjük a célunkat, be kell vezetni egy új típust, ami már biztosítja a szabadabb felosztást. Ezek alapján a partíciók típusai az alábbiak szerint alakulnak az MBR-en belül:

- **Elsődleges** (primary) partíció

  A fő rendszertöltő rekordban lévő fő partíciós táblában (Master Partition Table) elhelyezkedő partíciók. Egyes operációs rendszerek igénylik, hogy első (rendszer-) partíciójuk elsődleges legyen, ilyenek például az MS-DOS, Windows és a Minix. Más operációs rendszerek nem szabják ezt meg, ilyen például a Linux. A merevlemezen legfeljebb 4 elsődleges partíció lehet. A rendszerindításra kijelölt partíciót aktív partíciónak nevezzük.

- **Kiterjesztett** (extended) partíció

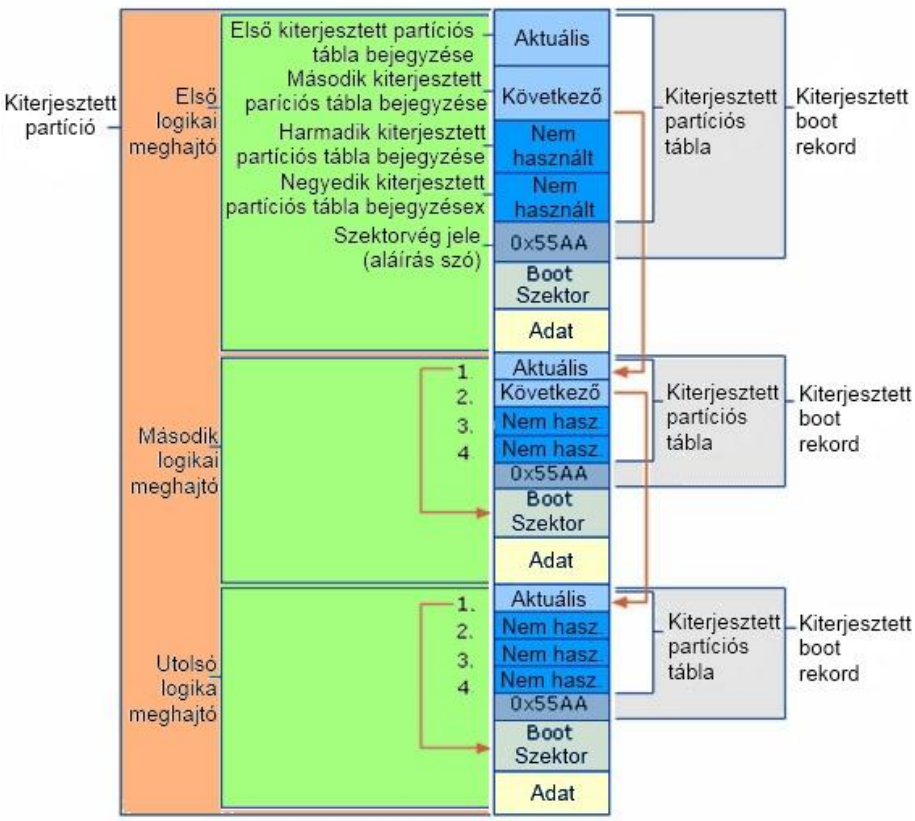
Mivel a legfeljebb négy partíció hamar kevésnek bizonyulhat és a fő rendszertöltő rekordban nincs több hely, szükségessé válhat a probléma megkerülése. A partíciótáblában *egyetlen kiterjesztett partíció lehet*, a többinek elsődlegesnek kell lennie. A kiterjesztett partíció egy olyan elsődleges partíció, amely nem fájlrendszert, hanem logikai partíciókat tartalmaz, így lehetővé válik több partíció használata. A kiterjesztett partíció aktuális struktúrája *egy vagy több kiterjesztett boot rekordot* (extended boot record - EBR) tartalmaz. Az első mindig a kiterjesztett partíció legelején található.

- **Logikai** meghajtó (logical drive)

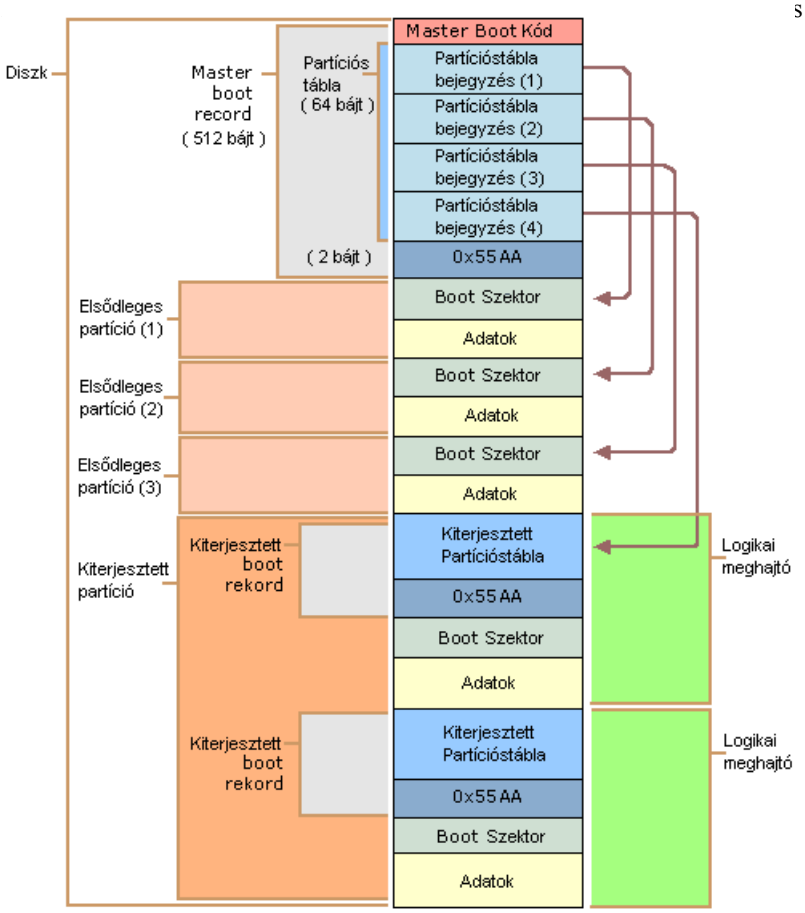
  Kiterjesztett partíción belül elhelyezkedő partíció. Fizikailag nem különül el attól. Mindig egy EBR vezeti be. Amennyiben további logikai meghajtó követi, úgy a hozzá tartozó EBR-ben található egy mutató a rákövetkező partíció EBR-jére, ezáltal egy láncolt listát alkotva. Ez azt jelenti, hogy egy kiterjesztett partíción létrehozható logikai meghajtók száma csak a rendelkezésre álló területtől függ.

  Az EBR szerkezete teljesen megegyezik az MBR struktúrájával, annyi megszorítással, hogy csak az első két partíció bejegyzés kerülhet felhasználásra, a maradék kettőnek üresnek kell lennie. A szektort végét jelentő két bájtos aláírás pedig ugyan az.

Ezt a láncolást az alábbi ábra szemlélteti:



s merevlemez partíciós struktúrája:



2. A betöltődés lépései

Amikor floppyról indítjuk az operációs rendszert a boot szektor olyan kódot tartalmaz, amely feladata a kernel betöltése a memória egy előre meghatározott helyére. Például egy Linux boot floppyn nem feltétlenül van fájlrendszer, a kernel egyszerűen egymást követő szektorokban található, mivel ez egyszerűsíti a boot folyamatát. Igaz, lehet fájlrendszerrel rendelkező floppyról is boot-olni, ilyen jellemzően a DOS indító lemeze.

Amikor merevlemezről boot-olunk a legegyszerűbb esetben - ilyet ír ki a DOS-os fdsik partíciókezelő program is - a master boot recordbeli (MBR) kód megvizsgálja a partíciós táblát (az MBR-belit is), hogy azonosítsa az aktív partíciót (azaz amelyik lett boot-olhatóvá téve), beolvassa annak boot szektorát, és elindítja az ott található kódot. Több operációs rendszer esetén már bölcsebb Boot manager programok használata, melyek segítségével menüből választhatunk a betöltendő (különböző operációs rendszereket tartalmazó) partíciók közül. A partíció boot szektorában található kód ugyanazt csinálja, mint egy floppy boot szektora: megkeresi majd beolvassa és elindítja a kernelt . 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. Ez már operációs rendszerenként különböző folyamat.

2.1. A DOS betöltése

A **DOS/360**-at, vagy röviden csak DOS-t az IBM 1966 júniusában adta ki. A DOS névvel a TOS (Tape Operating System, azaz mágnesszalag kezelésére képes operációs rendszer) ellenpárjaként lépett fel. Ezt az operációs rendszert speciális lyukkártyacsomagok összeállításával vezérelte az operátor vagy a programozó. Amikor a személyi számítógépek képesek lettek hajlékonylemezek használatára, a DOS elnevezés újra életre kelt, különböző gyártók különböző rendszereit jelentette, ezek közül a legfontosabb az IBM PC operációs rendszere, az MS-DOS, és a vele kompatibilis változatok, melyek uralták az IBM PC kompatibilis számítógépek piacát 1981 és 1995 között: PC-DOS, MS-DOS, FreeDOS, DR-DOS, Novell-DOS, OpenDOS, PTS-DOS,

ROM-DOS és mások. Ezen DOS változatok alapja a CP/M, melyet azonban kibővítettek a UNIX-hoz hasonló könyvtárszerkezet kezelésének képességével, és saját lemezformátumot vezettek be (FAT).

Single-user (egyfelhasználós) és single-tasking (egyfeladatos) rendszer, jogosultsági vagy védelmi rendszere nélkül. Ez alól csak egy kivétel volt: a TSR (Terminate and Stay Resident) programok, amelyek lefutása után a memóriában maradtak (64KB), így kódjuk kívülről is hívható, illetve futási állapotuk megőrizhető. A TSR kilépés előtt ráfűzi magát egy megszakításvektorra (INT 27H vagy INT 21H/31H), aminek aktiválása aktiválja a TSR-t magát. Pl.: az időzítő megszakítás hatására a TSR program rendszeres időközönként lefuthat. A TSR technika hasznosnak bizonyult a DOS, mint egyfeladatos operációs rendszer számos hiányosságának enyhítésére. Segítségével különféle hardver-illesztőprogramok, nyomkövetők és kisebb segédprogramok készültek (illetve a vírusok egyik jellemző tevékenysége, hogy fertőzni tudják a végrehajtható fájlokat így sokszorosítva magukat). A DOS készítői a rendszert csak egy egyszerű operációs rendszernek szánták viszont nagy sikere lett és a mikroszámítógépek elterjedésének idején azok 70%-án DOS futott.

A DOS operációs rendszer a következő lépések mentén töltődik be:

- a boot szektorban található kód a DOS BIOS bővítését betölti a `0000:0600` memóriacímre, amely az `IO.SYS` állományban található.
- ezután betöltődik a kernel, amely az `MSDOS.SYS` állományban helyezkedik el. [megj: a Windows 9x szériától kezdve a kernel össze lett fűzve az IO.SYS fájllal és az MSDOS.SYS mint egy konfigurációs fájl létezett tovább]
- a kernel ezután betölti a `CONFIG.SYS` állományt a konfigurációs paraméterek felolvasására, feltéve, hogy létezik ez a fájl. (pl. itt lehet megadni, hogy milyen parancsértelmezőt [shellt] szeretnénk használni - alapesetben ez a `COMMAND.COM`)

```
DEVICE=HIMEM.SYS (az 1 Mb feletti memóriarész kezelése)
DEVICE=MOUSE.SYS
DEVICE=EMM386.EXE
FILES=30 (az egyszerre megnyitva lehető fájl-ok száma)
COUNTRY=036 (A magyar szokásoknak megfelelően írja ki pl. a dátumot, számokat, stb.)
SHELL=COMMAND.COM (Itt adhatunk meg saját parancsértelmezőt.)
```

- a parancsértelmező betöltése és elindítása

  A parancsértelmező két részben helyezkedik el a memóriában.

  - *rezidens rész*

    állandóan a memóriában van. Ez a rész felel a hibakezelésért és a tranziens rész betöltéséért.

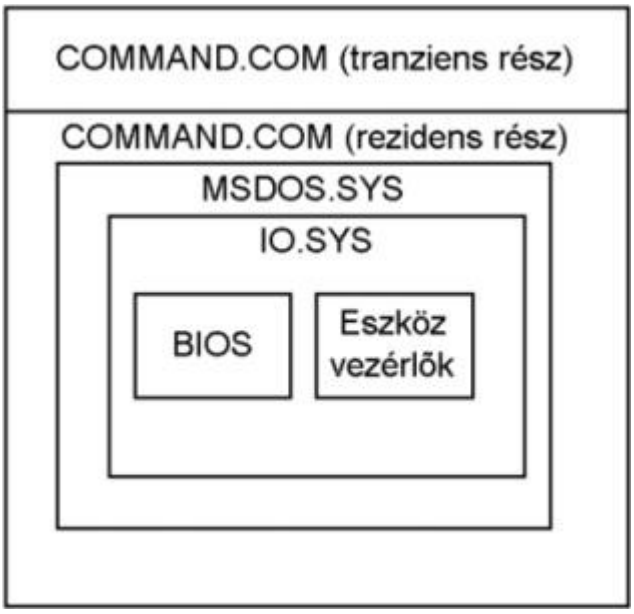
  - *tranziens rész*

    A hagyományos memóriának (<640 K) a felső régiójába töltődik. Ez nagyobb programok futtatásakor felülíródhat. Ha kilépünk a felhasználói programból, akkor a rezidens rész ellenőrzi a tranziens rész hibátlanságát, és ha kell, újratölti azt. A tranziens rész feladata a belső DOS parancsok értelmezése és a DOS programok futtatása (**.**BAT, **.**EXE,**.**COM).

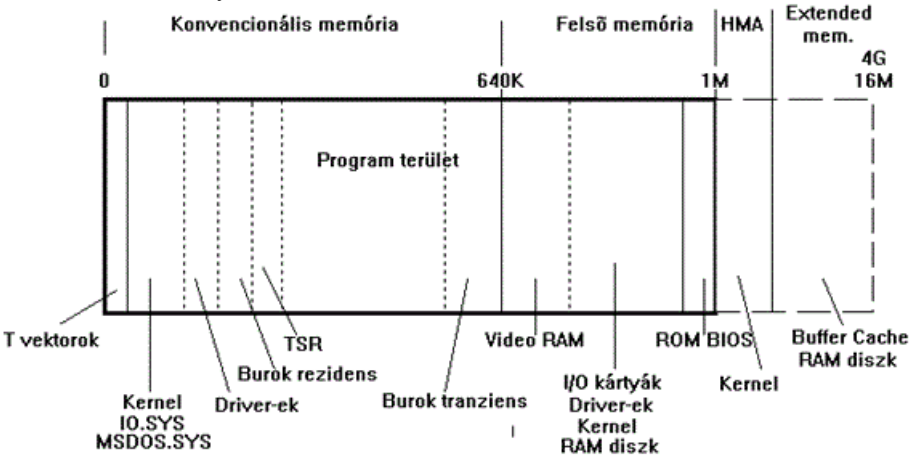
- az `AUTOEXEC.BAT` fájl elindítása, ha létezik. Ez a fájl a betöltődés után automatikusan lefuttatandó utasításokat tartalmazhatja.

```
SET PROMPT=$P$G (A prompt tulajdonságait jellemző környezeti beállítás, lásd help)
SET PATH=C:\DOS;C:\SYSTEMS (Az adott könyvtárakban keresi a végrehajtandó fájlt, ha az
aktuális könyvtárban nem találta meg.)
DOSKEY (Az előzőekben begépelt parancsokat hívhatjuk vele elő.)
SMARTDRV.EXE (Lemezgyorsító program)
REM ...
REM Feladata: a különböző környezeti (PATH, PROMPT, stb.) változók beállítása;
REM memóriarezidens programok (TSR)  betöltése (SMARTDRV, DOSKEY, stb.)
REM és egyéb olyan dolgok elvégzése, amelyeket nekünk kellene kézzel begépelnünk
minden rendszerindításnál.
```

A fentieket szemléltető ábra:



A fentebb bemutatott részek elhelyezkedése a memóriában:



A boot szektor által keresett BIOS bővítménynek és magának a kernelnek is összefüggő szektorokban kell lenniük és az első két bejegyzésként kell szerepelniük a mappában.

A DOS alapvetően a FAT (File Allocation Table - Fájl Allokációs Táblázat) fájlrendszert használja az állományok tárolására, erről részletesen a következő fejezetben olvashatunk. Amit itt még fontos megemlítenünk, az a DOS által alkalmazott algoritmus az egyes partíciók meghajtó betűjelhez történő rendelése.

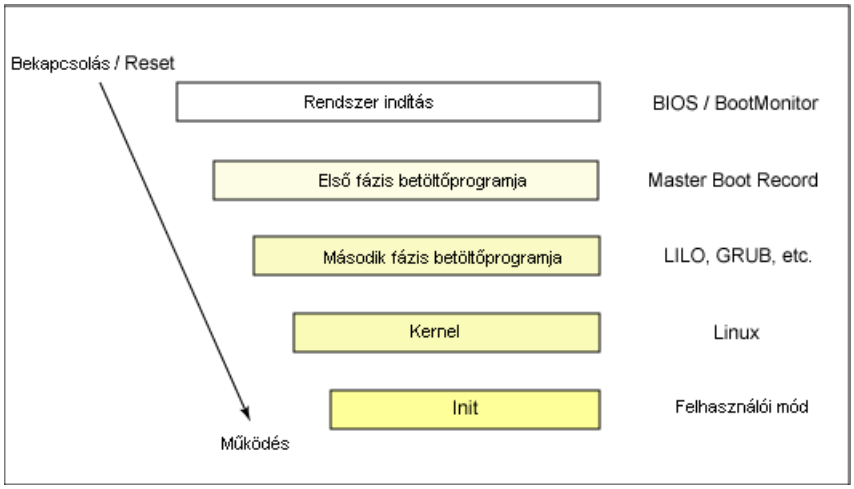
A DOS-ban a meghajtók azonosításra betűjelek kerülnek bevezetésre. A bevett gyakorlat szerint az "A" és "B" fent van tartva a floppy meghajtóknak. Azon rendszereken, ahol csak egy meghajtó található, a DOS mindkét betűt ahhoz rendeli. Ez megkönnyíti a floppyról floppyra való másolást, vagy egy program futtatását floppyról, miközben egy másikon fér hozzá az adatihoz. A merevlemezek eredetileg a "C" és "D" betűket kapták. A DOS csak egy aktív partíciót tudott meghajtóként kezelni. Ahogy több meghajtó támogatása is elérhetővé vált, ez átalakult egy olyan folyamattá, ahol első körben (Pri. Master, Pri. Slave, Sec. M., Sec. Sl.) az elérhető merevlemezek elsődleges partíciója kapott egy meghajtó betűt, aztán a következő körben a kiterjesztett partíciók logikai meghajtói, harmadsorban bármely nem aktív, elsődleges meghajtó kapott nevet (ahol az ilyen hozzáadott partíciók léteztek és DOS által ismert fájlrendszerrel rendelkeztek). Végezetül a DOS betűket rendel az optikai lemezmeghatókhoz, RAM diszkekhez és egyéb hardverhez. A betűhozzárendelések általában a merevlemezek féltérképezésekor történik, amikor a meghajtók betöltődnek, ugyanakkor az eszközvezérlők utasíthatják a DOS-t, hogy különböző betűket adjon a hálózati meghajtóknak, például tipikusan az olyan betűket, amelyek az ABC vége felé találhatók. Miután a DOS ezeket a betűket közvetlenül használja (nem úgy, mint a /dev mappa Unix-szerű rendszereknél), össze lehet ezeket kevertetni egy új hardver hozzáadásával, amelynek meghajtó betűre van szüksége. Példa erre az elsődleges partícióval rendelkező hardver hozzáadása, ahol a már létező hardver logikai meghajtókat tartalmaz a kiterjesztett partíciókon: az új meghajtó olyan betűt

fog kapni, amelyet egy kiterjesztett partíció kapott volna a logikai meghajtón. Sőt, egy pusztán kiterjesztett partíciókban logikai meghajtókkal rendelkező új merevlemez hozzáadása is megbontaná a RAM diszkek és optikai meghajtók betűsorrendjét. Ez a probléma fennállt a Windows 9x verzióin keresztül az NT-ig, amely megőrzi a betűket addig, amíg a felhasználó meg nem változtatja.

Fontos még kiemelni, hogy vannak fenntartott eszköznevek is a DOS-ban, amelyeket nem lehet fájlnévként használni, függetlenül a kiterjesztésüktől, ezeket perifériás hardverekre való alkalmazási output küldésére használják. Ezek a megkötések továbbá számos Windows verzióra is hatással vannak, néhány esetben ütközéseket és biztonsági sérülékenységet okozva. Egy nem teljes listája ezeknek a neveknek: NUL:, COM1: vagy AUX:, COM2:, COM3:, COM4:, CON:, LPT1: or PRN:, LPT2:, LPT3: és CLOCK$. A kettőspont nem minden esetben szükséges, pl: **echo "Eltüntetett kimenet > NUL".**

2.2. A Linux betöltése

A Linux betöltődési (rendszerindítási) folyamata sem tér el az általános modelltől. Azaz az áram rákapcsolása után az első lépések változatlanul a gép indulásával függenek össze: az alaplapi BIOS a videokártya BIOS-ának inicializálást követően inicializálja önmagát (SCSI eszközök esetén annak BIOS-a is betöltődik), megjelenik a hardverekről szóló összegző ablak és kezdődik a vadászat a boot szektor után. Itt is a már ismert lépések zajlódnak le, először az MBR kódja töltődik be, majd pedig a második fázisban az adott operációs rendszer kernelét indító kód.



2.2.1. Betöltőprogramok

A Linux betöltő programjai közül leggyakrabban a LILO (LInux LOader) és a GRUB (GRand Unified Bootloader) közül választhatunk telepítéskor. Miután a LILO nehézkesebben kezelhető, mint újabb társa a GRUB, ezért manapság inkább ezt telepítik. A GRUB egyik előnye, hogy már ismeri az alapvető fájlrendszereket (ext2, ext3, reiserfs) a LILO-val szemben, amely csak nyers szektoronként tekint a lemezre és ezért a kernel frissítése után mindig újra kellett inicializálni, hogy bootolható legyen. A GRUB éppen ezért vezeti be a három lépcsős betöltési fázist, beiktatva az első és a második köztes egy köztes (másfelediknek nevezett lépést) a kernelt tartalmazó fájlrendszer detektálására. Több operációs rendszer használata esetén a GRUB egy menüben kínálja fel az elérhető kerneleket. Ezek közül tudunk választani és igény esetén további indítási paramétereket is megadhatunk.

```
grub>kernel /bzImage-2.6.14.2
[Linux-bzImage, setup=0x1400, size=0x29672e]
grub>initrd /initrd-2.6.14.2.img
[Linux-initrd @ 0x5f13000, 0xcc199 bytes]
grub>boot
Uncompressing Linux... Ok, booting the kernel.
```

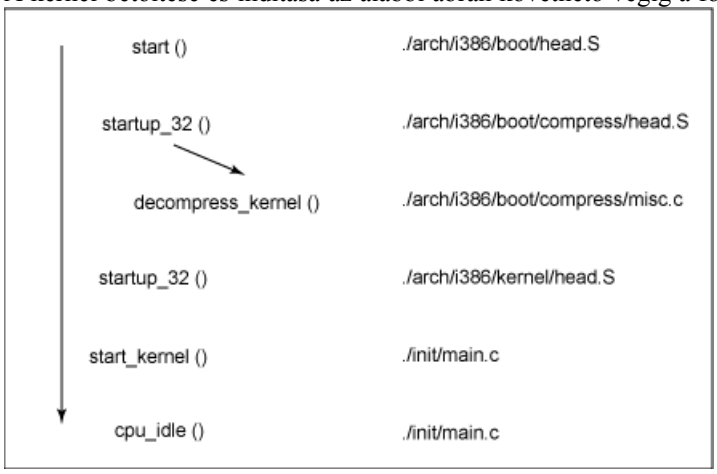
(A LILO érdekességeként említhetjük meg, hogy magának a LILO betöltésének és a kernel indításának folyamatát ötletesen úgy valósították meg, hogy a LILO karakterlánc szakaszosan (egyesével) kerül kiírásra. Az első "L" betűt akkor írja ki, miután a Lilo átmozgatta magát a kezdeti 0x00007c00 címről egy másik helyre, például a 0x0009A000 címre. Az "I" jelenik meg, mielőtt elkezdené a másodszintű betöltőkódját. A másodszintű betöltő írja ki a következő "L"-t, betölti a rendszermag részeire mutató leírókat, és végül kiírja a

végső "O" betűt. A leírók a 0x0009d200 címen helyeződnek el. A rendszerindítási üzenet és a parancssor konfiguráció függvényében kiíródik. A "tab" megnyomására a GRUB-hoz hasonlóan a promptnál teszi lehetővé, hogy a felhasználó rendszert válasszon, és parancssori opciókat adjon át a rendszermagnak, a meghajtóinak és az "init" programnak. Ezen kívül környezeti változók is megadhatók ennél a pontnál.)

2.2.2. A kernel betöltése

Miután a betöltőprogram a kernel képmását (image) betöltötte a memóriába, kezdetét veszi a következő fázis. A kernel önmagában nem futtatható, mert tömörített formában kerül letárolásra, ezért a fejlécében egy olyan rutin található, amely (minimális hardver ellenőrzést követően) kibontja magát a memóriába, majd meghívja a kernelt és elkezdődik a kernel bootja. A kernel neve lehet `zImage`, ha 512 bájtnál kisebb a mérete és `bzImage` (big compressed image) ha ezt meghaladja.

A kernel betöltése és indítása az alábbi ábrán követhető végig a fontosabb lépések és állományok megadásával.



Az első lépés az indító assembly rutin meghívása, amely (egy i386 esetén) a `./arch/i386/boot/head.S` fájlban található. Ennek a rutinnak a feladata a minimális hardver ellenőrzés és a startup_32 rutin meghívása (a `./arch/i386/boot/compressed/head.S` fájlból). Ez állítja be az alapkörnyezetet és kezdi meg a kernel kibontását (decompress_kernel rutin meghívása a `./arch/i386/boot/compressed/misc.c` fájlban). [ez az a pont amikor megjelenik a már említett: `Uncompressing Linux... Ok, booting the kernel.`] Ezután kerül meghívásra maga a kernel (a `./arch/i386/kernel/head.S` fájlban található - másik - `startup_32` függvény meghívásával), amely már alapvető memóriakezelést (lapozást) biztosít a folyamatok indításához. A kernel magja ez után (az `init/main.c` állományban található) `start_kernel()` függvény meghívásával indul el. Ebben a fázisban számos inicializáló folyamat zajlik le, többek között a megszakítások, a memóriakezelés és a kezdeti RAM diszk betöltése. A végén pedig elindul az első felhasználói módban futó folyamat: az init. Ezután az üresjárati folyamat következik, amely már lehetővé teszi az ütemezőnek a folyamatok feletti kontrollt (a mutlitasking megteremtését).

Pár megjegyzés a RAM diszk jelenlétére és szükségességéhez. A RAM diszk nem más, mint egy virtuális diszk, amely a rendszermemóriából lett leválasztva és ugyan úgy használható mint egy hagyományos meghajtó, azzal a különbséggel, hogy a sebessége drasztikusan nagyobb (míg a merevlemezek pozicionálási ideje 8-9 ms, addig a memória hozzáférési ideje 60-70 ns, azaz két nagyságrenddel gyorsabb). A linux kernel esetén a kezdeti RAM diszk (`initrd`) felcsatolásával egy átmeneti gyökér fájlrendszerhez jut a kernel, ami lehetővé teszi, hogy anélkül is eltudjon indulni sikeresen, hogy tényleges fizikai lemez is csatlakoztatva lenne. [gyökér (/) fájlrendszer nélkül nem tud működni] Egyrészről, ezen a kezdeti RAM diszken helyezhető el számos modul a kernelhez, hogy annak mérete minél kisebb lehessen, de mégis számos hardver konfigurációt támogasson. Másrészről beágyazott rendszerek esetén (ahol nincs fizikai meghajtó) ez a diszk szolgálhat a végleges gyökér fájlrendszerként is, illetve biztosíthat hozzáférést hálózaton keresztül elérhető fájlrendszerekhez is. Általánosságban azért a betöltés végére ez a kezdeti RAM diszk lecserélődik egy tényleges fájlrendszerre valamilyen merevlemezes egységen.

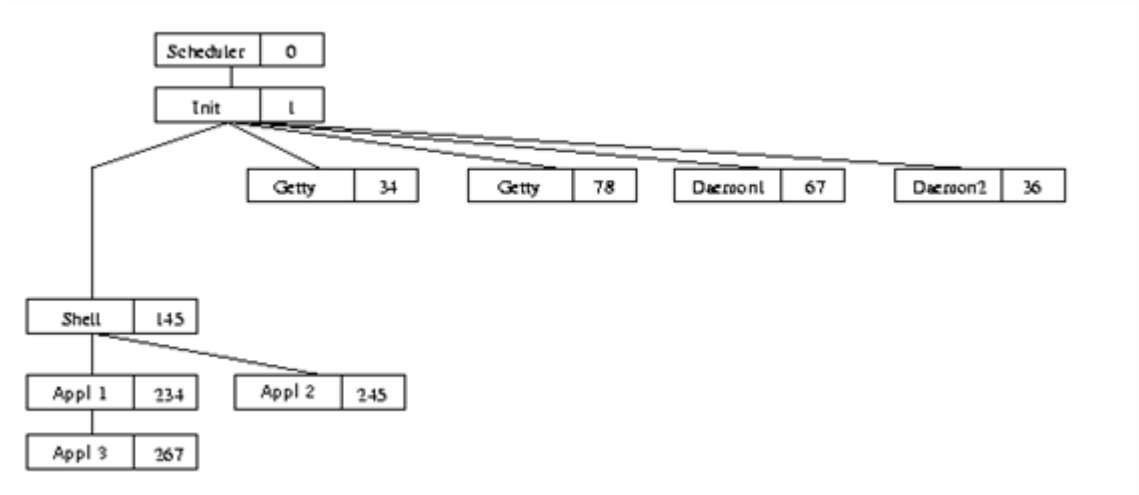
2.2.3. Az init folyamat

Miután a kernel betöltődött és inicializálódott, elindul az első felhasználói üzemmódban futó folyamat. Ennek a neve init (fizikailag többnyire - de nem kötelezően - a `/sbin/init` helyen található), és ez az első program,

ami már szabványos C könyvtári függvények használatával lett fordítva. Miután a Unix egy valódi preemptív multitaszking rendszert kínál - ami minden éppen futó folyamathoz meghatározott mennyiségű processzoridőt rendel - szükséges egy ütemező (Scheduler) aminek az azonosítója (PID) 0 lesz. Ez azt jelenti, hogy ez az első elindított folyamat, de a legtöbb Unix-ban ez a processz láthatatlan marad. Az ütemező - mint első processzus - az init folyamatot indítja, ami szükségszerűen mindig az 1-es PID-et kapja. Ez a folyamat a szülője a rendszer összes többi processzének. Ez a folyamat felelős a rendszer inicializálásáért és ezáltal az összes további processzért, amelyek menet közben indulnak el. Az init processz által indított processzeknek a két legfontosabb típusa:

- *démon*: a szolgáltatásokért felelősek (meghatározott helyzetekre meghatározott akciókat hajtanak végre), nem kötődnek terminálhoz, a háttérben futnak és közvetlenül az init indítja

- *felhasználói folyamat*: terminálhoz kötöttek (`getty` program) és csak bejelentkezés után indíthatóak.



Az indítás konfigurációját a `/etc/inittab` fájl segítségével végezhetjük el. Természetesen egy fájl túlságosan nagy és összetett lenne minden egyes viselkedés|konfiguráció (rendszerfolyamatok, szolgáltatások, ...) leírására, éppen ezért a Linux rendszerek futási szinteket határoznak meg.

Fontos megjegyezni, hogy ennek a hagyományos init démonnak megjelent egy - a szigorú szinkron és blokkolódó viselkedést leváltó - *esemény alapú helyettesítője*, amely **Upstart** néven ismert. Az Upstart *aszinkron* működésének köszönhetően sokkal jobban használható a mai modern gépek esetében, lehetővé téve, hogy az egyes események bekövetkezésekor reagáljon csak a rendszer - tipikusan ilyen az USB-s eszközök le- illetve felcsatlakoztatása. A tervezése során kiemelt figyelmet fordítottak arra, hogy visszafelé teljesen kompatibilis legyen a System V stílusú init folyamattal.

Első megjelenésével a 2006-ban debütált 6.10-es Ubuntu (Edgy Eft) kiadásában találkozhatunk, de azóta több gyártó is beemelte a rendszereibe (Debian, RedHat - a vállalati ágban is, openSUSE, de már a Google Chrome OS operációs rendszere és a Nokia Maemo 5 is ezt alkalmazza).

2.2.4. Futási szintek

Minden egyes futási szintnek saját beállításai vannak az indítandó processzekre vonatkozóan. Egy futási szint a rendszernek egyfajta szoftver konfigurációja, ami csak meghatározott processzeket indít el illetve állít le. Alapvetően 7 futási szint (8 - disztribúció függően lehet egy S szint is az 1-es szinonimájaként) létezik, ezek közül pedig néhánynak kitüntetett szerepe van. Fontos megjegyezni, hogy a rendszer indulásakor csak egy futási szint kerül futtatásra - és nem szekvenciálisan egymás után futtatódnak le. [az 5-ös futási szintre történő belépéshez nem hajtja végre az alatta lévő 4 szintet is]

A futási szintet határozzák meg a gép állapotát az indulás után. Ennek megfelelően az alábbi futási szinteket alakítják ki tipikusan:

- egyfelhasználós mód (single-user mode)

- többfelhasználós mód hálózat nélkül (multi-user mode without network services started)

- többfelhasználós mód hálózati támogatással (multi-user mode with network services started)
- rendszer leállítása
- rendszer újraindítása

A pontos beállítások disztribúcióról disztribúcióra változhatnak, ezért a telepítés előtt célszerű elolvasni a kiadási megjegyzéseket! Jellemzően a 4-es futási szint a leginkább eltérő, a többi alapvetően a következők szerint alakul:

- Standard futási szintek
  - 0 - Leállítás: a rendszer leállítására
  - S - egyfelhasználós mód: nincs hálózat és semmilyen szolgáltatás (démon) sem indul el [a disztribúciók többségében funkcionálisan az 1-es futási szintet használják erre]
  - 6 - újraindítás: a rendszer újraindítása
- Tipikus futási szintek
  - 1 - egyfelhasználós mód: szolgáltatások nem indulnak, hálózat nem elérhető, csak rendszergazda (root) bejelentkezés engedélyezett
  - 2 - többfelhasználós mód: szolgáltatások nem indulnak, hálózat nem elérhető
  - 3 - többfelhasználós mód hálózati szolgáltatások indulnak: normál rendszer indítás
  - 4 - nem használt: saját célokra alkalmazható
  - 5 - X11: a 3-as szint grafikus felülettel kiegészítve

3. fejezet - Fájlok és fájlrendszerek

Az operációs rendszer egyik legalapvetőbb feladata, hogy a használathoz és a működéshez szükséges adatokat elérhetővé tegye. Ez a fájlok és a fájlrendszerek területe és a továbbiakban ezek fogalmaival illetve használatával ismerkedünk meg.

1. Alapvető fogalmak

1.1. Fájl fogalma

A számítógépen lévő információtárolási egysége a fájl (file). Egy fájl tartalma a gép szempontjából vagy adat, vagy program. **Hagyományos** értelemben véve bináris vagy szöveges adatot tartalmazó állományok. A bináris fájlok között érdemes megkülönböztetni legalább a processzor által végrehajtható utasításokat tartalmazókat, amelyeket futtatható fájloknak is nevezhetünk. (megj: a szöveges fájlok között is létezik minden platformon végrehajtható, ezek szokták szkripteknek hívni. Windows alatt ezek a .bat kiterjesztésűek, míg az .exe és a .com a bináris változatuk.)

A fájlban tárolt adat tetszőleges, lehet szöveg, kép, hang stb. Az adatok formájára nézve nincs előírás, a gyakorlatban nagyon sokféle formátum létezik. A fájlt minden operációs rendszer használja, konkrét megjelenése azonban már az operációs rendszertől függ.

A fájlok alapvető jellemzői:

- fájlnév: a név amin keresztül elérhetjük
  - DOS esetén legalább 1, maximum 8 betű szóköz nélkül. Tartalmazhatja az angol ABC 26 betűjét, számjegyeket, kötőjelet és alulvonást.
  - Windows 95... 7: legalább 1, max. 255 betű, szóköz, több pont és ékezet is megengedett.
  - Linux ext2: legalább 1, max. 255 betű, szóköz, több pont és ékezet is megengedett.
- kiterjesztés: nem kötelező, DOS esetén maximum 3 karakter lehet. Általában a fájl jellegére utal.
- méret: a fájl mérete bájtokban.
- dátum: Általában három különbözőt is tárolnak: fájl létrehozásának, utolsó módosításának és utolsó hozzáférésnek dátuma.
- idő: Itt már csak a fájl létrehozásának és utolsó módosításának idejét tárolják.

Ezen felül pedig már fájlrendszer függő, hogy még milyen adatokat tárol. Lehet a fájl használatára utaló attribútumok, vagy fejlettebb esetekben jogosultságkezelés, netán tömörítés vagy titkosítás, esetleg a gyorsabb elérés érdekében többszintű indexelés.

1.2. Fájlrendszer fogalma

A fájlrendszer szűkebb értelemben nem más, mint a fájlok tárolására és rendszerezésére kialakított struktúra egy háttértároló eszközön (pl. floppy lemezen vagy merevlemezen vagy CD-ROM-on), vagy annak egy részén (pl.: merevlemez egy partícióján). A fájlok azonosítása a fájlnév alapján történik a fájlrendszeren belül. Kezdetben itt meg is állt minden, később a CP/M részéről bevezetésre került a - már rég elavultnak tekintendő - meghajtó betűjel (A:, C:, stb.) rendszere, míg a UNIX irányából a könyvtárszerkezet fogalma jelent meg. (A betűjelek használatának több hátránya is van, egyfelől korlátozott számban állnak rendelkezésre (az angol ábécé 26 betűjét használhatjuk) másfelől jelentősen csökkenhet az egész fájlrendszer áttekinthetősége (ha leülünk egy ismeretlen gép elé nem biztos, hogy azonnal rájövünk, mit rejt a T: vagy az S: meghajtó).)

A fájlrendszer UNIX esetén egy tiszta fogalom, mert egyetlenegy gyökér (root - /) könyvtárral rendelkezik, míg a DOS és Windows estén nem, mert ott meghajtónként van egy-egy. UNIX esetén további fájlrendszerek a meglévő struktúra tetszőleges könyvtárába csatolhatóak, azaz mountolhatóak (érdemes megjegyezni, hogy ez a

lehetőség Reparse Point (szabad fordításban kevés jelentést hordozó újraelemző pont) formájában a Microsoft Windows vonatkozásában is megjelent, bár jelentősen lassítva az ezzel az állományok elérését).

A fájlrendszerek manapság már sok más elemmel egészültek ki, mint például az a képesség, hogy meghatározhassuk egy felhasználó maximum mennyi tárterületet használhat fel (quota), de elérhető még a jogosultságkezelés különböző formái is, vagy a titkosítás, a tömörítés és akár a naplózás is. Utóbbi egy olyan fájlrendszer jellemző, ahol az egyes műveletek egy ún. tranzakciós területen is tárolódnak, így valami váratlan esemény esetén (pl: áramkimaradás) garantálják a fájlrendszer konzisztens állapotának megtartását, hosszú ideig tartó fájlrendszer ellenőrzés (fsck vagy scandisk) nélkül. Érdekes lehetőség még egy adott pont alatti tárterület dinamikus megnyújtásának (kibővítésének) lehetősége is vagy éppen egy egész partíció lementése fájlba és annak felcsatolása (pl: CD - DVD lemezek tartalmánál).

Mára azonban már nem csak helyi fájlrendszerekkel találkozhatunk. Szervereken lévő adatokhoz való hozzáférést is biztosíthatnak hálózati protokollok segítségével (pl., NFS, SMB, SSH vagy 9P kliensek), vagy lehetnek virtuálisak és csak a virtuális adatokhoz való hozzáférési mód miatt léteznek csak (például a Linux procfs fájlrendszere az aktuálisan futó folyamatok elérésére).

Precízebben meghatározva: *egy **fájlrendszer** absztrakt adattípusok halmaza, amelyeket adatok tárolására, hierarchikus rendezésére, kezelésére, megtalálására illetve navigálásra, hozzáférésre, és visszakeresésére valósítottak meg*.

A legismertebb fájlrendszerek egy adattároló eszköz segítségével biztosítják, hogy elérhető a terület elérhető legyen egy fix méretű blokkokból álló sorozat képében (ahol az általában 512 byte méretű blokkokat gyakran szektoroknak is nevezik, de ez a méret lehet akár 1,2 vagy 4 kilobyte is). A fájlrendszer feladata, hogy ezeket a szektorokat fájlokká valamint katalógusokká szervezze össze, és tartsa nyilván, melyik szektor melyik fájlhoz tartozik, és melyik szektorok nem használhatók már tárolásra. A legtöbb fájlrendszer ezen felül bevezeti még a klaszter (cluster) fogalmát is, amely a lemezről lefoglalható legkisebb tárterület méretét határozza meg, ez többnyire több blokkból áll és mérete formázáskor dől el.

Ennek ellenére egy fájlrendszernek nem feltétlenül kell tárolóberendezést használnia mindenre. Egy fájlrendszer használható az adatok szervezésére és megjelenítésére is, ha azok tárolása vagy elérése dinamikusan történik (például a korábban említett módok közül a hálózati kapcsolat segítségével vagy virtuálisan létrehozott fájlrendszerek).

Ami viszont mindenképp létezik, az a fájlnevek és a fájlok összekapcsolását végző **metaadatok**, ami többnyire valamilyen index. A későbbiekben ezt látni fogjuk a DOS esetén a FAT táblázatban, míg Unix vonalon az i-node számoknál. További kérdés, hogy a fájlrendszer ezen egyszerű összekapcsoláson túl biztosítja-e a hierarchiába történő szervezését. Ez nem törvényszerűen létező funkcionalitás, mert a korai DOS-os fájlrendszerek nem ismerték, első megjelenése pedig Unix környezetben Dennis Ritchie nevéhez fűződik, aki kísérleti jelleggel vezette először be és eleinte csak néhány szintű lehetett. Végső soron ez vezetett el a mappák megjelenéséhez.

A hagyományos fájlrendszerek az alábbi legalapvetőbb szolgáltatásokat nyújtják: létrehoznak, mozgatnak vagy törölnek fájlokat vagy mappákat. Biztosítják a csonkolás (truncate), a kibővítés (append to), a létrehozás (create), a mozgatás (move), a törlés (delete) és a helyben módosítás funkciókat a fájlokra. Nem támogatják viszont a csonkolás a fájl elejétől funkciókat, de esetleg megengednek korlátlan beszúrást a fájl tetszés szerinti helyén, vagy törlést a fájlon belül. Ezeken felül viszont már óriási eltérések adódnak az egyes fájlrendszerek között. Ami viszont még ezek előtt fontos, hogy megismerjük részletesebben a mappák működését.

1.3. Mappa fogalma

Mappa alatt egy fájlrendszeren belüli entitást értünk, ami valójában egy *speciális fájl* amely fájlok neveit és a fájlrendszer függvényében a nevekhez tartozó további információkat tartalmaz. UNIX alapú fájlrendszer (ext2) esetén például az iNode számot, míg mondjuk a Windowsnál használható FAT esetén a fájl további attribútumait, dátumit, stb. . Ami ebből levezethető, hogy ha a mappa fájlokat tárol és ha önmaga is egy fájl akkor következésképp a mappákban létrehozhatunk további (al)mappákat is.

Érdemes megjegyezni, hogy a mappa szó mellett a magyar nyelvben nagyon gyakran használjuk a könyvtár és katalógus szót is ezzel teljesen egyenrangú jelentésben, de a DOS idejében elterjedt volt a tartalomjegyzék *(directory, catalog)* kifejezés is. Nagyobb problémát nem is jelent számunkra, hiszen világos miről beszélünk. Azonban érdemes figyelembe venni, hogy fordítás során az eredeti angol szavakat használjuk. Ugyanis a

könyvtár fordítása lehetne library is a directory helyett, ami viszont nagyon helytelen, mert a szaknyelv ennek teljesen más jelentést tulajdonít, nevezetesen a programok által felhasználható függvénykönyvtárakat jelenti. Az értelmezésen sok múlhat, mert a tartalomjegyzék szó kifejezőbb abban az értelemben, hogy a működése hasonló egy telefonkönyvhöz, ami tartalmazza a nevek listáját, de magukat a dokumentumokat már nem. Azonban a mai operációs rendszerek a grafikus felület előretörése kapcsán már grafikus ikonokat használnak ennek reprezentálására, ami már sokkal inkább hasonlít egy mappára.

Érdekesség, hogy a Microsoft annak érdekében, hogy leegyszerűsítse a rendszermappák kezelését a Windows rendszeren belül, bevezette a speciális mappák fogalmát, amelyek a felhasználó számára a mappa koncepció által egységesen jelennek meg, annak ellenére, hogy telepítésenként és verziónként más és más helyen lehetnek ugyanazon mappák (Program Files, Windows, Document and Settings). Ezzel kikerülte az elérési utak elgépeléséből eredő hibákat. (A megoldás a környezeti változók használata lett.)

A mappa szemlélet láthattuk, hogy folyamatosan alakult ki, de ami minden fájlrendszer esetén létezik az a legfelsőbb szintű gyökér mappa, ami fizikai adathordozó esetén formázáskor kerül kialakításra. A könyvtárak használata ha támogatott, akkor fastruktúrát alkotnak: a gyökérkönyvtárnál feljebb nem léphetünk, és onnét mehetünk le a mappák almappáinak az almappáiig, ameddig csak szeretnénk. Unix rendszereken az egész fájlrendszer egyetlen fastruktúrát alkot egyetlen gyökérrel, míg Windows rendszereken minden meghajtóhoz külön gyökérkönyvtár tartozik.

Számos operációs rendszer ismeri az aktuális könyvtár, avagy munkakönyvtár fogalmát, ahol a kiadott parancsok végrehajtódnak. Unix alap rendszereknél erre a `pwd` parancs szolgál, míg Windows esetén a `cd` parancs paraméterek nélküli kiadása ekvivalens ezzel. Amennyiben szkripteken belül szeretnénk ezen információhoz hozzájutni, így használhatjuk Unix esetén a PWD, míg Windows esetén a CD környezeti változó értékét.

```
# Linux
[adamkoa@kkk ~]$ echo $PWD
/home/adamkoa
[adamkoa@kkk ~]$

# Windows
C:\Users\adamkoa\Documents\TAMOP-Op.r.jegyzet\Book>echo %CD%
C:\Users\adamkoa\Documents\TAMOP-Op.r.jegyzet\Book

C:\Users\adamkoa\Documents\TAMOP-Op.r.jegyzet\Book>
```

Az egyes operációs rendszerek más és más módon jelzik a mappákat. UNIX alatt a könyvtárak neve előtt a hozzáférési jogoknál egy `d` betű található az első helyen részletes lista esetén.

```
[adamkoa@kkk proba2]$ ls -l
total 8
drwxrwxr-x  3 adamkoa adamkoa 4096 Apr 27  2010 proba
```

Míg Windows platformon egy `<DIR>` karakterláncot láthatunk:

```
d:\TAMOP-OpRendszerek\Book> dir
 A meghajtóban (D) lévő kötet Data.
 A kötet sorozatszáma: 7895-009C

 d:\TAMOP-OpRendszerek\Book tartalma:

2011.04.30.  20:35    <DIR>          .
2011.04.30.  20:35    <DIR>          ..
2011.04.06.  18:11           374 207 book.xhtml
2011.04.30.  20:45           371 903 book.xml
2011.04.30.  20:44           371 717 book.xml.bak
2011.04.04.  20:00    <DIR>          images
2011.04.01.  14:01    <DIR>          meta
               3 fájl           1 117 827 bájt
               4 könyvtár  11 812 753 408 bájt szabad
```

A mappák használata a leghatékonyabb eszköz fájljaink hierarchikus elrendezéséhez. Azonban ahhoz, hogy egy adott fájlt használni is tudjuk ismernünk kell az élrési útját.

1.4. Elérési útvonal

Az elérési utak két csoportba sorolhatóak:

- **abszolút** elérési út: ez mindig `/` (vagy `\`) jellel kezdődik, és a gyökérkönyvtárból indulva minden mappát felsorolunk a célunkig, pl:

  ```
  /home/adatok/reszletek
  ```

  Megjegyzendő, hogy DOS illetve Windows esetén ez érelemszerűen az aktuális meghajtóra vonatkozik, amennyiben ettől eltérőre kívánunk hivatkozni, akkor az elérési út előtt meg kell adnunk a meghajtó nevét, pl: `D:`.

- **relatív** elérési út: már valamilyen mélységben "beástuk" magunkat a könyvtárrendszerben és nincs kedvünk az egészet a gyökértől kezdve újra felsorolni. Ilyenkor az aktuális mappához viszonyítva tudjuk megadni a fájl helyét.

  A példa kedvéért tegyük fel, hogy a `/home/adatok/` könyvtárban állunk és a `reszletek`re szeretnénk hivatkozni (relatívan):

  ```
  ./reszletek
  ```

Az elérési utak megadásánál az alábbi jelöléseket használhatjuk rövidítésként a mappáknál:

- `.` : az aktuális mappa
- `..` : a szülő mappa
- `/` vagy `\`: a gyökérmappa
- `~` : UNIX esetén a felhasználó saját (HOME) mappája

1.5. Rejtett fájlok

Tekintsünk vissza a mappák kiíratását szemléltető ábrákra. Figyeljük meg, hogy az első esetben (UNIX) nem látjuk az aktuális és a szülő mappára mutató hivatkozásokat, pedig tisztában vagyunk azzal, hogy ezeknek létezniük kell, mert nélkülük nem tudja a fájlrendszer fenntartani a hivatkozásokat az egyes mappák között. Míg Windows esetén azonnal látjuk a `.` és `..` bejegyzéseket. Ennek vannak előnyei és kisebb hátrányai is. Alapesetben előnyös, hogy egy újonnan létrehozott mappában nem látunk egyetlen egy bejegyzést sem, mert hisz mi is lenne benne és ráadásul az egyszerű földi halandó nem biztos, hogy azonnal tudja hova társítani, mert miért is van két mappa az újonnan létrehozott mappámban és mik is azok? Hát persze, hogy a szükséges hivatkozások a fájlrendszer működéséhez.

A UNIX alapesetben ettől megkíméli a felhasználót, hiszen ritkán tartalmaz ez számunkra fontos információt, mert vagy tudjuk, hogy létezik és akkor minden rendben, vagy nem, akkor meg miért terheljük ezzel az embert. Ahhoz hogy itt is láthatóvá váljanak, az `ls` utasítást fel kell paraméterezni.

```
adamkoa@kkk proba2]$ ls -la
total 32
drwxrwxr-x    3 adamkoa adamkoa  4096 Apr 27  2010 .
drwx--x--x  116 adamkoa adamkoa 12288 Apr 18 14:04 ..
drwxrwxr-x    3 adamkoa adamkoa  4096 Apr 27  2010 proba
```

Máris láthatóvá váltak. A megoldás, hogy a UNIX azt az ötletet követi, hogy azok a fájlok, amelyek neve .(pont)-tal kezdődik, rejtettnek tekint és alapesetben nem mutatja. Ezzel a szülő és az aktuális mappára mutató hivatkozást is elrejtve. (Fontos megjegyezni, hogy a pont a fájlnév része és nem külön kezelendő!) Míg Windows esetén a rejtett fájl tulajdonság egy attribútum segítségével szabályozható az `attrib` parancs használatával.

```
ATTRIB { +H | -H } [meghajtó:][elérési_út]fájlnév
```

Miután elrejtettünk egy fájlt, azt alapból nem látjuk. Láthatóvá csak úgy válik, ha külön kérjük a listázásukat. Ez DOS/Windows alatt a `dir` parancs `/A` módosítójának használatát jelenti:

```
dir /A:H
```

Ebben az esetben csak a rejtett fájlok fognak megjelenni.

1.6. Speciális fájlok Unix alatt

A Unix rendszerben található speciális fájltípusok:

- **link**: hivatkozás más fájlra (később részletes leírás találhatsz róla).

  A linkeket egy '`l`' betű azonosítja. (szintén később lesz részletezve, hogy léteznek még hard linkek is, amelyek megkülönböztethetetlenek)

```
lrwxrwxrwx termcap
```

- **nevesített csővezeték** (named pipe): folyamatok közötti kommunikációra ad lehetőséget, oly módon hogy az egyik alkalmazás kimenetét egy másik alkalmazás bemenetére köti. Egyszerű FIFO (First In First Out) pufferekről van szó, amikbe írni és amikből olvasni lehet. Például a különböző terminálokon futó processzeket is pipe-al lehet összekapcsolni.

  A pipeokat egy '`p`' betű azonosítja a hozzáférési jogok sztringjében és a létrehozásuk a **mkfifo** vagy **`mknode`** utasítással történhet. Részletesebb bemutatásuk a csővezetékekről szóló részben található.

```
prw-rw---- mypipe
```

- **socket**: speciális fájl melyet a pipeokhoz hasonlóan, folyamatok közötti kommunikációra használnak, de immár hálózatos környezetbe szerver-kliens kommunikáció során.

  Socketre példaként a /var/run/printer fájlt hozhatjuk fel azokon a rendszereken, amelyeken a printer démon fut, ez a fájl socket-ként létezik. Közvetlenül a felhasználó számára szolgáló (parancssoros) socket alkalmazás nincs, arra csak programon belül a `bind` rendszerhívás segítségével utasíthatjuk a rendszert.

  A socketeket az '`s`' betűvel jelzett fájlok jelentenek

```
srwxrwxrwx printer
```

- **eszköz** (device) fájlok: ezek a hardver elemeit reprezentálják a fájlrendszerben így hozzáférési jogokat adhatunk az egyes elemeknek, és utasításokban használhatjuk közvetlenül magát az eszközöket. Ide tartozik a billentyűzet, a terminál, a merevlemez, a memória, a floppy stb.

  Az eszközök a hozzáférési jogok sztringjében a `c` (karakteres elérésű) vagy a `b` (blokk elérésű eszköz) betű az eszköz kommunikációs módját mutatja, azaz karakterekben (kódtáblának megfeleltetve) vagy blokkokban (átalakítás nélkül) történik az átvitel.

```
crw------- /dev/kbd            # billentyűzet
brw-rw---- /dev/hda            # első IDE buszos HD (primary master)
# A merevlemezen található partíciók /dev/hda1 - /dev/hda15 eszköznévvel érhetőek el.
# Ebből az első négy szolgál az elsődleges partíciók jelölésére, a felette lévőek a
logikai meghajtók, illetve az 5-ös a
# kiterjesztett partíció.
brw-rw---- /dev/hdb            # második IDE buszos HD (primary slave)
brw-rw---- /dev/hdc            # harmadik IDE buszos HD (secondary master)
brw-rw---- /dev/hdd            # negyedik IDE buszos HD (secondary slave)

brw-rw----  /dev/sda           # első SCSI merevlemez egység
# Hasonlóan a merevlemezekhez /dev/sda1 - /dev/sda15 néven érhetőek el az egyes
partíciók.
# /dev/sdb ... /dev/sdd szintén azonos értelemben.

lrwxrwxrwx /dev/cdrom  -> hda  # link a CD-ROM meghajtóra

crw-rw---- /dev/ttyS0 to /dev/ttyS3 # 0 -3 sz. soros portok

crw------- /dev/tty1 - /dev/tty6    # virtuális konzolok (AltF1-F6)
```

A 3 legfontosabb, fizikailag nem létező eszköz:

- `crw-rw-rw- /dev/null`

Elfogad és elnyel minden bejövő adatot, kimenetet nem produkál. Teljesen hasonló, mint a DOS speciális fájlainál említett NUL eszköz. Tipikus felhasználása egy parancs kimenetének eltüntetése.

```
cat $filename 2>/dev/null >/dev/null
# Ha "$filename" nem létezik nem lesz hibaüzenet (2>)
# Ha "$filename" létezik, akkor a tartalma nem jelenik meg (>)
# Ez így tipikusan akkor hasznos, ha egy programnak a visszatérési értékét akarjuk
tesztelni
# és nem érdekes semmilyen kimenete sem.
# A 2> és a > részletes jelentése a -következő- átirányítást bemutató alfejezetben.
```

- `crw-rw-rw- /dev/random`

  Változó hosszúságú, véletlenszerűen generált karaktersorozatokat állít elő. Közvetlenül nem igazán hasznos, de nézzük az alábbi példákat a használatához:

```
Két bájtos decimális egész kinyerése:
od -An -N2 -i /dev/random
# -An kikapcsolja a cím megjelenítését
# -N a megadott méret bájtokban
# -ia kimeneti formátum megadása (egész)

Ha pedig egy tartományból szeretnénk kapni számokat - itt most 100 és 1000 között:
echo $(( 100+(`od -An -N2 -i /dev/random` )%(1000-100+1) ))
# itt most kihasználtuk, hogy a shell képes alapvető egész aritmetikás számításokat
végezni- ez a $(( ... )) jel jelentése
# illetve, hogy egy parancson belül egy másik parancs kimenetét szeretnénk
felhasználni - ez a ` ... ` rész jelentése.

Lehet használni átmeneti (temp) fájlok létrehozására is:
touch `od -An -N2 -i /dev/random`.tmp
```

- `crw-rw-rw- /dev/zero`

  Csupa 0 karakterekből álló karaktersorozatot állít elő. Használható például fájlok biztonsági törlésére úgy, hogy az eredeti tartalmat több lépcsőben nullákkal írjuk felül.

```
dd if=/dev/zero of=$FILE bs=$BLOCKSIZE count=$BLOCKS
# Fájl kinullázása - a $ jellel bevezetett változókat kell lecserélni a megfelelő
értékekre.
# fájl neve, blokkméret - fájlrendszer függő - a fájl mérete blokkokban megadva
# if= bemeneti fájl
# of= kimeneti fájl neve
```

1.7. Átirányítás

A szabványos bemenet (`stdin`), a szabványos kimenet (`stdout`) és a szabványos hibakimenet (`stderr`) átirányítása. Minden egyes elindított folyamat esetén három alapértelmezett eszköz kerül hozzárendelésre a folyamathoz. Ez a szabványos bemenet, ahonnan a program a futás során a beérkező adatokat olvassa, a szabványos kimenet, ahova a program ír és a szabványos hibakimenet (`stderr`), ahol a program a futás során fellépő hibákra adott hibaüzeneteit írja. Alapesetben a `stdin` a billentyűzet, az `stdout` és `stderr` pedig a képernyő - egészen pontosan pedig a szülő folyamat által használt fájlok, mert egy folyamat a kimenetét mindig a szülőjének adja át!

Mind a bemenet, mind pedig a kimenet (hibakimenet) átirányítható egy tetszőleges állományba. Az átirányítás jelöléseit a program utolsó paramétere után kell feltüntetni. Több átirányítás esetén azok végrehajtása balról jobbra történik.

`< állomány`: stdin átirányítása (a megadott állományból olvas)

`> állomány`: stdout átirányítása (a megadott fájlba ír, létező állomány estén annak tartalmának törlésével és felülírásával)

`>> állomány`: stdout átirányítása (a megadott fájlba ír, létező állomány esetén annak végéhez való hozzáfűzéssel)

`2> állomány`: stderr átirányítása (a megadott fájlba írja a hibaüzenetet)

`&> állomány`: stdout és stderr átirányítása ugyanabba a fájlba

`2>&1`: a stderr-t ugyanoda irányítja, ahova a stdout irányítva lett

`1>&2`: a stdout-ot ugyanoda irányítja, ahova a stderr irányítva lett

Példák:

`dir > lista.txt`	a dir átirányítása a lista.txt állományba - ha a fájl eddig nem létezett, létrehozza - ha létezett, felülírja
`dir >> lista.txt`	a dir átirányítása a lista.txt állományba - ha a fájl eddig nem létezett, létrehozza - ha létezett, a végéhez hozzáfűzi
`sort < nevek.txt > lista.txt`	a nevek.txt rendezése a lista.txt állományba

1.8. Csővezetékek

A cső vagy csővezeték (pipe, pipeline) a programok egy olyan sorozata, amelyek a szabványos folyamaik által vannak összekötve, azaz a Program1 kimenetét (stdout) a Program2 bemenetére (stdin) köti. A második program az első által produkált eredményt tekinti bemenetként. Több programból álló csővezeték is létrehozható. A cső létrehozása az esetleges átirányítások elvégzése előtt történik. Megadása: a két (vagy több) parancsot a | (függőleges vonal) jellel elválasztva adjuk ki egy sorban.

Ennek során egy névtelen csővezeték jön létre, ami a folyamatok közötti kommunikációt hivatott lekezelni és leggyakrabban az operációs rendszer I/O alrendszerén keresztül kerül megvalósításra. Emlékezzünk vissza, hogy Unix alatt speciális fájlként is létre lehet hozni őket, ezek lesznek a nevesített csővezetékek.

Példák névtelen csővezetékekre:

`dir | sort`	a dir kimenetének rendezése
`dir | sort > \temp\lista.txt`	a dir kimenetének rendezése, a kimenet a lista.txt állományba átirányítása
`dir | sort | more`	a dir kimenetének rendezése majd képernyőnkénti megjelenítése a more által

Részletes alkalmazásuk a későbbiekben tárgyalandó szűrőkről szóló alfejezetben. Most csak a szemléltetés érdekében lássuk, hogy a csővezetékek mennyire megkönnyíthetik a gép életét, azaz mennyire tehermentesíthetik a diszk alrendszert pl egy adatbázis feltöltés esetén ha az adatok tömörített formában vannak és így kikerülhetjük az átmeneti fájlok létrehozását:

```
mkfifo --mode=0666 /tmp/namedPipe
gzip --stdout -d file.gz > /tmp/namedPipe
```

Egy másik terminál ablakban pedig a mysql parancssorában:

```
LOAD DATA INFILE '/tmp/namedPipe' INTO TABLE tableName;
```

Hasonlóan demonstrálhatjuk a folyamatok közötti együttműködést, ha az egyik terminálon előkészítünk egy csővezetéket az adatok tömörítésére, míg egy másikban pedig szolgáltatjuk az adatokat:

```
mkfifo my_pipe
gzip -9 -c < my_pipe > out.gz
```

A másik ablakban pedig átadjuk a tömörítendő adatokat:

```
cat file > my_pipe
```

2. Fájlrendszerek Microsoft platformon

A Microsoft a korai operációs rendszereihez (MS-DOS, Windows) kifejlesztett egy fájlrendszert, a FAT-ot. Ma a Windows használja mind a FAT (File Allocation Table) mind pedig az újabb NTFS (New Technology File System) fájlrendszereket. A FAT rendszer korábbi változatai, a (FAT12 és FAT16) esetében a fájlnevek hossza korlátozott, és létezik egy korlát a gyökérkönyvtárban lévő bejegyzések számára is, de megkötések vannak a FAT-rendszerrel formázott lemezek vagy partíciók méretére is. A VFAT, amely a FAT12 és a FAT16 bővítése volt, és a Windows NT 3.5-nél jelent meg, de a Windows 95 számára is kiadva, megengedte a hosszú fájlneveket – long file names (LFN). A FAT32-ben néhány, a FAT12-ben és a FAT16-ban meglévő korlát megmaradt, de ezek elhanyagolhatók.

Az NTFS a Windows NT operációs rendszerrel egy időben jelent meg, megengedi a hozzáférési ellenőrző lista – access control list (ACL) alapú ellenőrzést, a többszörös fájl hozzáférést, a beégetett kapcsolatot, a jellemzővel való indexelést, a feltöltöttség követést, a tömörítést és a kapcsolódási pont létrehozást másik fájlrendszerek számára (ezek a „junction”-ök), azonban ezek a szolgáltatások nem teljesen dokumentáltak és eleinte külön segédprogramot kellett letölteni a használatukhoz.

A legtöbb operációs rendszertől eltérően, a Windows használja egyedül a meghajtó betűjel fogalmat felhasználói szinten, két lemez vagy partíció egymástól való megkülönböztetésére. Például, a C:\WINDOWS\ elérési út egy WINDOWS könyvtárat jelent, ami a C betűvel jelölt partíción helyezkedik el. A C meghajtó a leggyakrabban használt elsődleges lemezpartíció, ami a Windows-ban létrejött, és amiről betöltődik. Ez a „hagyomány” olyan erős, hogy a régebbi Windows változatoknál az a meghajtó, amin a Windows rendszer található, az csak a C meghajtó lehet. A „C” meghajtóval kapcsolatos hagyomány egészen a DOS-ig nyúlik vissza, ahol a korábbi szakaszban ismertetett módon történik a betűjelek kiosztása. Kiegészítésként még érdemes tudni, hogy a hálózati meghajtókhoz szintén hozzá kell rendelni egy-egy betűt a MAP paranccsal. Viszont mióta a Windows grafikus felhasználói felület segítségével áll kapcsolatban a használójával, a Windows kézikönyvei a katalógusokra úgy hivatkoztak, mint egy olyan mappára amely fájlokat tartalmaz, és ezeket a grafikus mappa ikonnal jelölték.

Mikor használjunk FAT-ot?

Ha több mint egy operációs rendszert futtatunk a számítógépünkön, akkor mindenképpen érdemes lehet egy közös FAT partíciót létrehozni. Minden olyan program vagy adat, amit két vagy több rendszer között kell megosztanunk FAT16 vagy esetleg FAT32-es fájlrendszerre kell helyeznünk (kivéve, ha a másik rendszer is NT alapú). De ilyenkor észben kell tartanunk, hogy a FAT fájlrendszeren lévő adatokat semmi sem védi meg. Más szóval hozzáférhet az adatokhoz és olvashat, módosíthat, és ráadásul törölheti is azokat. A legtöbb esetben ez a hálózaton át is lehetséges. Tehát ne tároljunk fontos és érzékeny adatokat FAT16, FAT32 fájlrendszerű partíción vagy merevlemezen.

2.1. Fájl allokációs táblázat (File Allocation Table - FAT)

A fájlok elhelyezkedését a lemezen a DOS a fájl elhelyezkedési táblában (File Allocation Table - FAT) tartja nyilván. Minden fájlhoz a FAT-ben egy bejegyzés-láncolat tartozik, melynek első elemét a katalógus-bejegyzés egyik mezője tartalmazza. A FAT egy önindexelt tömb, melynek minden mezője a lánc következő elemének megfelelő indexet tartalmazza, azaz megmutatja, hogy melyik logikai szektorban helyezkedik el a fájl következő része, honnan lehet megtudni a rá következő szektor sorszámát.

A FAT kulcsfontosságú eleme a DOS-nak, hiszen sérülésekor a lemezen található fájlok elérhetetlenné válhatnak. Fontossága miatt a lemezeken több (általában kettő) példány is található. Ezt részint a lemez fizikai sérülésének lehetősége, másrészt a logikai hibák létrejöttének lehetősége (pl. áramszünet a FAT módosítása alatt) indokolja. Normális esetben a második FAT példányba mindig csak ír a DOS, de az első FAT példány sérülése esetén a második példányból próbálja meg kinyerni az információkat, ami alapján korrigálhatja az első példányt is.

Egy partíció egyenlő mértékű klaszterekre, vagyis folyamatos hely blokkokra osztható. A klaszter méret függ a használt FAT típusától és a partíció méretétől. Tipikusan elmondható, hogy a klaszter méretek valahol 2 kb és 32 kb között vannak. Minden egyes fájl elfoglalhat egyet, vagy többet ezekből a klaszterekből a méretétől függően, ezért egy fájlt az ilyen klaszterek egy láncolata jellemez. Azonban ezek a láncok nem feltétlenül egymást követően tárolódnak egy diszk területén, hanem gyakran töredeznek az adatterületen. Minden egyes FAT fájlrendszer verzió különböző méretű FAT bejegyzéseket használ, a méretet mutatja a név, pl. FAT16 esetén 16 bites bejegyzések vannak, míg a FAT32 32 biteseket. A különbség azt jelenti, hogy a FAT táblája a FAT32-nek nagyobb számú klasztereket fed le, mint a FAT16, nagyobb partíció méreteket is megengedve. Ez hatékonyabb helykihasználást tesz lehetővé, mint a FAT16, mert ugyanazon a meghajtón a FAT32 tábla kisebb

klasztereket címez meg, amely így kevesebb helyet pazarol. Jegyezzük meg, hogy a FAT32 csak 28 bitet használ a lehetséges 32-ből, a fennmaradó négy bit általában 0, de ezek fenn vannak tartva és érintetlennek kell maradniuk. A FAT olyan bejegyzések sorozata, amely minden klaszter lefed a partíción. minden bejegyzés a következő dolgok egyikét jegyzi fel:

(0)000h	a bejegyzés (klaszter) nem használt
(0)002h-(F)FEFh	a fájlt tartalmazó következő klaszter sorszáma
(F)FF0h-(F)FF7h	fenntartott terület fizikailag hibás terület jelölése ((F)FF7h)
(F)FF8h-(F)FFFh	a klaszter a lánc utolsó eleme (a fájl vége)

A főkatalógus

A főkatalógus - vagy más néven gyökér-könyvtár - minden meghajtó legfelső könyvtára. E könyvtárnak nincs tulajdonos könyvtára és nem is lehet törölni. Azonban míg más könyvtárak mérete dinamikusan változik és a bennük tárolható fájlok és könyvtárak (azaz katalógusbejegyzések) számának csak a lemez kapacitása szab határt, addig a főkatalógus mérete már a formattáláskor determinált (ez a méret a boot-szektorból kiolvasható) és utólag - a lemezen lévő adatok mozgatása nélkül - nem változtatható. Ezen kívül a gyökér-könyvtár nem tartalmaz két speciális, - más könyvtárakban azonban megtalálható - az aktuális könyvtárat jelentő '.' ill. a tulajdonos-könyvtárat jelentő '..' bejegyzést. Emellett a főkatalógus szigorúan egymás után következő szektorokon (a FAT-ok után) foglal helyet, míg más könyvtárak - a fájlokhoz hasonlóan - akár széttöredezve (nem egymást követő szektorokon) is elhelyezkedhetnek.

Katalógusbejegyzések

A mappabejegyzés egy speciális fájltípus, amely egy könyvtárat képvisel (manapság úgy ismert, mint mappa). Minden fájl vagy bejegyzés, amely itt tárolódik egy 32 bájtos bejegyzés a táblában. Ezen bejegyzések fájlokat vagy újabb - a könyvtáron belüli - (al)könyvtárakat definiálhatnak. Minden egyes ilyen feljegyzi a nevét, kiterjesztését, jellemzőit,a keletkezés dátumát és időpontját, az első klaszter címét a fájlnak és végül a méretét a fájlnak.

A DOS fájlnevek a következő karaktereket tartalmazhatják:

- nagybetű A-Z-ig
- számjegyek 0-9-ig
- !, #, $,%,&,’,(,),-,@,^,_,`,{,},~
- csak FAT32 esetén: +, ., , , ;, =, [,]
- ASCII értékek 128-255 között
- a sorvégi szóközök megengedettek, de nem képzik a név részét.

Bájt Ofszet	Hossz	Leírás
0x00	8	a fájl/könyvtár neve. A fájl nevének első karaktere speciális jelentéssel bír, ugyanis ez határozza meg a fájl-bejegyzés érvényességét. Értékei és értelmezésük a következő:

0x00	a bejegyzés még nincs használva
0x05	a fájl nevének első karaktere E5h (a 32 alatti ASCII kódú karakterek ugyanis nem jeleníthetőek meg, míg az E5h igen)
0x2E	a bejegyzés az említett két speciális bejegyzés ('.','..') egyike (a kérdést a második karakter vizsgálata dönti el)
0xE5	a bejegyzésnek megfelelő fájl/könyvtár törlésre került, így az már nem valós. A helyreállító segédprogramok (pl. undelete) ezt a karaktert helyettesítik egy nyomtathatóval a visszaállítás során.

0x08	3	kiterjesztés

0x0b 1 Fájl attribútumok

Bit	Mask	Description
0	0x01	a fájl csak olvasható (read-only) (nem módosítható/törölhető)
1	0x02	a fájl rejtett (hidden)
2	0x04	a fájl rendszerfájl (system)
3	0x08	a bejegyzés kötetnév (volume label)
4	0x10	a bejegyzés könyvtár (directory)
5	0x20	a fájl még nem került archiválásra (archive)
6	0x40	eszköz (csak belső használatra, sose található diszken)
7	0x80	nem használt

A 0x0F érték egy hosszúfájlnév bejegyzést takar.

0x0c 1 Fenntartott; NT és későbbi verziók kódolási információja

0x0d 1 Létrehozás ideje finom felbontásban (10 ms-es tartományban)

0x0e 2 Létrehozás ideje (óra, perc, másodperc)

0x10 2 Létrehozás dátuma. (év, hónap, nap)

Az év 0 értéke 1980-at jelöl, a 127 - legnagyobb érték - pedig 2107-et!

0x12 2 Utolsó hozzáférés dátuma.

0x14 2 Fenntartott.

0x16 2 Utolsó hozzáférés ideje.

0x18 2 Utolsó módosítás dátuma

0x1a 2 Első klasztere címe.

0x1c 4 Fájl mérete. [4 bájt = 32 bit => 4GB!]

A főkatalógusok kivételével minden könyvtár legalább két bejegyzést tartalmaz: az aktuális könyvtárat reprezentáló '.' és a tulajdonos-könyvtárat meghatározó '..' nevű könyvtárakat. A bejegyzés nevének ill. kiterjesztésének fel nem használt karakterei 32-es ASCII kódú (szóköz) karaktereket tartalmaznak.

Az könyvtár-bejegyzéseket a fájloktól az attribútum 4. bitje különbözteti meg. Az alkönyvtárak gyakorlatilag katalógusbejegyezések sokaságából álló fájloknak tekinthetők. Elhelyezkedésük - csakúgy mint a fájloké - a FAT alapján határozható meg.

A 3. bitet bekapcsolva tartalmazó bejegyzésből általában az egész lemezen csak egyetlen egy található, mégpedig a gyökérkönyvtárban. Ez a bejegyzés a lemez kötet-azonosítóját definiálja, melynek lekérdezése és állítása a DOS LABEL parancsa segítségével lehetséges.

A hosszú fájlnevek kezelése pedig egy trükk segítségével valósul meg. Miután a régi DOS-os programok kötetcímkékkel nem foglalkoznak, így használatuk tökéletes lehet a régi DOS-os nevek mellett a hosszú verziójuk tárolásához. Egy fájlnév esetleg több ilyen fájlt is igényel a tároláshoz, mert egy bejegyzésben maximum 26 karaktert tárolnak el.

2.2. NTFS

Az NTFS (New Technology File System) a Microsoft Windows NT és utódainak fájlrendszere. A Windows 95, 98, 98SE és ME nem képesek natív módon olvasni az NTFS fájlrendszert, bár léteznek programok erre a célra is. Az NTFS a Microsoft korábbi FAT fájlrendszerét váltotta le, melyet az MS-DOS és a korábbi Windows verziók esetén használtak. Az NTFS több újdonsággal rendelkezik a FAT fájlrendszerrel szemben, mint például a metaadatok támogatása, fejlettebb adatstruktúrák támogatása a sebesség, a megbízhatóság és lemezterület-felhasználás érdekében, valamint már rendelkezik hozzáférésvédelmi listával és megtalálható benne a naplózás is. Legnagyobb hátránya a korlátozott támogatottsága a nem-Microsoft operációs rendszerek oldaláról, mivel a pontos specifikáció a Microsoft szabadalma. Az NTFS-nek három verziója létezik: v1.2 – NT 3.51, NT 4 v3.0 – Windows 2000 v3.1 – Windows XP, Windows Server 2003 és 2008, Windows Vista, Windows 7

Az NTFS-en belül minden fájlokkal kapcsolatos információt (fájlnév, létrehozás dátuma, hozzáférési jogok, tartalom) metaadatként tárolnak. Ez az elegáns, bár absztrakt megközelítés lehetővé tette újabb fájlrendszer funkciók létrehozását a Windows NT fejlesztése során – egy érdekes példa az Active Directory által használt indexelő mezők hozzáadása. A fájlnevek Unicode (UTF-16) formátumban vannak tárolva, azzal a változtatással, hogy a fájlrendszer nem ellenőrzi az UTF-16 szerinti szabványosságot.

Az NTFS B+-fákat használ a fájlrendszer adat tárolására. Bár bonyolult megvalósítani, rövidebb hozzáférési időt biztosít bizonyos esetekben. Egy fájlrendszer naplót használnak magának a fájlrendszer integritásának (de nem az egyes fájloknak) a biztosítására. Az NTFS-t használó rendszerek biztonságosabbak, ami egy kiemelten fontos követelmény a Windows NT-k korábbi verzióinak instabil mivolta miatt. A megvalósítás részletei nem nyilvánosak, így külső gyártóknak nagyon nehéz NTFS-t kezelő eszközöket előállítani.

Az NTFS 5.0 a harmadik NTFS-verzió volt, amit a Microsoft közzétett. Több új lehetőséget tartalmazott: kvóta, sparse fájl-támogatás (ami lehetővé teszi, hogy egy nagy fájl üres részei ne foglaljanak helyet), elosztott link követés és titkosítás (Encrypting File System, EFS).

NTFS Log (napló)
: Az NTFS egy naplózó fájlrendszer, amely a belső összetett adatstruktúrák és indexek konzisztens mivoltát biztosítja egy rendszer összeomlás esetén is, lehetővé téve a nem rögzített változások visszavonását a kötet újbóli csatlakoztatásakor. (Azaz nem magát az adatot védi.)

USN Journal (**U**pdate **S**equence **N**umber **Journal**, azaz szekvencia napló)
: Minden egyes kötet esetén fenn van tartva egy napló, amely a köteten történt változásokat naplózza. Az egyes bejegyzések rendelkeznek egy azonosító számmal (szekvencia) és a módosult fájl nevével és a végrehajtott módosítás leírásával. Ezek az adatok a művelet visszavonását nem teszik lehetővé.

Alternatív adatfolyam (ADS – alternative data stream)
: Az alternatív adatfolyamok (ADS) lehetővé teszik egy fájl csatolását több adatfolyamhoz. Például a szöveg.txt nevű fájl tartalmazhat egy ADS-t szöveg.txt:titok.txt néven (fájlnév:ads formátumban), amit csak az ADS név ismeretében vagy speciális fájlkezelő programokkal lehet elérni. Az alternatív adatfolyamok nem fedezhetőek fel a fájl méretének vizsgálatával, de elvesznek az eredeti fájl törlésével, illetve mikor a fájlt ADS-t nem támogató meghajtóra másolják (például egy FAT partícióra, floppy lemezre, hálózati megosztásra). Bár az ADS hasznos szolgáltatás, észrevétlenül csökkentheti a szabad területet.

Kvóta
: A fájlrendszer kvótákat az NTFS 5-ben vezették be. Lehetővé teszik az adminisztrátorok számára, hogy korlátozzák az egyes felhasználók által lefoglalható tárterületet. Lehetővé teszi az adminisztrátor számára azt is, hogy lekérdezze az egyes felhasználók által lefoglalt terület méretét. Beállítható, hogy a felhasználó mikor kapjon figyelmeztetést, majd mikor tiltsa le a lemezfoglalást a rendszer. Az alkalmazások, melyek lekérik a szabad tárterületet, a kvótának megfelelő szabad területet fogják visszakapni.

Encrypting File System (EFS)
: Erős, és a felhasználó számára átlátszó, fájl és könyvtár titkosítást biztosít az NTFS köteteken. Az EFS együttműködik az EFS szolgáltatással, a Microsoft CryptoAPIjával és az EFS File System Run-Time Library-vel (FSRTL).

  Az EFS szimmetrikus kulccsal (File Encryption Key, FEK) titkosít, mivel azzal nagyobb teljesítmény érhető el, mint egy aszimmetrikus kulcsú megoldással. A szimmetrikus kulcsot egy, a titkosítást igénybe vevő felhasználó publikus kulcsával titkosítják, és ezt a titkosított adatot a titkosított fájl alternatív adatfolyamába mentik. A visszafejtéshez a felhasználó privát kulcsával visszafejtik a szimmetrikus kulcsot, majd a szimmetrikus kulccsal visszafejtik a titkosított fájlt. Mivel ez a folyamat rendszer szinten történik, a felhasználó számára láthatatlan. Arra az esetre, ha a felhasználó elvesztené a privát kulcsát (például adatvesztés történt), helyreállító ügynökök visszafejthetik a fájlt.

  Ezért is függ a titkosítás erőssége a felhasználók által használt jelszavaktól, ami sebezhetővé teszi az eljárást helyi támadásokkal szemben.

Fájltömörítés
: Az NTFS képes a fájlok tömörítésére az LZ77 algoritmus (melyet a ZIP fájl formátumban is használnak) használatával. Bár a tömörített fájlok írása és olvasása transzparens módon történik, a Microsoft ajánlása

szerint a tömörítés használata kerülendő szerverrendszereken és profilokat tároló hálózati megosztásokon, mert jelentős processzorterhelést idéz elő.

Kötet csatolási pont
: Hasonló a Unix csatolási pontokhoz, ahol egy másik fájlrendszer gyökerét csatolják egy könyvtárba. Az NTFS esetén ez lehetővé teszi fájlrendszerek csatolását külön meghajtó betűjel (például C:) használata nélkül.

Könyvtár csatlakozások (junction)
: Hasonló a kötet csatolási ponthoz, de kötetek helyett könyvtárakra hivatkozik. Például ha a C:\A könyvtár a C:\B könyvtárra hivatkozik, akkor megnyitása esetén a C:\B könyvtár tartalmához fog hozzáférni. A könyvtár csomópont megegyezik a Unix szimbolikus hivatkozásaival, bár a Unix szimbolikus linkek fájlokra és könyvtárakra is alkalmazhatóak.

Hard link (kemény kötés)
: A hard link hasonlít a könyvtár csomópontokhoz, de fájlokra, és nem könyvtárakra érvényes. A hard link csak az azonos köteten elhelyezkedő fájlokon használható, mivel egy Master File Table (MFT) rekord jelzi a hivatkozást.

3. Fájlrendszerek UNIX alatt

A mai modern UNIX-rendszerek számos különböző típusú állományrendszert támogatnak. Ezek két nagy csoportra oszthatók: lokális és távoli állományrendszerek. Az előbbi a rendszerhez közvetlenül csatlakoztatott berendezéseken tárolja az adatokat, míg az utóbbi lehetőséget nyújt a felhasználónak, hogy távoli gépeken elhelyezkedő állományokhoz férjen hozzá.

A korai UNIX-rendszerek egyetlenegy típusú állományrendszer kezelésére voltak képesek, így a rendszerek fejlesztői választásra kényszerültek. Mint azt majd a későbbiekben látni fogjuk, a Sun Microsystems által kidolgozott vnode/vfs interfész tette lehetővé több állományrendszer egyidejű alkalmazását is.

Az alap koncepciót a System V sorozat állományrendszere (az s5fs) alkotta, ennek alapjait vette át a Linux is. Azonban készült a BSD vonalon egy másik rendszer, amely Fast File System (FFS) néven jobb hatékonyságú, robusztusabb és több funkcionalitást biztosít korábbi társánál.

A Unix más operációs rendszerekhez hasonlóan logikai szinten, nem pedig diszk szinten kezeli az állományrendszert. A kernel logikai blokkokban szervezi az állományrendszert, egy blokk lehetséges mérete: $512*2^k$ byte, ahol a k kitevő tipikus értékei a 0–5 tartományba esnek, megválasztásánál az alábbi szempontokat szokás figyelembe venni:

- minél nagyobb a blokkméret, annál hatékonyabb az adathozzáférés, kisebb az adategységre jutó keresési időveszteség, nagyobb az átvitel sávszélessége
- minél kisebb a blokkméret, annál kisebb a fájlok tördelődése

A kezdeti s5fs implementációban 512, majd a későbbiekben 1Kb-os blokkméretet alkalmaztak.

3.1. Az i-node táblázat

A fájlok ezen blokkokban tárolódnak, és a hozzáférést az i-node értékeken keresztül tudjuk elérni. Az i-node az index-node rövidítése, ezzel utalnak arra, hogy a fájlt alkotó blokkokat egy indexelt struktúrán keresztül érik el. Az i-node-ok az egyes fájlokra vonatkozó minden információt tartalmaznak, kivéve a fájl nevét. A fájl nevét és az i-node azonosítóját (ami egy egyedi pozitív egész szám és számozása: 1, 2, 3, ... alakú) a mappákban tárolja az operációs rendszer. Ez a mappabejegyzés reprezentálja a fájlt. Az adatblokkok pedig az egyes fájlokban tárolt információkat tartalmazzák. Egy tipikus i-node szerkezete a következő:

- állományleíró (jellemzők a név kivételével)
- cím mutatók

Az állományleíró az alábbi információt tartalmazza egy fájlról:

- a fájl típusát és hozzáférési jogait
- a tulajdonos felhasználót és csoportot (ezáltal tudja kezelni a jogokat)
- a fájl méretét (blokkokban)
- időbélyegeket az i-node létrehozásának, módosításának és utolsó hozzáférésének
- egy referencia számlálót, ami megadja az i-node-ra mutató (fájlrendszeren belüli) hivatkozások számát

Ezen meta-információk után jönnek az állomány fizikai elhelyezkedésért felelős címmutatók. Ebből három féle létezik:

- közvetlen (direkt)
- egyszeresen indirekt
- kétszeresen indirekt
- háromszorosan indirekt.

Közvetlen mutatóból 12 db létezik. Ezek fájladatot tartalmazó adatblokkokra mutathatnak. 1 Kb blokkméret esetén a direkt címmutatókkal max. 12 Kb méretű fájlra lehet hivatkozni. Ha a fájl kisebb helyet foglal el, akkor maradnak üres direkt címmutatók. [A címmutató mező mérete 4 bájt!]

A fenti séma problémája, hogy az indirekciók időigényesek, nagy fájlok esetén lassul az adatelérés. Azonban a bevezetés időszakára jellemző statisztika szerint az állományok kb. 85%-a kisebb 8 Kb-nál (így elférnek a direkt blokkokban), és ezen állományok kb. 48%-a kisebb 1 Kb-nál! Ebből következik, hogy indirekt elérésre ritkán van szükség, így az effektív adathozzáférés gyors. Nagy fájlok esetén pedig a hosszabb keresési idő elkerülésére egyéb megoldásokat is alkalmaznak a gyorsítás érdekében (például buffer-cache).

Ha a fájl mérete pedig meghaladja a 12 blokkot, akkor az i-node következő mezője egy olyan blokk címét fogja tartalmazni, ami `blokkméret/4` bájt további adatblokk direkt címmutatóját tartalmazhatja. Ha a fájlméret miatt szükséges, úgy a következő i-node bejegyzés (a kétszeres indirekt címmutató) egy olyan blokkra fog mutatni, mely további blokkméret/4 bájt indirekt címmutatót tartalmazhat, melyek mindegyike 1-1 olyan blokkra mutathat, melyek megint blokkméret/4 bájt direkt címmutatót tartalmazhatnak. (Ez 1 Kb-os blokkméret esetén 256 db mutatót jelent, 512 bájtos esetén 128 db-ot, 4 Kb esetén pedig már 1024-et)

Nézzük példák segítségével, hogy ez mekkora maximális fájlméretet tesz lehetővé:

- 512 bájtos blokkméret esetén

12 blokkra direkt címmutató mutat	6K	
128 direkt címmutató az egyszeres indirekciós blokkban	64K	(a címmutatók 4 byte-osak és 512/4 = 128)
128*128 direkt címmutató a kétszeres indirekciós blokkban	8192K	
128*128*128 direkt címmutató a háromszoros indirekciós blokkban	1 048 576K	

Összesen:	1 056 838K	= 1 GB

- 1 Kb-os blokkméret esetén

12 blokkra direkt címmutató mutat	12K	
256 direkt címmutató az egyszeres indirekciós blokkban	1024K	(a címmutatók 4 byte-osak és 1024/4 = 256)
256*256 direkt címmutató a kétszeres indirekciós blokkban	65 536K	

256*256*256 direkt címmutató a háromszoros indirekciós blokkban	16 777 216K	

Összesen:	16 843 788K	= 16 GB

- 4 Kb-os blokkméret esetén

12 blokkra direkt címmutató mutat	48K	
1024 direkt címmutató az egyszeres indirekciós blokkban	4096K	(a címmutatók 4 byte-osak és 4096/4 = 1024)
1024*1024 direkt címmutató a kétszeres indirekciós blokkban	4 194 304K	
1024*1024*1024 direkt címmutató a háromszoros indirekciós blokkban	4 294 967 296K	

Összesen:	4 299 165 744K	= 4 TB

Mint már említettük, a fájl neve nem jelenik meg az i-node-ban. Az i-node csak a hivatkozás azokra az adatblokkokra, amiket a fájl használ, és az a hely, ahol az operációs rendszer az attribútumokat tárolja.

Az alkalmazott fájlrendszertől függően a gyökérkönyvtár egy meghatározott i-node-on van, legtöbbször az 1 vagy 2 azonosítójú i-node-on. Linux alatt minden katalógus (folder, könyvtár, mappa) nem egyéb, mint egy fájl, ami az illető könyvtárban lévő fájlok neveiből és i-node számaiból álló listát tartalmazza.

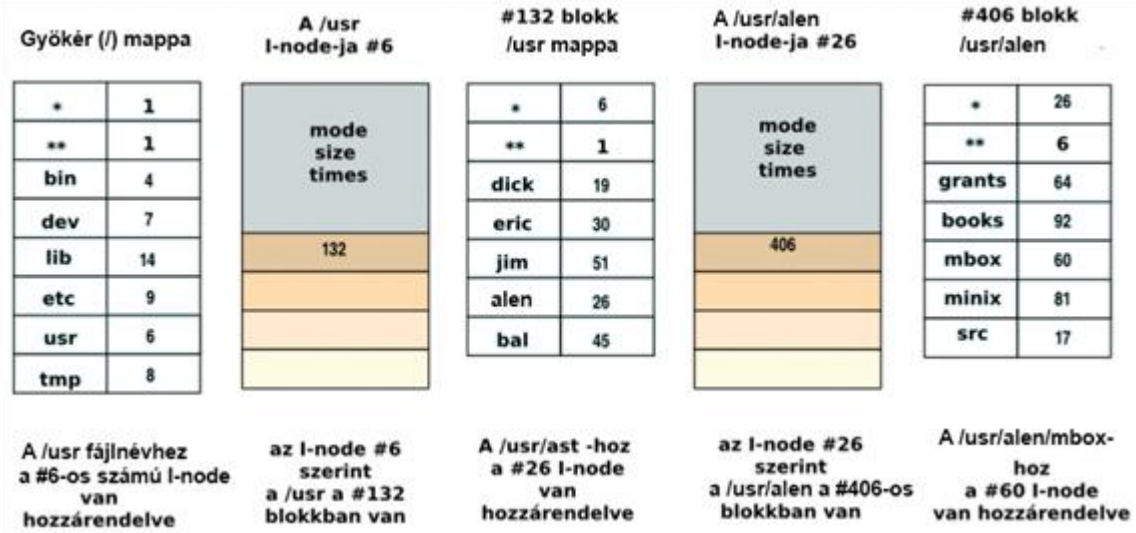
Ebből könnyen következik, hogy a Unix fájlrendszerének miért nem okoznak problémát a következő fejezetben bemutatandó hard link-ek, vagy is azok a fájlok, amelyeknek több nevük van. Egyszerűen az említett listába különböző fájlnevek vannak bejegyezve, melyekhez ugyanazok az i-node számok tartoznak.

Megjegyzés: Két nagyon fontos információ a hard linkek esetén:

- Hard link-et csak egy fájlrendszeren belül lehet használni! Nem lehetséges egy fájlrendszerben egy másik fájlrendszerbe mutató hard link-et létrehozni, mert az i-node számok által hivatkozott blokkok ott más fájlt alkotó adatblokkokhoz vannak hozzárendelve.

- Hard link-ek csak fájlokra mutathatnak : könyvtárra mutató hard link-et azért nem lehet létrehozni, hogy elkerüljük a könyvtárfában a ciklus megjelenését.

A következő ábra azt mutatja be, hogy az előbb elmondottak alapján milyen lépésekkel lehet eljutni a gyökérkönyvtárból a */usr/alen/mbox* file-hoz. * az aktuális könyvtárat, ** az aktuális könyvtár szülőkönyvtárát jelenti.



Az i-node számokat az ls parancs `-i` módosítójával tehetjük láthatóvá, de a stat parancs is láthatóvá teszi számunkra ezt az információt.

```
[adamkoa@kkk run]$ ls -i /etc/passwd
3710987 /etc/passwd
[adamkoa@kkk run]$ stat /etc/passwd
  File: `/etc/passwd'
  Size: 3149            Blocks: 16         IO Block: 4096   regular file
Device: 901h/2305d      Inode: 3710987     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2011-05-07 20:17:50.000000000 +0200
Modify: 2010-12-13 16:58:40.000000000 +0100
Change: 2010-12-13 16:58:40.000000000 +0100
[adamkoa@kkk run]$
```

3.2. Link

Mi is az a link?

A link egy mutató (hivatkozás) valamely adathordozón található adatokhoz (általában fájlokhoz). Két típusa létezik:

A **hard** link:

- Hivatkozás fájlokra. (Könyvtárakra és speciális fájlokra nem! Ezzel zárják ki a ciklikusságot a fából)
- A hard linkek nem hivatkozhatnak az egyik partícióról a másikra, csak egy fájlrendszeren belül. Azaz például a /dev/hdb meghajtón nem hozhatunk létre olyan hard linket amely a /dev/hda meghajtón lévő tartalomra vonatkozik, mert az i-node számok által hivatkozott blokkok máshoz tartoznak.
- A hard linkek egymástól teljesen megkülönböztethetetlenek. Lényegében ugyan arra a tárterületre több néven helyezünk el hivatkozást a fájlrendszerben. Ezt láthatjuk is, ha egy fájl jellemzőiről részletes listát kérünk, itt a linkszámláló (második oszlop) értéke fogja mutatni, hogy a fájlrendszeren belül mennyi a hivatkozások száma.
- A fájl ténylegesen akkor törlődik, ha a linkszámláló értéke a törlést követően 0-ra csökkenne.

A **soft** link:

- Felrúgja a hard link megszorításait.
- A létrejövő mappa bejegyzés teljesen független lesz a hivatkozott fájltól.
- Hivatkozás programokra, fájlokra és könyvtárakra amik a merevlemezen bárhol lehetnek. Windowson nevelkedettek számára picit hasonlíthat a Windowsos parancsikonra, de nem szabad általánosítani, hogy azzal egyenértékű lenne, SŐT egyáltalán nem!
- Ha a soft link célját törlik, átnevezik vagy áthelyezik a lemezen máshova akkor a linket töröttnek (broken link) nevezzük.

Összegezve, a hard link-ekkel szemben a szimbolikus link-ek lényegesen rugalmasabbak, bár hibára is hajlamosabbak. Ebben az esetben egy létező fájlra történő hivatkozásról van szó, ami azt jelenti, hogy a szimbolikus link a valóságban egy fájl, ami csak más egy másik fájl nevét tartalmazza. Annak érdekében, hogy a rendszer ezt a fájlt ne szövegfájlként értelmezze, ezért külön típusként jelöljük, a fájltípus megjelenítésénél egy **l** áll a kötőjel helyett.

Ennek köszönhetően a szimbolikus linkek esetében mindig meg tudjuk különböztetni az eredeti fájlt a link-től a hard link-kel ellentétben, amely esetében nincs lehetőség erre a különbségtételre. A szimbolikus linkek a partíció határán túl mutató linkek létrehozását is lehetővé teszik. Az persze nincs biztosítva, hogy a link olyas valamire is mutat, ami létezik. Ha egy fájlra mutató szimbolikus linket hozunk létre, majd ezt követően töröljük a fájlt, akkor a szimbolikus link egy nemlétező elemre fog mutatni.

A szimbolikus link további előnyének tekinthető, hogy könyvtárakra is mutathatnak. Így az is lehetséges, hogy egy mappát, ami már nem illeszkedik abba a partícióba, amelyikbe várnánk, egy másik partícióban tároljuk és az eredeti partícióban egyszerűen egy szimbolikus linket állítunk be az új pozícióra.

A linkek az ln paranccsal hozhatóak létre. Alapesetben hard linket készít a paraméteréül kapott fájlra.

Szintaktika:

```
ln [opciók] célfájl [linknév]
```

Az opciók közül a leglényegesebb a `-s` kapcsoló mert ekkor, nem hard hanem soft linket hoz létre. (Ezt a régebbi Unix változatok nem támogatják.)

Példa az ln parancs használatára:

```
[adamkoa@kkk linkproba]$ echo "alma dio mogyoro" >fajl1
[adamkoa@kkk linkproba]$ ls -li
total 8
9158657 -rw-rw-r--  1 adamkoa adamkoa 17 May  7 20:45 fajl1
[adamkoa@kkk linkproba]$ ln fajl1 fajl1.hard
[adamkoa@kkk linkproba]$ ls -li
total 16
9158657 -rw-rw-r--  2 adamkoa adamkoa 17 May  7 20:45 fajl1
9158657 -rw-rw-r--  2 adamkoa adamkoa 17 May  7 20:45 fajl1.hard
[adamkoa@kkk linkproba]$ ln -s fajl1 fajl1.soft
[adamkoa@kkk linkproba]$ ls -li
total 20
9158657 -rw-rw-r--  2 adamkoa adamkoa 17 May  7 20:45 fajl1
9158657 -rw-rw-r--  2 adamkoa adamkoa 17 May  7 20:45 fajl1.hard
9158658 lrwxrwxrwx  1 adamkoa adamkoa  5 May  7 20:45 fajl1.soft -> fajl1
[adamkoa@kkk linkproba]$ rm fajl1
[adamkoa@kkk linkproba]$ ls -li
total 12
9158657 -rw-rw-r--  1 adamkoa adamkoa 17 May  7 20:45 fajl1.hard
9158658 lrwxrwxrwx  1 adamkoa adamkoa  5 May  7 20:45 fajl1.soft -> fajl1    # törött
link lesz, kiírásnál pirosan kiemelve mutatja a célfájl hiánya miatt!
[adamkoa@kkk linkproba]$
```

Szimbolikus linkek használatát igen jól tudja demonstrálni, hogy egyes programok esetén különböző nevekkel látják el az eredeti programot és az futása során ellenőrzi, hogy milyen néven lett elindítva és annak megfelelően változtatja a működési módját. Tipikus példa erre a bemeneten érkező szövegben mintaillesztést végző grep parancs viselkedését befolyásoló elnevezések.

```
[adamkoa@kkk linkproba]$ ls -li /bin/*grep
4898821 lrwxrwxrwx  1 root root     4 Feb  8  2010 /bin/egrep -> grep    # kibővített
mintaillesztés
4898852 lrwxrwxrwx  1 root root     4 Feb  8  2010 /bin/fgrep -> grep    # gyors működés
4898833 -rwxr-xr-x  1 root root 89152 Jun  1  2009 /bin/grep             # normál
viselkedés
[adamkoa@kkk linkproba]$
```

3.3. Extended File System

A Linuxhoz készült első fájlrendszer amely alapjaiban a Unix fájlrendszert implementálta és átvéve az FFS szoft link lehetőségét próbálta kiküszöbölni a Minix korlátozásait. 1992-ben jelent meg, de nagyon hamar, már 1993 januárjában megjelent a second extended filesyste (ext2), hogy a kezdeti hiányosságokat és következetlenségeket egy egységes felülettel váltson ki, valamint lehetővé tegye a Linux kernel számára az idegen fájlrendszerek használatát is egy virtuális réteg bevezetésével VFS (Virtual File System) néven.

Sokáig meghatározó volt a legnagyobb Linux disztribúciók között, mindaddig amíg meg nem jelent 2001-ben a third extended filesystem (ext3) többek között naplózási és online dinamikus bővíthetőségi funkcióval.

Összefoglalva az eddigieket egy táblázatba:

3.1. táblázat - Diszk felépítése

Egy diszk felépítése			
boot szektor	boot program	(pl. 8 MB-os) cilinder-csoportok	
Egy cilinder-csoport (egymás alatti gyűrűk a diszkfelületeken) **felépítése**			
szuperblokk	cilinder-csoport blokk	i-node tábla	adatblokkok
A szuperblokk főbb adatai (cilinder-csoportonként ismételve)			
• blokkméret (1/2, 1, 2, 4... KB) • fragment méret (1/2, 1/4... blokk)	fájlrendszer méret (blokk)	• i-node száma • adatblokkok száma	cilinder-csoport felépítését leíró adatok
Egy cilinder-csoport blokk főbb adatai			
• használt i-node száma • használt adatblokkok száma	• szabad blokkok száma és helye(bittérkép) • szabad i-node-ok száma és helye(bittérkép)	szabad fragmentek száma és helye(bittérkép)	
Egy i-node adatai			
A fájl tipusa (reg., dir., dev, sym.link, mount, stb.)	user, group, rwx bitek"link"-ek száma	utolsó módosítás,elérés, i-node mód.(32bit = Y2038!)	• A fájl hossza (byte-okban) • 12+1+1+1 adatblokk-cím
Egy könyvtár egy sorának szerkezete			
Fájlnév (max. 14-255 jel)	A fájl i-node-jának száma		
A 12+1+1+1 blokkcím (A csak bináris nullákat tartalmazó blokk címe is bináris nulla, az adatblokk nem lesz letárolva !)			
12 adatblokk címe	egy indirekt cím	egy duplán indirekt cím	egy triplán indirekt cím

3.4. Virtual File System

A korai UNIX-rendszerek azt a filozófiát követték, hogy egy rendszerben csak egyetlen állományrendszer létezhet és ebben a megközelítésben az állományok leírására tökéletesen elégséges volt az i-node absztrakció. Azonban hamar világossá vált, hogy ez a korlátozás indokolatlan, több állományrendszer (beleértve a több típust is) számos előnnyel járhat. A megvalósításához azonban már nem elegendők az inode-ok alkalmazása a heterogenitás miatt.

Az új leíró adatszerkezet a virtuális csomópont (vnode) és a virtuális állományrendszer (vfs) lett. Az új absztrakció bevezetésének célja, hogy

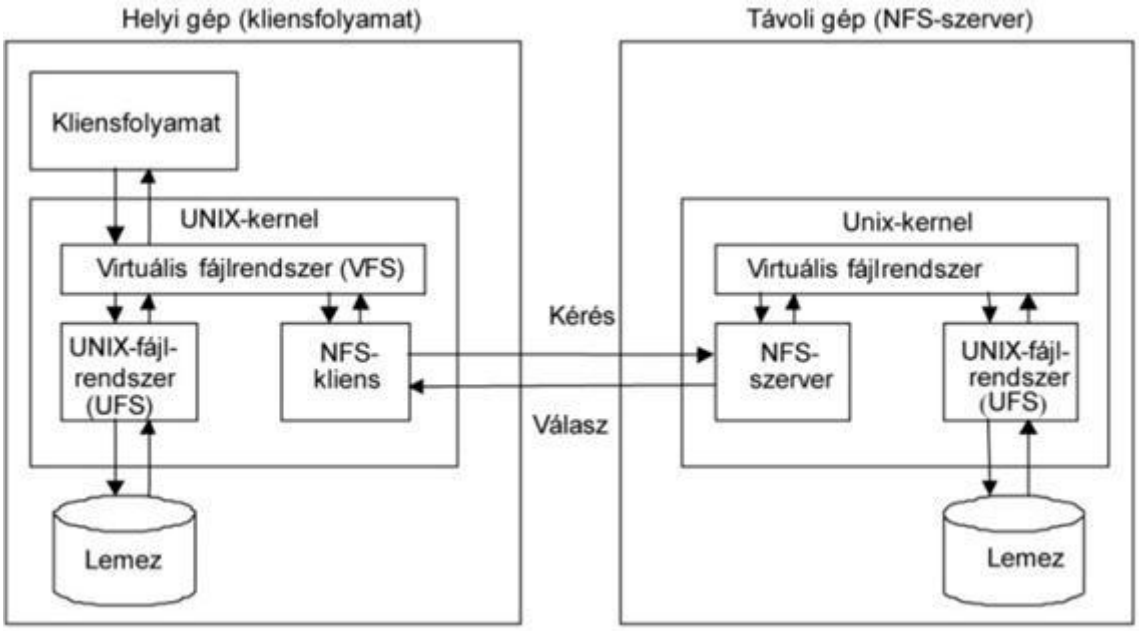
- egyszerre támogasson több állományrendszert (UNIX és nem UNIX alapút),
- különböző diszk partíciók tartalmazhassanak különböző állományrendszert, viszont csatlakoztatás (mount) esetén egységes „képet” mutassanak,
- támogassa a hálózaton történő állományok osztott használatát,
- modulárisan bővíthető legyen.

Az absztrakciót tulajdonképpen két adatszerkezet a vnode és a vfs, illetve azok kezelési módja valósítja meg, amelyek eléréshez egy egységes felületet definiál. Ezáltal könnyen lehet újabb fájlrendszerek támogatását a rendszerhez adni, mert mindösszesen a "szerződésnek" (felületnek - API) megfelelő fájlrendszer illesztő programot kell elkészíteni.

A VFS működésében sokat kölcsönöz az eredeti i-node alapú UNIX fájlrendszertől, mert a vnode valójában az i-node által tartalmazott információkat veszi át és az adott fájlrendszert kezelő program feladata, hogy a szerződésben meghatározott függvényeket megvalósítva az adatmezőket feltöltse az adott fájlrendszerből kinyerhető információkkal. Egy FAT partíció esetén például a jogosultságkezelés értelmét veszti, mert maga a FAT ezt a fogalmat nem ismeri, ezért a függvény a jogok kitöltésénél mindenkinek minden jogot megad, miután más lehetőség itt nincs.

A vfs pedig egy olyan root fájlrendszer, amire a többi állományrendszert csatlakoztatni lehet (fel lehet mount-olni). A rendszer minden egyes állományrendszer típushoz tartalmaz egy *vfs* struktúrát, ami tárolja az állományrendszerhez kapcsolódó, az állományrendszer kezeléséhez elengedhetetlen információkat. Továbbá minden egyes olyan virtuális csomópontban *(vnode)*, amely egyben egy állományrendszer gyökér csomópontja.

A VFS segítségével lehet megvalósítani például távoli gépek fájlrendszereinek elérést olyan módon, mintha azok a helyi gépen lennének -ez az NFS (Network File System). Ezt mutatja az alábbi ábra:



3.5. További fő fájlrendszerek Linux alatt

A 2.4 és újabb kernelek esetén már nem csak az ext2 használható, hanem 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. Emellett még a legkifinomultabb fájlrendszer sem helyettesíthet egy józan mentési stratégiát.

- ReiserFS

  A 2.4-es kernelkiadás egyik fontos eleme. Legfontosabb előnyei:

  - jobb lemezterület-kihasználás

    A ReiserFS fájlrendszerben az összes adat 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 lemez hozzáférési teljesítmény

Kis fájlok esetén az adatok és az i-node információ általában egymás mellett kerül tárolásra. Ez az információ egyetlen lemez I/O-művelettel kiolvasható, azaz csak egy lemezhozzáférés szükséges a kívánt információ lekéréséhez.

- gyors visszaállítás rendszerösszeomlás után

  A legutolsó metaadat-módosításokat nyomkövető napló segítségével a fájlrendszer ellenőrzése nagyon nagy fájlrendszerek esetén is csak néhány másodpercet vesz igénybe.

- megbízhatóság az adatnaplózás használatával

  A ReiserFS az adatnaplózást és a sorbarendezett adatmódokat is támogatja.

- Ext3

  Az ext3 nem vadonatúj tervezési elvekre épül, hanem az ext2-re. A két fájlrendszer szorosan kapcsolódik egymáshoz, így az Ext3 fájlrendszer egyszerűen ráépíthető egy Ext2 fájlrendszerre. A legfontosabb különbség az Ext2 és Ext3 között, hogy az Ext3 támogatja a naplózást. Röviden összefoglalva, az Ext3-nak három nagy előnye van:

  - egyszerű és nagyon megbízható frissítés Ext2-fájlrendszerekről
  - megbízhatóság
  - teljesítmény.

  Számos 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, bár mértéke szabályozható.

- 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.

4. File System Hierarchy Standard (FHS)

A UNIX könyvtárstruktúra esetén e szabvány célja, hogy könnyebb legyen a rendszerek adminisztrálása, illetve a programok írása, ha a megfelelő fájlok egy előre adott helyen találhatóak. Az FHS megpróbálja a UNIX szokásokat és az aktuális tendenciákat követni, ezáltal a Linux rendszer ismerős lesz a más Unix rendszereken gyakorlatot szerzettek számára is.

A teljes könyvtárfa úgy lett megtervezve, hogy kisebb részekre lehessen bontani, melyek akár külön lemezpartíción is elhelyezkedhetnek, hogy a korlátozott lemezmérethez igazodhasson, és könnyebb legyen a biztonsági mentés, valamint egyéb rendszeradminisztrációs feladatok ellátása. A legfontosabb részei a gyökér `/` (root), `/usr`, `/var` és `/home` fájlrendszerek. Mindegyik résznek más a feladata. A könyvtárfát úgy tervezték meg, hogy jól működjön hálózatba kötött linuxos gépek esetében is, amikor egyes részek csak olvasható módon kerülnek megosztásra, például CD-ROM-ról vagy NFS-en keresztül.

Az egyes főbb részek szerepe a következő:

- A `gyökér (/)` fájlrendszernek minden gépnél egyedinek kell lennie (általában egy helyi lemezen található, bár lehetne ramdiszken vagy hálózati meghajtón is, a legfontosabb, hogy létezzen, ahogy ezt a betöltődésnél már olvashattuk), és tartalmaznia kell mindazt, ami a rendszerindításhoz kell addig a pillanatig, amikor már a többi fájlrendszer is felcsatolható (mount). Ezért a gyökér fájlrendszer tartalma csak az egyfelhasználós (single user) üzemmódhoz elegendő. Ez a rendszer tartalmazza továbbá a meghibásodott rendszer javításához és az elveszett fájlok biztonsági mentésből való visszaállításához szükséges eszközöket is.

- Az `/usr` fájlrendszer tartalmazza mindazokat a parancsokat, könyvtárakat, kézikönyv oldalakat és egyéb változatlan fájlokat, amelyek a normális működéshez szükségesek. Az /usr könyvtárban lévő fájlok nem

specifikusak egyetlen gépre vonatkozóan sem, s nem is kell megváltoztatni őket normális működés során. Így elképzelhető a fájlok hálózaton keresztüli megosztására is, ami további előny lehet, hogy a rendszer karbantartása könnyebb lesz, mert csak a központi /usr partíción kell az alkalmazásokat frissíteni, nem minden gépen külön-külön. De még helyi lemez esetén is megtehetjük, hogy az /usr könyvtárat csak olvasható módon csatlakoztatjuk, ami csökkenti a fájlrendszer sérülésének veszélyét.

- A `/var` fájlrendszer változó fájlokat tartalmaz, mint például a spool könyvtárak (a levelezéshez, hírekhez, nyomtatáshoz), naplófájlokat (log file), formázott kézikönyv oldalakat és ideiglenes fájlokat. Valaha minden, ami most a /var könyvtárban van, az /usr könyvtárban volt, de emiatt nem lehetett az /usr könyvtárat csak olvasható módban csatlakoztatni.

- A `/home` fájlrendszer tartalmazza a felhasználók munkakönyvtárait (home), azaz minden igazi adatot a rendszerben. A home könyvtárak külön fájlrendszerbe való elkülönítése megkönnyíti a biztonsági mentéseket; a többi részt általában nem, vagy csak ritkábban kell elmenteni, mivel ritkábban változnak. Egy nagy /home fájlrendszert esetleg érdemes néhány további fájlrendszerre szétszedni, ami egy plusz elnevezési szint bevezetését követeli meg a /home alatt, mint például /home/hallgatok és /home/dolgozok képében.

Noha a különböző részeket fájlrendszereknek neveztük az eddigiekben, nem feltétlenül szükséges, hogy valóban külön fájlrendszereken legyenek. Egy kicsi, egyfelhasználós rendszeren akár egyetlen fájlrendszeren is tárolhatók, ami egyszerűbbé teszi a dolgokat. A könyvtárfa esetleg máshogy is feloszthető fájlrendszerekre, annak függvényében, hogy mekkora lemezeink vannak és hogy hogyan foglaljuk a helyet különféle célokra. A fontos az, hogy minden *szabványos név* működjön, például, ha a /var és az /usr egy partíción van, az /usr/lib/libc.a és /var/log/messages neveknek működniük kell, mondjuk úgy, hogy a /var alatti fájlokat az /usr/var könyvtárba másoljuk és a /var csak egy szimbolikus hivatkozás az /usr/var könyvtárra.

Most pedig lássuk a fontosabb könyvtárak nevét és szerepét:

3.2. táblázat - Az FHS főbb könyvtárai

gyökér könyvtár	Mindennek az alapja, minden más könyvtárnak és fájlnak ő a szülője. A gyökér könyvtárra a fájlrendszerben a / (per) jellel bárhonnan hivatkozhatunk.
/bin	A rendszer használatához nélkülözhetetlen futtatható állományok találhatóak . Több bin könyvtár is található ezen kívül, például a /usr/bin és a /usr/sbin. Bár ez nem törvényszerű, de általában a bin könyvtárakba a minden felhasználó által futtatható állományok kerülnek. A /bin és /sbin az alaprendszerhez szükséges programokat tartalmazza, a felhasználó által telepített programok a /usr/bin, /usr/sbin alá kerülnek.
/boot	Ebben könyvtárban találhatók a bootoláshoz használt fájlok. A rendszermag (kernel), illetve LILO/Grub rendszerbetöltő használata esetén annak konfigurációs állománya is. A gyökérben található még egy vmlinuz fájl is (esetenként bzImage), mely egy hivatkozás a /boot/vmlinuz-ra, azaz a rendszermagra.
/cdrom	Link, általában a /media/cdrom könyvtárra. Ez utóbbi alá csatolódik be a CD/DVD meghajtó egység.
/dev	A rendszerben levő hardver eszközökre hivatkozó fájlokat tartalmazó könyvtár. Ezek az eszközök lehetnek karakteres illetve blokkeszközök. Karakteres eszköz pl. a terminál, blokkeszköz pl. winchester (a primary master winchester a hda1 nevet kapja, ill. ha SATA csatolófelületű sda1-et).
/etc	A rendszer beállításait tartalmazó szöveges fájlokat

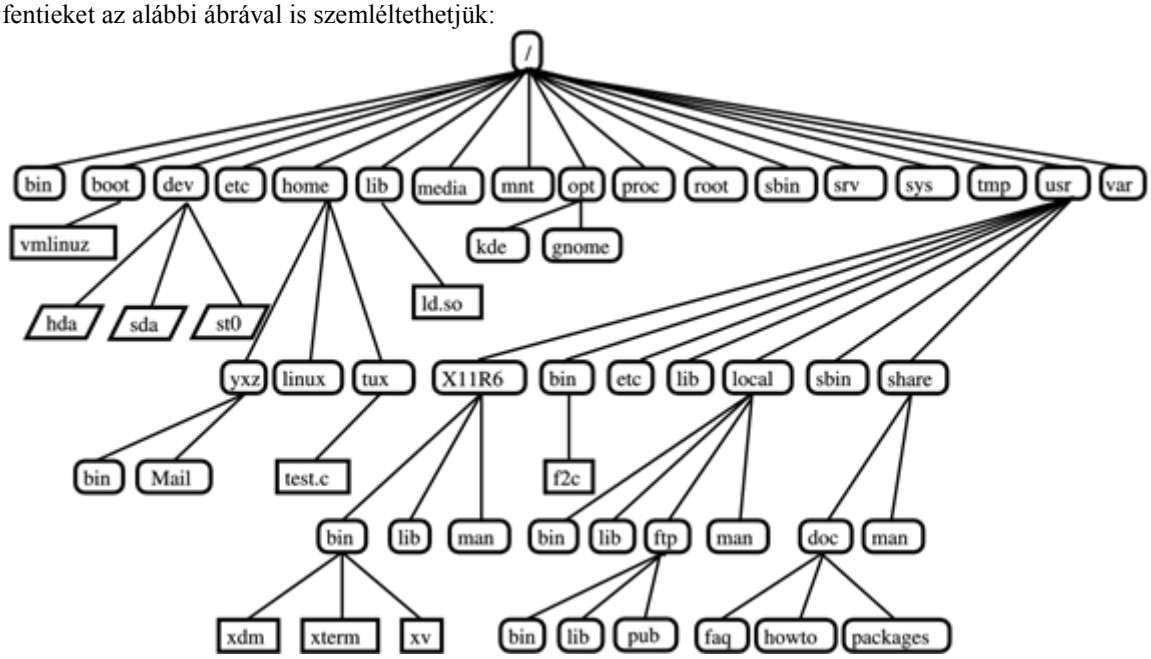
	találjuk itt. A könyvtár neve az angol et cetra (stb.) szó rövidítése. Ellentétben a Windowsos registry megoldással Linux alatt minden konfigurációs állomány egyszerű szövegfájlba van mentve, aminek nagy előnye, hogy az állományok akkor is egyszerűen elérhetők, ha a rendszer egyébként használhatatlan.
	
	Természetesen emellett az egyes programok felhasználó specifikus beállításokkal is rendelkeznek, ezeket a /home könyvtárakban tárolja a rendszer, rejtett mappákban.
	
	Fotnosabb fájlok és könyvtárak:
	
	• /etc/rc vagy /etc/rc.d vagy /etc/rc?.d
	
	Szkriptek, vagy szkripteket tartalmazó könyvtárak, melyeket induláskor vagy futási szint-váltáskor futtat a rendszer.
	
	• /etc/passwd
	
	A felhasználók adatbázisa a következő mezőkkel: felhasználói név (username), Felhasználói azonosító (UID), elsődleges csoportazonosító (GID), valódi név (illetve bármilyen leíró információ - telefonszám, szobaszám, ...), home könyvtár és bejelentkezési shell.
	
	`adamkoa:x:500:500:Adamko Attila:/home/adamkoa:/bin/bash`
	
	• /etc/fstab
	
	Itt találhatók a rendszerindításkor automatikusan (mount -a paranccsal) felcsatolt fájlrendszerek.
	
	• /etc/group
	
	Hasonló az /etc/passwd fájlhoz, de a csoportokat írja le, nem a felhasználókat. Lásd a group kézikönyv oldalát.
	
	`adamkoa:x:500:`
	
	• /etc/inittab
	
	Az init konfigurációs fájlja.
	
	• /etc/issue
	
	A getty üzenete a bejelentkezési prompt előtt. Általában egy rövid rendszerleírást vagy egy üdvözlő üzenetet tartalmaz. A tartalma a rendszeradminisztrátortól függ.
	
	• /etc/magic
	
	A file parancs konfigurációs fájlja. Különböző fájlformátumok leírását tartalmazza, amely alapján a file megpróbálja kitalálni a fájl típusát. Lásd a magic és a file kézikönyv oldalakat.

	• /etc/motd A napi üzenet, mely a sikeres bejelentkezés után automatikusan megjelenik. • /etc/mtab A pillanatnyilag csatlakoztatott fájlrendszerek listája. • /etc/shadow Az árnyékjelszó-fájl (shadow password file) azokon a gépeken, amelyekre árnyékjelszavakat kezelő programokat telepítettek. Az árnyékjelszavak használata esetén a jelszavak az /etc/passwd helyett az /etc/shadow fájlban tárolódnak, amelyet csak a root olvashat. Ez megnehezíti a jelszavak feltörését. • /etc/profile, /etc/csh.login, /etc/csh.cshrc Ezeket a fájlokat a Bourne illetve a C burok hajtja végre bejelentkezéskor vagy induláskor. Ez lehetővé teszi a rendszeradminisztrátornak, hogy minden felhasználónak azonos beállításokat adjon. Lásd a megfelelő shell kézikönyv-oldalát.
/home	A felhasználók "otthona". A felhasználókhoz tartozó állományokat és személyes beállításokat tartalmazó könyvtár. A home könyvtárra, bárhol is vagyunk éppen a fájlrendszerben a ~ (hullámvonal) jellel mindig hivatkozhatunk. Ezalatt a könyvtár alatt a felhasználónak teljes joga van, viszont rajta kívül azonban leginkább csak olvasási joga van alapból.
/lib	Úgynevezett librarykat (függvénykönyvtárakat - a Windows alatt a DLL-hez hasonló állományokat) tartalmazó könyvtár amelyek a programok futtatásához szükségesek. A kernel modulok is itt találhatóak.
/lost+found	Ez egy speciális könyvtár, jelen esetben egy ext típusú fájlrendszerrel szerelt partícióról van szó, ez a könyvtár nem is a Linux, mint inkább a fájlrendszer része Minden egyes ext partíción van egy lost+found könyvtár. Ha hiba miatt leállt a rendszer (pl.: áramszünet) akkor bootoláskor egy fájlrendszer ellenőrzésre kerül sor (nagyjából mintha a Windows alatt a Scan Disket futtatnák). A sérülten talált fájlok kerülnek ide.
/media	Beállítástól függően általában e könyvtár alá kerülnek befűzésre a cserélhető médiák (CD/DVD-Drive, USB PenDrive, floppy, stb.)
/mnt	A rendszerbe csak az aktuális munkamenetre, ideiglenesen felcsatolt fájlrendszereket tartalmazó mappa. Ilyen ideiglenesen felcsatolt fájlrendszer lehet például egy CD lemez tartalma. Hagyományosan a winchesterek felcsatolási pontja, de a mai rendszerek jelentős része erre a célra automatikusan a /media mappát jelöli ki. Mivel ebben a könyvtárstruktúrában nincs kiemelt "volume" egy-egy meghajtónak, mint

	Windows alatt a C:, D:, stb., így egy-egy eszközt tetszőleges helyre befűzhetünk a fájlrendszerbe.
/opt	Ez egy kicsit homályos pont. A hivatalos leírás szerint külsős programok települnek ebbe a könyvtárba, de a rendszerek nagy részén üresen áll...
/proc	Itt találhatóak az éppen futó folyamatok fájlként leképezve, sorszámozva, illetve információk a rendszerről: processzorról, memóriáról, stb. Ez egy virtuális fájlrendszer.
/root	A rendszergazda saját, külön bejáratú használatú könyvtára. (A rendszergazda "home" ja).
/sbin	A rendszer működéséhez nélkülözhetetlen állományok kerültek ide. Pl.: init, route, ifup, stb. Általában ebbe a könyvtárakba kerülnek azok a rendszereszközök, melyeket csak rendszergazdák használnak vagy a használatukhoz rendszergazdai jog szükséges.
/sys	A másik számunkra homályos kérdés. 2.6-os kernellel együtt jelent meg ez az újfajta eszközkezelési metódus, ebben a könyvtárban található meg a sysfs számára egy komplett fa. Szintén egy kincsesbánya, de egy átlag felhasználó ritkán téved erre.
/tmp	Az átmeneti (temporary) állományokat tartalmazó könyvtár. A rendszer a tartalmát a boot során törli, így fontos információt itt ne tároljunk. Ez a másik olyan könyvtár (home-on kívül), amely alapértelmezettben írható minden felhasználó számára.
/usr	Másodlagos hierarchia belépési pontja. A felhasználókra vonatkozó adatokat tárolja. Általában a többfelhasználós alkalmazások nagy része itt található. A /usr könyvtár alatt található a telepített programok nagy része és hagyományból ide kerülnek a forrásokat (/usr/src). Ugyanitt találhatók a dokumentációk is. Struktúráját tekintve pedig a gyökér mappa szerkezetét ismétli nagyvonalakban, pl bin,sbin,lib mappanevek használatával. • /usr/bin • /usr/include/ - standard include fájlok helye • /usr/lib - függvénykönyvtárak a binárisokhoz • /usr/sbin • /usr/src • /usr/local - Harmadlagos hierarchia belépési pontja, specifikus dolgok az adott gépre.
/var	Változó állományokat tartalmazó könyvtár. (logok, nyomtatási sorok, beérkező e-mailek). Számos szolgáltatás gyűjtőkönyvtára. Itt találhatók a naplófájlok, egyes programok hosszabb ideig tárolt, mégis átmeneti fájljai, alapértelmezettben a felhasználói levelezőfiókok, stb. • /var/log

	• /var/mail • /var/run • /var/spool • /var/tmp
/vmlinuz	Hivatkozás a /boot/vmlinuz-ra, azaz a rendszermagra. Esetenként neve bzImage.

A fentieket az alábbi ábrával is szemléltethetjük:



4. fejezet - Állománykezelés

A következőekben tekintsük át az egyes platformokon használható parancssori eszközöket az állományok kezelésére:

1. DOS-os parancsok

Két nagy csoportra lehet osztani a DOS utasításait. Vannak belső utasítások *(a parancsértelmező tudja értelmezni)* és külső utasítások *(külső programot futtat)*. A legfontosabb DOS utasítások, amelyek az állománykezelést végzik azok a belső parancsok.

Fontos megjegyezni, hogy minden olyan esetben, ahol több fájlon szeretnénk a tevékenységet végre hajtatni, lehetőségünk van metakarakterek használatára, hogy egy mintával válasszuk ki a feldolgozandó fájlokat. A két **metakarakter**, amit használhatunk, az a `*` és a `?` . A ? jelentését tekintve egy tetszőleges karakterre tud illeszkedni, míg a * tetszőlegesen sok karaktert vehet fel értékül. Azaz a könyvtár tartalmának kilistázásához használt DIR parancs önmagában az összes fájlt megjeleníti, de a `DIR *.txt` már csak a a txt kiterjesztéssel megadottakat, míg a `DIR a?.txt` az betűvel kezdődő és két hosszúságú txt fájlokat listázza ki.

4.1. táblázat - Belső parancsok

Belső parancsok			
`DATE`	A dátum beállítása. (a gép által kiírt minta szerint)	`TIME`	A rendszeridő beállítása. (a gép által kiírt minta szerint)
`VOL`	A lemez címkéjét és sorszámát jeleníti meg.	`VER`	Képernyőre írja a rendszer verziószámát.
`SET`	A környezeti változók megjelenítését, változtatását teszi lehetővé.	`PATH`	Keresési út beállítása futtatható állományokhoz
`REN`	Átnevezi a megadott könyvtárat, vagy fájlt.	`PROMPT`	A parancssor készenléti jelét (C:\>) változtathatjuk meg.
`TYPE`	Szöveges fájl tartalmát jeleníti meg a képernyőn.	`CLS`	Képernyő törlése. (a parancsfájlokban is)
`DEL`	Törli a fájlokat. (A parancsfájlokban is használható.)	`COPY`	Állományok másolása.
`RD`	Törli az üres könyvtárat.	`CD`	Az aktuális könyvtár megváltoztatása.
`MD`	Könyvtár létrehozása parancssorból és parancsfájlokban is.	`DIR`	Képernyőre listázza a könyvtárakat és fájlokat.
`ATTRIB`	Fájl attribútumok megjelenítése vagy megváltoztatása.	`LABEL`	Lemez kötetcímkéjének létrehozása, módosítása vagy törlése

4.2. táblázat - Külső parancsok

Külső parancsok			
`SYS`	Az MS-DOS rendszerfájloknak és a parancsértelmezőnek egy megadott lemezre	`FORMAT`	Lemezek formatálását teszi lehetővé.

Külső parancsok			
	másolása		
HELP	Súgó indítása.	UNDELETE	A DEL paranccsal töröltek visszaállítására.
EDIT	Szövegszerkesztő MS-DOS módban ASCII fájl (parancsfájlok, levelek, e-mailek) készítésére.	DISKCOPY	Egy floppy tartalmát másolja floppyra.
FDISK	A merevlemez konfigurálása (kialakítása) a felhasználás előtt.	PRINT	Háttér nyomtatást végez (a nyomtatás alatt a gépen dolgozhatunk).
SCANDISK	Lemezhibák feltárása, javítása.	MEM	A memória felhasználás kiíratása a képernyőre.

1.1. Könyvtárak kezelése

- **DIR:** a DIR parancs az adott könyvtár tartalmának kiírására való. Használata:

```
DIR [meghajtó:][elérési út][fájlnév] [/P] [/W] [/A[[:]attribútumok]][/O[[:]rendezés]]
[/S] [/B] [/L] [/V] [/4]
```

A szögletes zárójelben lévő elemeket nem kötelező használni. Jelentésük:

[meghajtó:][útvonal][fájlnév]Megadja a listázandó meghajtót, könyvtárat, illetve fájlokat. (Lehet bővített fájlmegadás vagy több fájl megadása is.)

- **/P** várakozás minden képernyő után
- **/W** több oszlopos listázási formátum használata
- **/A** A megadott attribútumú fájl megjelenítése, attribútumok:
  - **A** Archiválandó fájlok
  - **D** Könyvtárak
  - **H** Rejtett fájlok
  - **R** Írásvédett fájlok
  - **S** Rendszerfájlok
  - - logikai nem előtag (tagadás)
- **/O** A fájlok rendezett sorrendű listázása rendezés:
  - **N** Név szerint (ábécérendben)
  - **S** Méret szerint (előbb a kisebbek)
  - **E** Kiterjesztés szerint (ábécérendben)
  - **D** Dátum és idő szerint (előbb a korábbiak)
  - **G** Előbb a csoportkönyvtárak
  - **A** A legutóbbi hozzáférés szerint (előbb a korábbiak)
  - - Fordított sorrend (előtag)
- **/S** Adott könyvtárban és alkönyvtáraiban lévő fájlok megjelenítése

- **/B** Egyszerű formátum (fejléc és összegzés nélkül
- **/L** Kisbetűk használata
- **/V** Részletes mód
- **/4** Négyjegyű évszámok (/V -vel együtt hatástalan).

- **TREE:**a TREE parancs a fastruktúra lekérdezésére szolgál; szemléletesen kirajzolja a könyvtárrendszer szerkezetét. Használata:

```
TREE [meghajtó:] [/F]
```

A[/F]kapcsoló hatására a fájlok neveit is kilistázza.

- **MD vagy mkdir:** az MD parancs a könyvtárak létrehozására szolgál. Használata:

```
MD [meghajtó:] [ÚTVONAL]
```

ahol a [meghajtó:] jelöli azt a meghajtót, amelyen a könyvtárat létre szeretnénk hozni, az [ÚTVONAL] a fa struktúrán belül egy út, melynek a végén a létrehozandó alkönyvtár neve szerepel. Az útvonalban a könyvtárneveket *backslash* (\ ) karakter választja el.

- **RD vagy rmdir:** az RD parancs a könyvtárak törlésére szolgál. Használata:

```
RD [meghajtó:] [ÚTVONAL]
```

A törlés feltétele, hogy a könyvtár üres legyen.

- **CD vagy chdir:** Az aktuális könyvtár megváltoztatása. Használata:

```
CD könyvtár
```

Belépés a megadott könyvtárba

CD .. Visszalépés a szülőkönyvtárba

CD\ Visszalépés a gyökérkönyvtárba

- **ATTRIB**: Fájlattribútumok megjelenítése vagy módosítása.

```
ATTRIB [+R | -R] [+A | -A ] [+S | -S] [+H | -H] [+I | -I]
[meghajtó:][elérési_út][fájlnév] [/S [/D] [/L]]
```

- + Attribútum beállítása.
- - Attribútum törlése
- **R** Írásvédett fájlattribútum.
- **A** Archiválandó fájlattribútum.
- **S** Rendszer fájlattribútum.
- **H** Rejtett fájlattribútum.
- [meghajtó:][elérési_út][fájlnév] A feldolgozandó fájlok.
- /S Az egyező fájlok feldolgozása az aktuális könyvtárban és az összes alkönyvtárban.
- /D Könyvtárak feldolgozása is.

- **PATH:** A keresési útvonal beállítása, a futtatható fájlokhoz. Futtatható fájlok keresési útjának megjelenítése vagy beállítása. Azaz, ha nem akarunk teljes elérési utat használni az egyes programok indításához, a programot tartalmazó mappát erre fel kell venni.

```
PATH [[meghajtó:]útvonal[;...]]
```

Az aktuális keresési út megjelenítéséhez paraméterek nélkül írd be a PATH parancsot.

1.2. Fájlkezelés

- **COPY:** A parancs - nevéből következően - fájl(ok) másolására szolgál egyik könyvtárból a másikra, beleértve ebbe az egyik meghajtóról a másikra való másolást is. Maga a parancs igen összetett, sok lehetőséget biztosít. Ebben a fejezetben a parancsnak csak az alapjait ismertetjük. Használata:

```
COPY [meghajtó:][ÚTVONAL]FÁJLNÉV[.KIT] [meghajtó:][ÚTVONAL][FÁJLNÉV][.KIT]
```

A parancs leírását egyszerűbb úgy megjegyezni, hogy a parancsszó után le kell írni, *(honnan) mit, hová* akarunk másolni.

Érdekes lehetőség a fájlok összefűzése, ami történhet - `/a` -karakteresen (alapértelmezett) és binárisan - `/b` - is, azaz EOF jelig, vagy tartalomtól függetlenül.

```
copy /a alpha.txt + beta.txt gamma.txt
copy /b alpha.mpg + beta.mpg gamma.mpg
```

- **MOVE:** Állományok mozgatása, könyvtárak átnevezése.
- **DEL vagy ERASE:** Törli a fájlokat. (A parancsfájlokban is használhatod.) Használata:

```
DEL  [/P] [/F] [/S] [/Q] [/A[[:]attribútumok]] [meghajtó:][ÚTVONAL][FÁJLNÉV][.KIT]
```

  - /P Rákérdez minden egyes fájl törlése előtt.
  - /F Az írásvédett fájlok kényszerű törlése.
  - /S Megadott fájl törlése az összes alkönyvtárból.
  - /Q Csendes mód, ne kérdezzen rá a törlésre globális helyettesítő karakter esetén.
  - /A Fájlok kiválasztása attribútumaik alapján. attribútumok R (Írásvédett fájlok), S (Rendszerfájlok), H (Rejtett fájlok), A Archiválandó fájlok), - Negálást jelentő előtag

  A metakarakterek itt is alkalmazhatóak:

```
*.*  aktuális mappa összes fájlja
*.* /s az aktuális mappa és összes almappájának összes fájlja törlődik
```

- **REN:** Átnevezi a megadott könyvtárat, vagy fájlt. Használata:

```
REN [meghajtó:][ÚTVONAL]RÉGI_FÁJLNÉV[.KIT] ÚJFÁJLNÉV[.KIT]
```

Használata pici figyelmet igényel, mert egyben tud mozgatni és átnevezni is:

```
ren c:\windows\filex.txt \temp\filey.txt
```

Hosszú fájlnevek esetén külön figyelmet kell szentelni a szóközök kezelésére, mert a szóköz általános elhatároló karakternek tekintendő és ezért ha a név tartalmazza, akkor idézőjelek segítségével kell végeni:

```
ren c:\"Documents and Settings"\"All Users"\Desktop\filex.txt filey.txt
ren "c:\Documents and Settings\All Users\Desktop\filex.txt" filey.txt
```

Metakarakterek használata esetén is figyelnünk kell, mert lehetőség van csoportos átnevezésre is, pl minden htm kiterjesztésű fájl átnevezése .html-re:

```
ren *.htm *.html
```

- **FC vagy COMP:** Két kijelölt fájl összehasonlítása. Használata:

```
COMP [meghajtó:][ÚTVONAL][FÁJLNÉV][.KIT] [meghajtó:][ÚTVONAL][FÁJLNÉV][.KIT]
```

A parancs először a két fájl hosszát ellenőrzi, s ha már az sem egyezik, leáll. Ha a hossz azonos, akkor továbbmegy, a "belső" eltéréseket kijelzi, de a tizedik eltérés után leáll. Hibát jelez akkor is, ha a fájl végén nem talál `^Z` jelet, noha a fájl ettől még lehet jó, ha nem ASCII fájl. A nevekben használhatók a helyettesítő karakterek is. Ez teremti meg a lehetőségét annak, hogy egy menetben több fájlt is összehasonlítsunk.

```
C:\ ... \Book>fc book.xml book.xml.bak
Comparing files book.xml and BOOK.XML.BAK
***** book.xml
                        <listitem>
                            <para><emphasis role="bold">DEL vagy ERASE:</emphasis>
Törli a fájlokat.
                                (A parancsfájlokban is használhatod.)
Használata:</para>
***** BOOK.XML.BAK
                        <listitem>
                            <para><emphasis role="bold">DEL vagy erase:</emphasis>
Törli a fájlokat.
                                (A parancsfájlokban is használhatod.)
Használata:</para>
*****
```

- **TYPE:** Szöveges fájl tartalmát jeleníti meg a képernyőn. Használata:

```
TYPE [meghajtó:][ÚTVONAL]fájlnév[.KIT]
```

Ebben a parancsban a helyettesítő karakterek nem használhatók. Ha az állománynak van kiterjesztése, azt is ki kell írni!

2. Linux parancsok

2.1. Általános fájl kezelő parancsok

2.1.1. touch

Létrehoz egy üres fájlt, vagy ha a fájl már létezik akkor pedig módosítja az utolsó hozzáférés és módosítás idejét egyidejűleg az aktuálisra.

Szintaktika:

`touch` [-a | -m] [állománynév(ek)]

- -a: csak a hozzáférési idő módosítása
- -m: csak a módosítási idő módosítása

2.1.2. cp

Fájl másolása egy helyről egy másik helyre.

Szintaktika:

```
cp [forrásfájl] [célfájl]
```

A fájl típusától függ, hogy mappán belül, egy másik mappába másol egy fájlt vagy egy mappát. Mappa másolás előtt ne felejtsük el, hogy külön kell beállítani, hogy rekurzívan működjön.

```
cp -r [i ] forrásmappa(k) célmappa(k)
```

Ha a felülírás lehetőségét szeretnénk elkerülni, akkor használjuk az -i módosítót is, hogy interaktívan visszakérdezzen ilyen esetekben.

2.1.3. mv

Egy fájlt mozgat a könyvtárrendszerben egy helyről egy másikra vagy egy fájlt átnevez.

Szintaktika:

```
mv [-i] [forrásfájl] [célfájl]
```

Amennyiben a második argumentum fájlnév, akkor átnevezés történik, ha mappanév, akkor pedig átmozgatás. A forrásfájl lehet fájl is mappa is, a hatása ugyan az.

2.1.4. rm

Töröl egy vagy több fájlt. Vigyázat, mindent töröl, nem kérdez rá, ezért minden esetben toldjuk meg egy -i kapcsolóval, így rákérdez minden egyes elemre törlés előtt.

Szintaktika:

```
rm [kapcsoló(k)] fájlnév...
```

Kapcsolók:

- -f: kényszerített, hibajelzés elmaszkolása
- -i: interaktív
- -r|-R : rekurzív törlés

2.1.5. find

Megadott feltételeknek eleget tevő állományokat keres. A keresés nagyon erőforrás igényes és jelentősen leterheli a rendszert így mindig próbáljuk meg a keresési feltételeket leszűkíteni.

Szintaktika:

```
find elérési_útvonal kifejezés [tevékenység]
```

Kifejezések a keresendő fájlok megadásához:

Rengeteg kapcsolót tartalmaz (bővebben a manuálban találsz leírást)

- `-name név`: adott nevű fájlok keresése
- `-type fájltípus` :Adott fájltípusú fájlokat keres (pl: d - mappa)
- `-mtime [+/-]szam` : a legutolsó módosítás ideje napokban
- `-atime [+/-]szam` : a legutolsó hozzáférés ideje szintén napokban
- `-user userid` : melyik felhasználó tulajdonában van a fájl
- `-group csoportid` : melyik csoporté a fájl
- `-perm jogosultság` : hozzáférési jogosultság (3db oktális számjegy)
- `-size [+/-]szam[c]` : a megadott méretnél nagyobb vagy kisebb fájlok keresése (a méret blokkokban értendő, a c módosító esetén viszont bájtokban)
- `-a` : és kapcsolat a keresési feltételek között
- `-o` : vagy kapcsolat a keresési feltételek között

Tevékenység:

Találat esetén az adott fájlra végrehajtja a parancsot

- `-exec parancs { } \;` : ha találat van lefut a parancs

- `-ok parancs { } \;` : ha találat van lefut egy olyan parancs amely felhasználói inputot fog kérni
- `-ls` : listázza a talált fájlokat

Példa a find parancs használatára:

```
adamkoa@it:~$ find /hol/keresek -name valami* -a -size +256c -exec rm{ } \
```

Megkeres minden "valami"-vel kezdődő és 256 bájtnál nagyobb állományt, majd törli azt.

Ahova nincs jogosultságunk belépni ott hibaüzeneted ad, ennek kiszűrésére toldjuk meg a 2>/dev/null kapcsolóval, mely hatására a hibaüzenet nem jelenik meg a képernyőn (átirányítás a semmibe).

```
adamkoa@it:~$ find /hol/keresek -name valami* -ls 2>/dev/null
```

További példák:

```
find . -type d    # könyvtárak keresése az aktuális mappában
find . -mtime +90 # amelyek nem lettek módosítva az elmúlt 90 napban
find ~ -perm 777 -a -size 400 # a home mappán belüli 400 blokknál nagyobb és mindenki
által módosítható fájlok keresése
```

2.2. Könyvtárkezelő parancsok

2.2.1. pwd

Kiírja az aktuális könyvtár abszolút elérési útját.

2.2.2. cd

Könyvtárváltás a cd után megadott könyvtárba. Felfele a `cd ..` paranccsal léphetünk (lehetséges többet is, pl. `cd ../../..`). Önmagában kiadva a cd parancs a saját felhasználói home könyvtárunkba ugrik.

Példa a `cd` és a `pwd` parancs használatára:

```
adamkoa@it:~$ cd /
adamkoa@it:/$ cd bin
adamkoa@it:/bin$ cd ~
adamkoa@it:~$ pwd
/home/adamkoa
adamkoa@it:~$ cd ..
adamkoa@it:/home$ cd /var
adamkoa@it:/var$ cd ..
adamkoa@it:/$ cd /var/spool
adamkoa@it:/var/spool$
```

2.2.3. ls

Kiírja a megadott (alapesetben aktuális) könyvtár által tartalmazott fájlokat.

Szintaktika:

```
ls [kapcsolók] [fájlnév]
```

Használható kapcsolók (a kapcsolók kombinálhatóak is)

- -l : minden információ megjelenítése
- -a : megjeleníti azon állományokat is melyek neve ponttal kezdődik (rejtett fájlok kiírása)
- -i : ekkor a fájl neve mellé kiírja az iNode számát is
- -d: az adott mappa információnak megjelenítése a tartalma helyett (csak mappán van értelme)
- -R: rekurzív listázás az adott mappától kezdve

Példa az ls parancs használatára:

```
[adamkoa@kkk proba]$ ls
link2.txt  linkproba  link.txt  new_file2.txt  new_file.txt  p2.txt  sed.txt
szimbolikus.txt  test  tmp
[adamkoa@kkk proba]$ ls -l
total 64
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 May  8 13:58 linkproba
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
-r--rw-rw-  1 adamkoa fuse      25 Mar  9  2010 p2.txt
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
lrwxrwxrwx  1 adamkoa adamkoa    8 Mar 23  2010 szimbolikus.txt -> link.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
lrwxrwxrwx  1 adamkoa adamkoa    5 Mar 23  2010 tmp -> /tmp/
[adamkoa@kkk proba]$ ls -la
total 88
drwxrwxr-x    4 adamkoa adamkoa  4096 May  7 20:34 .
drwx--x--x  116 adamkoa adamkoa 12288 May 11 20:07 ..
-rw-rw-r--    1 adamkoa adamkoa     0 Mar  9  2010 link2.txt
drwxrwxr-x    2 adamkoa adamkoa  4096 May  8 13:58 linkproba
-rw-r-----    1 adamkoa adamkoa     0 Mar 23  2010 link.txt
-rw-rw-r--    1 adamkoa adamkoa    30 Apr 14  2010 new_file2.txt
-rw-rw-r--    1 adamkoa adamkoa    30 Apr 14  2010 new file.txt
-r--rw-rw-    1 adamkoa fuse       25 Mar  9  2010 p2.txt
-rw-rw-r--    1 adamkoa adamkoa    26 Apr 20  2010 sed.txt
lrwxrwxrwx    1 adamkoa adamkoa     8 Mar 23  2010 szimbolikus.txt -> link.txt
drwxrwxr-x    2 adamkoa adamkoa  4096 Apr 27  2010 test
lrwxrwxrwx    1 adamkoa adamkoa     5 Mar 23  2010 tmp -> /tmp/
[adamkoa@kkk proba]$ ls -l linkproba/
total 24
-rw-rw-r--  1 adamkoa adamkoa 17 May  7 20:45 fajl1.hard
lrwxrwxrwx  1 adamkoa adamkoa  5 May  7 20:45 fajl1.soft -> fajl1
prw-rw-r--  1 adamkoa adamkoa  0 May  8 13:58 my_pipe
-rw-rw-r--  1 adamkoa adamkoa 40 May  8 13:58 out.gz
[adamkoa@kkk proba]$ ls -ld linkproba/
drwxrwxr-x  2 adamkoa adamkoa 4096 May  8 13:58 linkproba/
[adamkoa@kkk proba]$ ls -lR linkproba/
linkproba/:
total 24
-rw-rw-r--  1 adamkoa adamkoa 17 May  7 20:45 fajl1.hard
lrwxrwxrwx  1 adamkoa adamkoa  5 May  7 20:45 fajl1.soft -> fajl1
prw-rw-r--  1 adamkoa adamkoa  0 May  8 13:58 my_pipe
-rw-rw-r--  1 adamkoa adamkoa 40 May  8 13:58 out.gz
[adamkoa@kkk proba]$
```

2.2.4. mkdir

Létrehoz egy alkönyvtárat, az aktuális könyvtárban.

Szintaktika:

```
mkdir [könyvtárnév]
```

2.2.5. rmdir

Törli a paraméteréül kapott könyvtárat ha az létezik és üres.

Szintaktika:

```
rmdir [könyvtárnév]
```

Példa a `mkdir` és a `rmdir` parancs használatára:

```
adamkoa@it:~$ ls -l
összesen 36
-rw------- 1 adamkoa prog1 48 2007-04-16 11:23 nevek.txt.
-rwx------ 1 adamkoa prog1 16589 2007-02-12 18:26 xy
```

```
-rw-r--r-- 1 adamkoa prog1 61 2007-02-12 18:22 xy.c
-rw------- 1 adamkoa prog1 196 2007-02-12 18:26 xy.log
-rw------- 1 adamkoa prog1 6 2007-02-12 18:26 xy.out
drwx------ 2 adamkoa prog1 144 2007-04-12 15:10 zh2
adamkoa@it:~$ mkdir newdir

adamkoa@it:~$ ls -l
összesen 36
-rw------- 1 adamkoa prog1 48 2007-04-16 11:23 nevek.txt.
drwx------ 2 adamkoa prog1 48 2007-04-25 22:01 newdir
-rwx------ 1 adamkoa prog1 16589 2007-02-12 18:26 xy
-rw-r--r-- 1 adamkoa prog1 61 2007-02-12 18:22 xy.c
-rw------- 1 adamkoa prog1 196 2007-02-12 18:26 xy.log
-rw------- 1 adamkoa prog1 6 2007-02-12 18:26 xy.out
drwx------ 2 adamkoa prog1 144 2007-04-12 15:10 zh2

adamkoa@it:~$ rmdir newdir
adamkoa@it:~$ ls -l
összesen 36
-rw------- 1 adamkoa prog1 48 2007-04-16 11:23 nevek.txt.
-rwx------ 1 adamkoa prog1 16589 2007-02-12 18:26 xy
-rw-r--r-- 1 adamkoa prog1 61 2007-02-12 18:22 xy.c
-rw------- 1 adamkoa prog1 196 2007-02-12 18:26 xy.log
-rw------- 1 adamkoa prog1 6 2007-02-12 18:26 xy.out
drwx------ 2 adamkoa prog1 144 2007-04-12 15:10 zh2
adamkoa@it:~$
```

5. fejezet - Szűrők

A szűrők olyan programok, amelyek egy adatfolyam feldolgozására hivatottak. Alapvető működésű elvük, hogy a standard bemeneten fogadják az adatokat, majd az eredményt a standard kimenetre irányítják. Legalapvetőbb használatuk épp emiatt a csővezetékekben történik. Természetesen ettől el lehet térni - a már korábban említett - átirányítási lehetőségek segítségével felül definiálhatjuk a bemenetet és a kimenetet egy eszközzel vagy fájllal.

1. DOS

A DOS esetén nem igazán vannak jelen bonyolultabb szűrő programok, az elérhető három az alábbi:

- **MORE:** Képernyőoldalakra tördelve jelenít állományokat. Használata:

```
MORE [meghajtó:][ÚTVONAL]FÁJLNÉV[.KIT]

parancs | more
vagy
more < inputfilename
```

- **FIND**: Karakterlánc keresése fájlban vagy fájlokban.

```
FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] "karakterlánc" [[meghajtó:][elérési út]fájlnév[
...]]
```

  - /V A karakterláncot NEM tartalmazó sorok megjelenítése.
  - /C A tartalmazó soroknak csak a számát jeleníti meg.
  - /N A megjelenített sorokat sorszámmal együtt jeleníti meg.
  - /I Kereséskor nem különbözteti meg a kis- és a nagybetűket.
  - /OFF[LINE] Ne hagyja ki az offline állapotú fájlokat.
  - "karakterlánc" A keresendő szöveg megadása.
  - [meghajtó:][elérési út]fájlnév A keresendő fájl vagy fájlok.

  Ha nincs megadva elérési út, a FIND a parancssorba beírt szöveget vagy másik parancstól átirányított szöveget keres.

- **SORT**: a bemenet sorainak rendezése.

```
SORT [/R] [/+n] [/M kilobájtok] [/L területi_beállítás] [/REC rekordok_bájtjai]
[[meghajtó1:][elérési_út1]fájlnév1]
     [/T [meghajtó2:][elérési_út2]] [/O [meghajtó3:][elérési_út3]fájlnév3]

sort < inputfilename > outputfilename
```

2. Linux

2.1. cat

Minden argumentumként megadott fájl(oka)t összefűz, és a standard kimenetre írja a végeredményt. Amennyiben nincs fájlnév megadva, úgy a standard bemenetről olvas. Példa a cat parancs használatára:

```
adamkoa@it:~$ cat > x.txt
átirányítja a standard
outputot az x.txt
fájlba                          # bevitel befejezése CTRL+D billentyűkombinációval
adamkoa@it:~$ cat
Vajon miért ír ki mindent 2x? :-)
Vajon miért ír ki mindent 2x? :-)
adamkoa@it:~$
```

A UNIX világában számos apró kis névjáték van, így alakult ki a `tac` programocska is, ami a cat szó fordítottja - és a működése is a fordítottja, azaz a sorokat az utolsótól az elsőig írja ki.

2.2. colrm

A standard inputon érkező szöveg megadott oszlopait törli, majd az így kapott szöveget a standard outputra írja.

Szintaktika:

```
colrm [kezdőindex] [végindex]
```

Példa a colrm parancs használatára:

```
adamkoa@it:~$ cat nevek.txt                 # kiíratás
Áron 5 34
Attila 3 20
Barna 5 31
Betti 2 8
adamkoa@it:~$ cat nevek.txt | colrm 2 4     # törlünk a 2-tól az 5. oszlopig
Á 5 34
Ala 3 20
Ba 5 31
Bi 2 8
adamkoa@it:~$
```

2.3. cut

A standard inputon (vagy a megadott fájlból) érkező szöveg soraiból törli a megadott sorszámú karaktereket, majd az eredményt a standard outputra írja.

Szintaktika:

```
cut [kapcsoló] [indexek] [fájlnév]
```

Kapcsoló: -c :Az eredeti működés ellentéte. Csak a megadott sorszámú karaktereket írja ki.

Példa a cut parancs használatára:

```
adamkoa@it:~$ cut 3           # kiírja a 3. karaktert törli
abcdefgh
abdefgh
adamkoa@it:~$ cut -c 3        # kiírja a 3. karaktert az adatfolyamban
abcdefgh
c
adamkoa@it:~$ cut -c 1,3-8    # kiírja az 1 helyen és 3-tól 8-ig található karaktereket
abcdefghijk
acdefgh
adamkoa@it:~$
```

Kapcsoló: -d : A cut alapesetben a TAB karaktert használja a bemeneten érkező karakterfolyam mezőkre bontására, amennyiben ettől eltérőt szeretnénk használni, akkor ezt a kapcsolót használva megadhatjuk a mezőket egymástól elválasztó karaktert.

Kapcsoló: -f : amennyiben a bemenet tagolható, akkor az egyes részekre, amelyet tovább szeretnénk engedni a kimenetre, azokat itt kell felsorolni.

```
[adamkoa@kkk ~]$ cat /etc/passwd | tail
n4sadm:x:512:515:SAP System Administrator:/home/n4sadm:/bin/csh
sdb:x:513:514:MmaxDB Software Owner:/home/sdb:/bin/bash
sqdn4s:x:514:515:SAP Database Administrator:/home/sqdn4s:/bin/csh
arato:x:501:501:Arató Mátyás:/home/arato:/bin/bash
uuidd:x:104:105:UUID generator helper daemon:/var/lib/libuuid:/sbin/nologin

[adamkoa@kkk ~]$ cat /etc/passwd | cut -d: -f1,5,7 | tail
n4sadm:SAP System Administrator:/bin/csh
sdb:MmaxDB Software Owner:/bin/bash
sqdn4s:SAP Database Administrator:/bin/csh
```

```
arato:Arató Mátyás:/bin/bash
uuidd:UUID generator helper daemon:/sbin/nologin
[adamkoa@kkk ~]$
```

2.4. grep

A grep valójában nem egy, hanem három parancsot takar: grep, az egrep (extended) és az fgrep (fast). Grep a megnevezett bemeneti fájlokban a megadott mintához illeszkedő sorokat keres. Amennyiben nincs bemenő fájlnév megadva, a standard bemenetet olvassa.

Szintaktika:

```
grep [minta] [fájlnév] [kapcsolók]
```

Alapértelmezés szerint grep a mintához illeszkedő sorokat a standard outputra írja. A mintát a fentebb leírt reguláris kifejezésekkel lehet megadni.

Kapcsolók:

- -c : elhagyja a szokásos kimenetet, ehelyett az illeszkedést mutató sorok számát írja ki minden fájl esetére
- -i : nem különbözteti meg a kis- és nagybetűket sem a mintában, sem a bemeneti fájlban
- -l : elhagyja a szokásos kimenetet, és csak azon fájlok neveit adja meg, amelyekből származna kimenet
- -v : megfordítja az illeszkedés értelmét: a mintához nem illeszkedő sorokat választja ki
- -w : csak azokat a sorokat választja ki, amelyekben az illeszkedés teljes szavakból származik. Azaz az illeszkedést mutató szövegrész előtt és után nem állhat betű, szám vagy aláhúzásjel.

A minta lehet egyszerű sztring, és használhatók metakarakterek is. Ha a mintát megtalálja egy sorban akkor azt mondjuk, a minta illeszkedik.

2.4.1. Minta metakarakterek

A minta felépítéséhez a grep a reguláris kifejezések nyelvét használja, amely a Perl konvencióját alkalmazza. Lássuk az egyes jeleket és azok jelentését:

Metakarakterek:

5.1. táblázat - Metakarakterek

karakter	jelentése
.	Bármely karakter Pl.: M.m illeszkedik minden karaktersorra ahol a nagy M és a kis m betű között pontosan egy bármilyen karakter van.
[felsorolás]	A felsorolásban lévő bármelyik karakterre illeszkedik. Pl.: [aeiou] Bármely angol magánhangzóra illeszkedik.
[tól – ig]	A határok által megadott karakterek mindegyikére illeszkedik. Pl.: [a-z] Bármely a és z közötti bármelyik karakterre illeszkedik.
[^minta]	Bármely karakterre illeszkedik a mintát alkotó karakterek kivételével (negálás)
^	A sor eleje. Pl.: ^A Csak a sor elején álló nagy A betűre illeszkedik
$	A sor vége. Pl ^$ Üres sorokra (sor eleje után rögtön a vége) illeszkedik.
[:karakterosztály:]	előre definiált karakterosztályok vannak (lásd man

karakter	jelentése
	grep) • [:alpha:] betűkre illeszkedik • [:lower:] kisbetűkre illeszkedik • [:upper:] nagybetűkre illeszkedik • [:digit:] számjegyekre illeszkedik stb.

Ismétlődés jelzők:

5.2. táblázat - Ismélődés jelzők

karakter	jelentése	példa
?	előző tag 0-szor vagy egyszer	Pl: al?ma Illeszkedik az alma és az ama karaktersorra.
*	előző tag 0-szor vagy többször	Pl.: al*ma illeszkedik: ama alma, allma, alllma stb.
+	előző tag 1 szer vagy többször	Pl.: al+ma u.a. mint előbb, de az ama nem illeszkedik
{n}	előző tag pontosan n szer	Pl.: k{8} pontosan 8 k betűhöz illeszkedik
{n,}	előző tag legalább n-szer vagy többször	
{ ,m}	előző tag maximum m-szer	
{n,m}	előző tag legalább n-szer legfeljebb m-szer	Pl.: r{2,4} legalább kettő, maximum 4 egymás melletti r betűre illeszkedik

2.5. head

Kiírja az állomány első néhány sorának tartalmát a standard outputra.

Szintaktika:

```
head [filenév]
```

2.6. paste

A paraméteréül kapott fájlok sorait egymás mellé fűzi és az eredményt a standard kimenetre írja.

Szintaktika:

```
paste [fájlnév1] [fájlnév2]
```

2.7. rev

Visszafele kiírja a bemenetként kapott sorokat.

Szintaktika:

```
rev [fájlnév]
```

Példa a rev parancs használatára:

```
adamkoa@it:~$ rev nevek.txt
norÁ
alittA
anraB
itteB
icaL
ótáneR
adamkoa@it:~$
```

2.8. sed

Folyamszerkesztő (stream editor), tulajdonképpen egy nem interaktív programozható szövegszerkesztő/manipuláló program. A program a megnevezett fájlokat (alapértelmezés szerint a standard bemenetet) a standard kimenetre másolja, de közben egy parancsokat tartalmazó szkriptnek megfelelően megszerkeszti.

Használat:

```
sed [kapcsolok] [script] [fajl]
```

Kapcsolók:

- -n : Alapértelmezésben a SED igen sokat ír a képernyőre, minden sor után folyamatosan tájékoztatva a felhasználót. Ezen kapcsoló megadásakor csak akkor ír a kimenetre mikor a szkriptben erre a p módosítóval utasítást adunk.

A szkript:

A szkript nagy általánosságban sed 's/mit/mire/módosító' alakú. A sed végignézi az összes sort (egyenként), és ha talál olyan szövegrészt, ami illeszkedik a "mit" mintára, lecseréli a "mire" mintának megfelelő szövegre. A szkriptben a "mit" helyén az összes fentebb ismertetett reguláris kifejezés használható a keresett minta definiálására. Ha nem adunk meg "módosító"-t, akkor hiába van több olyan szövegrész is a sorban, amire illeszkedik a "mit" minta , csak az elsőt cseréli!

A sed 's/mit/mire/N' csak az N-edik előfordulást cseréli (minden sorban újrakezdi a számolást), míg a sed 's/mit/mire/g' az összes előfordulást cserélni.

```
# A p (kiíratás) módosító használata: adamkoa@it:~$ cat nevek.txt | sed -n '2,5p'  #
csak a 2-tól az 5. sorig írja ki Attila Barna Betti Laci
adamkoa@it:~$ sed -n '/Zoli/p' nevek.txt     # csak a "Zoli" mintának megfelelő sorokat
írja ki
Zoli

adamkoa@it:~$ ls -l | sed -n '/xy.*/p'       # csak az "xy"-nal kezdődő fájlneveket írja
ki
-rwx------ 1 adamkoa prog1 16589 2007-02-12 18:26 xy
-rw-r--r-- 1 adamkoa prog1 61    2007-02-12 18:22 xy.c
-rw------- 1 adamkoa prog1 196   2007-02-12 18:26 xy.log
-rw------- 1 adamkoa prog1 6     2007-02-12 18:26 xy.out
adamkoa@it:~$

# A d (törlés) módosító használata:

adamkoa@it:~$ ls -l | sed '/xy.*/d'          # csak azokat a fájlneveket írja ki,
amelyek nem "xy"-nal kezdődnek
összesen 36
-rw------- 1 adamkoa prog1 66  2007-04-26 14:24 nevek.txt
-rw------- 1 adamkoa prog1 0   2007-04-26 16:22 x.txt
drwx------ 2 adamkoa prog1 144 2007-04-12 15:10 zh2
adamkoa@it:~$

# Az s (csere) módosító használata:

adamkoa@it:~$ ls -l | sed 's/ /:/g'          # minden szóközt ":"-ra cserél, a /g
biztosítja az összes előfordulás cseréjét
összesen:36
-rw-------:1:adamkoa:prog1::::66:2007-04-26:14:24:nevek.txt
-rw-------:1:adamkoa:prog1:::::0:2007-04-26:16:22:x.txt
```

```
-rwx------:1:adamkoa:prog1:16589:2007-02-12:18:26:xy
-rw-r--r--:1:adamkoa:prog1::::61:2007-02-12:18:22:xy.c
-rw-------:1:adamkoa:prog1:::196:2007-02-12:18:26:xy.log
-rw-------:1:adamkoa:prog1:::::6:2007-02-12:18:26:xy.out
drwx------:2:adamkoa:prog1:::144:2007-04-12:15:10:zh2
adamkoa@it:~$ ls -l | sed 's/ /:/2'          # csak a 2. előfordulást cseréli
összesen 36
-rw------- 1:adamkoa prog1 66 2007-04-26 14:24 nevek.txt
-rw------- 1:adamkoa prog1 0 2007-04-26 16:22 x.txt
-rwx------ 1:adamkoa prog1 16589 2007-02-12 18:26 xy
-rw-r--r-- 1:adamkoa prog1 61 2007-02-12 18:22 xy.c
-rw------- 1:adamkoa prog1 196 2007-02-12 18:26 xy.log
-rw------- 1:adamkoa prog1 6 2007-02-12 18:26 xy.out
drwx------ 2:adamkoa prog1 144 2007-04-12 15:10 zh2
adamkoa@it:~$
```

2.9. sort

Szövegfájl sorainak rendezése.

Szintaktika:

```
sort [kapcsolók] [be_fájlnév] [ki_fájlnév]
```

Kapcsolók:

- -o : az eredményt a kimeneti állományba írja az alapértelmezés szerinti kimenet helyett
- -r : visszafele (csökkenő sorrendbe) rendez
- -n : Szöveg elején álló számok numerikus összehasonlítása. A számok előjelből és 0 vagy több számjegyből, és tizedespont utáni további számjegyekből állhatnak
- -u : A bemenet ismétlődő sorai a kimeneten pontosan csak egyszer fognak szerepelni
- -k : a rendezés alapjául szolgáló kulcs oszlop megadása, ha elhagyjuk, akkor az egész sort tekinti egybe.

```
[adamkoa@kkk proba]$ ls -l
total 60
-rw-rw-r--  1 adamkoa adamkoa    0 May  9 15:32 core
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
-r--rw-r--  1 adamkoa adamkoa   25 Mar  9  2010 p2.txt
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
lrwxrwxrwx  1 adamkoa adamkoa    8 Apr 27  2010 szimbolikus.txt -> link.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
lrwxrwxrwx  1 adamkoa adamkoa    5 Apr 27  2010 tmp -> /tmp/
[adamkoa@kkk proba]$ ls -l |sort -k 5
total 60
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
-rw-rw-r--  1 adamkoa adamkoa    0 May  9 15:32 core
-r--rw-r--  1 adamkoa adamkoa   25 Mar  9  2010 p2.txt
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
lrwxrwxrwx  1 adamkoa adamkoa    5 Apr 27  2010 tmp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    8 Apr 27  2010 szimbolikus.txt -> link.txt
[adamkoa@kkk proba]$ ls -l |sort -k 5 -n
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
-rw-rw-r--  1 adamkoa adamkoa    0 May  9 15:32 core
total 60
lrwxrwxrwx  1 adamkoa adamkoa    5 Apr 27  2010 tmp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    8 Apr 27  2010 szimbolikus.txt -> link.txt
-r--rw-r--  1 adamkoa adamkoa   25 Mar  9  2010 p2.txt
```

```
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
[adamkoa@kkk proba]$
```

Több oslzop szerint is lehet rendezni:

```
[adamkoa@kkk proba]$ ls -l |sort -k5n -k9
total 64
-rw-rw-r--  1 adamkoa adamkoa    0 May  9 15:32 core
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
lrwxrwxrwx  1 adamkoa adamkoa    5 Jun  5 15:21 temp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    5 Apr 27  2010 tmp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    8 Apr 27  2010 szimbolikus.txt -> link.txt
-r--rw-r--  1 adamkoa adamkoa   25 Mar  9  2010 p2.txt
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
[adamkoa@kkk proba]$ ls -l |sort -k5n -k9r
-rw-r-----  1 adamkoa adamkoa    0 Mar 23  2010 link.txt
-rw-rw-r--  1 adamkoa adamkoa    0 Mar  9  2010 link2.txt
-rw-rw-r--  1 adamkoa adamkoa    0 May  9 15:32 core
total 64
lrwxrwxrwx  1 adamkoa adamkoa    5 Apr 27  2010 tmp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    5 Jun  5 15:21 temp -> /tmp/
lrwxrwxrwx  1 adamkoa adamkoa    8 Apr 27  2010 szimbolikus.txt -> link.txt
-r--rw-r--  1 adamkoa adamkoa   25 Mar  9  2010 p2.txt
-rw-rw-r--  1 adamkoa adamkoa   26 Apr 20  2010 sed.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file.txt
-rw-rw-r--  1 adamkoa adamkoa   30 Apr 14  2010 new_file2.txt
drwxrwxr-x  2 adamkoa adamkoa 4096 Apr 27  2010 test
[adamkoa@kkk proba]$
```

2.10. uniq

Az egymást követő ismétlésekkel dolgozik. Az inputon érkező rendezett adathalmazból alapértelmezésben eltünteti a duplikált sorokat.

Szintaktika:

```
uniq [kapcsolók] [fájlnév]
```

Kapcsolók:

- -u : csak a nem azonos sorokat írja ki
- -d : csak a duplikált sorokat írja ki
- -c : kiírja a sor elé, hogy az adott sor hányszor fordult elő

2.11. wc

A Word Count bájtok (karakterek), szavak (whitespace karakterekkel tagolva), sorok megszámlálására alkalmas.

Szintaktika:

```
wc [kapcsolók]
```

Kapcsolók:

- -c : karaktereket számol
- -w : szavak számol

- -l : sorokat számol

Példa a wc parancs használatára:

```
adamkoa@it:~$ wc nevek.txt
10 11 66 nevek.txt
adamkoa@it:~$
```

2.12. tail

Kiírja az állomány utolsó néhány sorának tartalmát a standard outputra.

Szintaktika:

```
tail [kapcsolók] [filenév]
```

Kapcsolók:

- -f : hozzákapcsolódik az állományhoz és mikor az állomány tartalma megváltozik akkor kiírja az utolsó sorait. Ez igen hasznos funció aktív rendszereken az aktív folyamatok log állományit nyomonkövetni, úgy hogy közben más folyamatot is futtathatunk feltéve ha a tail-t háttérfolyamatként indítottuk el.

2.13. tr

A tr karakterek lecserélésére, vagy törlésére használható.

Szintaktika:

```
tr [mit] [mire]
```

Példa a tr parancs használatára:

```
adamkoa@it:~$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
Megyek, iszom valamit.
MEGYEK, ISZOM VALAMIT.
adamkoa@it:~$ tr abcd AZ
ab
AZ
abc
AZZ
abcd
AZZZ
adamkoa@it:~$
```

2.14. tee

Szintaktika:

```
tee [kapcsoló] [fájlnév]
```

Az adatokat a standard inputról veszi. Paramétere egyetlen fájl amibe a beérkező adatot beleírja, majd a standard outputon az adatot továbbdobja. A -a kapcsoló használata esetén a fájlt folytatja (append), ellenkező esetben a tartalmát törli.

Példa a tee parancs használatára:

```
adamkoa@it:~$ date | tee ido.txt | ...
adamkoa@it:~$
```

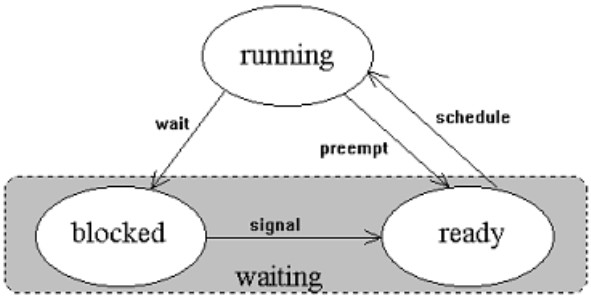
Az aktuális idő az "ido.txt" állományba kerül majd a pipe fut tovább a dátummal.

6. fejezet - Folyamatkezelés

A Linux többfeladatos (multitask) és többfelhasználós (multiuser) rendszer. Ebből következik, hogy akár egy felhasználó egy időben több programot is futtathat. Az elindított program a processz, azaz folyamat, más megfogalmazásban egy végrehajtható fájl "élő" változata. Gyakran taszknak is nevezik. A folyamatok jól definiált hierarchiát alkotnak. Minden folyamatnak pontosan egy szülője (parent) van, és egy vagy több gyermek folyamata (child process) lehet. A folyamat hierarchia tetején az init folyamat helyezkedik el. Az init folyamat az első létrehozott felhasználói folyamat, a rendszer indulásakor jön létre. Minden felhasználói folyamat az init folyamat leszármazottja. Néhány rendszer folyamat, mint például a swapper és a page daemon (a háttértár kezelésével kapcsolatos folyamatok), szintén a rendszer indulásakor jön létre és nem az init folyamat leszármazottja. Ha egy folyamat befejeződésekor még léteznek aktív gyermek folyamatai, akkor azok árvákká (orphan) válnak és azokat az init folyamat örökli - majd egyben meg is szünteti azokat. A Linux minden egyes feladathoz két számot (PID, process identificator - feladat azonosítót és a PPID, parent process identification - szülő azonosítója) rendel. A rendszer a PID-et automatikusan növeli. Az init folyamat PID-je 1.

Az "életre keltett" folyamatok szekvenciálisan hajtódnak végre, azaz a felhasználó csak akkor kapja vissza a készenléti jelet, ha a végrehajtás befejeződött - alapvetően szinkron módon működik. A processz futhat előtérben (billentyűzetet és a képernyőt magához ragadva), és háttérben (manuálisan is elő lehet idézni, melynek formája:`parancsnév&`). Ha egy előtérben futó folyamatot szeretnénk háttérbe helyezni, a suspend funkcióhoz rendelt karakterrel (ez rendszerint a Ctrl+Z) tudjuk az előtérben futó folyamatot felfüggeszteni. Ezután pedig a megfelelő parancsokkal tudjuk folytatni a futását, háttérbe helyezni, végleg leállítani, stb.).

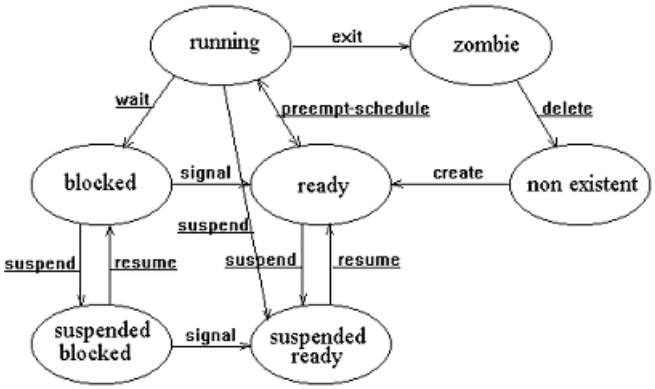
Minden processz önálló entitás a saját programszámlálójával és kontextusával. Lehet közöttük együttműködési kapcsolat, akár szinkron, amikor az egyik processz készít valamilyen output-ot, ami a majd másik processz bemenete lesz, illetve aszinkron, amikor már két futó folyamat kommunikál. Szinkron esetben a két processz futásának relatív sebességétől függően előfordulhat, hogy a második processznek várnia kell, amíg az első a kimenetét elkészíti. A második u.m. blokkolt amíg az inputja elkészül. Kérdés merülhet fel, hogyan "billen" ki ebből az állapotból a blokkolt processz. Másrészt az is előfordulhat, hogy egy processz ugyan nem vár semmire, tehát futhatna, de az operációs rendszer egy másik processznek adja át a CPU-t: ekkor is "vár" a processzünk, most a CPU-ra, ezt az állapotát feltétlenül meg kell különböztetni az inputra való várakozástól. Azt mondhatjuk, hogy a processzek - életük során - különböző állapotokban (state) lehetnek, az állapotok között különböző állapotátmenetek lehetségesek. A legegyszerűbb és legáltalánosabb állapot és állapotátmenet diagram a következő:



Egy speciális háttérfolyamat a daemon. Ezek nagy részét a Linux rendszer már a rendszerbetöltéskor elindítja. Számos démon fut a háttérben és figyeli, pl. a lokális hálózatba belépőket, a nyomtatási kérelmeket, stb. Pl.: inetd (tcpd), ftpd, httpd. Tipikusan a szolgáltatásokkal lehet őket azonosítani. Talán ezért is lehet őket a /sbin mappa alatt található `service` utasítással vezérelni. Inaktív állapotban várakoznak arra, hogy a szolgáltatásuk igénybevételre kerüljön, ekkor aktív állapotba kerülnek, kiszolgálják a kérést, majd visszamennek inaktív állapotba.

Egy másik speciális helyzetű folyamat a zombie, a már halott (leállt), de még a rendszerből el nem tűnt folyamat: akkor lehetséges ez, ha a gyermek folyamat már kilépett, de a szülő még nem fogadta a gyermek visszatérési értékét - még nem vett tudomást gyermeke haláláról (befejeződéséről).

A rendszeren belüli egyes állapotok és a közöttük fellelhető állapotátmeneteket az alábbi ábra szemlélteti:



1. Folyamatkezelő parancsok

1.1. ps

A Process Sate parancs processzusok állapotát jeleníti meg.

Szintaktika:

```
ps [kapcsolók]
```

Kapcsolók:

- -e : az összes folyamat megjelenítése
- -f: részletes lista
- -u username : megjeleníti az adott felhasználó összes folyamatát

Mezők jelentése:

- PID : a folyamat azonosítója
- TTY : a vezérlő terminál azonosítója
- STAT : a folyamat állapota
- TIME : a processz által eddig elhasznált processzor idő
- CMD : a processz neve

Példa a ps parancs használatára:

```
adamkoa@it:~$ ps
PID   TTY   TIME     CMD
15573 pts/4 00:00:00 bash
16407 pts/4 00:00:00 ps
adamkoa@it:~$
```

1.2. pstree

Az initből induló folyamathierarchiát lehet a parancs segítségével megtekintetni fa szerkezetű ábrázolásban.

1.3. nohup

Mikor a rendszerből kijelentkezünk (azaz a bash bezáródik) minden gyerekfolyamatát a rendszer automatikusan kilövi. Lehetőségünk van azonban arra is, hogy egy folyamatot immúnissá tegyünk kilépésünkre. Hosszan, több óráig, több napig futó programokat a `nohup` paranccsal indíthatunk.

Példa a nohup parancs használatára:

```
adamkoa@it:~$ nohup program
adamkoa@it:~$
```

1.4. top

A top a kill és a ps parancs egyesített változata mely folyamatosan futva listázza az éppen aktív folyamatokat, információt nyújt a rendszer állapotáról és terheltségi mutatóiról illetve lehetőséget ad szignálok küldésére folyamatok számára. A top-ot a parancssorban kiadott top utasítással indíthatjuk.

2. Jelzések, szignálok

A Linux rendszer a folyamatok vezérlését a folyamatoknak küldött ún. szignálok segítségével végzi: a Ctrl+Z billentyű például egy STOP szignált küld az előtérben futó processznek. Igen sok (kb. 60 db) szignál létezik, ezek közül csak néhányat tárgyalunk. Processzt megszüntetni szintén szignál(ok) segítségével lehet: az előtérben futó program a Ctrl+C megnyomására egy INT szignált kap, amely rendszerint a program elhalálozását vonja maga után. Háttérben futó folyamatainkat a kill paranccsal állíthatjuk le.

2.1. kill

A kill a nevével ellentétben nem csak folyamatok megölésére használható, hanem tetszőlegese signalt küldhetünk vele bármely folyamatnak melynek tudjuk a tudjuk a PID számát és rendelkezünk a folyamat kezeléséhez megfelelő jogokkal. Alapértelmezés szerint (signal paraméter nélkül használva) a kill egy TERM (terminate) szignált küld a megadott folyamatnak.

Szintaktika:

```
kill [signal] [PID]
```

Példa a kill parancs használatára:

```
adamkoa@it:~$ ps
PID TTY STAT TIME COMMAND
310 pp0 S    0:00 -bash
313 pp0 R    0:00 ps
321 pp0 R    0:00 find -name= doksi

adamkoa@it:~$ kill 321

adamkoa@it:~$ ps
PID TTY STAT TIME COMMAND
310 pp0 S    0:00 -bash
334 pp0 R    0:00 ps
adamkoa@it:~$
```

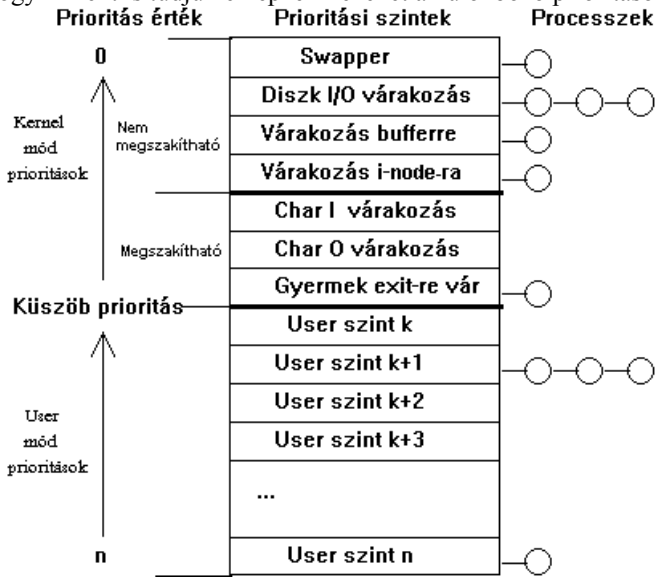
Ha más (nem TERM) szignált akarunk küldeni, a kill parancsot megfelelően paraméterezni kell. Folyamatot megölni még a HUP (hangup) és a KILL (9-es) szignálokkal is lehet. (a nohup parancs ezen HUP szignál ellen teszi immúnissá a folyamatot.) A sokféle látszólag azonos hatású szignál oka, hogy korántsem azonos hatásúak: például a HUP és a TERM szignálokat a folyamat felülbírálhatja, saját szignál-kezelő rutint állíthat be. Ezeket a szignálokat a folyamat kapja meg, és alapértelmezés szerinti kezelő rutinjuk lép ki. A KILL szignál hatására viszont a kernel öli meg a folyamatot, annak megkérdezése nélkül. Ezért nem probléma Unixban, ha egy folyamat "lefagy", végtelen ciklusba kerül: egy KILL szignál mindig megoldja a problémát. Szignált csak saját processzeinknek küldhetünk (kivéve a root-ot, aki bármely processzről rendelkezhet). Fontos még az ALARM szignál. Ezzel a rendszert megkérhetjük, hogy megadott idő elteltével küldjön egy jelet. Ezt használják időzítési célokra, többek között a sleep utasítás is így működik.

3. Prioritás

A prioritás (az ütemezési - scheduling - prioritás) azt szabja meg, hogy ha több folyamat is van egyszerre futóképes állapotban, akkor a kernel milyen arányban ossza meg a rendelkezésre álló CPU időt az egyes processzek között. Unixban a prioritás számszerű értéke minél kisebb, annál több CPU időt fog kapni a folyamat. Prioritás értéke 19-től -20-ig terjed és a negatívabb érték magasabb prioritást jelent. A processzek prioritását a "top" paranccsal vagy a "ps" parancs -l opciójával a PRI oszlopban lehet megnézni.

Minden folyamat három prioritással rendelkezik: egy alapprioritással (base priority), amely állandó, egy ütemezési prioritással (sceduling priority), amely a program futásakor nő, és egy ún. "nice" prioritással, amely (bizonyos határok között) felhasználó által változtatható. Ütemezéskor e három érték bizonyos szabályok szerint képzett összegét használja a rendszer: az ütemező algoritmus döntési pontján mindig a legalacsonyabb összeggel rendelkező processz kapja meg a vezérlést (ezért kell ebbe az összegbe az elhasznált CPU idővel növekvő tagot is tenni: egyébként mindig csak a legmagasabb prioritású folyamat futna).

Az alábbi ábra mutatja, hogy miként is tudjuk elképzelni ezeket a különböző prioritásokat:



A "`nice -n növekmény parancs`" utasítás szolgál arra, hogy a "parancs"-ot a megnövelt nice prioritás értékkel futtassuk. Erre akkor lehet szükség, ha valami számításigényes, hosszan futó programot indítunk, de nem akarjuk jelentősen lassítani az interaktívan dolgozók munkáját. Ezt a "nice" értéket egyébként a "top" "r" parancsával is megváltoztathatjuk. Nem privilegizált felhasználó csak növelni -azaz gyengíteni- tudja folyamatai nice értékét (illetve a visszacsökkentéskor nem tudja az induló érték alá csökkenteni), a root felhasználó természetesen tetszőlegesen állíthat prioritást.

Már futó processz esetén a renice parancs segítségével tudunk a prioritáson változtatni.

4. Előtér, háttér

Háttérfolyamatot előtérbe hozni az **fg** paranccsal tudunk. Ha egy másik programmal szeretnénk foglalkozni, de azt akarjuk hogy az előtérben lévő folyamat tovább fusson a háttérben, akkor a Ctrl+Z billentyűkombinációval megállíthatjuk (ekkor várakozó, "stopped" állapotba kerül), majd háttérbe helyezni a **bg** paranccsal tudjuk**.** Ha a folyamat futásképes, (nem vár mondjuk terminál inputra) akkor a háttérben tovább fog futni. Kilépéskor, ha vannak még háttérben futó vagy várakozó folyamataink, a rendszer erre figyelmeztet a "`You have running jobs`" vagy "`You have stopped jobs`" üzenettel. Ha közvetlenül ez után még egyszer kiadjuk a logout parancsot, a shell leállítja a háttérfolyamatokat és kiléptet bennünket a rendszerből. Ha az fg és bg parancsokat argumentum nélkül használjuk, mindig a legutoljára hivatkozott folyamatra vonatkoznak. Ettől eltérő esetben hivatkozhatunk a feladatra a job azonosítójával, amit a `jobs` parancs ad meg, vagy hivatkozhatunk a nevével is. Minkét esetben egy % jellel kell bevezetni a paramétert. (Grafikus felület esetén erre a legjobb szemléltető példa az xeyes nevű program többszöri megnyitása és ezek szabályozása.)

5. Ütemezett végrehajtás

A UNIX multitaszking képessége nem korlátozódik csak a jelenben a futó folyamatokra. Tartalmaz programokat, amelyek lehetővé teszik a programok ütemezett futtatását, akár egyszeri, akár ismétlődő időközönként. Egy adott időben az `at` program segítségével indíthatunk el folyamatokat, ismétlődő esetekben pedig a `corntab` alkalmazással.

5.1. at

Az `at` parancs lehetővé teszi a programok,parancsok vagy shell skriptek egy jövőbeli dátum és időben történő futtatását. Például ha e-mailben szeretnénk egy fájl tartalmát elküldeni, vagy a find segítségével szeretnénk egy keresést indítani, akkor amikor a rendszer terheltsége alacsony, például egy hajnali órában.

Az at használatához az alábbi lépéseket hajtsuk végre:

- Adjuk meg az at parancsot egy időspecifikációval,ahol az idő meghatározása lehet:

  $at 10:30am today

  $at midnight

  $at 12:01 1 jan 2012

  Lásd még a man oldalt további példákért.

- Az at promptjánál (`at>`) adjuk meg a végrehajtandó parancsot.

- Több utasítás megadásához üssünk Enter-t, vagy CTRL-D-t a befejezéshez.

Ezután egy azonosító rendelődik az ütemezett feladathoz és bekerül a végrehajtási sorba. A sor állapotát az `atq` paranccsal tekinthetjük meg, ha pedig el szeretnénk távolítani egy ütemezett feladatot, akkor az `atrm [job#]` paranccsal tehetjük meg.

Ha csak egy parancsot szeretnénk futtatni, akkor azt megtehetjük az interaktív mód használata nélkül is: `$at [időspec][szkriptfájl neve]`. Például a `$at midnight whoison` segítségével megnézhetjük kik dolgoznak még éjfélkor is a gépen.

5.2. crontab

A crontabbal lehetőségünk nyílik időzített programfuttatásra, melynek kimenetéről e-mailben kapunk értesítést.

Mindig csak egy-egy felhasználóra vonatkozó crontabot lehet módosítani. Csak a superuser adhat meg a magáétól különböző felhasználónevet, illetve más crontab könyvtárat a parancshoz. Általában a -e opció jeleneti a saját crontab-unk módosítását. A crontab-ok módosításához a vi vagy a joe szövegszerkesztőt, használja a parancs.

Az egyes mezők tartalmazhatnak időpontot, időintervallumot, skip faktoros időintervallumot, szimbolikus intervallumot a hét napjaira, illetve az év hónapjaira, valamint további részintervallumokat vesszővel elválasztva. A crontab file -ban lévő üres, vagy kettős kereszttel kezdődő sorokat a parancs nem veszi figyelembe. Ha megadtuk a hét és a hónap egyik napját is, akkor a crontab bejegyzés le fog futni minden héten a megadott napon, valamint minden hónapban a megadott napon (a két feltétel vagy kapcsolatát képezzük.).

Példa a crontab parancs használatára:

```
# m h dom mon dow command
# PERC ÓRA NAP HÓNAP AHÉTEGYNAPJA PARANCS # MIN HOUR DAY MONTH DAYOFWEEK COMMAND
# minden nap reggel 6:10-kor
10 6 * * * date

# minden második órában az óra végén
0 */2 * * * date

# minden második órában reggel 11-től este 7-ig , valamint este 8-kor
0 23-7/2,8 * * * date

# este 11-kor negyedikén, valamint minden hétfőn, kedden, és szerdán
0 11 4 * mon-wed date 0 11 4 * mon-wed date

# január elsején délután 4-kor
0 4 1 jan * date 0 4 1 jan * date

# óránként egyszer, és minden kimenet a log file -ba menjen
0 4 1 jan * date >>/var/log/messages 2>&1 0 4 1 jan * date >>/var/log/messages 2>&1
```

6. Irányító operátorok

`p1 && p2` : p2 akkor fut le, ha p1 hiba nélkül fejezi be a futást (0-val tér vissza) - azaz csak sikeres futás után induljon a p2.

`p1 || p2` : p2 csak akkor fut le, ha p1 nem 0-val tér vissza - azaz hiba esetén kell valamit tenni, egyébként semmit.

hibakód lekérdezése: `echo $?`

7. Parancsbehelyettesítés

Parancsbehelyettesítés:

- `` `parancs` ``

  vagy

  ```
  $(parancs)
  ```

  pl.:

  ```
  [adamkoa@kkk ~]$ echo "Most" `who | wc -l` "db felhasználó van bejelentkezve."
  Most 1 db felhasználó van bejelentkezve.
  [adamkoa@kkk ~]$
  ```

Parancsbehelyettesítés **egymásba ágyazható**.

7. fejezet - Egyéb hasznos programok

1. Felhasználó és csoport kezelő parancsok

- **w**

  A w információkat jelenít meg arról, hogy éppen hány felhasználó van a gépen és hogy mit csinálnak. A fejléc megmutatja - ebben a sorrendben - az időt, mióta működik a rendszer, jelenleg hány felhasználó van belépve és a rendszer átlagos terhelését az elmúlt 1, 5 és 15 percben. Az alábbiak minden felhasználónak megjelennek: azonosító, a tty neve, a távoli host, ahonnan bejelentkezett, a belépés ideje, a "henyélési" idő (azaz mióta nem adott inputot a gépnek), JCPU, PCPU és az éppen futó programjaik parancssora.

  Példa a w parancs használatára:

```
[adamkoa@kkk proba]$ w
 15:59:54 up 158 days,  4:18,  1 user,  load average: 0.13, 0.09, 0.08
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
adamkoa  pts/1    iad010.inf.unide 15:31    0.00s  0.04s  0.00s w
[adamkoa@kkk proba]$
```

- who

  A bejelentkezett felhasználók kilistázása.

```
[adamkoa@kkk proba]$ who
adamkoa  pts/1        May 22 15:31 (iad010.inf.unideb.hu)
[adamkoa@kkk proba]$
```

- whoami

  Kiírja az aktuális felhasználó nevét és user id számát. Egyenértékű az `id -un` paranccsal.

```
[adamkoa@kkk proba]$ whoami
adamkoa
[adamkoa@kkk proba]$ id -un
adamkoa
```

- id

```
[adamkoa@kkk proba]$ id
uid=500(adamkoa) gid=500(adamkoa)
groups=500(adamkoa),507(svnusers),512(fuse),528(wwwadmin)
context=user_u:system_r:initrc_t
[adamkoa@kkk proba]$
```

- groups

  Kiírja azon csoportok neveit melyeknek az aktuális felhasználó a tagja.

```
[adamkoa@kkk proba]$ groups
adamkoa svnusers fuse wwwadmin
[adamkoa@kkk proba]$
```

- users

  Kiírja a bejeretnekezett felhasználók nevét egy sorba.

```
[adamkoa@kkk proba]$ users
adamkoa
[adamkoa@kkk proba]$
```

- passwd

  Ezen parancs segítségével lehet az aktuális felhasználó jelszavát megváltoztatni.

2. Egyéb

- date

  Az aktuális dátumot és időt kiírja az standard outputra.

  Példa a date parancs használatára:

```
adamkoa@it:~$ date
2007. máj. 1., kedd, 19.14.08 CEST
adamkoa@it:~$
```

- clear

  Törli a konzolon látható összes szöveget.

- logout

  Kijelentkezés a shellből.

8. fejezet - Archiválás

Az archiválás nagyon fontos lépés az adatok biztonságos megőrzésének folyamatában. A UNIX esetén ez tradicionálisan a '70-as évek szalagos mentéséhez kötődően a tar program segítségével történik, melyet aztán a helytakarékosság érdekében lehet tömöríteni is. Az archiválás többféle stratégia szerint is történhet, ezek lehetnek inkrementális, differenciális vagy teljes mentések.

1. tar

A Tape Archiver segédprogram GNU változata. Nem tömörít, csak egybemásol. Ha az argumentumban nem szerepel kimeneti fájlnév, a tar a standard kimenetre írja a végeredményt.

Szintaktika:

```
tar [kapcsolók] [fájlnév]
```

Kapcsolók:

- -c : új archív fájlt hoz létre
- -x : kicsomagolja a fájlokat az archív fájlból
- -t : az archív fájl tartalmát listázza
- -v : a feldolgozott fájlokat beszédesen listázza, kijelzi az összes tömörített vagy kicsomagolt fájl tömörítési arányát és nevét
- -f : a megadott fájl- illetve eszköznevet használja (alapértelmezés: /dev/rmt0, amennyiben van szalagos egység a rendszerben és a TAPE környezeti változó be van állítva)
- -r: további fájlok hozzáfűzése az archívumhoz

```
adamkoa@morse:~>tar -c hello.c
hello.c000064400026400000764000000000056115662077520111260ustar  adamkoaik main( ) {
        printf("hello, world");
 }
adamkoa@morse:~>
```

Aktuális mappa elmentése:

```
[adamkoa@kkk proba]$ tar cvf proba.tar .
./
./test/
./link.txt
./link2.txt
tar: ./proba.tar: file is the archive; not dumped
./linkproba/my pipe
./linkproba/out.gz
[adamkoa@kkk proba]$
```

Tartalom megjelenítése:

```
[adamkoa@kkk proba]$ tar tvf proba.tar
drwxrwxr-x adamkoa/adamkoa   0 2011-05-22 15:31:48 ./
drwxrwxr-x adamkoa/adamkoa   0 2010-04-27 16:58:47 ./test/
-rw-r----- adamkoa/adamkoa   0 2010-03-23 17:19:12 ./link.txt
-rw-rw-r-- adamkoa/adamkoa   0 2010-03-09 18:48:30 ./link2.txt
prw-rw-r-- adamkoa/adamkoa   0 2011-05-08 13:58:41 ./linkproba/my_pipe
-rw-rw-r-- adamkoa/adamkoa  40 2011-05-08 13:58:41 ./linkproba/out.gz
[adamkoa@kkk proba]$
```

Fájl hozzáadása:

```
[adamkoa@kkk proba]$ tar -rvf proba.tar ../phonebook
tar: Removing leading `../' from member names
```

```
../phonebook
[adamkoa@kkk proba]$
```

Archívum kibontása:

```
[adamkoa@kkk proba]$ tar -xvf proba.tar
```

Illetve csak bizonyos fájlok kibontása

```
[adamkoa@kkk proba]$ tar -xvf proba.tar link.txt link2.txt
```

2. gzip

Fájlok ki és becsomagolására használhatjuk. Alapesetben a fájlt helyettesíti a tömörített verzióval és a .gz kiterjesztést fűzi hozzá.

Szintaktika:

```
gzip [kapcsolók] [fájlnevek]
```

Kapcsolók:

- -d : kitömörít
- -l : listázza a tömörített állomány tartalmát
- -r : ha fájlnév helyett könyvtárnevet adunk meg akkor minden az abban található fájlt és alkönyvtárat is betömörít

```
adamkoa@it:~$ gzip xy.c

adamkoa@it:~$ ls -l
összesen 48
-rwx------ 1 adamkoa prog1 16589 2007-02-12 18:26 xy.c
-rw-r--r-- 1 adamkoa prog1 84 2007-02-12 18:22 xy.c.gz
adamkoa@it:~$
```

Nagyon egyszerűen köthető össze a tar programmal, miután a tar alapból a standard kimenetre ír, a gzip pedig képes onnan olvasni:

```
[adamkoa@kkk proba]$ tar cv . | gzip -c > proba.tgz
./
./test/
./link.txt
./linkproba/my_pipe
./linkproba/out.gz
[adamkoa@kkk proba]$ ls
linkproba  link.txt  proba.tgz test
[adamkoa@kkk proba]$
```

3. compress / uncompress

A gzip Solarisos párja. A compress parancs fájlt tömörít. A tömörített fájl ".Z" kiterjesztésű, a tulajdoni jogok, a módosítási és hozzáférési idők nem változnak. Ha az argumentumban nem szerepel fájlnév, a parancs a standard bemenetet tömöríti a standard kimenetre. A tömörített fájlok kitömörítését az uncompress paranccsal végezhetjük.

II. rész - A gép, mint eszköz a munkához!

Tartalom

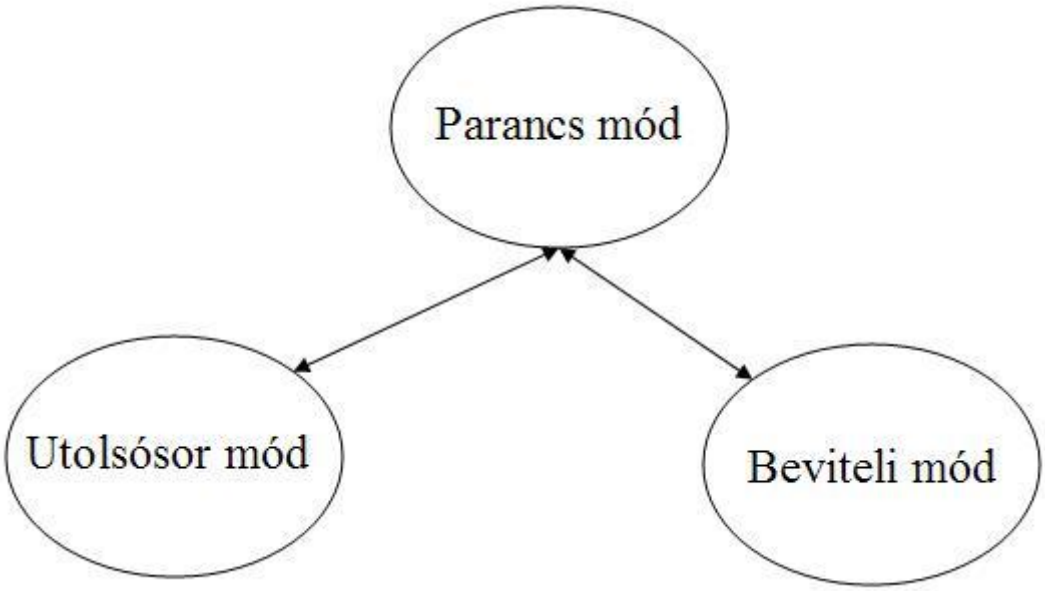
9. fejezet - Szövegszerkesztők

1. VI

A vi egy szabványos UNIX szövegszerkesztő, amely - az előbbi félmondatból is következően - valamennyi UNIX rendszeren megtalálható. Nyilvánvaló tehát, hogy mindenképpen érdemes megismerkedni legalább az alapszintű használatával, hiszen bárki kerülhet olyan helyzetbe, hogy semmiféle egyéb editor nem áll rendelkezésére (például nem tudunk a normál módon bejelentkezni a rendszerünkre, csak egy rendszer-helyreállító bootlemez segítségével) és az ilyen esetekben adunk majd hálát a Mindenhatónak azért, hogy voltunk olyan előrelátók, és megtanultuk a vi alapszintű használatát.

Használata:

Csak gépeljük be, hogy vi. És már benne is vagyunk a vi kellős közepében. Természetesen előfordulhat, hogy a vi egy klónja található a rendszerünkön. Szabványos esetben ilyenkor is elég a vi szót begépelnünk, amely egy link az adott klónra (pl. vim, nvi).



- A Parancs mód:

  A vi indításakor az úgynevezett parancsmódba kerülünk. Innen érhetjük el a szerkesztő többi funkcióját. Bárhol is el vagyunk veszve a vi-n belül a parancs módba az ESC gomb lenyomásával bármikor visszatérhetünk!

  A parancs módban kiadható parancsok

  - h, j, k, l : mozgás a szövegen belül
  - w, b : lépegetés szavanként
  - ^ : ugrás az adott sor elejére
  - $: ugrás az adott sor végére
  - x : a kurzor alatti karakter törlése
  - dw : a következő szó törlése
  - dd : a következő sor törlése

- cw : a következő szó cseréje
- yy : másolás
- y(: egy mondat másolása
- y{ : egy bekezdés másolása
- p : beillesztés
- a : váltás beviteli módba, a kurzor után bővítjük a szöveget
- o : váltás beviteli módba, új sorban folytatjuk a szöveg bevitelét
- : : váltás utolsó sor módba

A másolás és a beillesztés úgynevezett puffereken át történik. Ez nagyon hasonló a Windowsban található vágólaphoz. Így egy szövegrész a vi-n belül több helyre is beilleszthető. 9 db számmal illetve 26, az angol abc kisbetűivel jelzett puffer van. A betűvel jelzettekbe csak mi tölthetünk adatot, míg a számmal jelzett puffereket a vi az yy parancs kiadásakor automatikusan, sorrendben tölti fel a kimásolt szövegrésszel. Ha a betűvel jelzett pufferekbe töltünk adatot, akkor mindig idézőjelt kell írnunk a puffer neve előtt.

Példa az yy és a p parancsokra:

"ayy a kimásolt szöveg az a pufferbe kerül

3p a 3 sorszámú pufferből beillesztjük a benne tárolt szöveget

"ap az a bufferből beillesztjük a benne lévő szöveget

- Utolsó sor mód:

  Utolsósor módba parancsmódból a kettősponttal lehet átváltani. Ekkor a szövegszerkesztő utolsó sorába gépelhetjük az utasításokat.

  Az utolsósorban kiadható parancsok

  - :q : kilépés (ha nincs a szerkesztett fájl mentve akkor rákérdez, hogy mentsük-e)
  - :q! : kényszerített kilépés (nem kérdez rá a mentésre)
  - :w : mentés
  - :w [fájlnév] : mentés más néven
  - :wq : mentés és kilépés
  - :wq! : mentés és kilépés (csak olvasható fájlok esetén is felülírja a fájl tartalmát)
  - :G : az utolsó sorra ugrik
  - :21 : az adott (most pl.: 21.) sorra ugrik
  - :set [belső változó] : az adott belső változó állapotát állítja
  - :set all : minden belső változó alapértelmezésre áll vissza
  - :![parancs] : futtatja a parancsot anélkül, hogy a szerkesztőből kilépne

- Beviteli mód:

  Ekkor szabadon módosíthatjuk a szöveget.

2. JOE

Leírás

A Joe erőteljes, képernyő-orientált ASCII-editor. Felhasználói felülete hasonlít sok felhasználóbarát PC-s szövegszerkesztőhöz. A Micro-Pro-féle WordStar vagy a Borland "Turbo"-nyelveinek felhasználói otthonosan fognak mozogni benne. A Joe mégis jellegzetes Unix-os képernyő-orientált editor, és legfőképpen programok és szövegek szerkesztésére alkalmas.

A Joe számos más szövegszerkesztőt is emulál (WordStar, PICO, GNU-EMACS, stb).

Használat:

A szövegszerkesztő indításához gépeld be, hogy joe, azután a szerkeszteni kívánt nulla vagy több fájl nevét. Ha új fájlt szerkesztesz, az új fájl nevét megadhatod mind a szövegszerkesztő indításakor, mind a fájl elmentésekor. A fájlnevek módosított szintaxisa lehetőséget nyújt programok kimenetének, a szabványos bemenetnek és kimenetnek, illetve fájlok vagy eszközök részeinek szerkesztéséhez. Ha már a szövegszerkesztőben vagy, szöveget gépelhetsz be, és speciális vezérlőkarakter-szekvenciákat használhatsz más szerkesztő feladatok elvégzéséhez. Hogy megtudd, mik a vezérlőkarakter-szekvenciák, nyomj Ctrl+K+H-t a súgóhoz a szövegszerkesztőben.

10. fejezet - Batch fájlok

Batch: szó szerint kötegelt feldolgozást jelent. A mi esetünkben pedig a DOS olyan eszközét, ahol egy szöveges fájlban parancsok írhatunk le egy-egy sorba és az operációs rendszer ezt sorról sorra feldolgozza. Programozási eszközök:

- környezeti változók (SET), paraméterek
- üzenetek kiírása (ECHO) akár ANSI szabványos színkódokkal is!
- megjegyzés (REM)
- menükészítéshez billentyűzet olvasás és kiértékelés (CHOICE)
- növekményes ciklus (FOR)
- feltételvizsgálat (IF)
- ugrás (GOTO) adott címkére

Nézzük az egyes utasításokat részletesebben:

- **SET** : környezeti változók beállítása

  `SET VALTOZO=ERTEK` A VALTOZO nevű környezeti változó beállítása ERTEK értékűre. A változó értékére úgy tudunk hivatkozni, hogy `%` jelek közé zárjuk.

- **ECHO** ('visszhang'): egy sor kiírása a képernyőre.

  Az előbbiekben létrehozott környezeti változó értékének kiíratása:

```
set teszt_valtozo=teszt
echo A létrehozott változó értéke az alábbi:
echo teszt_valtozo=%teszt_valtozo%
```

- **REM**: 'megjegyzés'

  A shell a rem-mel kezdődő sorokat nem hajtja végre, tehát ide megjegyzéseket írhatunk.

- **SHIFT**: eltolás

  A pozicionális paraméterek számozását tolja el eggyel: azaz a második paraméter kerül az első helyére, a harmadik a második helyére, ... és így tovább

- Menü készítése (**CHOICE**)

  Várakozás arra, hogy a felhasználó kiválasszon egyet a választékhalmazból.

  `CHOICE [/C[:]választék] [/N] [/S] [/T[:]c,nn] [szöveg]`

  /C[:]választék - Az engedélyezhető billentyűket adja meg. Az alapértelmezés Y,N

  /N A prompt karakterlánc végén nem jeleníti meg a választékot és a kérdőjelét (?).

  /S Kis- és nagybetűk megkülönböztetése.

  /T[:]c,nn - Alapértelmezett választás nn mp után.

  szöveg - A megjelenítendő karakterlánc

  ERRORLEVEL a megnyomott billentyű választékban elfoglalt helye. Pl: Valaszt.bat

```
@echo off
cls
choice /C:I,N /T:N,5 Akarsz valami tolem?(5 mp -et kapsz,hogy eldöntsd)
If errorlevel==2 goto nemakar
If errorlevel==1 goto akar
:nemakar
echo nem akar
goto vege
:akar
echo de mennyire
:vege
pause
```

- **IF**: Feltételtől függő feldolgozás kötegelt programokban.

  `IF [NOT] ERRORLEVEL szám parancs`

  `IF [NOT] karakterlánc1==karakterlánc2 parancs`

  `IF [NOT] EXIST fájlnév parancs NOT`

  Az egyes ágak jelentése:

  `ERRORLEVEL szám` :Igaz értéket ad, ha a legutóbb futtatott program nagyobb vagy egyenlő kilépési kódot adott vissza, mint a megadott szám.

  `karakterlánc1==karakterlánc2` :Igaz értéket ad, ha megadott karakterláncok egyeznek.

  `EXIST fájlnév` :Igaz értéket ad, ha a megadott fájlnév létezik.

- **FOR**

  Ez a parancs lehetővé teszi, hogy egy Dos parancsot többször végrehajtson a rendszer anélkül, hogy a parancsot ismételten megadnánk.

  `FOR változó IN ( mondat ) DO parancs`

  változó - egyenként felveszi a mondat egyes elemeinek értékét, és ezzel az értékkel hajtódik végre a DO utáni parancs (feltéve, hogy végrehajtható)

  formája:

  `%karakter` parancsfájlon kívül

  `%%karakter` parancsfájlon belül

  mondat : legalább egy (egymástól szóközökkel elválasztott) elemből áll. Az egyes elemekkel együtt a parancsnak értelmesnek kell lennie.

  parancs : az ismétlődő Dos parancs, paramétereivel együtt. Legtöbbször a paraméterei között szerepel a változó is.

- **GOTO**

  `GOTO címke`

  Címke jelentése: A címke egy karakterekből álló jelsorozat. A címkéket a rendszer az első nyolc karakterük alapján különbözteti meg. A végrehajtás a batch programban a címke utáni első DOS paranccsal folytatódik. Ha a végrehajtás sorrendjének megfelelően jut a vezérlés egy címkére, akkor a címke hatástalan.

```
@ECHO OFF
:cimke
echo Ez az először szereplő címke
goto cimke
:cimke echo ez a másodszor szereplő címke
```

1. Batch programok paraméterezése

A Batch program paramétereire a `%1 ... %9` szimbólumokkal hivatkozhatunk. A `%0` szimbólum a parancsfájl nevét tartalmazza abban a formában, ahogyan azt indítottuk. Ezek alapján tegyük paraméterezhetővé a MENT.BAT programunkat. Ezáltal ez a program alkalmassá válhat tetszőleges fájl(-ok) floppyra mentésére.

```
@ECHO OFF REM **********MENTÉS FLOPPYRA **********

IF " %1"==" " GOTO NOPARAM

ECHO Tégy egy lemezt a floppy meghajtóba,
ECHO majd nyomj meg egy billentyűt
PAUSE > NUL

REM Másolás ************************************
COPY %1 A:\

ECHO Ellenőrzés ************************************
FC /b %1 A:\%1
GOTO VEGE

:NOPARAM
ECHO Paraméterre is szükségem van!
ECHO Helyes indítás:
ECHO Ment fájlazonosító

:Vege
ECHO Befejeztem a program futtatást.
ECHO További jó munkát kívánok!
```

A batch program helyes indítása: `MENT b*.doc`

Ekkor csak azok a fájlok lesznek floppyra mentve, amelyeknek a neve B betűvel kezdődik és a kiterjesztése DOC. Ha paraméter nélkül indítjuk, akkor hibajelzések sorát kapjuk a DOS-tól.

2. Példák

1. Az alábbi batch fájl a gyökérkönyvtár könyvtárait kimenti konyvtarak.txt néven, míg a fájljait fajlok.txt néven.

```
@echo off
dir \ | find "<DIR>" >konyvtarak.txt
dir \ | find/v "<DIR>" >fajlok.txt

rem A \ jel jelenti a gyökérkönyvtárat. A könyvtárak szűrése: a DIR szóval van-
rem nak jelezve, így tudjuk őket kiszűrni egy szűrő jellel és a find parancs
rem segítségével.
rem A szűrő működése: a szűrőjel (függőleges vonal) előtti parancs eredményét,
rem azaz a könyvtárlistát megkapja a szűrőjel utani parancs, azaz a find.
rem A fájlok szűrése: a find utáni "/v" paraméter a negáció + meg ki kellene
rem szűrni az elejére és a végére irt tájékoztató sorokat is néhány find-dal.
rem Azaz a valódi megoldás igazából az alábbi 1 hosszú sor lenne:
rem Win Me/XP-n is működik és a fajlok1.txt nevű fájlt hozza létre.

dir \ | find /v "<DIR>" | find ":" | find /v "A kötet sorozatszáma:" | find /v
"meghajtóban lévő kötet:" | find /v ":\ könyvtára" | find /v ":\ tartalma:"
>fajlok1.txt
```

2. Kérjük be a felhasználó nevét és üdvözöljük!

```
@echo off
if exist nev.txt goto end
cls
echo Hogy hívnak? (vegen F6+Enter)
copy con nev.txt > nul
echo Szia,
type nev.txt
del nev.txt
```

```
:end

rem A 2. sor óvatosság, nem kötelező beírni. Ha létezik a nev.txt fájl, kilep a
rem program és nem fut le (mert felülírna a mar létező fájlt).
rem A megoldás lényege, hogy valójában egy fájlba írjuk be a nevet. Ezt a fájlt
rem (nev.txt) a program végen letoroljuk, mert nem lesz többet rá szükségünk.
rem Az F6+Enter valójában Ctrl+Z es Enter, a fájl végé jele a DOS-ban.
rem A "copy con" végén az átirányítás a "semmibe" (nul) egy "finomság", azt
rem eredményezi, hogy nem fogja kiírni a képernyőre: "1 fajl másolása megtörtént."
```

3. Kérjünk be soronként szavakat, majd számoljuk meg mennyit írtunk be, valamint ezek közül mennyi tartalmazott 'a' betűt!

```
@echo off
if exist szavak.txt goto end
echo Kérek szavakat, soronként egyet. Vége: F6+Enter!
copy con szavak.txt >nul
echo Szavak száma:
type szavak.txt | find /c /v ""
echo Az "a"-t tartalmaz˘ sorok:
type szavak.txt | find "a"
del szavak.txt
:end

rem A find-nal kiíratjuk azon sorok számat (/c paraméter), amelyek nem tartal-
rem mazzák (/v paraméter) az üres karakterláncot (""). Vagyis megszámoljuk a
rem nem üres sorokat.
rem Ha nem írunk be semmit és rögtön F6+Entert ütünk, számtalan hibaüzenetet kapunk.
```

4. A batch fájl a paraméterben megadott „f”-re listázza ki a gyökérkönyvtár fájljait, „k”-ra a gyökérkönyvtár alkönyvtárait. Ha a paramétersor üres, adjon hibajelzést.

```
@echo off
if "%1"=="f" goto f
if "%1"=="k" goto k
goto hiba
:f
dir\|find/v "<DIR>"
goto end
:k
dir\|find "<DIR>"
goto end
:hiba
echo HIBA! Irjon a param,tersorba "f"-t vagy "k"-t!
:end

rem A fenti megoldás előnye, hogy ha a paraméter nem "f" es nem "k", hanem bar-
rem mi más, még üres is lehet, a program hibaüzenetet ad.

rem Példa a program indítására: 4pelda.bat f   vagy: 4pelda.bat k
```

5. A batch fájl írja ki a képernyőre az első paraméterben megadott fájl attribútumait, majd e fájlt tegye csak olvashatóvá (read only), és újból jelenítse meg az attribútumokat.

```
@echo off
attrib %1
attrib +r %1
attrib %1

rem A program léfutása után az adott fájl "read only" marad. A feloldáshoz az
rem alábbi parancsot kell kiadni a prompt jel után: attrib -r fajlnev
```

6. A batch fájl vizsgálja meg, hogy az 1. paraméterben megadott fájl létezik-e. Ha igen, jelenítse meg a tartalmát és alatta azt, hány sorból áll. Ha nem, írja ki: „Nem találom a … fájlt!” (a … helyébe a megadott fájlnév kerüljön)!

```
@echo off
if exist %1 goto van_fajl
echo Nem tal lom a(z) %1 f jlt!
goto end
```

```
:van_fajl
type %1
echo.
type %1|find /c /v ""
:end

rem A 2. sor vizsgálja a fájl létezését!
rem A sorok megszámolása (type+find parancs).
rem Ahhoz, hogy a sorok számának kiírása biztosan a sor elején legyen,
rem egy soremelést is beteszünk ("echo."=soremelés) (ha a kilistázott fájl végen
rem nincs Enter, a sor végére írja ki a számot).
```

7. A batch fájl a billentyűzetről kérjen be néhány szót, soronként egyet, majd írja őket ABC sorrendben a képernyőre. Utána tegye fel a kérdést: „Folytassuk?” Ha a válasz igen, írja ki fordított ABC-ben is!

```
@echo off
if exist id.txt goto end
echo Irjon be néhány szót, soronként egyet. Az utolsó szó után: Enter, F6, Enter
copy con id.txt >nul
type id.txt|sort
choice /n Folytassuk? (I vagy N)
if errorlevel 2 goto vege
sort /r id.txt
:vege
del id.txt
:end

rem Windows XP alatt nem működik!

rem A szavak beírása valójában egy fájlba történik.
rem Ezen ideiglenes falj (id.txt) létezését is ellenőrizzük ovatos-
rem sagbol (lasd 08.bat).
rem Az ABC-be kiírás kétféleképpen történhet: type+sort, illetve csak a sort
rem paranccsal (a /r fordítva rendez, lásd sort /?), mindkettőre láthatunk
rem egy-egy példát.
rem A végen letoroljuk az ideiglenesen létrehozott id.txt fájlt.

rem Windows XP alatt a choice parancs megszűnt. Helyét a SET vette át, a /P módosító
segítségével!!!
```

8. Szimuláljuk a Unix shellekben megtalálható which parancsot, azaz adjuk meg az elérési útvonal nélkül kiadható (PATH-ban szereplő) parancs pontos elérési útvonalát!

```
@echo off
if "%2"=="" goto scr
dir /s %1 > %2
goto end
:scr
@echo on
dir /s %1
:end
```

11. fejezet - Shell programozás alapjai

1. Shell változók

- Shell változók:
  1. Lekérdezésük az env paranccsal történik.
  2. Típusaik:
     - Csak a bash használja őket: a BASH_ kezdetűek
     - "Környezeti változók": TERM, HOME, EDITOR, PATH
     - Felhasználói változók: amelyeket a felhasználó hoz létre
  3. Beállításuk:

     ```
     név=[érték]
     ```

     módon történhet. Ha nem adunk meg értéket, akkor a változó értéke a null lesz.

     ```
     export név=érték
     ```

     paranccsal a gyermek-folyamatok számára is látható lesz a változó.
  4. Shell változók neve nem kezdődhet csak betűvel vagy aláhúzással.
  5. Shell változók törlése unset-tel:

     ```
     unset A
     ```
  6. Readonlyval csak olvashatóvá tehetjük a változókat:

     ```
     readonly valtozo
     ```

     A csak olvashatóság **megszüntetése nem lehetséges** sem az unset sem az újradeklarálás segítségével!

2. Shell változók behelyettesítése

Shell változók behelyettesítése: Egy példa:

```
[user@localhost ~]$ A=ma
[user@localhost ~]$ AA=holnap
[user@localhost ~]$ echo AA
holnap
[user@localhost ~]$ echo ${A}A
maA
```

- Alapértelmezett érték: `${nev:-word}`
  - Ha nev null vagy nem beállított, akkor a word lesz kiírva, ha pedig nev-nek van értéke, akkor azt írja ki.

    ```
    [user@localhost ~]$ A=
    [user@localhost ~]$ B=ertek
    [user@localhost ~]$ echo ${A-semmi}
    semmi
    [user@localhost ~]$ echo ${B-semmi}
    ertek
    ```
- Értékadás: `${nev:=word}` (pozícionális és speciális karaktereknél nem működik)
  - Ha nev értéke null vagy nem beállított, akkor megkapja word értékét. A word értéke pedig kiíródik a képernyőre.

```
[user@localhost ~]$ A=
[user@localhost ~]$ B=ertek
[user@localhost ~]$ echo ${A=$B}
ertek
[user@localhost ~]$ echo $A
ertek
[user@localhost ~]$ B=valami
[user@localhost ~]$ echo ${B=$A}
ertek
[user@localhost ~]$ echo $B
valami
```

- Hiba, ha nem beállított, vagy null: `${nev:?word}`
  - Ha a nev null, vagy nem beállított, akkor kiírja a word értékét a standard hibakimenetere, és ha a shell nem interaktív, akkor kilép.

```
[user@localhost ~]$ unset A
[user@localhost ~]$ echo ${A?Hiba}
bash: A: Hiba
```

- Alternatív érték: `${nev:+word}`
  - Ha nev null vagy nem beállított, akkor nem ír ki semmit, ha van értéke, akkor pedig a word értéket írja ki.

```
[user@localhost ~]$ A=
[user@localhost ~]$ echo ${A+ertek}
[user@localhost ~]$ A=valami
[user@localhost ~]$ echo ${A+ertek}
ertek
```

- Adott karakterek kiírássa: `${nev:eltolas:hossz}`
  - A nev eltolas számú karakterét követő hossz db karakterét kapjuk meg. Kiválóan használható egy mappa összes adott kiterjesztésű fájljának adott kiterjesztésűre való átírására.

```
[user@localhost ~]$ A=senkit
[user@localhost ~]$ echo ${A:3:2}
ki
```

- Hossz: `${#nev}`
  - a nev hosszát írja ki.

```
[user@localhost ~]$ A=string
[user@localhost ~]$ echo ${#A}
6
```

- Csere: `${nev/minta/sztring}`
  - A nev-en belül a minta karaktersorozatot a sztring karaktersorozatra cseréli.

```
[user@localhost ~]$ A=mintasztring
[user@localhost ~]$ echo ${A/tasztring/imum}
minimum
[user@localhost ~]$ echo ${A/ta/imum}
minimumsztring
```

- `${!nev@}`
  - A nev karakterlánccal kezdődő változók listája

3. Tömbök

Tömbök

- Az indexelés 0-tól indul.

- Az index csak egész érték lehet.
- Létrehozás módjai:

```
tomb[0]=valami
tomb_0=valami
```

A következő a 10. elemet beállítja ertekre, az őt megelőzőeket (ha még addig nem léteztek) nem fogja létrehozni. Csak azokat az indexeket követi nyomon, amelyek rendelkeznek értékkel.

```
tomb[10]=ertek
```

A következő két utasítás ekvivalens:

```
tomb=ujertek
tomb[0]=ujertek
```

- Egyszerre több érték is megadható:

```
tomb=(ertek1 ... ertekn)
```

Ilyenkor az első indextől kezdve sorfolytonosan tölti fel a tömbelemeket értékkel.

```
tomb=([0]=ertek1 [3]=ertek2 [2]=ertek3)
```

Csak a bash ismeri!

- A kiírás sorfolytonos:

```
echo "${tomb[*]}"
echo "${tomb[@]}"
```

A * vagy @ végigfut az egész tömbön, és kiírja az értékeket az alapértelmezett elválasztóval határolva.

- `echo ${!tomb[@]}`

jelen esetben: 0 2 3

4. Néhány shell változókban eltárolt információ

Néhány shell változókban eltárolt információ:

- IFS - internal field separator: Bemeneti elválasztó jel(ek)et tartalmazza. (Alapértéke: "\t\n ")
- OFS - output field separator: Az alapértelmezett kimeneti elválasztó jel(ek)et tartalmazza. (csak AWK használatakor!!!)
- SHLVL a bejelentkezés óta megnyitott shell-ek száma.
- RANDOM egy véletlen értéket ad vissza 0 és 32767 között
- PWD ugyanaz, mint a hasonló nevű parancs (az aktuális munkakönyvtár nevét adja meg)
- UID az aktuális felhasználó UID-jét tartalmazza
- $# a kapott paramétek száma
- $0 a futó program neve - shell szkriptek esetén elérési útvonallal együtt
- $* az összes argumentumra hivatkozik - egy értékként tekintve az összeset IFS-el elválasztva
- $@ az összes argumentumra hivatkozik - u.a., mint $*
- "$@" az összes argumentumra hivatkozik - az értékeket egyenként tekintve: "$1" "$2" ...
- $1 .. $9 pozícionális paraméterek

- $? a legutoljára lefutott program visszatérési értéke (hiba kódja)
- $$ a saját PID
- $! az utolsó háttérbe indított folyamat azonosítója

5. Shell szkriptek létrehozása

Shell szkriptek létrehozása:

- szerkesztés
  - szövegszerkesztővel
  - cat-tel

    ```
    cat > shell.sh
    ```

    `^D`-vel tudunk kilépni a cat-ből.

- `chmod u+x filenév`

  paranccsal adhatunk, vagy csak olvasási joggal az

  ```
  sh filenév
  ```

  utasítással dolgoztathatjuk fel a szkriptet. *Megjegyzés: cat szkript | sh szerkezet is futtat egyszerre.*

- A shell szkript első sorában adhatjuk meg azt, hogy milyen parancsfaldolgozóval akarjuk feldolgoztatni. pl.:

  ```
  #!/bin/bash
  ```

Ha szeretnénk látni, hogy az egyes utasítások végrehajtása után mi történik, akkor használjuk a '-x' opciót, amely így a PS4 prompt után mindig megjeleníti az aktuális eredményt.

```
#!/bin/bash -x
```

Megjegyzések is megadhatóak, erre a '#' karaktert használjuk. A sor további részét nem dolgozza fel

```
#!/bin/bash
# ez megjegyzés sor lesz
echo "Hello"
```

5.1. Példák

- futtathatóvá teszi a paraméterként kapott fájlokat:

  ```
  chmod u+x $*
  ```

- A futtató felhasználó teljes nevét írja ki:

  ```
  grep $UID /etc/passwd | cut -d : -f 5
  ```

  A minta egészen pontosan :$UID:

- A paraméterként kapott UID-hez tartozó felhasználó teljes nevét írja ki:

  ```
  grep $1 /etc/passwd | cut -d : -f 5
  ```

  Megjegyzés: Ha paramétert vár a szkriptünk, akkor teszteléskor adjunk neki megfelelő számú paramétert. Legalább esélye legyen a helyes lefutásra.

- A bemeneti mezőelválasztó átdefiniálása:

  ```
  myIFS=$IFS
  IFS=":"
  ```

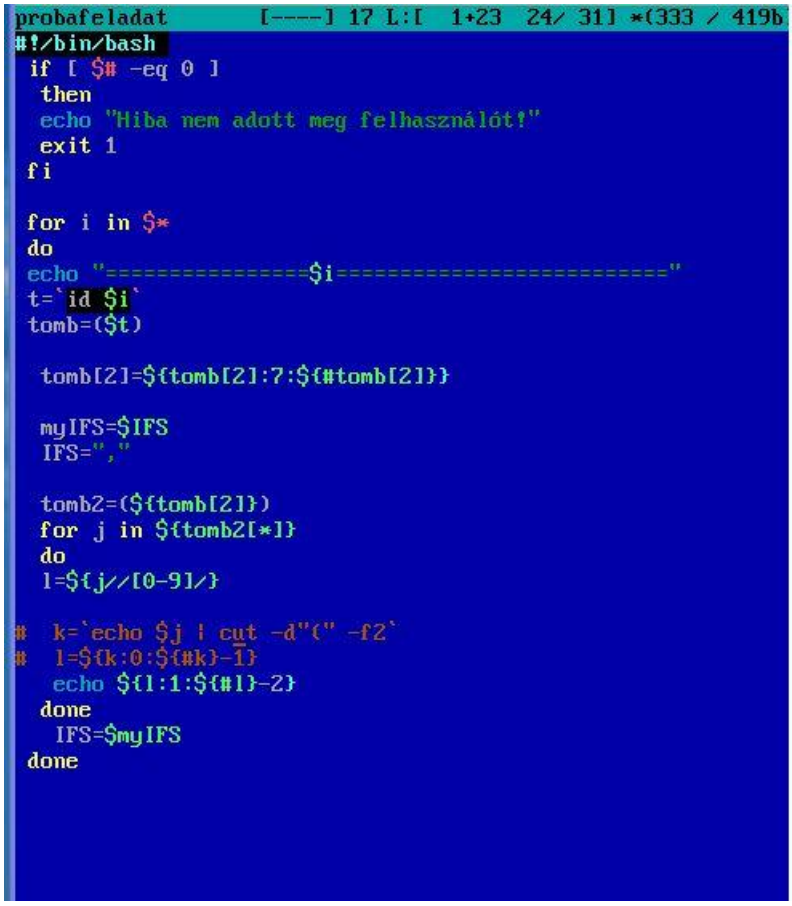
```
sztring="elso:2:3madik"
tomb=($sztring)
echo $tomb
echo ${tomb[2]}
echo "Meg ervenyes a ':' , mint IFS..."
echo $tomb-${tomb[1]}:${tomb[2]}
echo "Most elnyomjuk, mert egy karakterláncba fogjuk össze:"
echo "$tomb-${tomb[1]}:${tomb[2]}"
IFS=$myIFS
```

- Komplex példa:

```
if [ $# -eq 0 ]
  then echo "Usage: `basename $0` username"
  exit 1
else
# a /etc/group file tartalma:
# <soreleje>csoportnév:x:tag1,tag2,tag3<sorvége>
# egrep-pel lehet szűrni
# lehetséges előfordulása a felhasználónévnek:
# soreleje van előtte és : van mögötte
# a felsorolásban van
# ekkor előtte vagy : van (első tag) vagy , van
# mögötte pedig vagy , van (nem utolsó tag) vagy sorvége
# az egrep szerkezete:
# (...|...) vagy-vagy
# [xy] lehetséges karakterek felsorolása
# $1 a paraméter
# ^ a soreleje, $ a sorvége
# miután a [...] jelek között a meta-karakterek elvesztik jelentésüket, (...) lesz
szükség a $ miatt!

# a grep eredménye azok és csak azok a sorok, ahol pontosan $1 szerepel
# ebből kell az első oszlop, amire a cut-tal lehet szűrni
 cat /etc/group | egrep "(^$1:|[,:]$1([,]|$))" | cut -d":" -f1
fi
```

- Ugyan ez kicsit másképp:



```
probafeladat       [----] 17 L:[  1+23  24/ 31] *(333 / 419b
#!/bin/bash
 if [ $# -eq 0 ]
  then
  echo "Hiba nem adott meg felhasználót!"
  exit 1
 fi

 for i in $*
 do
 echo "================$i=========================="
 t=`id $i`
 tomb=($t)

  tomb[2]=${tomb[2]:7:${#tomb[2]}}

  myIFS=$IFS
  IFS=","

  tomb2=(${tomb[2]})
  for j in ${tomb2[*]}
  do
  l=${j//[0-9]/}

#  k=`echo $j | cut -d"(" -f2`
#  l=${k:0:${#k}-1}
   echo ${l:1:${#l}-2}
  done
   IFS=$myIFS
 done
```

12. fejezet - Shell szkript kilépési kódja

```
exit [n]
```

Ha megadunk ennek az utasításnak egy számot az n helyén, akkor azzal a számmal, mint visszatérési értékkel kilép a szkriptből. Ha nem, akkor a visszatérési érték az utoljára lefuttatott program visszatérési értékével tér vissza.

13. fejezet - Shell programozás - alapvető utasítások

1. Az " és az ' közti különbség

Az " és az ' közti különbség:

- "-ek közti szövegben a változó-behelyettesítés végrehajtódik. pl echo után
- '-ok közti szövegben nem hajtódik végre a változó-behelyettesítés.

2. Feltételes elágaztatás

1. Az if:

   -
   ```
   if feltétel
   then utasításlista
   [elif feltétel
     then utasításlista] ...
   [else utasításlista]
   fi
   ```

   - vagy egy sorba írva:

   ```
   if feltétel; then utasítások; [ elif utasítások; then utasítások; ] ... [else
   utasítások;] fi
   ```

   - Működése: kiértékelődik a feltételként megadott kifejezés vagy parancs. Ha a visszatérési értéke 0, akkor lefut a then ágba írt utasítássorozat, ha nem 0 akkor ha van else ág, akkor az abba írt utasítások futnak le. Csak akkor lépünk ki az ifből, amikor elérjük a fi-t. Nincs csellengő else, mert az if fi keret egyértelműen megadja, hogy mire vonatkozik az else.
   - Egy példa az if-re:

   ```
   if date | cut -d ' ' -f 2 | grep "Oct" ; then echo "Október van." ; else echo "Nem
   Október van." ; fi
   ```

   Ha október van akkor kiírja, hogy október van, ha nem, akkor azt írja ki, hogy nem október van.

   - Kicsit bonyolultabban, ami már paraméterként várja a keresett hónapot:

   ```
   # Aktuális hónap
   honap=`date | cut -d ' ' -f 2`
   echo $honap | grep $1 > /dev/null
   if [ $? -eq 0 ]
   then echo "Igen, most $1 hónap van."
   else echo "Nem, most $honap van!"
   fi
   ```

   - Komplex példák:

   ```
   if echo ${1?"Hiba, nincs parameter!"} >/dev/null
   then
       if who | grep $1 >/dev/null
       then echo "Dolgozik:"
            ps -u $1
       else
            echo "Nem dolgozik."
       fi
   fi
   ```

Ha nem kap paramétert, akkor a hiba, nincs paraméter szöveg íródik ki, majd leáll a szkript. Ha kap paramétert, akkor megnézi, hogy a paraméter értékével azonos nevű felhasználó be van-e jelentkezve. Ha be van jelentkezve, akkor kiírja, hogy dolgozik, illetve kilistázza az általa futtatott folyamatokat. Ellenkező esetben, a nem dolgozik feliratot kapjuk.

```
if [ $(date | cut -f 4 -d " " | cut -c 1-2) -lt 10 ]
then echo "Jo reggelt!"
elif [ $(date | cut -f 4 -d " " | cut -c 1-2) -lt 19 ]
then echo "Jo napot!"
else echo "Jo estet!"
fi
```

A szkript kiírja az aktuális napszaknak megfelelő üdvözlő üzenetet, amennyiben a date kimenete *"Mon Nov 12 16:20:45 CET 2007"* alakú.

2. A `test` utasítás:

Mondhatni multifunkciós, mivel kapcsolóinak köszönhetően több vizsgálatra is használható, pl:

- -b fájl blokk eszköz az adott fájl
- -c fájl karakteres eszköz a fájl
- -d fájl könyvtár a fájl
- -e fájl létezik a fájl
- -f fájl hagyományos file a fájl
- -h fájl szimbolikus link
- -k fájl sticky bit aktív
- -r fájl olvasható
- -w fájl írható
- -x fájl futtatható
- f1 -nt f2 f1 újabb, mint f2
- f1 -ot f2 f1 régebbi, mint f2
- f1 -ef f2 f1 és f2 inode száma megegyezik
- -z string a string hossza 0
- S1=S2 S1 és S2 megegyezik
- S1!=S2 S1 nem egyenlő S2-vel
- kif1 -a kif2 logikai művelet: kif1 és kif2
- kif1 -o kif2 logikai művelet: kif1 vagy kif2
- k1 -eq k2 k1 egyenlő k2
- k1 -lt k2 k1 kisebb k2
- k1 -gt k2 k1 nagyobb k2
- k1 -ne k2 k1 nem egyenlő k2
- ! kif a kifejezés tagadása

```
test kifejezés
```

ugyanaz, mint a

```
[ kifejezés ]
```

mert a `[` egy link a `test`-re.

- Példa a használatára a komplex példát átírva:

```
if [ $# -ne 1 ]
then
    echo "A skript hasznalata `basename $0` parameter"
    exit
else
    if who | grep $1 >/dev/null
    then echo "Dolgozik:"
         ps -u $1
    else
         echo "Nem dolgozik."
    fi
fi
```

- Másik példa:

```
if [ -z "$1" ]
then
    echo "A skript hasznalata `basename $0` parameter"
    exit
else
    sor=`cat /etc/passwd | grep $1`
    if [ -z "$sor" ]
    then
        echo "Nincs $1 nevu felhasznalo!"
    else
        nev=`echo $sor | cut -f5 -d:`
        echo $nev
    fi
fi
```

3. Helyettesítő karakterek feloldása

A shell mielőtt elindítaná a parancs által hivatkozott programot, megvizsgálja, hogy a paramétereket nem kell-e átalakítania. A paraméter-sztringet az OFS mentén feldarabolja, majd ezekben a helyettesítő karakterek előfordulását keresi. Amennyiben szerepel benne, akkor azokat is feloldja.

- Azaz ha a programunknak *.txt kifejezést adunk át, akkor megpróbálja behelyettesíteni a megfelelő fájlneveket, és külön-külön paraméterként átadni azokat. Ha nem talál a mintára illeszkedő fájlokat, akkor a *.txt szöveget adja át paraméterként.

Ha ezt el akarjuk kerülni, akkor kell '*.txt' módon le kell védeni a feloldástól.

Természetesen paraméterként átadhatóak shell változók is, hivatkozni normál módon - $ jellel - kell. Az ' '-jel szintén kivédhetjük a kiértékelésüket.

4. Többirányú elágaztatás

A case

```
case szó in
minta1) parancs1 ; ... utolsó_parancs ;;
mintax|mintay) p1; ... pn ;;
[Mm]?nta) p1 ;;
...
[*) pd ;;]
esac
```

A működését tekintve, az első olyan mintára, ami ráilleszthető a kapott szóra, lefuttatja a megadott parancsokat, majd kilép a case-ből. Alap esetben nincs default ág, de egy kis trükkel megoldható, hogy legyen, mivel ha beírjuk a *-ot a minták közé, akkor biztosan illeszkedni fog mindenre. Éppen emiatt **a * mindig az utolsó minta legyen, ha van!** Használhatjuk a [Mm] illetve ? jelöléseket is, ahol az [Mm] halmaz jelentése, hogy M vagy m karakter áll az adott pozíción, illetve a ? egy karaktert helyettesíthet. A | jel pedig a lehetséges minták elválasztására szolgál.

Egy példa:

```
case $# in
3) ;;
*) echo "Hiba: Nem megfelelo szamu operatort kaptam." ; exit 1;;
esac
case $2 in
JAN|jan|Jan|01|1) if [ ${#3} -gt 1 ] ; then echo "20$1-01-$3"
                                       else echo "20$1-01-0$3" ; fi ;;
[Ff][Ee][Bb]|02|2) if [ ${#3} -gt 1 ] ; then echo "20$1-02-$3"
                                         else echo "20$1-02-0$3" ; fi ;;
[Mm][Aa][Rr]|03|3) if [ ${#3} -gt 1 ] ; then echo "20$1-03-$3"
                                         else echo "20$1-03-0$3" ; fi ;;
[Aa[Pp][Rr]|04|4) if [ ${#3} -gt 1 ] ; then echo "20$1-04-$3"
                                          else echo "20$1-04-0$3" ; fi ;;
[Mm][Aa][Jj]|05|5) if [ ${#3} -gt 1 ] ; then echo "20$1-05-$3"
                                         else echo "20$1-05-0$3" ; fi ;;
[Jj][Uu][Nn]|06|6) if [ ${#3} -gt 1 ] ; then echo "20$1-06-$3"
                                         else echo "20$1-06-0$3" ; fi ;;
[Jj][Uu][Ll]|07|7) if [ ${#3} -gt 1 ] ; then echo "20$1-07-$3"
                                         else echo "20$1-07-0$3" ; fi ;;
[Aa][Uu][Gg]|08|8) if [ ${#3} -gt 1 ] ; then echo "20$1-08-$3"
                                         else echo "20$1-08-0$3" ; fi ;;
[Ss][Zz][Ee]|09|9) if [ ${#3} -gt 1 ] ; then echo "20$1-09-$3"
                                         else echo "20$1-09-0$3" ; fi ;;
[Oo][Kk][Tt]|10) if [ ${#3} -gt 1 ] ; then echo "20$1-10-$3"
                                       else echo "20$1-10-0$3" ; fi ;;
[Nn][Oo][Vv]|11) if [ ${#3} -gt 1 ] ; then echo "20$1-11-$3"
                                       else echo "20$1-11-0$3" ; fi ;;
[Dd][Ee][Cc]|12) if [ ${#3} -gt 1 ] ; then echo "20$1-12-$3"
                                       else echo "20$1-12-0$3" ; fi ;;
*) echo "Hiba: Nem sikerult feldolgozni a kapott datumot, ismeretlen formatum." ;;
esac
```

A skript a paraméterként kapott '07 JAN 8', '07 Jan 8', '07 jan 8', '07 01 8', '07 1 8', '07 JAN 08', '07 Jan 08', '07 jan 08', '07 01 08', '07 1 08' formátumú dátumot yyyy-mm-dd alakúra konvertálja.

5. Mintaillesztés

- A c karakter általában a c-re illeszkedik, kivéve, ha **+ * . \ [ ] ^ $** -ről van szó.
  - . egy tetszőleges karakterre
  - ^ a sor elejére
  - $ a sor végére illeszkedik
- Egy karakter kiválasztása kartersorozatból:
  - [...] a ... helyén álló karakterek közül pontosan egy fordulhat elő az adott pozíción
  - [^...] a ... karakterek nem fordulhatnak elő az adott pozíción
  - [c1-c2] a c1-c2 tartomány egy eleme fordul elő az adott pozíción pl.: [0-9] [a-z] [B-Q]
- Jelentést módosító jelek: *** + ( ) { }**
  - * az ismétlő karakter, a * előtt álló kifejezés akárhány előfordulására illeszkedik
  - + az előtte álló kifejezés előfordulásaira illeszkedik, de az előfordulások száma legalább 1 kell legyen.

- () csoportosítás a zárójelek közti kifejezés egy egység lesz, vonatkozhat rá * vagy + is
- {n} {n,} {n,m} ezek segítségével az előfordulások száma adható meg:
  - {n} az előtte álló kifejezés **pontosan n-szer** forduljon elő.
  - {n,} az előtte álló kifejezés **minimum n-szer** forduljon elő.
  - {n,m} az előtte álló kifejezés **minimum n-szer és maximum m-szer** forduljon elő.

- A speciális karakterek használata:
  - grep-nél, sed-nél ha valamit speciális karakterként akarok tekinteni, akkor \-t kell elé írni, de awk-nál nem.
  - Shell szkriptben a * az aktuális könyvtár tartalmára vonatkozik, a | ugyanolyan jelentéssel bír, mint amit a case-nél is tanultunk.

- Példa:

```
'\(^\| \)\((*\(36\|06\))*[- ]1[- ]\)*[0-9]\([- ]*[0-9]\{3\}\)\{2\}\( \|$\)'
```

Ez a budapesti telefonszámokra illeszkedő minta.

```
'\(^\| \)
```

A sor elején vagy szóközzel kezdődik,

```
\((*\(36\|06\))*[- ]1[- ]\)*
```

Ha van körzetszám, akkor azt (36) vagy (06) előzi meg, 1-es a körzetszám, és az 1-es előtt ill. után szóköz, vagy - állhat,

```
[0-9]\([- ]*[0-9]\{3\}\)\{2\}
```

1 számjegy, szóköz vagy kötőjel és 2 db 3 számjegyből álló blokk szóközzel, vagy kötőjellel elválasztva,

```
\( \|$\)
```

szóköz, vagy sorvége van a végén.

6. Ciklus szervezés

6.1. For ciklus

A for ciklus:

```
for valtozo in szavak_lista
do
    utasítások;
done
```

Egy sorba írva:

```
for valtozo in szavak_lista ; do utasítások; ; done
```

- A változó a lista minden elemének felveszi az értékét, és végrehajtja ezekre az értékekre az utasításokat.
  - *Amennyiben nem adunk meg listát, akkor a ciklus a pozicionális paraméterek listáján fog végigmenni! [ilyenkor az* in *kulcsszó sem kell]*
  - A lista lehet a *, *.txt, files[1-4] stílusban is megadva, ha az aktuális munkakönyvtár fájljait akarjuk használni.

pl.: Könyvtár listázása:

```
for i in *
do echo $i
done
```

Ha ki akarjuk írni a fájlok típusát is, akkor így kell megírnunk a skriptet:

```
for i in *
do
 if [ -f $i ]
   then echo "Hagyomanyos file: $i";
 elif [ -d $i ]
   then echo "Konyvtar: $i";
 else echo "Egyeb: $i"
 fi
done
```

Ha tudjuk, hogy hányszor akarjuk lefuttatni a ciklusmagot (pl. 100-szer) akkor gépelhetünk egy listát a for-nak 1-től 100-ig a számokat az alapértelmezett elválasztó karakterrel elválasztva, vagy lustábbaknak marad a `seq 1 100` parancs-behelyettesítés listakénti megadása. Fontos, hogy **parancsbehelyettesítést végezzünk**, mert egyébként a parancsot egyszerű szólistának tekinti! Például:

```
for i in `seq 1 10`
do echo -n $i
done
```

Így kiírjuk a számokat egymás mellé 1-től 10-ig. A -n kapcsoló az echo mellett általában (de ez nem minden rendszeren működik!) megakadályozza, hogy az echo letörje a sort (nem lesz újsorjel a sor végén). illetve egy másik példa

```
files=$(cat filelist)
for file in $files
 do
     echo $file;
 done
```

Ha C stílusú programozásra szeretnénk használni a for-t akkor azt is megtehetjük:

```
for ((i=1; i<11; i++))
do echo $i
done
```

Csupán arra kell odafigyelnünk, hogy kettős zárójelbe kerüljön a "ciklus feje", illetve, hogy nem hagyhatjuk ki a do, done keretet sem.

Példák a pozicionális paraméterek használatára:

```
echo Argumentumok száma: $#

for arg in $*
do
   echo $arg
done
```

Lefuttatva:

```
$ args a b c

Argumentumok száma: 3
a
b
c

$ args 'a b' c

Argumentumok száma: 2
a
b
c
$
```

Különbség, ha "$@" módon hivatkozunk:

```
echo Argumentumok száma: $#

for arg in "$@"
do
   echo $arg
done
```

Lefuttatva:

```
$ args a b c

Argumentumok száma: 3
a
b
c

$ args 'a b' c

Argumentumok száma: 2
a b
c

$ args

Argumentumok száma: 0
$
```

6.2. While ciklus

A while ciklus:

```
while parancs
do utasítások;
done
```

A while utáni parancs ha nullával tér vissza, akkor lefut a ciklusmag, majd megint "kiértékelődik" a parancs, míg egyszer "hamis" nem lesz.

```
while true
#végtelen ciklus, a true helyettesíthető a : utasítással is! Kilépés CTRL-C
do
  date;
  sleep 3;
done
```

A szkript futtatása egy végtelen ciklus keretében az idők végezetéig (vagy míg Ctrl+C-t nem nyomunk) kiírja a dátumot, majd vár 3 másodpercet, és újra. Másik példa, amely addig fut amíg a paraméterként megnevezett felhasználó ki nem lép:

```
while who|grep $1 >/dev/null
 do
  echo "Dolgozik meg $1..."
  sleep 3
 done
```

A fentebbi for ciklusos pozicionális paramétereket feldolgozó szkript while ciklussal:

```
while [ "$#" -ne 0 ]
do
      echo "$1"
      shift
done
```

6.3. Until ciklus

Az until:

```
until parancs
do utasítások;
done
```

Működése hasonló a while-hoz, csak itt a parancs nem nullával való visszatérése esetén fut le a mag.

```
until false
do
  date;
  sleep 3;
done
```

Az előzőleg már bemutatott végtelen ciklusunk until-lal készített párja.

6.4. Kitörés a ciklusokból

A ciklusokat azonnal is befejeztethetjük a `break` utasítással. A break mindig a legbelső ciklusból lép ki. A break opcionálisan elfogad egy argumentumot, ekkor a számmal megadott darabszámú ciklusból lép ki

```
while true
do
        cmd=$(getcmd)

        if [ "$cmd" = quit ]
        then
                break
        else
                processcmd "$cmd"
        fi
done
```

vagy

```
for file
do
       ...
       while [ "$count" -lt 10 ]
       do
              ...
              if [ -n "$error" ]
              then
                     break 2
#mind a for és mind a while ciklus is befejeződik!
              fi
              ...
       done
       ...
done
```

A ciklus hátralévő részének átugrása: `continue`. A `break`-hez hasonlóan elfogad egy opcionális paramétert(szám), ami a legbelső n db ciklus átugrását eredményezi.

```
for file
do
       if [ ! -e "$file" ]
       then
              echo "$file not found!"
              continue
       fi
       #
       # Process the file
       #
       ...
done
```

A szkript az argumentumként kapott fájlneveket megvizsgálja, hogy létezik-e ilyen nevű fájl, és ha igen, akkor kiírja a sorait ABC sorrendbe rendezve.

```
while test $# -gt 0
 do
  if test !-f $1
```

```
   then echo "Hiba"
   shift
   continue
  fi
  sort $1
  shift
 done
```

6.5. Érdekességek

Ciklus háttérben futtatása:

```
$ for file in memo[1-4]
> do
>         run $file
> done &                      Send it to the background
[1] 9932
```

Ciklus kimenetének átirányítása:

```
$ for i in 1 2 3 4
> do
>         echo $i
> done > loopout              Redirect loop's output to loopout
```

A hibakimenet is leválasztható:

```
while [ "$endofdata" -ne TRUE ]
do
          ...
done 2> errors
```

Átirányítás csővezetékbe:

```
$ for i in 1 2 3 4
> do
>         echo $i
> done | wc -l
       4
$
```

6.6. Összetettebb példák

Megvizsgálja a szkript, hogy kapott-e paraméterként valamit, és ha nem, akkor kiírja, hogy hogyan kell használni. Ellenkező esetben ha a 2. paraméter null, akkor 5 másodpercenként megnézi, hogy be van-e jelentkezve az első paraméterben kapott felhasználó, és ha igen, akkor küld egy levelet a megadott címre. Ha megadtunk valamit a 2. paraméterben, akkor a megadott ideig vár, nem 5 másodpercig.

```
if [ -z $1 ]
  then
    echo " Nem adtal meg Júzernevet!";
    exit 1;
fi
IDO = 5;
if [!-z $2 ]
  then
    IDO=$2;
fi
OK=1;
while [ $OK -eq 1 ]
  do
    if who | cut -d "-f1" | grep $1
    echo "Belépett" | mail -s Alert! hallgato;
  else
  sleep $IDO;
fi
done
```

Ugyanez tömörebben:

```
if [ $# -eq 0 ]
then echo "Usage: `basename $0` username [delay]"
else
  IDO=${2:-5};
  until who | grep $1 >/dev/null
    do sleep $IDO
    done
  echo "$1 belepett" | mail -s Alert x@y.hu
fi
```

Ez a szkript egyesével olvas be karaktereket, amíg nem kap egy '.' karaktert, amire kilép. A beolvasott karaktereket kiértékeli, hogy betűk, számok, vagy speciális karakterek, majd az eredményt ki is írja.

```
while true
do
read -s -n1 -p">" be
case $be in
[a-zA-Z]) echo "$be - betu" ;;
[0-9]) echo "$be - szamjegy" ;;
[\'\|\$\ß\#\+\-\*\/\\\=\,\!\"\?\:]) echo "$be - specialis vagy irasjel" ;;
\.) echo "GoodBye!" ; exit 0 ;;
*) echo egyeb ;;
esac
done
```

7. Aritmetikai műveletek elvégzése

7.1. expr

expr: Az expr parancs szintaxisa:

•

```
expr arg1 operator arg2 ...
```

Az expr logikai vagy aritmetikai kifejezésekként kezeli az argumentumokat és kiértékeli azokat. Az eredményt a standard outputra küldi. Az egyes argumentumokat szóközökkel kell elválasztani, a metakaraktereket semlegesíteni kell. A kifejezések argumentumai lehetnek sztringek vagy egész(!) számok. Kilépési kódjai (EXIT STATUS):

- 0 If the expression is neither NULL nor 0.
- 1 If the expression is either NULL or 0.
- 2 For invalid expressions.
- >2 An error occurred.
- A használható függvények:
  - + összeadás
  - - kivonás
  - * szorzás
  - / osztás
  - % maradékos osztás
  - Példa:

```
$ cat >szamol
  if test $# -ne 2
  then
    echo A programnak pontosan 2 argumentuma kell legyen.
```

```
    exit 1
  fi
  SUM=`expr $1 + $2`
  DIFF=`expr $1 - $2`
  PROD=`expr $1 \* $2`
  echo \$1= $1
  echo \$2= $2
  echo $1 + $2 = $SUM
  echo $1 - $2 = $DIFF
  echo $1 \* $2 = $PROD
CTRL-D
```

- Példa az expr használatára:

```
i=0
while expr $i \< 10 >/dev/null
# fontos \ a < előtt !!!
 do
  echo $i
  let i++;
# rövidebben: echo $((i++))
# vagy i=`expr $i+1`
 done
```

Bonyolultabb feladatok megoldására is alkalmas.

- Összehasonlító operátorok:

```
arg1 = arg2     Displays a 1 if arg1 and arg2 are equal. For example,
$ NUM=5
$ PREV=6
$ expr $NUM = $PREV     # NUM and PREV are not equal so
0                       # 0 is displayed.
arg1 \> arg2    Displays a 1 if arg1 is greater than arg2.
$ expr dog \> cat       # Since d has a higher ASCII
1                       # value than c a 1 is displayed.
arg1 \>= arg2   Displays a 1 if arg1 is greater than or equal to arg2.
arg1 \< arg2    Displays a 1 if arg1 is less than arg2.
arg1 \<= arg2   Displays a 1 if arg1 is less than or equal to arg2.
arg1 !> arg2    Displays a 1 if arg1 is not equal to arg2.
```

- Feltételes operátorok:
  - arg1 \& arg2 Returns the value of arg1 if arg1 and arg2 are not 0 or NUL; otherwise, 0 is returned. Arg1 can be another expression.

```
$ VAR=""        # Set VAR to NUL
$ PRT=""        # Set PRT to NUL
$ expr $VAR \& $PRT     # Return 0, arg1 and arg2
0       # are NUL
$ VAR="hplj"    # Set VAR to hplj printer
$ PRT="hpij"    # Set PRT to hpij printer
$ expr $VAR \& $PRT     # VAR is set, return its
hplj    # value
```

  - arg1 \| arg2 Returns the value of arg1 if it is not 0 or NUL; otherwise, the value of arg2 is returned. Arg1 may consist of another expression (arg1 operand arg2).

```
$ VAR=""        # Set variable VAR to NUL
$ expr $VAR \| undefined        # If VAR is NUL return
undefined       # the undefined string
```

- Sztring mintaillesztő operátorok:
  - arg1:arg2

```
$ expr string : str     # First 3 characters match
3
$ expr string : ing     # Last 3 match but comparison
0                       # must start at beginning of arg1
```

```
$ expr string : strg    # Arg2 must match arg1 in its entirety
0
$ expr string : '.*'    # .* is a regular expression that matches any number of any
characters
6
$ expr string : '.*i'   # .*i matches any set of characters ending with i
4

 You may find times when you need to return the matching portion of the string instead
of the number of matching characters. You can use the \(...\) notation to perform this
function. For example:
$ expr string : '\(.*\)'        # Returns all chars in arg1 string
$ expr string : '..\(..\)'      # Skips first 2 chars of arg1, returns next 2
ri
$ expr string : '\(...\)'       # Returns first 3 chars of arg1
str
$ expr string : '.*\(...\)'     # Returns last 3 chars of arg1
ing
$ expr string : '..\(.*\)..'    Returns center of arg1 first and last 2 chars removed
ri
$ expr string : '\(stx\)'       # Returns nothing
$ expr string : 'st\(.*\)'      # Return all of arg1 after skipping st
ring
```

- index string character-list A karakterlista legkorábban előforduló elemének a pozíciója.

```
$expr hello lo
3
```

- length string A sztring hossza
- substr string integer-1 integer-2

```
$expr substr hello 2 2
el
```

7.2. bc

bc:

- Komoly műveleteket végeztethetünk el vele. Előnye, hogy állíthatjuk a pontosságát, viszont fontos, hogy **standard inputról olvas**, illetve **standard outputra ír**.
- Példa a használatára:

```
[user@localhost ~]$ echo 5/2 | bc
2
```

Az eredmény 2 lesz, mert a pontosságot, ha nem állítjuk be, vagy negatív értéket adunk meg neki, akkor a default 0 pontossággal dolgozik.

```
[user@localhost ~]$ echo "scale=2; 5/2" | bc
2.50
```

A scale=2; azt eredményezi, hogy 2 tizedesjegy pontossággal kapjuk meg az eredményt.

- További műveletek

```
#square root
$ echo 'scale=30;sqrt(2)' | bc
1.414213562373095048801688724209

#power
$ echo '6^6' | bc
46656

#zárójelek
$ echo '(6^6)^6' | bc

#obase és ibase
```

```
$ echo 'obase=16;255' | bc
FF

$ echo 'obase=2;12' | bc
1100

$ echo 'ibase=2;obase=A;10' | bc
2
#Vigyázat
$ echo 'ibase=2;obase=10;10' | bc
10
# mert az obase-t már az ibase számrendszerében értelmezi!

$ echo 'ibase=2;obase=A;10000001' | bc
129

$ echo 'ibase=16;obase=A;FF' | bc
255
```

- A bc használata szkriptben:

```
valtozo=`echo "scale=2; 5/2" | bc`
```

- A bc parancssorban:

```
$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.

scale=5
57/43
1.32558

quit
```

- A bc fájlok használatával:

```
scale=2

/* C-style comments
   are allowed, as are spaces */

print "\nConvert Fahrenheit to Celsius\n\n"
print "Temperature in Fahrenheit: " ; fah = read()
print "\n"
print "Equivalent Temperature in Celsius is: "
(fah - 32.0) * 5.0 / 9.0
quit

$ bc -q filename

Convert Fahrenheit to Celsius
Temperature in Fahrenheit: 61
Equivalent Temperature in Celsius is: 16.11
```

7.3. $((kifejezés))

belső parancs, a kifejezés helyére primitív műveletek kerülhetnek.

```
i=0
while [ $i -lt 10 ]
do
 echo $i
 : $((i++))
done
```

7.4. let

let:

- Példa:

```
let sorszam+=1
```

Megnöveli 1-gyel a sorszam értékét.

8. A HERE Document

HERE document:

```
parancs << SZTRING
[input]...
SZTRING
```

Addig olvas be adatokat, amíg a SZTRING kifejezést teljes egészében egy új sorban meg nem találja. pl.:

```
cat >file.txt <<SOROK
első sor
2. sor
További Sorok
még mindig SOROK
SOROK
```

ekkor a file.txt tartalma:

```
első sor
2. sor
További Sorok
még mindig SOROK
```

9. Beolvasás

A read:

```
read változó
```

változó-ba olvas be egy sort (a következő enterig).

- -d elválasztó jel (a terminátor karakter megváltoztatása, hogy csak addig olvasson)
- -s (silent mode) nem echozza vissza a beolvasott karaktereket
- -p"prompt" (nincs szóköz!) a sor elejére kiír egy prompt-ot, és utána olvas be.
- -n n db karaktert olvas be, és azután visszatér

Példa:

```
read -p"Tortenjen valami?[Igen]/Nem" be
: ${be:=Igen}
case $be in
[Ii][Gg][Ee][Nn]) echo "Tortenik valami..." ;;
[Nn][Ee][Mm]) echo "Majd maskor..." ;;
*) echo "Nem ertem." ;;
esac
read -p"Kilepeshez nyomj valamit..." -s -n1
```

A szkript kiírja a "Tortenjen valami?[Igen]/Nem" kérdést, majd az erre adott választ kiértékeli. Ha Igen-t gépelünk be (nem case sensitive) vagy, ha csak entert ütünk, akkor a "Tortenik valami..." sztringet kapjuk, ha pedig nem-et gépelünk, akkor a "Majd maskor..." sztringet kapjuk.Ettől eltérő esetekben a "Nem ertem." szöveg lesz a válasz. A :${be:=Igen} sor elején a : azért kell, mert az utána írt kifejezést a bash lenyeli, nem próbálja meg az értékét értelmezni. Ha a szkript kilépett a case-ből, akkor kiírja a "Kilepeshez nyomj valamit..." feliratot, majd vár pontosan egy billentyű leütéséig. A leütött billentyű nem fog megjelenni a képernyőn a -s miatt.

Kiegészítés: Ha a read után több változó nevét adjuk meg, akkor a beolvasott sort az IFS értéke alapján feldarabolja és elhelyezi a változókban. Ha több "szelet" áll elő mint változó, akkor az utolsó változóba helyezi a "maradékot", ami már nem fért az előző változókba.

```
read a b c
ez itt most mi is lesz?
echo "$a -=- $b -=- $c"
ez -=- itt -=- most mi is lesz?
```

10. Példák:

1. Szűrő írása:

   ```
   • while read LINE
     do
      utasítások;
     done <filenév
   ```

   Lesz a szkript váza, ahol addig olvas a filenév nevű fájlból, amíg el nem éri az EOF-ot (a file végét).

   ```
   • while read LINE
     do
      case $LINE in
      *$1*) echo $LINE ;;
      esac
     done </etc/passwd
   ```

   A /etc/passwd-ből kiolvassa a sorokat egyesével, majd összeveti őket a *$1* mintával, majd ha azt találja, hogy az első argumentum egyezik az aktuális sorral, akkor kiírja a sort az outputra.

2. Különböző működés a hívási névtől függően:

   - pl.: Megtehetjük, hogy a tar ha maketar néven hívjuk a szkriptünket, akkor a kapott file-t vagy könyvtárat összetarolja, ha pedig listtar néven, akkor kilistázza a kapott tar file tartalmát.

     ```
     case `basename $0` in
     maketar) TARARG=" -cvf $1.tar $1" ;;
     listtar) TARARG=" -tvf $1" ;;
     *) echo "Hiba" ; exit 1 ;;
     esac
     tar $TARARG
     ```

3. `Program opció paraméter...` típusú utasítások kezelése:

   - pl.:

     ```
     if [ $# -e 2 ]
     then case `basename $0` in
     maketar) TARARG=" -cvf $1.tar $1" ;;
     listtar) TARARG=" -tvf $1" ;;
     *) echo "Hiba" ; exit 1 ;;
     esac
     tar $TARARG
     fi
     ```

4. Maximum 3 opció 1 paraméter esetén:

   ```
   sorszam=0
   for i in $*
   do
     let sorszam=sorszam+1
     case $i in
     -a) a opció utasításai ;;
     -b) b opció utasításai ;;
     -c) c opció utasításai ;;
     *) if [ $# -ne $sorszam ]
        then echo "Hibas parameter..."
        exit 1
   ```

```
    fi ;;
esac
```

14. fejezet - Shell programozás - függvények és érdekességek

1. Függvények

1.1. Függvények definiálása

Függvények készítésére két módunk van:

- ```
  function fv_nev{
   utasítások
   ...
  }
  ```

  Vagy a function elhagyásával.

- ```
  fv_nev(){
   utasítások
   ...
  }
  ```

Ettől még annyiban eltérhetünk, hogy a `{` kerülhet új sorba is.

Fontos, hogy a fv_nev nem lehet már létező alias, mert akkor hibát kapunk.

Példa: ls -l végrehajtása, ha meghívjuk a longlist függvényt

```
longlist(){ /bin/ls -l; }
```

Megjegyzés: Azt, hogy egy parancs honnan fut le, a

```
which parancs
```

utasítással kérdezhető le. Az egyes állományok típusát pedig a

```
file allomany
```

utasítás írja ki.

1.2. Speciális esetek

Speciális esetek:

- Függvény hívása a függvényben:

  ```
  fv1()
  {
  utasítások...
  fv2
  utasítások...
  }
  fv2()
  {
  fv2 utasításai
  }
  ```

  Csak akkor működik helyesen, ha fv1 lefutását megelőzően már lefutott a fv2.

- Függvény definiálása függvényen belül:

  ```
  fv1(){
  ...
  fv2(){
  ```

```
...
}
...
}
```

Ha az fv2 hívása megelőzi az fv1 hívását, akkor hibát kapunk.

- Függvény definiálása then ágban:

```
if feltétel
then
   fv()
   {
   ...
   }
fi
```

A függvény csak akkor érhető el, ha már lefutott a then ág.

1.3. NO_EXIT - Példa a függvényekre

1. NO_EXIT - Példa a függvényekre:

```
[ [ NO_EXIT -eq 1 ] ] && exit() { true; }
```

Függvény deklaráció bárhol állhat, ahol utasítás is. Ez a példa az exit utasítást "maszkolja" egy exit függvénnyel, ami csak 0 visszatérési értéket állít be, de semmi egyéb hatása nincs.

1.4. A függvények paraméterei

A függvények paraméterei:

- Függvényen belül a $1 $2 $3 ... $9 jelülések új értelmet nyernek, ugyanis itt már nem a szkript által kapott pozícionális paramétereket takarják, hanem a függvény paramétereit. Nincs paraméterlista, csak a híváskor a függvény neve után írt, alapértelmezett elválasztóval elválasztott sztringek egy-egy paraméterként átadódnak a függvénynek:

```
fv_nev par1 par2 par3
```

- Az üres sztring, és a semmi közötti különbség:

```
fv(){
 if [ -z $1 ]
 then echo "Nem kaptam 1. parametert."
 else echo "Az 1. parameter: $1"
 fi
 if [ -z $2 ]
 then echo "Nem kaptam 2. parametert."
 else echo "Az 2. parameter: $2"
 fi
}
```

Ez a függvény ellenőrzi, hogy kapott-e első illetve második paramétert, majd ha igen, kiírja őket. Teszteljük le a függvényt a következő esetekre:

-
```
fv
Kimenet:
Nem kaptam 1. parametert.
Nem kaptam 2. parametert.
fv ""
Kimenet:
Nem kaptam 1. parametert.
Nem kaptam 2. parametert.
fv elso
Kimenet:
Az 1. parameter: elso
Nem kaptam 2. parametert.
fv elso masodik
```

```
Kimenet:
Az 1. parameter: elso
Az 2. parameter: masodik
fv "" masodik
Kimenet:
Nem kaptam 1. parametert.
Az 2. parameter: masodik
```

Ha üres sztringet kap a függvény, akkor a paraméter kiértékelése szempontjából ugyanazt a választ kapjuk, mint a semmire, de az üres sztring egy paraméternek számít, míg a semmi nem számít paraméternek. Így ha nem akarunk első paramétert adni a függvénynek, de másodikat igen, akkor "" jeleket kell tenni az első paraméter helyére.

1.5. A függvények visszatérési értéke

A függvény visszatérési értéke az utoljára végrehajtott utasítás visszatérési értéke lesz, kivéve, ha "`return (n)`"-nel térünk vissza, amikor ha írunk valamilyen kifejezést a return után, akkor az lesz a visszatérési érték. A visszatérési érték lekérdezésére ezúttal is a $? változót használhatjuk.

1.6. A függvények által deklarált változók

A függvények által létrehozott változókat a hívó is látja, illetve a hívó változóit a függvény is. Minden változó pontosan egy példányban létezik, ezért deklarálhatunk a local kulcsszóval lokális változókat.

Példa a faktoriális számítására:

- lokális változók használata nélkül:

```
fakt(){
if [ $1 -gt 1 ]
then
 fakt $(($1-1))
else
 return 1
fi
retrun $((${1}*${?}))
}
```

- illetve lokális változók használatával:

```
fakt(){
local x
x=$(($1-1))
if [ $1 -gt 0 ]
then
 fakt $x
 let result=$1*$?
else
 result=1
fi
return $result
}
```

1.7. A névtelen függvények

Az untilhoz és a repeathez hasonlóan a függvényeknél is megtehetjük, hogy beleirányítunk egy fájlt, vagy a kimenetét egy fájlba írjuk ki:

```
fv(){
...
} < input.txt
```

Ezt kicsit elegánsabban a következő módon szokták megoldani:

```
fv(){
 {
 ...
```

```
 } < input.txt
}
```

Ez azért jó, mert így a < vagy a > helyett állhat | (azaz pipe) is. Az ennél a példánál megjelenő plusz belső blokkot (amibe beleirányítjuk az input.txt-t) névtelen függvénynek nevezzük. Ilyenekkel már találkoztunk korábban, amikor a '{' '}' illetve a '(' ')' közötti különbséget vizsgáltuk. Röviden a különbség az, hogy a kerek zárójelek közé írt utasításoknak új shellt nyit a bash, míg a kapcsos zárójeles névtelen függvényt a jelenlegi shell értelmezi.

2. Shift-elés

A shift utasítás:

A pozícionális paramétereket n-nel balra tolja (a $1 helyére kerül a $n). Ha nincs n megadva, akkor az alapértelmezett 1 értékkel dolgozik.

Példa:

```
while test $# -gt 0
 do
  case $1 in
   -a) ... ;;
   -b) ... ;;
   -c) ... ;;
   *) if [ $# -eq 1 ]
      then ...
      else echo "Hiba"
      fi ;;
  esac
  shift;
 done
```

Az itt látható Max 3 opció 1 paraméter szkript shift-eléses megoldása. Példák paraméterek kezelésére

Első verzió, sok sok if-elif:

```
if [ $# -eq 0 ]
then
  echo "Kevés paraméter!"
  exit 1;
fi
if [ $1 = "-t" ]
then
  if [ $3 = "-m" ]
  then
    until who | grep $4 >/dev/null
    do
    ido=${2:-5}
    echo "LÉPJEN BE!"
    sleep $ido;
    done
    user=`who |cut -d" " -f1 |uniq`
    echo "nem jelentkezett be" | mail -s "hiba" user
    echo "mail sent"
    echo "Belépett, exiting...";
    exit 1;
  fi
  until who | grep $3 >/dev/null
  do
  ido=${2:-5}
  echo "LÉPJEN BE!"
  sleep $ido;
  done
  echo "Belépett, exiting...";
elif [ $1 = "-m" ]
  then
  if [ $2 = "-t" ]
  then
    until who | grep $4 >/dev/null
```

```
    do
    ido=${3:-5}
    echo "LÉPJEN BE!"
    sleep $ido;
    done
    echo "Belépett, exiting...";
    user=`who |cut -d" " -f1 |uniq`
    echo "nem jelentkezett be" | mail -s "hiba" user
    echo "mail sent"
    exit 1;
  fi
  until who | grep $2 >/dev/null
  do
  ido=2;
  echo "LÉPJEN BE!"
  sleep $ido;

  done
  echo "Belépett, exiting...";
  user=`who |cut -d" " -f1 |uniq`
  echo "nem jelentkezett be" | mail -s "hiba" user
  echo "mail sent"
else
  until who | grep $1 >/dev/null
    do
    ido=1
    echo "LÉPJEN BE!"
    sleep $ido;
    done
    echo "Belépett, exiting...";
fi
```

Második lehetőség: [nem teljes - de próbálja figyelni, hogy létező felhasználónév lett-e megadva ...]

```
if [ $#-eq0 ]
then
 echo "Hiba: Kevés p"
 exit 1;
fi
teszt=`cat /etc/passwd | cut -d":" -f1`
if [ $teszt|grep $1 ]
then
 usr = $1;
elif [ $teszt|grep $2 ]
then
 usr = $2;
elif [ $teszt|grep $3 ]
then
 usr = $3;
elif [ $teszt|grep $4 ]
then
 usr = $4;
fi
if [ -z $usr ]
then
 echo "Nem létező felhasználó"
 exit 1;
fi
if [ $*|grep "-t" ]
then
 ido=`echo $* | cut -d"-t" -f2 | cut -d" " -f2`
fi
until who|grep $usr >/dev/null
 do sleep ido;
done
if [ $*|grep "-m" ]
then
 echo "belépett"|mail -s 'Alert!' `whoami`;
fi
echo "Belépett!";
```

Harmadik verzió: [már ciklus figyeli ...]

```
if [ $# -eq 0 ]
then
 echo "Hiba: Tobb parametert adjal meg!"
 exit 1;
fi
ido=5;
m=0;
until [ $# -eq 1 ]
do
 if echo $1 | grep "\-t" > /dev/null
 then
  shift 1;
  if [ $1 -gt 0 ] 2> /dev/null
  then
   ido=$1;
  else
   echo "Hiba";
   exit 1;
  fi
 fi
 if echo $1 | grep "\-m" > /dev/null
 then
  m=1;
 fi
 shift 1;
done
until who | grep "^$1 " > /dev/null
do sleep $ido;
done
if [ $m -eq 0 ]
then
 echo "Belepett"; echo $ido;
else
 echo "mail"
 echo $ido;
 echo "Belepett" | mail -s "Alert" `whoami`
fi
```

3. Pozícionális paraméterek értékének átírása

1. A shell változók `set` ill. `env` parancsokkal történő kilistázása már ismert, most nézzük meg a **set** használatának egy új módját:

   ```
   set word1[ wordi]...
   ```

   Ez a parancs a szkriptben kiadva felülírja a kapott pozícionális paramétereket a word1 word2 ... wordn szavakkal ($1=word1 $2=word2 ... $n=wordn módon).

2. Példa, a set használatára:

   ```
   set `date`
   echo $6 $2 $3 $1
   ```

   A szkript az aktuális dátumot fogja kiírni "2007 Nov 12 Mon" formában (feltéve, hogy az eredeti formátum a "Mon Nov 12 16:20:45 CET 2007" volt). Egy másik gyakori alkalmazási példa, az IFS-el kombinálva:

   ```
   $ line="Micro Logic Corp.:Box 174:Hackensack, NJ 07602"
   $ IFS=:
   $ set $line
   $ echo $#                          Mennyi paraméter lett beállítva(set-telve)?

   3

   $ for field; do echo $field; done

   Micro Logic Corp.
   Box 174
   Hackensack, NJ 07602
   $
   ```

4. Getopts, avagy egy másik megoldás az opciók kezelésére

A getopts használata:

```
getopts opciósztring változó
```

Ahol az opciósztring a:b:c ha -a -b -c opciók vannak, és ezek közül a -a ill. a -b opció paramétert is vár (ezt jelzi az opció utáni ':'). Ebben az esetben a

```
getopts a:b:c KAPCSOLO
```

utasítás a 'szkript -a "a parméter" -b "b paraméter" -c' típusú hívást képes kezelni. Az éppen aktuális opcióhoz tartozó paraméter (ha tartozik hozzá) az $OPTARG változóban tárolódik, az eddig feldolgozott opciók száma pedig az $OPTIND változóban.

Ha **nem ismert opciót** talál, akkor a változóba '**?**' értéket helyez el!

Példa:

```
while getopts i:o:v KAPCSOLO
 do
  case $KAPCSOLO in
  "i") INPUT=$OPTARG ;;
  "o") OUTPUT=$OPTARG ;;
  "i") VERBOSE="igaz" ;;
  "?") echo "Tanuld meg hasznalni" ; exit 1 ;;
  esac
 done
eval "FILENEV=$"$#
```

5. xargs

A shell védekezik a puffer-túlcsordulás ellen, ezért ha például egy mappában van 8000 fájl, és törölni akarjuk az összes a betűvel kezdődőt (és még ebből is van mondjuk 1000), akkor a *The parameter list is too long* üzenetet fogjuk kapni. Ezt elkerülendő, ha nem tudjuk, mennyi filera fog illeszkedni a minta, amit a szkriptben megszabunk, akkor használjuk az xargs-ot.

Működése:

```
argumentum listát előállító utasítás | xargs program
```

A túlcsordulást az xarg úgy védi ki, hogy a paramétereket a standard inputról várja, és így adja át egyesével, vagy előre megszabott méretű csomagokban a paramétereket az xargs mögött álló programnak. Pl.:

```
cat filelist.txt | xargs rm
```

Törli a listán található fájlokat. Az előbb említett csoportosítás pedig így működik:

```
cat filelist.txt | xargs -n 20 rm
```

Így az rm 20-as csoportokban kapja meg a fájlneveket. A csoportosítás egy kicsit látványosabb, ha olyan paranccsal végezzük, aminek van kimenete, pl.:

```
ls | xargs -n 2 echo ---
```

Kilistázza a könyvtár tartalmát úgy hogy minden sor "--- file1 file2" formátumban tartalmaz két fájlt (kivéve az utolsó sort, ha abba már csak egy fájl jut). A fentebb említetthez hasonló fájl törlési probléma megoldása pedig a következő:

```
ls | egrep ^a[0-9] | xargs -n 20 rm
```

Törli az összes 'a'-val kezdődő és számjeggyel folytatódó nevű fájlt a könyvtárból.

6. A ":" utasítás

A : utasítás - az üres parancs

- Visszatérési értéke minidg 0. - hasonló, mint a true
- Bárhol szerepelhet, ahol utasítás állhat.

```
while :
do
 ut.-ok
done
if feltétel
 then :
 else
  ut.-ok
fi
: > file
: >>file
```

- Ha van argumentuma, akkor azt a shell kiértékeli:

```
: ${VALTOZO:=25}
Ha nem lenne a : utasítás, akkor az értékadás után a shell automatikusan lefuttatná az
eredményt!
```

7. eval - Parancssor újbóli feldolgozása - közvetett változó használat

Az eval utasítás:

Példa: - Parancssor újboli feldolgozása

```
OUTPUT=output.txt
echo Hello > $OUTPUT
Ami megjelenik: Hello > output.txt
Ami megoldja:
eval echo Hello > $OUTPUT
Ennek hatására már a kimeneti fileba megy az echo kimenete.
```

Példa:

```
x=3
valtozo_neve="x"
parancs="echo $"$valtozo_neve
eval $parancs
```

Ekkor az utolsó sor kiírja a $x értékét. Ugyanezt elérhetjük az

```
x=3
valtozo_neve="x"
eval echo \$$valtozo_neve
```

szkripttel is. De mutatóként is használható, vegyük az alábbi példát:

```
eval $valtozo_neve=50          Eltárolja az 50 értéket a valtozo_neve által mutatott
változóba
echo $x                        Mi történt, nézzük ...
```

Egy másik használati példa:

```
$ cat last
eval echo \$$#

$ last egy ketto harom negy
negy

$ last *                        Utolsó fájl neve ...
zoo_report
```

8. Szignálok

Szignálok küldésére a kill utasítás használható.

A küldhető szignálok egy része:

- 9 kill
- 15 term
- 2 interrupt
- 0 exit
- DEBUG debug

Szignálok elkapására illetve elengedésére a trap utasítás szolgál. Elkapás (a szignál maszkolása):

```
trap utasítások jel(ek)
```

pl.:

```
trap 'echo EXIT jel jött;' EXIT
```

Az egyetlen olyan szignál, ami nem maszkolható, a kill szignál (nem lehet örökéletű folyamat). Az eleresztés:

```
trap jel(ek)
```

pl.:

```
trap EXIT
```

Ha nem akarunk semmilyen utasítást végrehajtani, amikor a maszkolandó szignált kapjuk, csak nem szeretnénk, ha a felhasználó mondjuk egy védett folyamatnak EXIT szignált küldene, akkor így kell eljárnunk:

```
trap '' EXIT
```

15. fejezet - Komplex példa

Az alábbi példa egy komplex program, amely bemutatja a korábban tárgyalt összes lehetőséget arra, hogy miként is tudjuk felhasználni a megszerzett ismereteket. Maga az alkalmazás egy telefonkönyv lesz, amely egy szöveges fájlban tárolja az adatokat, a felhasználóval pedig interaktív formában kommunikál.

```
# Először a szükséges függvényeket készítjük el!
#
#================================================
#
# Személy felvitele a telefonkönyvbe:
#

felvesz(){

echo "Új bejegyzés fevitele!"
echo "A befejezéshez egy üres sor (ENTER) kell a végén ütni!"

elso=
bejegyzes=

while true
do
      # sor beolvasása
      echo ">> \c"
      read line

      # üres sor kezelése
      if [ -n "$line" ]
      then
              bejegyzes="$bejegyzes$line^"

              if [ -z "$elso" ]
              then
                      elso=$line
              fi
      else
              break
      fi
done

echo "$bejegyzes" >> $TELEFONKONYV
# bejegyzések rendezése
sort -o $TELEFONKONYV $TELEFONKONYV

echo
echo "$első elem felvéve!"
}
#=========================================
#
# Személy keresése
#
keres(){

nev=$1
# átmeneti fájl segítségével
grep "$nev" $TELEFONKONYV > /tmp/talalatok$$

if [ ! -s /tmp/talalatok$$ ]
then
      echo "Nem található $nev a telefonkönyvben!"
else
      #
      # Találatok megjelenítése
      #
      while read line
      do
               display "$line"
      done < /tmp/talalatok$$
fi
```

```
rm /tmp/talalatok$$
}
#=======================================
#
# Bejegyzés megjelenítése
#
kiir(){
echo
echo "-------------------------------------"

#mentés
szoveg=$1
# mezőelválasztó átdefiniálása
IFS="^"
# bemenet feldarabolása
set $szoveg

# kiírása
for line in "$1" "$2" "$3" "$4" "$5" "$6"
do
        printf "| %-34.34s |\n" $line
done

echo "|       o                   o       |"
echo "-------------------------------------"
echo
}
#=============================================

#
# Törlés a telefonkönyvből
#
torol(){

nev=$1
#
# találatok átmeneti fájlba gyűjtése

grep "$nev" $TELEFONKONYV > /tmp/talalatok$$

if [ ! -s /tmp/talalatok$$ ]
then
        echo "Nem található ilyen bejegyzés!"
        exit 1
fi

#
# Találatok megjelenítése egyesével és megerősítés kérése
#
while read line
do
        kiir "$line"
        echo "Bejegyzés törlése (i/n)? \c"
        read valasz < /dev/tty

        if [ "$valasz" = i ]
        then
                break
        fi
done < /tmp/talalatok$$

rm /tmp/talalatok$$

# ha törlést választunk, akkor eltávolítjuk a fájlból
if [ "$valasz" = i ]
then
        if grep -v "^$line$" $TELEFONKONYV > /tmp/telko$$
        then
                mv /tmp/telko$$ $TELEFONKONYV
                echo "Kijelölt bejegyzés törölve!"
        else
```

```
                echo "Bejegyzés nem lett rörölve!!!"
        fi
fi
}
#================================================
#
# Bejegyzés módosítása
#
modositas(){

nev=$1
#
# Bejegyzés kikeresése

grep "$nev" $TELEFONKONYV > /tmp/talalatok$$
if [ ! -s /tmp/talalatok$$ ]
then
       echo "Nem található $nev bejegyzés!"
       exit 1
fi

#
# Találatok megjelenítése egyesével és megerősítés kérése
#

while read line
do
        kiir "$line"
        echo "Bejegyzés módosítása (i/n)? \c"
        read valasz < /dev/tty

        if [ "$valasz" = i ]
        then
                break
        fi

done < /tmp/talalatok$$

rm /tmp/talalatok$$

if [ "$valasz" != i ]
then
        exit
fi

#
# szövegszerkesztő indítása a kijelölt bejegyzésre
#

echo "$line\c" | tr '^' '\012' > /tmp/ed$$
echo "Változások bevitele az alábbival: ${EDITOR:=/bin/ed}"
trap "" 2           # ne lépjen ki DELETE gomb leütésére - el kell kapni a signált!
$EDITOR /tmp/ed$$

#
# Régi bejegyzés eltávolítása - új felvitele
#

grep -v "^$line$" $TELEFONKONYV > /tmp/telko$$
{ tr '\012' '^' < /tmp/ed$$; echo; } >> /tmp/telko$$
# utolsó echo a tr törlése miatt kell!

# rendezés
sort /tmp/telko$$ -o $TELEFONKONYV
rm /tmp/ed$$ /tmp/telko$$
}
#=======================================

kiir_mindet(){
#
# Összes bejegyzés megjelenítése
#
```

```
IFS='^'     # hasonló mint fentebb ...
echo "-------------------------------------------------------"
while read line
do
     # darabolás
     set $line
     #
     # bizonyos mezők mutatása
     #
     eval printf "\"%-40.40s %s\\n\"" "\"$1\"" "\"\${$#}\""
done < $TELEFONKONYV
echo "-------------------------------------------------------"
}

# alapértelmezett helye a telefonkönyvnek ...
: ${TELEFONKONYV:=$HOME/telefonkonyv}

export TELEFONKONYV

if [ ! -e "$TELEFONKONYV" ]
then
       echo "$TELEFONKONYV nem létező fájl!"
       echo "Létrehozza most (i/n)? \c"
       read valasz

       if [ "$valasz" != i ]
       then
              exit 1
       fi
       > $TELEFONKONYV || exit 1        # exit ha nem jött létre!
fi

#
# ha kapott paramétert, akkor arra keresünk is egyből ...
#
if [ "$#" -ne 0 ]
then
        keres "$@"
        exit
fi

#
# A DELETE gombra signál elkapás helyezése, hgoy a ciklus fusson
trap "continue" 2

#
# Iterál mindaddig amíg a felhasználó a kilépést nem választja ...
#
while true
do
        #
        # Menü kiírása
        #
        echo '
        Telefonkönyv:
             1. Személy keresése
             2. Személy felvitele
             3. Személy törlése
             4. Személy módosítása
             5. Bejegyzések listázása
             6. Kilépés a programból
        Kérem válasszon a fenti lehetősége közül (1-6): \c'
        #
        # választás feldolgozása
        #
        read valasztas
        echo
        case "$valasztas"
        in
            1) echo "Keresendő név: \c"
               read nev
```

```
               if [ -z "$nev" ]
               then
                      echo "Keresés megszakítva ..."
               else
                      keres "$nev"
               fi;;

            2) felvesz;;

            3) echo "Törlendő személy neve: \c"
               read nev
               if [ -z "$nev" ]
               then
                      echo "Törlés megszakítva ... "
               else
                      torol "$nev"
               fi;;

            4) echo "Módosítandó személy neve: \c"
               read nev
               if [ -z "$nev" ]
               then
                      echo "Módosítás megszakítva ..."
               else
                      modositas "$nev"
               fi;;

            5) kiir_mindet;;

            6) exit 0;;

            *) echo "Rossz gombleütés!\a";;
         esac
done
```