Szemethy Tivadar

Unix ismertető

Tartalomjegyzék

1. Unix? Linux!

1.1. Pár szó a Unix világról

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

1.2. A Unix rövid története

Mert a Unix meglehetősen régi dolog. Első változatát 1969-ben készítette Ken Thomson és Dennis Ritchie az AT&T Bell Laboratóriumában egy PDP-7 típusú számítógépre. A rendszer magját 1973-ban átírták C nyelvre - ennek köszönheti a Unix mind a mai napig legnagyobb előnyét, a könnyű hordozhatóságot. Az AT&T kezdetben ingyen az amerikai egyetemek rendelkezésére bocsátotta a Unix forráskódját, így tíz éven belül százezer fölé emelkedett a működő Unix rendszerek száma (ne felejtsük, ez még a „hőskorszakban" volt - a 80-as évek elején százezer számítógép még hatalmas nagy szám volt). A gyors terjedésnek azonban jelentkeztek a hátulütői is: nem volt egységes ellenőrzése senkinek sem a forráskód, a rendszer egysége felett, így számos (helyi módosításokon alapuló) változat alakult ki, amelyek közül a két legjelentősebb a Berkeley egyetemen kifejlesztett BSD Unix, amely jelenleg a 4.4-es verziónál tart, illetve az AT&T „hivatalos" változata a System V (System Five, amely a 4. Release-nél tart, rövidítve SVR4), amelyet a Unix System Laboratories fejleszt tovább. (Az USL-t egyébként a Novell vásárolta fel a közelmúltban). Ezen fő változatok mellett számos kisebb-nagyobb alváltozat van forgalomban még napjainkban is.

Amint a Unix egyre népszerűbbé kezdett válni a kereskedelmi szférában, egyre több cég ismerte fel egy egységes Unix szabvány fontosságát, és több egységesítő, szabványosító bizottság és csoportosulás kezdett dolgozni. Az USL köré tömörülő cégek az SVR4 mögé sorakoztak fel, a BSD irányból érkezők pedig az OSF (Open Systems Foundation) ajánlását, az OSF/1-et támogatják. Időközben független (nem az érdekelt cégek támogatásával létrejött) bizottságok is próbálták valamennyire egységesíteni a BSD és System V ajánlásokat, és az IEEE kidolgozta (az ANSI és az ISO támogatásával) a „POSIX" (Portable Operating System Interface (x)) ajánlást, amely igyekszik egyesíteni a két fő szabványt. Végül 1993-ban a Novell átruházta a UNIX védjeggyel kapcsolatos jogait az X/Open konzorciumra, amely lényegében a világ összes jelentősebb gyártó és felhasználó szervezetét tömöríti, így tulajdonképpen többékevésbé függetlennek mondható.

Fontos megjegyezni a „UNIX" és a „Unix" terminológia közötti különbséget: a szakmában meghonosodott szokás szerint „UNIX"-szal jelölik az „igazi", az USL licensszel rendelkező, USL forráskódból származó rendszereket, míg a „Unix" jelölés általában jelöl mindenféle

„Unix típusú” rendszert, amelyek a „józan ész” alapján Unixnak mondhatók, függetlenül attól, hogy van-e valami közük az USL-hez vagy sem. Újabban lehetővé vált, hogy bármely gyártó - függetlenül attól, hogy USL forrásból származik-e a rendszere vagy sem - kérje az X/Open-t, hogy egy szoftver-specifikáció és teszt alapján (ezt nevezik SPEC 11.70-nek) minősítse termékét „igazi” UNIX-nak, a UNIX védjegy viselésére jogosultnak, azonban ez a specifikáció annyira új még, hogy mindezidáig egy cég sem kért ilyen minősítést. A közeljövőben azonban, amint a cégek befejezik termékeik hozzáigazítását a specifikációhoz, tömegesen várható az egységes szabványnak megfelelő UNIX rendszerek megjelenése.

Természetesen, mivel a Unix nagyon könnyen hordozható, már elég korán megszülettek az Intel-PC-alapú Unixok is, először csak oktatási célokra (pl. a már 286-oson működő XENIX), majd megjelentek a már komoly munkára is képes PC-s Unix verziók, melyeknek kereskedelmi ára 500 és 2000 $ között van.

1.3. Hogy kerül ide a Linux?

1.3.1. A lehetőség

Amikor a Unix még csak az egyetemi és akadémiai szférában volt közismert, kialakult körülötte egy hatalmas programkörnyezet: minden egyetem, kutatóintézet elkészítette saját megoldásait felmerülő számítástechnikai problémáira (a szövegszerkesztéstől, táblázatkezeléstől kezdve a mindenféle apró utility-n keresztül a különböző fordítóprogramokig), és mivel ezek az intézmények non-profit szervezetek voltak, elkészült szoftvereiket publikussá tették. Forráskódban, C nyelven adták közre ezeket a programokat, egyre inkább a terebélyesedő Hálózat segítségével, és a C nyelv és az egységes környezet miatt minden Unix felhasználó lefordíthatta, használhatta, módosíthatta és továbbfejleszthette őket szinte teljes szabadsággal. Ennek a folyamatnak az eredményeként alakult meg Richard Stallman kezdeményezésére az FSF (Free Software Foundation) alapítvány, melynek célja egy szabadon (forráskódban is) ingyen hozzáférhető szoftverkörnyezet biztosítása bárki számára, illetve ennek részeként a GNU project (GNU is Not UNIX), amely pedig egy minél teljesebb Unix rendszert kíván létrehozni és biztosítani. Ennek jogi megfogalmazása a GPL (GNU General Public 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.

1.3.2. Linus Torvalds operációs rendszert ír

Megvolt tehát a GNU környezet: fordítók, segédprogramok, és a szabadon terjeszthető XFree grafikus felület, egy olyan operációs rendszer mag hiányzott csak, amely bizonyítottan szabad (nem tartalmaz copyright alá eső USL vagy BSD kódot). Ennek megírását kezdte el helsinki egyetemista korában Linus Torvalds, hogy aztán több száz segítőjével együtt létrehozza azt, amit ma Linuxként ismerünk: egy teljes, szabad operációs rendszert bárki 386-os PC-jére. Bár Linus Torvalds egyedül kezdett hozzá operációs rendszere elkészítéséhez, ma már a Linux oly sokfelé ágazott és akkorára nőtt (ma már talán ez a legtöbb PC-s hardvert támogató szoftver,

és átírása más architektúrákra - Sun, MIPS, DEC Alpha, 68000 stb. - folyamatban van), hogy Linus maga már leginkább csak koordinálja a fejlesztéseket.

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.

1.3.3. Az Internet közössége felkarolja a fejlesztést

Szót kell itt még ejtenünk még egy fontos dologról, ami nélkül a Linux operációs rendszer nem születhetett volna meg: ez pedig az Internet világhálózat, amely összekötötte a fejlesztőket, és amelyen mindig bárki számára ingyen hozzáférhetők az egyes Linux változatok. A Unix és az Internet mindig is szorosan kötődött egymáshoz: az első IP (Internet Protocol) implementációkat BSD Unix rendszerekre készítették, a Unixot nyíltsága, jól bővíthetősége miatt választották a fejlesztők. Az Internetbe kötött gépek legnagyobb része Unix alatt fut, az Internet szolgáltatásait maximálisan csak Unix alatt tudjuk kihasználni. Manapság talán ez a Unix (Linux) rendszerek gyors terjedésének fő oka. A Linux az Interneten született: bár támogat más hálózati protokollokat is, igazán egyszerűen Internetes környezetbe illeszthető.

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.

1.4. Kernelek, disztribúciók, verziók

1.4.1. Folyamatosan fejlődő szoftver

Mivel a Linux rendkívül gyorsan fejlődik, (hetente, de néha két-három naponta is jelenik meg új verzió), szükséges talán már itt az elején kis rendet tenni a verziószámok dzsungelében. Legelőször is jegyezzük meg, hogy ha „stabil” verzióról beszélünk, akkor is csak saját felelősségünkre használhatjuk a Linuxot: mivel nem áll mögötte kereskedelmi cég, nincs aki a megbízható működésért garanciát vállaljon. (Ha hajlandók vagyunk áldozni rá, vásárolhatunk support-ot valamely erre szakosodott cégtől - bár Linuxra szakosodott cégek jelenleg leginkább csak az USA-ban működnek - az ingyenes Linuxhoz azonban nem jár garancia.)

Ha lefagy, és fontos adataink vesznek el, csak magunkra számíthatunk - azonban a több százezer Linux felhasználó tanúsága szerint a Linuxnak nem igazán szokása lefagyni. Megjegyzendő egyébként, hogy szoftvereik hibamentességére gyakran nagy szoftvercégek sem vállalnak garanciát.

1.4.2. A kernel verziószámáról

Linux verziószám alatt az ún. „kernel”, az operációs rendszer mag verziószámát értjük, ez a tulajdonképpeni Linux, azonban mint minden operációs rendszer esetén, önmagában ez nem jó semmire: arra való, hogy más (felhasználói) programokat futtassunk vele (rajta). A kernel verzió-számozása a következő: x.y.z, ahol az x a fő verziószám, ez jelenleg 1 (és még sokáig

az is marad), az y, a középső szám az al-verziószám, amely ha páros, akkor az egy „stabil” béta-verziót jelent, és ha páratlan, akkor pedig alfa-verziót, amely tényleg teljesen fejlesztői változat. A harmadik szám, a z pedig az ún. patch-level: az apróbb változtatásokat sorszámozzák ezzel. E sorok írása idején a legutolsó stabil, béta állapotú verzió száma 1.2.11 volt, a fejlesztői verzióé pedig 1.3.10.

1.4.3. A disztribúciókról

Szokás még az ún. disztribúció verziószámáról is beszélni: a disztribúció egy Linux kernelen alapuló teljes (működőképes) Unix rendszer, segédprogramokkal, alkalmazásokkal együtt. Egy disztribúció elkészítése tulajdonképpen a C forrásban meglévő utility-k, programok lefordításából, könyvtárstruktúrába helyezéséből és összekonfigurálásából áll. Sokféle disztribúció létezik, ingyenesek is és kereskedelmiek is - a továbbiakban mi a legjobban elterjedt „Slackware” disztribúciót fogjuk ismertetni, amely ingyenes és az Internetről sok helyről letölthető. Elhangzott, hogy léteznek kereskedelmi (nem ingyenes) disztribúciók is - talán ez furcsa lehet egy szabad szoftver esetén, azonban mint azt már a GPL ismertetésekor láttuk, ezek a cégek nem a Linux-ért kérnek pénzt, hanem a disztribúció összeállításáért, illetve esetleg a disztribúcióban lévő, Linux alatt futó, de nem ingyenes szoftverek használatáért.

2. Hogyan működik?

2.1. Memóriakezelés, többfeladatos működés

A Linux - a DOS-tól eltérően - valódi többfeladatos (multitask) operációs rendszer. Kihasználva az Intel 80386 processzor nyújtotta fejlett tár és taszkkezelési lehetőségeket, valódi időosztásos környezetet biztosít. A 386-os processzor többféle üzemmódjai közül a Linux a „supervisor" üzemmódot használja a kernel, az operációs rendszer mag futtatására, ebben az üzemmódban a kernelnek hozzáférése van a gép összes fizikai erőforrásához, a felhasználói folyamatok (processzek) pedig ún. „user" üzemmódban futnak. A 386-os processzoron lehetőség van több, egymástól független „user" módú ún. taszk definiálására, amelyek egymástól védettek, nem tudják egymás és a felügyelő kernel memória-területét kiolvasni vagy módosítani, és a gép közvetlen hardver erőforrásaihoz sincs hozzáférésük (a valóságban ez úgy működik, hogy „user" módban csak a processzor utasításkészletének egy része hajtható végre, nem megengedett utasítás végrehajtásának kísérletekor vagy nem megengedett memóriacímre hivatkozáskor a vezérlés hibaüzenettel visszakerül a supervisor módban futó felügyelő programhoz, vagyis a kernelhez). Így biztosítható az egyes felhasználói programok egymástól való védelme. Mivel az egyes folyamatoknak a gép fizikai erőforrásaihoz (pl. winchester, képernyő) sincs közvetlen hozzáférésük, bármilyen perifériaműveletet csak a kernel meghívása útján végezhetnek, így mód nyílik például biztonságos file-rendszer megvalósítására is. Nincs az a trükk, amivel az egyik felhasználó el tudna olvasni olyan adatot, amihez nincs engedélyezett hozzáférése - amennyiben persze a biztonsági rendszerben nincsenek programozói hibák és a rendszer konfigurációja megfelelő. Az emberi gyarlóságot nem tekintve a Linux biztonságos többfelhasználós operációs rendszernek tekinthető.

A kernel teljes mértékben, fizikai szinten hozzáfér a gép erőforrásaihoz. Fizikai, a lehető legalacsonyabb szinten kezeli is a hardvert, a legnagyobb teljesítmény elérése érdekében. A boot-olást (rendszer betöltést) kivéve egyáltalán nem használja a BIOS-t: ennek oka, hogy a BIOS programja a DOS figyelembevételével készült, és nem alkalmas multitaszk operációs rendszer futtatására.

Memóriakezelésében szintén kihasználja a 386 által nyújtott lehetőségeket: lapozásos virtuális memóriakezelést használ, ahol a fizikai memóriát kiegészíthetjük a winchesterről vett virtuális memóriával (page vagy swap terület). A teljes memóriát lapokra osztja, ezen virtuális lapokat rendeli hozzá az egyes folyamatokhoz, és gondoskodik róla, hogy az éppen szükséges lapok a fizikai memóriában legyenek. Itt kell megemlíteni, hogy a Linux használja a virtuális tárkezelés mindkét (gyakran összekevert) fajtáját, a lapozást (paging) és a tárcserét (swapping) is. Lapozásnál folyamatoktól függetlenül, a rendszer arra ügyel, hogy a szükséges lapok a fizikai memóriában legyenek, ha azok esetleg diszken vannak, akkor gondoskodik memóriába olvasásukról, illetve ha a fizikai memória teli van, akkor a ritkábban használt lapokat a diszkre írja. Tárcserénél pedig a rendszer figyelemmel kíséri az egyes folyamatok aktivitását is, és ha szabad memóriára van szükség, egy inaktív folyamat egészét háttértárra írja, felszabadítva ezzel a folyamat által használt összes fizikai memóriát. A Linux a két módszer keverékét használja: amíg bővében van a memóriának, csak egyes lapokat lapoz ki/be, de például ha úgy látja, hogy egy folyamat hosszú ideje inaktív, és nem csak egy-két lapnyi memóriára van szükség, akkor az adott folyamathoz tartozó összes fizikai lapot diszkre menti.

2.2. Automatikus diszk-gyorsítótár

Szorosan kapcsolódik a memória-lapkezelés mechanizmusához a Linux buffer cache kezelési módszere: a buffer cache a Unix rendszerek diszk-eléréshez használt gyorsítótárja, amelyet a kernel kezel, mivel minden folyamat csak és kizárólag a kernel meghívásával végezhet diszkműveletet. Linuxban a buffer cache mérete dinamikusan, a rendszer-terheléstől függően változik: mindig az éppen szabad fizikai memória egészét erre a célra használja. A diszk-írások is a buffer cache-n keresztül történnek: minden írás először a cache memóriába kerül, és vagy egy megadott idő elteltével íródik ki diszkre, vagy pedig akkor, ha a rendszer számára „elegendő" kiírnivaló összegyűlt. Ezért fontos az, hogy a megfelelő shutdown procedúra (a rendszer „lelövése", leállítása) végrehajtása nélkül soha ne kapcsoljuk ki a gépet: kikapcsolás előtt mindig szükséges a diszk tartalmának szinkronizálása a memóriában lévő állapottal, a nyitott file-ok lezárása - ezen lépések elmulasztása esetén kikapcsoláskor a diszk tartalma helytelen lehet, információk, egész file-ok veszhetnek el. Ugyanez történhet persze áramszünet vagy más hiba esetén is: ez az az ár, amit a Unix file-rendszer által nyújtott nagyobb teljesítményért fizetnünk kell. A Unix-ok általában (így a Linux is) rendelkeznek funkciókkal az esetleges információvesztés minimalizálására, illetve a korrekt file-rendszer visszaállítására - tulajdonképpen egy átlagos Linux rendszerben váratlan rendszerösszeomláskor maximum csak a legutóbbi 30 másodperc munkája veszhet el - nagyon kicsi valószínűségű extrém esetektől eltekintve. De az ilyen adatvesztés veszélye minden, a diszk-írást bufferelő rendszerben fennáll.

2.3. Fejlett memóriakezelési technikák

2.3.1 A „demand paging" módszer

A fejlett memóriakezelés lehetővé teszi további, a teljesítményt növelő megoldások használatát is: ezek egyike az ún. „demand paging", ami azt jelenti, hogy egy futtatható file végrehajtásakor nem az egész file töltődik be a memóriába, hanem mindig csak azok a lapjai, amelyekre a végrehajtás során éppen szükség van: gondoljunk csak bele, minden programnak vannak olyan részei (inicializálás, hibakezelő részek), amelyek csak egyszer, vagy éppenséggel egyszer sem futnak le - ezeket aztán vagy soha be sem tölti a rendszer a memóriába, vagy pedig miután lefutottak, rögtön fel is szabadítja az általuk elfoglalt területet.

2.3.2. Megosztott programkönyvtárak

Többfeladatos működés esetén, amikor egyszerre több program tartózkodik a fizikai memóriában, hasznos a Linuxban szintén használt „osztott könyvtár" (shared library) mechanizmus, vagy más néven a dinamikus (futásidejű) programösszefűzés (link). Alapötlete az, hogy mivel mindegyik program C-ben íródott és ugyanabban a környezetben fordult le, ezért valószínűleg lesz egy csomó olyan függvény (például a képernyőkezelő könyvtári függvények), amelyeknek kódja minden programban ugyanaz, és felesleges minden programmal együtt a memóriába tölteni őket, elég csak egyszer - csak a programok tudják, hol keressék a memóriában ezeket a függvényeket. Így minden programba elég egy „csonk"-nak (stub) nevezett programrészlet beépítése, amelyik a dinamikus linker segítségével gondoskodik a megfelelő függvény megtalálásáról illetve memóriába töltéséről, amennyiben az még nem lenne betöltve.

2.3.3. A copy-on-write mechanizmus

Még egy memóriakezelési finomság van, amelyet Unix rendszerekben előszeretettel használnak: ez pedig az úgynevezett copy-on-write (írásnál másold) mechanizmus, amely a következőképpen működik: multitaszkos operációs rendszerben gyakran lehet arra szükség, hogy bizonyos memória-lapokat több folyamat között megosszunk, vagy a legtipikusabb példa: Unix-ban új folyamat létrehozása mindig egy másik folyamat memóriájának lemásolásával történik. Mivel viszont egy memórialapra „többfelől", több folyamat memóriatérképéből tudunk hivatkozni, nem is kell azt a lapot lemásolni, csak elhelyezni az ugyanarra a lapra mutató hivatkozásokat a megfelelő helyeken. Mivel a memóriamásolás meglehetősen időigényes dolog, ezzel kritikus helyen (a felhasználó szemszögéből holtidőnek számító folyamatlétrehozáskor) sok időt takaríthatunk meg, és most már csak arra kell vigyáznunk, hogy ha az ugyanarra a lapra hivatkozó több folyamat közül valamelyik módosítani akarja a lapot, (mivel mindegyik folyamat azt hiszi, a memórialap csak az övé) akkor készítsünk csak annak a folyamatnak a számára másolatot, amelyet aztán módosíthat kedve szerint.

2.4. A folyamatok ütemezése

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

Nem szabad azonban egy fontos dologról megfeledkeznünk: A Unix nem valós-idejű (real-time) operációs rendszer, ami annyit jelent, hogy ha több folyamat fut egyszerre, és az egyiktől elkerül a vezérlés, akkor valamekkora (rendszerint nagyon rövid) idő múlva vissza is fogja majd kapni - a két aktív (futó) állapot közti időre azonban nincs szigorú felső korlát. Az esetek 99.9999999 százalékában ez az idő (még egy leterhelt - de nem indokolatlanul túlterhelt rendszeren is) pár tized másodperc - azonban soha nem mondhatjuk, hogy biztosan csak ennyi. Ezt a tényt pontos és rövid időzítéseket használó programoknál nem árt szem előtt tartani.

2.5. Többfelhasználós működés

2.5.1. A terminál koncepció

A Linux, eltérően a PC-n eddig megszokott operációs rendszerektől, nemcsak többfeladatos, ahol egy felhasználó egyidejűleg több programot futtathat (mint például az MS-Windows és az OS/2), hanem többfelhasználós is, vagyis egyidejűleg több felhasználó használhatja ugyanazt a rendszert, és mindegyikük akár több programot is futtathat. Ennek megvalósításához azonban szükség van néhány új fogalom, koncepció bevezetésére: Rögtön első problémaként jelentkezik az, hogy egy PC-nek csak egy billentyűzete, és (kevés kivételtől eltekintve) csak egy monitora van, amit értelemszerűen egyszerre csak egyvalaki használhat. A Unix filozófia

minden egyes bejelentkezett felhasználóhoz hozzárendel egy-egy úgynevezett terminált: egy terminál pedig egy billentyűzet + megjelenítő egység (leggyakrabban szöveges display) együttesét jelenti. Az adott Unixos géphez legközvetlenebbül csatolt terminált (Linux esetén a gép saját billentyűzetét és monitorát) konzol terminálnak (console terminal) nevezzük, ez abból a szempontból kitüntetett, hogy bizonyos rendszeradminisztrációs feladatok csak innét hajthatók végre. További terminálok csatolhatók még a géphez soros vonalon (ez a legősibb Unixos megoldás, egy terminálemulációs szoftver és egy soros kábel segítségével akár kidobandó XT-inket is egyszerűen soros terminállá alakíthatjuk, illetve direkt erre a célra készült terminálokhoz manapság már fillérekért hozzájuthatunk). De köthetünk soros vonalra modemet is: ekkor a felhasználói terminál a telefonvonal „túlsó végén" lesz, távolról is elérhetővé téve rendszerünket.

A hálózaton vagy grafikus felületen keresztül bejelentkezett felhasználókhoz ún. pszeudo-terminálokat rendel a rendszer, ahol is a billentyűzet és a képernyő annak a gépnek a billentyűzetéhez és képernyőjéhez rendelődik, amely előtt a felhasználó ül. A terminálok megnevezése a szakzsargonban tty (angolul betűzve ejtik), illetve a pszeudo-terminálóké pty vagy ttyp, a Linux ez utóbbi megnevezést használja. Minden Unix rendszer többféle, képességeik alapján osztályozható terminált képes kezelni: mivel nem alakult ki egységes szabvány, és a nagygépek hőskorában sokféle gyártó sokféle terminált gyártott, ezért a Unix rendszereket általában felkészítik a sornyomtató típusú, sorüzemmódú, kurzor-címzésre nem képes terminálóktól kezdve a modernebb, színkezelésre és akár ANSI grafikára képes terminálokig bezárólag sokféle termináltípus kezelésére. Az egyes terminálokat gyártó és típusra utaló megnevezéssel azonosítják: Amennyiben csak konzolról használjuk gépünket, elég annyit tudni, hogy a konzol terminál azonosítója „console".

2.5.2. Az egyes felhasználók megkülönböztetése

Az egyes felhasználók azonosítására a „login név" (account, témaszám) rendszert használja a Unix: minden felhasználónak van egy (maximum 8 karakter hosszú, konvenció szerint kisbetűvel írott) azonosítója, és ehhez tartozik a maximum 8-16 karakter hosszú jelszó. A finomabb hozzáférés-hierarchia kialakítása érdekében a felhasználókat csoportokba (groups) oszthatjuk: minden felhasználónak van egy elsődleges csoportja (pl. student), és ezenkívül tartozhat még más csoportokhoz is (pl. texusers). A csoportneveket is konvenció szerint kisbetűvel írják. Belsőleg a rendszer minden egyes felhasználóhoz az egyedi felhasználó-néven kívül még egy numerikus felhasználó- és (esetleg több) csoport-azonosítót rendel (UID - user identification és GID - group identification). Léteznek kitüntetett felhasználónevek is, illetve legalább egy, amelyik minden rendszeren megvan: ez a „root" felhasználó, a rendszergazda azonosítója, aki felelős az adott rendszer karbantartásáért és üzemeltetéséért, és akinek a rendszeren „mindent szabad": ő az, akinek a rendszer egészéhez hozzáférése van. Fel kell itt hívni a figyelmet egy kialakult, de elkerülendő rossz gyakorlatra: sokan, akik csak egyedül használják Linuxos gépüket, minden tevékenységüket „root" hozzáférési jogokkal végzik, mondván, hogy így jobban hozzáférnek a rendszerhez, nem kell mindenféle korlátozásokkal vesződniük. Nem érdemes ezt a gyakorlatot folytatni, mert egy jól bekonfigurált rendszerben normál (nem privilegizált) felhasználókent is kényelmesen elérhetünk minden szolgáltatást, amire hétköznapi felhasználás során szükségünk lehet - viszont figyelmetlenséggel vagy elgépeléssel sokkal kisebb kárt tudunk okozni. Egy root jogokkal kiadott hibás vagy át nem gondolt utasítás egy pillanat alatt az egész rendszerünket jóvátehetetlenül tönkreteheti. Megjegyzendő még, hogy „root"-tal ekvivalens felhasználót akár többet is hozhatunk létre, az összes olyan felhasználó, akinek UID-je 0 (felhasználó létrehozásakor ezt megadhatjuk) root-tal ekvivalens

lesz, de általában nem érdemes ezt tennünk: egy rendszeren éppen elég egy darab teljhatalmú felhasználó.

2.6. A file-okról

2.6.1. A Unix file koncepció

Mielőtt továbblépnénk, feltétlenül szükséges megismerkednünk még egy, az egész rendszert átható szervező elvvel: ez pedig a file koncepció. Unixban ugyanis (túlzás nélkül) az égvilágon minden file. Definíció szerint (kissé tudományosan) a file olyan „kommunikációs végpont", ahova vagy byte-folyamot (byte stream) tudunk írni, vagy byte-folyamot tudunk onnét olvasni (esetleg mindkettőt). File-ként kezelhető tehát a billentyűzet (csak olvasható), a szöveges képernyő (csak írható), a nyomtató, de még a fizikai memória tartalma is (!). A gépben lévő winchester szektorait (ha nem csak egyes file-okat akarunk elérni) is egy speciális file olvasásával illetve írásával érhetjük el, ugyanígy a hálózati eszközöket is a file-kezelés szabályainak megfelelően használhatjuk. Ez utóbbi három esetben persze szükség van a megfelelő jogosultságokra: a fizikai memória tartalmát vagy direktben a winchester szektorait „mezei" felhasználó nem láthatja. Ezt a file-orientált szervezésmódot (amely tulajdonképpen egy Unix rendszer fő szervező ereje) folyamatosan érdemes észben tartanunk: ha egyszer kellően átláttuk logikáját, sok addig bonyolultnak, lehetetlennek tartott művelet roppant egyszerűvé válik.

2.6.2. A file-rendszer struktúrája

A Unix file rendszere a DOS-hoz hasonló könyvtár rendszerű (helyesebben szólva a DOS hasonlít a Unixhoz), azzal a különbséggel, hogy itt nincsenek különböző meghajtók, hanem a rendszerben lévő minden file-t egy gyökérkönyvtárból kiindulva (ennek jelölése / könyvtár) elérhetünk. Fontos különbség még, hogy a Unix file-nevek nem a DOS-ból ismert 8+3 szabály szerint formálódnak: egy file-név max. 255 karakter hosszú lehet, és tetszőleges karaktert tartalmazhat. (Vigyázat, tartalmazhat nem nyomtatható, nem gépelhető karaktereket is! Gonoszul elnevezett file-okkal sok bosszúságot okozhatunk magunknak és felhasználótársainknak.) Fontos különbség továbbá, hogy a Unix (nemcsak a file-nevekben, mindenhol) különbséget tesz kis- és nagybetű között (case sensitive). Az ALMA, Alma, alma, almA tehát mind-mind különböző file-okat jelentenek. Mivel file-névben pont is lehet, ezért Unixban is adhatunk kiterjesztéseket a file-oknak: ezek azonban a rendszer számára semmilyen jelentéssel nem bírnak: csak a felhasználók eligazodását segíti elő, ha betartjuk a megfelelő file-elnevezési konvenciókat. Azt sem a file-névből tudja meg a rendszer, hogy például az adott file futtatható-e vagy sem: a következő részben látni fogjuk, miből is derül ez ki. Előtte azonban még egy fontos dolgot tisztáznunk kell: hogyan valósul meg az, hogy a gépben több fizikai eszköz van (floppy, esetleg több winchester), és mégis az összes file-t egy könyvtárstruktúrában látjuk? Úgy történik, hogy van egy kitüntetett diszk (vagy partíció, esetleg ramdisk) amelyen a gyökér könyvtár (/ könyvtár) található, és a többi úgynevezett filesystem-et (filesystem-nek Unix alatt az egy diszken, partíción, egy rendszerbe szervezett file-ok összességét nevezzük) pedig a file-rendszer valamely alkönyvtárába lehet beilleszteni (mounting, igazán jó magyar terminológia mindezidáig nem született rá). Tehát ha van például egy /mnt/floppy alkönyvtárunk a rendszerben, és van egy floppy diszkünk, amelyen lévő filesystem-ben a /alma, /narancs és a /alma/starking könyvtárak vagy file-ok vannak, akkor a floppy „felmount-olása" után a /mnt/floppy/alma, /mnt/floppy/narancs és a /mnt/floppy/alma/starking könyvtárakat vagy file-

okat fogjuk látni. Láttuk azt is, hogy Unixban az egyes könyvtárak szeparálására a DOS \ (backslash) karakterével ellentétben a /-t (slash) használják. Nem érdemes eltéveszteni, mert amint később látni fogjuk, a backslash karakternek is megvan a saját szerepe.

Unixban kétféleképpen adhatunk meg file elérési útvonalat, ez hasonlít a DOS-nál megszokottra: abszolút módon, a gyökérkönyvtárból indulva, amikor az elérési út neve /-rel kezdődik, vagy relatív módon, például ../../alma/starking, ami azt jelenti, hogy az aktuális könyvtárból lépjünk felfelé kettőt, és onnét lépjünk az alma könyvtárba, ha a starking-ot akarjuk megtalálni.

2.6.3. Többféle file-rendszer kezelése

A Linux képes arra, hogy többféle fizikai és logikai szervezésű filesystem-et egy könyvtárszerkezetben kezeljen: támogatja többféle Unix-os filesystem formátum mellett a DOS FAT file-rendszert, tudja olvasni az OS/2 HPFS file-okat, ismeri a CD-s file-formátumokat, és tudja kezelni a TCP/IP hálózat felett működő hálózati file-rendszert, az NFS-t is.

2.6.4. További érdekességek a file-rendszerről. A link

Még két Unix-os specialitásról kell beszélnünk: az egyik az, hogy abból hogy /mnt/floppy/narancs, nem derül ki, hogy ez a narancs egy könyvtár-e vagy egy file - ennek eldöntéséhez a file neve egyedül nem elég. A másik specialitás ismertetéséhez kicsit jobban bele fogunk mászni a Unix file-rendszerek rejtelmeibe. A tárgyalt sajátosság a „link”, az a tulajdonság, hogy egy file-nak egyszerre több neve is lehet: több néven is tudunk ugyanarra a file-ra hivatkozni (pl. megcsinálhatjuk, hogy a /gyumolcs/alma és a /holnap/reggeli file-nevek fizikailag ugyanarra a file-ra mutassanak). Ez pedig úgy lehetséges, hogy egy Unix filesystemben minden egyes file-hoz tartozik egy inode-nak nevezett rekord, amely tartalmazza a file nevét, létrehozásának dátumát és minden egyéb, a file-lal kapcsolatos adatot, kivéve a file tartalmát - többek között azt az információt is, hogy fizikailag a diszk melyik szektorában kezdődik az a file. Létrehozhatunk egy újabb inode-ot, amelyben a file neve más, de a kezdő szektor száma azonos: és máris két helyről hivatkozhatunk ugyanarra a file-ra. Törléskor pedig mindaddig csak az inode-okat töröljük, amíg el nem érünk az utolsóhoz, amikor már csak egy darab inode címzi meg a file-t: ennek törlésekor szabadítjuk fel a file által elfoglalt diszkhelyet. Az ilyen típusú linket „hard link”-nek nevezzük, és van egy hátránya: csak ugyanazon a diszken illetve partíción lévő file-ra tudunk ilyet létrehozni, mert az inode-on belül nem tudunk mondjuk egy cserélhető floppy vagy CD-ROM szektorszámaira hivatkozni - egy másik diszkről. Ezért találták ki az ún. szimbolikus vagy „soft link”-et, amely szintén egy link, de úgy működik, hogy a link maga egy speciális file, amiben a hivatkozott file neve van. Itt általában érdemes relatív nevekkel dolgozni, így később egész könyvtárszerkezetek, részfák másolásakor, mozgatásakor kevesebb problémába ütközünk. Mozgatás alatt az is értendő, ha az adott filesystem-et a könyvtárstruktúra egy más pontjára illesztjük be!

Láthattuk, hogy a .. itt is a szülő könyvtárra hivatkozás, a . pedig logikusan az aktuális könyvtárra utal. Unixban (mint minden), a könyvtárak is file-ként jelennek meg - ennek az is az előnye, hogy a könyvtár neve, a könyvtárban lévő . file és a könyvtár közvetlen alkönyvtáraiban lévő .. hivatkozások mind-mind ugyanarra az inode-ra, a könyvtár inode-jára mutató hard linkek. Ugye milyen logikus?

Ennyi elméleti alapozás után már éppen itt az ideje, hogy billentyűzetet ragadjunk, és elkezdjük egy Linux rendszer felfedezését a gyakorlatban is!

3. Hogyan használjuk?

3.1. Jelentkezzünk be!

Most már (elméletben) elég jól ismerjük a rendszer felépítését: kezdjük el használni is. Egy Linux gép terminálja elé ülve valami hasonlót láthatunk:

```
Welcome to Linux 1.3.10

claudius login:
```

Ez a gépen futó kernel verziója, illetve a gép neve (a Unix-os gépeknek nevük van!), valamint a login: prompt jelzi azt, hogy a terminál kész a bejelentkezés fogadására. (Ezek a bejelentkezés előtti üzenetek természetesen konfigurálhatók). Gépeljük be tehát a user-nevünket és a jelszavunkat, és pár sor újabb üzenet után (pl. hogy mikor jelentkeztünk be ezelőtt utoljára), megkapjuk a parancsértelmező prompt-ját, jelen esetben:

```
claudius:~$
```

Jelentése: legelőször áll a gép neve, utána kettősponttal elválasztva az aktuális könyvtár (a tilde jel egy rövidítés - a felhasználó ún. home könyvtárát jelenti, amiről mindjárt megtudjuk mi is az), a prompt végét pedig egy $ karakter jelzi. Ha root privilégiumokkal jelentkeztünk volna be, a prompt végén egy # karaktert látnánk - a rendszer ezzel is figyelmeztet különleges hatalmunkra. A home könyvtárról: a rendszeren minden felhasználónak van egy otthona, ahova bejelentkezéskor a rendszer az aktuális könyvtárat (current working directory) állítja: ez a könyvtár a felhasználó tulajdonában van, itt tartja file-jait, ide dolgozik, és ha akarja, el is rejtheti a könyvtár tartalmát a többi felhasználó kíváncsi szeme elől. Ez Linuxokon általában a /home/felhasznalo_nev vagy ritkábban a /usr/users/felhasznalo_nev nevű könyvtár. Rövidítve - mint már láttuk - a ~ jellel hivatkozhatunk rá.

3.2. A parancsértelmező, avagy shell

3.2.1. Miért van rá szükség?

Egy Unix rendszermag szigorúan véve nem tartalmaz felhasználó-interakciót: a kernel csupán arra képes, hogy futtasson más programokat, illetve hozzáférést biztosítson a rendszer erőforrásaihoz - ezen szolgáltatásait azonban programozói szinten, ún. rendszerhívások (system calls, vagy röviden syscalls) formájában kínálja. Már a rendszer indításakor szükség van tehát egy (illetve több) programra, amely biztosítja a bejelentkezés lehetőségét, illetve bejelentkezéskor is minden felhasználó számára el kell indítani egy programot, amely lehetővé teszi a felhasználó számára, hogy a rendszerrel kommunikáljon, más programokat indítson el stb. Ez a program a parancsértelmező vagy shell (szokás - elég szerencsétlenül - burokprogramnak is fordítani), amely nem csinál mást, minthogy olvassa a felhasználó billentyűzetéről érkező parancsokat, és ezeket a kernel számára emészthető rendszerhívásokká alakítja, a végeredményről pedig a képernyőn tájékoztatja a felhasználót. Fontos megjegyezni, hogy ilyen szempontból a shell semmiféle speciális tulajdonsággal nem bír: tulajdonképpen bármely programot megadhatunk ún. default shell-nek (cifra magyarsággal alapértelmezés szerinti parancsértelmező). Ha azt akarjuk, hogy valamely felhasználó csak a „vi” szövegszerkesztőt legyen képes futtatni, állítsuk be számára default shell-ként a „vi” programot: ekkor bejelentkezés

után rögtön a „vi” indul el, a szövegszerkesztés befejezése (a „vi”-ből kilépés) után pedig a felhasználó visszakerül a login: prompthoz. Sőt, olyan programot is megadhatunk default shellnek, amely egyáltalán nem olvas a billentyűzetről és nem ír a képernyőre - bár szegény felhasználó számára a bejelentkezések nem lesznek túl érdekfeszítőek - a program lefutását is csak onnét veszi észre, hogy visszakerült a login: prompthoz. Egy hasznos információ: a rendszerből kilépni a shell promptnál a „logout” paranccsal lehet. Hasonló az „exit” parancs is, azonban ez csak az első, a „login shell” után indított shellekből léptet ki, a rendszerből teljesen kilépni „logout”-tal kell.

3.2.2. Milyen shellek vannak?

Általában persze nem valamilyen célprogramokat adnak meg shell-ként, illetve célprogramokat: shell célprogramokat, amelyek fő célja, hogy a felhasználót segítsék a rendszer használatában (ami más programok indítását jelenti). A Unix hosszú története során a sokféle fejlesztő és felhasználó ízlését tükrözvén, többféle shell program alakult ki: a legelterjedtebbek: a legelső, az ősi „sh”, a Bourne által megírt shell, amely egyfajta közös nevezőt jelent, mert minden Unix rendszer tartalmazza, aztán az sh továbbfejlesztett és C programozók ízlésvilágához igazított változata a „csh”, a sokféle funkciót magában foglaló Korn shell, a „ksh”, valamint a csh kényelmi és kiterjesztett funkciókkal jól megpakolt változata a „tcsh”, és végül a Linux rendszereken általánosan használt „bash”, a Bourne Again SHell, amely az sh egy alaposan továbbfejlesztett változata. A shell lehetőségei nem merülnek ki csupán interaktív parancs-futtatásban: mindegyik shell-nek egy saját kis programnyelve van, amelyben interpretált kis programocskákat (ún. shell script-eket) írhatunk. A shell script programozással később részletesen foglalkozunk majd.

3.2.3. Környezeti változók

Fontos tulajdonsága még a shellnek, hogy definiálhatunk benne (a DOS-hoz hasonló) ún. környezeti változókat (enviroment variables), amelyek közül a fontosabbak: a PATH változó, amely megmondja, mely könyvtárakban keresse a rendszer a futtatható file-okat, a HOME változó, amely a home könyvtár útvonalát tartalmazza, a LOGNAME és a HOSTNAME változók, amelyek a login nevet illetve a gép nevét tartalmazzák, illetve a TERM változó, amely a használt terminál típust azonosítja.

3.2.4. Különbségek az egyes shellek között

Alapvető vonásaiban minden shell azonos: programot elindítani mindegyikben a programnév begépelésével lehet, mindegyik azonosan értelmezi a file-név metakaraktereket, és végül is amit egyik shellben meg tudunk csinálni, az több-kevesebb nehézséggel egy másikban is elérhetjük - bár néha eltérő szintaktikával. Például a környezeti változók állítása már nem ugyanazzal a szintaktikával történik az egyes shellekben, az elért hatás azonban ugyanaz. Lényegi különbség nincs igazán a shellek között, talán csak annyi, hogy az újabbakban sok olyan - gépelést megtakarító, gyorsító - funkció van amelyek a régebbi shellekben még nem voltak meg. A továbbiakban bash-ban fogunk dolgozni, mert - bár létezik Linuxra az összes említett shell - mégis a Linux világban ez a legelterjedtebb.

Térjünk tehát vissza gépünk promptjához, amelyről most már tudjuk, hogy a shell írja ki (a ~ rövidítést is a shell értelmezi).

3.2.5. Néhány hasznos ismeret a bash-ról

A bash számos kényelmi funkciója közül néhány: Az előzőleg begépelt parancsokat visszahívhatjuk a felfelé nyíllal, a parancssort szabadon szerkeszthetjük (mozogni a nyílbillentyűkkel lehet). Előző parancsot visszahívni még a felkiáltójellel is lehet: a ! mögé rövid sztringet írva megkeresi a legutóbbi olyan parancsot, amely azzal a sztringgel kezdődött, és azt futtatja le. A ! használatával előhívott parancs egyébként módosítható, át is szerkeszthető. File-név kiegészítést a „Tab”-bal kérhetünk: ennek hatására a shell megpróbálja kiegészíteni az eddig begépelt file-név kezdeményt. Ha ez egyértelműen lehetséges, beírja a file-nevet, ha nem, sípol egyet, és a „Tab” újbóli megnyomására megkapjuk a lehetséges file-nevek listáját. A file-név kiegészítés valamennyire intelligens: sor elején (első szóban) csak futtatható file-okra, parancsok nevére egészít ki (viszont a PATH-ot is végignézi), argumentumoknál viszont az aktuális könyvtár (vagy a megkezdett file-névnek megfelelő könyvtár) összes file-jára működik.

Még két, a bash-hoz tartozó file-ról érdemes beszélnünk, mindkettő a home könyvtárunkban van, és shell parancsokat tartalmaz, amelyeket a bash bizonyos körülmények között végrehajt. Az első a .profile, amelyet bejelentkezéskor hajt végre a rendszer, a másik pedig a .bashrc, amelyet minden bash indításkor (tehát akkor is, ha már bejelentkezünk és újabb shellt indítunk). A .profile-ban szokás például a PATH és más globális változók értékét saját ízlésünk szerint konfigurálni, a .bashrc-ben pedig például „alias”-okat definiálhatunk: „alias”-ok segítségével parancsoknak más neveket adhatunk, hosszú parancsoknak rövideket vagy nehezen megjegyezhető nevűeknek valami memorizálhatóbbat. DOS mániákusok például írhatják a következőt:

```
claudius:~$ alias dir='ls -l'
```

Ezután a „dir” parancs jelentése „ls -l” lesz. Az eddig (vagy előre) definiált alias-okat a paraméterek nélkül kiadott „alias” paranccsal nézhetjük meg, egy alias-t kitörölni pedig az „unalias alias_nev” paranccsal lehet. „Folytatható” alias-t (ahol az alias helyettesítése után az alias név után írt szavakat is továbbviszi a shell), az alias végére írt $* szimbólummal definiálhatunk.

3.3. Parancsok futtatása

3.3.1. Az ls parancs

Már tudjuk, hogy programot indítani nevének begépelésével lehet. Ha csak a file-nevet adtuk meg, akkor a rendszer végignézi a PATH változóban szereplő könyvtárakat, és ha valamelyikben megtalálja a file-t és ha az futtatható, akkor elindítja. Vigyázat! Az aktuális könyvtárban csak akkor nézi meg, ha a PATH-ban meg van adva a . is, mint könyvtár. Ha a file-nevet útvonallal együtt adtuk meg, akkor csak az adott helyen lévő file-ot próbálja meg végrehajtani (pl. /bin/ls, vagy hivatkozás az aktuális könyvtárban lévő file-ra: ./ls). Futtassuk tehát az ominózus ls parancsot, amely hasonlóan a DOS „dir” parancsához, kilistázza az adott könyvtárban lévő file-ok neveit:

```
claudius:~$ ls
LaTeX/          egyeb/                  pi*
Mail/           hun.kbd                 xtermdos.zip
News/           linuxmop-0.01.README    yp-clients-1.5.patch
WWW/            linuxmop-0.01.tar.gz
ch-1-k.ps       lx-1.3.tgz
```

A parancs a file-neveket abc rendben írja ki. A file-név mögött lévő / azt jelenti, hogy egy alkönyvtárról van szó, a * pedig futtatható file-t jelez. Ezeknek az információknak a jelzéséhez az ls-t külön paraméterezni kell: alapértelmezés szerint csak a file-ok neveit írná ki.

3.3.2. Ismét pár szó a környezeti változókról

Ezek a paraméterek előre be vannak állítva munkakörnyezetünkben: az ls parancs opcióként használja az LS_OPTIONS környezeti változóban beállított sztringet. Nézzük meg:

```
claudius:~$ echo $LS_OPTIONS
--8bit --color=tty -F -T 0
```

Az egyes opciók jelentésével nem kell most törődnünk - elég azt látni, hogy környezeti változó értékét az „echo” paranccsal írathatjuk ki, a változó neve elé egy $ jelet írva (környezeti változók esetén a változó neve elé írt $-ral érhetjük el, hogy a változó neve mint sztring helyébe a shell helyettesítse a változó tartalmát). Nézzük meg néhány további változó értékét:

```
claudius:~$ echo $HOME
/usr/users/tiv
claudius:~$ echo $LOGNAME
tiv
claudius:~$ echo $PATH
/usr/local/bin:/bin:/usr/bin:.:/usr/X11/bin:/usr/andrew/bin:
/usr/openwin/bin:/usr/games:.:/usr/TeX/bin
```

Az „echo” paranccsal egyébként nemcsak környezeti változók értékét írathatjuk ki: a parancs ugyanis nem csinál mást, minthogy az argumentumként kapott sztringe(ke)t egyszerűen kiírja - esetünkben is a változó-helyettesítést a shell végzi. Környezeti változónak értéket adni a valtozo_nev=ertek utasítással lehet. Például:

```
claudius:~$ valtozo=valami
claudius:~$ echo $valtozo
valami
```

Megkülönböztetünk ún. lokális (csak az éppen futó shell alatt érvényes) változókat, illetve globális, egész bejelentkezésünk ideje alatt minden indított programban érvényes változókat (például a PATH-nak ilyennek kell lennie). Környezeti változót globálissá az „export” utasítással tehetünk. Egy könyvtár hozzáadása a PATH-hoz:

```
claudius:~$ PATH=/alma/jonatan:$PATH
claudius:~$ export PATH
```

Vagy akár össze is vonhatjuk:

```
claudius:~$ export PATH=/alma/jonatan:$PATH
```

Mint látható, a PATH-ban az egyes könyvtárnevek kettősponttal vannak elválasztva. Ha a behelyettesítendő környezeti változó után rögtön szöveget akarunk írni (mondjuk a környezeti változó egy sztring része), akkor a ${valtozo_nev} szintakszist használjuk, ezzel világosan el lehet különíteni a változó nevét a körülvevő egyéb karakterektől. Az éppen definiált összes környezeti változó nevét és értékét az „env” paranccsal írathatjuk ki.

3.3.3. Még pár szó az ls parancsról

Térjünk most vissza az ls parancshoz! Az ls tipikus Unix parancs, amennyiben egy kis, rövid önálló futtatható program, amelynek argumentumokat és opciókat adhatunk meg, és futása végeredményét a kimenetre (jelen esetben ez a képernyő) írja. Nézzünk meg néhány gyakran használt opciót:

```
claudius:~$ ls -a
./                Mail/                pi*
../               News/                xtermdos.zip
.bash_history     WWW/                 yp-clients-1.5.patch
.elm/             ch-1-k.ps
.netrc            egyeb/
.newsrc           hun.kbd
.profile*         linuxmop-0.01.README
.rhosts           linuxmop-0.01.tar.gz
LaTeX/            lx-1.3.tgz
```

Most az előzőeken kívül kiírt még egy csomó ponttal kezdődő nevű file-t is: ezek a file-ok az előbb is ott voltak a könyvtárban, csak az ls alapértelmezés szerint nem írja őket ki: konvenció szerint ezen file-ok legtöbbje ún. inicializáló file, amelyekre normál munkánk során általában nem vagyunk kíváncsiak. Például a .profile egy olyan file, amelybe azokat az utasításokat írjuk, amelyeket szeretnénk hogy minden bejelentkezéskor végrehajtson a rendszer: tudjuk hogy ez a file ott van, de nem szeretnénk, hogy minden egyes ls kiadásakor foglalja a képernyőn a helyet. Hasonló a helyzet a többi ponttal kezdődő file esetén is.

Még egy fontos ls opció:

```
claudius:~$ ls -l
drwxr-xr-x 2 tiv users 1024 Jun 10 11:10 LaTex/
drwx------ 2 tiv users 1024 Jul 18 23:27 Mail/
drwx------ 2 tiv users 2048 Jun 30 11:24 News/
drwxr-xr-x 3 tiv users 1024 May 15 14:56 WWW/
-rw-r--r-- 1 tiv users 203839 Jun 5 13:31 ch-1-k.ps
drwxr-xr-x 2 tiv users 1024 Jan 15 1995 egyeb/
-rw-r--r-- 1 tiv users 613 Apr 2 15:52 hun.kbd
-rw-r--r-- 1 tiv users 136 Jun 29 17:39 linuxmop-0.01.README
-rw-r--r-- 1 tiv users 159303 Jun 29 17:37 linuxmop-0.01.tar.gz
-rw-r--r-- 1 tiv users 1374657 Jun 21 15:08 lx-1.3.tgz
lrwxrwxrwx 1 tiv users 17 Jul 20 09:01 vmessages -> /var/adm/messages
-rw-rw-r-- 1 tiv users 276578 May 25 09:10 xtermdos.zip
-rw------- 1 tiv users 577 Feb 8 14:31 yp-clients-1.5.patch
```

3.3.4. A file-típusok

„-l”-lel (long listing) minden egyes file-ról „bő” (verbose) információt kaphatunk. Az egyes mezők jelentése:

A legelső karakter a file típusára utal:

- d ha a file egy könyvtár (directory)
- b ha a file block típusú speciális file
- c ha a file karakter típusú speciális file
- l ha a file szimbolikus link
- s ha a file socket
- \- ha a file „egyszerű” (plain) file

(A b,c,s típusú file-okat később majd tárgyaljuk: egyelőre fogadjuk el, hogy ilyenek is vannak).

3.3.5. A hozzáférési jogosultságok

A következő 3*3 karakter a file hozzáférési jogosultságait adja meg: az első három karakter jelenti, hogy a file tulajdonosa mit tehet a file-lal:

r ha olvashatja(read)
w ha írhatja(write)
x ha végrehajthatja (execute)
- ha az adott jog nincs számára megadva

A következő három karakter ugyanilyen formában jelenti a file-tulajdonos csoport (group) jogait, vagyis hogy a file tulajdonos csoportba tartozó felhasználók mit tehetnek az adott file-lal, az utolsó három karakter pedig a külvilág (others), az egyéb felhasználók jogait jelenti. A file tulajdonosa természetesen tetszés szerint megadhatja a hozzáférési jogokat, nemsokára látni fogjuk hogyan. Azt tehát, hogy egy file futtatható-e (az operációs rendszer megkísérelheti-e futtatni) Unix alatt nem kiterjesztés, vagy más egyéb file-névbe olvasztott információ dönti el, hanem a megfelelő x attribútum. Az is igaz, hogy ha egy file-ra csak x jogunk van, akkor csak futtatni fogjuk tudni azt a file-t, olvasni nem - tehát nem fogjuk tudni lemásolni sem! Mint már láttuk, szöveges file is lehet futtatható: a shell parancsokat tartalmazó shell script-ek is egyszerű szöveges file-ok, beállított x attribútummal. Ha egy ilyen file-t indítunk el, a shell egyszerűen soronként olvassa és hajtja végre a beleírt utasításokat, hasonlóan a DOS-os .BAT file-okhoz. A végrehajtási jog („x") könyvtárak esetén a könyvtárban való keresés jogát jelenti. Ha valamely műveletünk megfelelő jogosultságok hiányában meghiúsul, a sokat átkozott „Permission denied." hibaüzenetet láthatjuk.

3.3.6. A file-okhoz tartozó egyéb információk

A jogok után következő mező a sorban az úgynevezett link számláló (link count): megadja, hogy az adott filesystem-en hány hard link mutat az adott file-ra (tulajdonképpen megmondja, hogy hány néven hivatkozhatunk a file-ra). Ez a számláló csak a hard linkek számát mutatja - szimbolikus linkek számolására nincs lehetőség.

A következő mező a file tulajdonos user-neve (jelen esetben „tiv"), a következő pedig a tulajdonos csoport azonosítója („users"). Ezután a file hossza következik byte-okban, majd az ún. „mtime" mező, a file utolsó módosításának dátuma. Legvégül a file neve látható. Szimbolikus linkeknél (a fenti példában a „vmessages" file ilyen) a -l opció hatására egy -> nyíl után az ls kiírja, melyik másik file-ra mutat a szimbolikus link.

3.3.7. Unix parancsok paraméterezése

Az ls parancsnak azt is megadhatjuk, melyik könyvtárat listázza (ha nem adunk meg semmit, az aktuális könyvtár tartalmát írja ki). A gyökér-könyvtár és mondjuk a /usr/users/tiv/ch-1-k.ps file lehető legteljesebb listázása így nézhet ki:

```
claudius:~$ ls -la / /usr/users/ch-1-k.ps
```

Argumentumként akár több file- vagy könyvtárnevet is megadhatunk, és amint láttuk, az opciókat összevonhatjuk. Általánosságban, egy tipikus Unix parancs hívási formája a következő:

```
parancsnév [opció(k)] [argumentum(ok)]
```

Ha egy paramétereket igénylő parancsot paraméterek nélkül, vagy nyilvánvalóan hibás paraméterezéssel hívunk meg, általában a parancs rövid segítséget ír ki a helyes paraméterezésről.

3.4. Még néhány alapvető file-kezelő parancs

Aktuális könyvtárat a „cd” paranccsal válthatunk - a DOS-tól eltérően argumentumok nélkül kiadva azonban nem az aktuális könyvtárat írja ki, hanem visszavisz bennünket a home könyvtárunkba. Az aktuális könyvtár nevét a „pwd” (print working directory) paranccsal kérdezhetjük le:

```
claudius:~$ pwd
/usr/users/tiv
claudius:~$ cd /usr/local
claudius:/usr/local$ cd ../..
claudius:/$ pwd
/
claudius:/$ cd
claudius:~$ pwd
/usr/users/tiv
```

File-ot másolni a „cp” (copy) paranccsal tudunk, („cp [opciók] mit hova” szintaktikával). Hasznos opciói a -r (rekurzív másolás) illetve a -i (interactive) amely ha már létezik a cél-file, felülírása előtt rákérdez, biztosak vagyunk-e a dolgunkban.

File mozgatása illetve átnevezése az „mv” (move) paranccsal lehetséges: az argumentumok sorrendje ugyanolyan mint a cp-nél. A mv parancs használható egész könyvtárstruktúrák mozgatására is, de csak egy filesystem-en belül.

Könyvtárat létrehozni a „mkdir dir_nev” utasítással lehet, letörölni pedig az „rmdir dir_nev” paranccsal. Törölni csak üres könyvtárat lehet.

Linkeket az ln [-s] letezo_filenev uj_filenev paranccsal hozhatunk létre - ha megadjuk a -s opciót, szimbolikus link keletkezik, ha nem akkor hard link.

File-okat törölni az „rm” (remove) utasítással lehet. Figyelem! Unix rendszerekben letörölt file visszaállítására NINCS LEHETŐSÉG, legfeljebb biztonsági másolatból. Ezért az rm használata előtt mindig kétszer gondoljuk meg, valóban meg akarunk-e szabadulni attól a file-tól. Hirtelenkezű felhasználóknak találták ki az rm -i opciót, amely minden egyes file letörlése előtt még egyszer visszakérdez. Megfelelő önbizalommal rendelkezők használhatják a -f opciót: ez viszont nem kérdez semmit, sőt ha esetleg nem lenne jogunk a file törlésére, akkor sem szól vissza: amit tud, letöröl csendben, amit nem, azt ott hagyja. Amennyiben egy file a mi tulajdonunkban áll, de (talán elővigyázatosságból?) levettük róla a „w” jogot (írás - ennek számít a törlés is), a „sima” rm nem törli le a file-t, hanem megkérdezi, felülbírálja-e a törlési jog hiányát (saját file esetén ezt megtehető). A -f opció viszont nem kérdez semmit: átállítja a jogokat és töröl szó nélkül (amennyiben persze teheti). Az rm, mkdir, rmdir parancsoknak több file-nevet is megadhatunk argumentumként: az „rm a b c” mindhárom file-t törli. Ez veszélyes is lehet: az „rm a*” utasítás törli az összes „a”-val kezdődő nevű file-t (a metakarakterekről bővebben majd később), a gépelési hiba folytán keletkezett „rm a *” parancs pedig először törli az „a” nevű file-t (ha van ilyen), aztán pedig az aktuális könyvtárban lévő összes többit... (a * minden file-névre illeszkedik). Van az rm-nek még egy gyakran használt, de nagyon veszélyes opciója, ez pedig a -r (rekurzív törlés). Egész könyvtárstruktúrákat törölhetünk vele pillanatok alatt. És - ezt nem lehet elégszer hangsúlyozni: ha egyszer valamit letöröltünk, nincs visszaút!

3.5. File-ok tulajdonságainak manipulációja

3.5.1. File attribútumok

File-attribútumok megváltoztatása, további attribútumok: Mint említettük, a file tulajdonosa tetszés szerint beállíthatja a file védelmi attribútumait: erre szolgál a „chmod” parancs, melynek formátuma:

```
chmod [opciók] [ugoa...][+-=][rwx...] file_nev
```

(Az opciók közül most a -R érdemel említést: ezzel a parancs működése a könyvtárstruktúrában rekurzívvá tehető.) Az egyes betűk jelentése:

u user: a file tulajdonosra vonatkozó jogok
g group: a csoportjogok
o other: mindenki más jogai
a all: mindhárom csoportra vonatkozó művelet

A műveletek:

\+ jog (attribútum) megadása
\- a jog megvonása
= az attribútumok legyenek a maszk-kal pontosan egyenlők

Ezután pedig az attribútumokból képzett maszk következik. Néhány példa:

```
claudius:~$ ls -l alma
-rw-r--r-- 1 tiv users 5 Jul 20 10:14 alma
claudius:~$ chmod u-w alma
claudius:~$ ls -l alma
-r--r--r-- 1 tiv users 5 Jul 20 10:14 alma
claudius:~$ chmod g=rwx alma
-r--rwxr-- 1 tiv users 5 Jul 20 10:14 alma*
claudius:~$ chmod a=r alma
claudius:~$ ls -l alma
-r--r--r-- 1 tiv users 5 Jul 20 10:14 alma
claudius:~$ chmod u+w alma
claudius:~$ ls -l alma
-rw-r--r-- 1 tiv users 5 Jul 20 10:14 alma
```

Még két (pontosabban három) attribútumot fontos itt megemlítenünk: az első az ún. „sticky” attribútum (jele „t”), amely futtatható file esetén azt jelenti, hogy a program lefutása után az operációs rendszer a program kódját megpróbálja a memóriában tartani, így legközelebbi indításkor nem kell megvárni a program betöltődését. Ezt a funkciót azonban nem minden Unixban valósították meg, és napjainkban a „demand paging” technikák miatt egyre inkább jelentőségét veszti. Legjobb tudomásom szerint ez a funkció Linuxban sincs implementálva. Ha a „t” attribútum könyvtárra van kiadva, azt jelenti, hogy hiába van az egész könyvtárra írási jogom, mégis csak a saját tulajdonomban lévő file-okat törölhetem le. Amennyiben ugyanis egy könyvtárra írási jogom van, ebből a könyvtárból törölhetem a mások által létrehozott file-okat is, még akkor is, ha magukra a file-okra nincs írási jogom. Viszont ha a „sticky bit” be van

állítva, csak a saját file-jaimat pusztíthatom. A másik két fontos attribútum még a „set-UID” és a „set-GID” bitek: jelentésük, hogy (futtatható programnál van értelme használni őket) a futó program a program file tulajdonos felhasználó vagy csoport jogaival fog rendelkezni (de csak a futás idejére!) függetlenül attól, ki is indította el. Hogy mire jó ez? Képzeljük el, hogy valaki meg akarja változtatni a jelszavát (amire a „passwd” parancs való)! Ekkor át kell írni a csak root által írható kódolt jelszófile-ban a felhasználó jelszavát - de hogy ezt bármelyik felhasználó megtehesse, és ne kelljen minden jelszóváltáskor a rendszergazdát felkeresni, a „passwd” programnak root jogokkal kell futnia. Természetesen a „passwd” programba bele vannak építve a megfelelő biztonsági mechanizmusok arra, hogy mindenki csak a saját jelszavát legyen képes módosítani.

A rendszer belsőleg ezeket a file attribútumokat (file permission bits) numerikusan tárolja, és a chmod parancsnak is megadhatjuk numerikusan, mégpedig oktális szám formájában. Például:

```
claudius:~$ chmod 467 alma
claudius:~$ ls -l alma
-r--rw-rwx 1 tiv users 5 Jul 20 10:14 alma
```

Amiből észrevehetjük, hogy az egyes számjegyek jelentik az egyes felhasználó(csoport)ra vonatkozó engedélyeket, kiszámításuk pedig: x = 1, w = 2, r = 4, és az egyes számjegyekbe a kívánt módok összegét kell írni.

3.5.2. File-ok tulajdonjogának állítása

File-ok tulajdonjogát a „chown [-R] user file” paranccsal állíthatjuk, (ezt csak a root teheti meg, saját magunk nem adhatunk át file-t másoknak), a csoport tulajdonjogot pedig a „chgrp [-R] group file” utasítással állíthatjuk. Ezt az utasítást felhasználók is használhatják, amennyiben mindkét csoport tagjai, és a file tulajdonosai is egyben.

3.6. Egy gyakran használt parancs: a man

Unix alatt is rendelkezésre áll egy on-line help rendszer, amelyet a „man” (manual pages - kézikönyv) utasítással hívhatunk elő. Ez, mint egy valódi kézikönyv, nem arra szolgál, hogy ebből sajátítsuk el a Unix használatát - sokkal inkább referencia jellegű, az egyes parancsok, függvények, file-formátumok rövid, de pontos ismertetésével. Ha tudjuk, hogy könyvtár tartalmát az „ls” paranccsal listázhatjuk ki, de nem emlékszünk melyik opciójával kaphatjuk meg a bő (long) listát, csak gépeljük be: „man ls” és máris megkapjuk a ls parancs teljes és minden opcióra kiterjedő leírását. A manual, mint egy rendes kézikönyv, fejezetekre van osztva, és ha esetleg előfordul két azonos címszó, a fejezetszám megadásával tehetjük kérdésünket egyértelművé.

Az egyes fejezetek:

- A shell-ből, átlagfelhasználó számára használható parancsok (segédprogramok, utility-k, mint az ls és a chgrp)
- A kernel rendszerhívásai, C szintakszis szerint megadva - programozók számára érdekes
- A rendszerkönyvtárakban (programkönyvtárakról - libraries - van szó) található könyvtári függvények leírásai, szintén C programozóknak
- A rendszer néhány fontosabb állományának formátuma

- Szintén állomány- és adatformátumok
- A rendszeren található játékprogramok leírásai
- Perifériaállományok, speciális file-ok formátumainak specifikációja, a különböző periféria-vezérlő parancsok leírása. Rendszerprogramozók számára emészthető formában.
- A rendszeradminisztrátor számára futtatható parancsok.

Néhány fejezet alfejezetekkel is rendelkezik, pl. 3x, 3tcl ahol is ezek a jelölések az egyes programcsoportok közti jobb eligazodást segítik. Fejezetre hivatkozni a man parancsban a kért lap (manual page) elé írt fejezetszámmal lehet: létezik például a „chown" utasítás mellett chown rendszerhívás is: az utasításról a „man 1 chown" paranccsal, míg a rendszerhívásról a „man 2 chown"-nal kaphatunk segítséget. A man parancs kimenete valamilyen tördelőprogram segítségével kulturáltan, képernyőoldalakra tördelve jelenik meg - általában a „space" billentyűvel lapozhatunk előre egy képernyőoldalnyit, „b"-vel lapozhatunk vissza, „Enter-rel" görgethetünk csak egy sort, és „q"-val léphetünk ki. A „h"-val a lapozó program parancsairól kérhetünk gyors segítséget.

Minden manual oldalnak van egy egysoros címe, ezekben keresgélhetünk is a „man -k szó" paranccsal: ekkor kilistázódnak azok a manual oldalcímek, amelyek tartalmazzák a keresett szót - először a címhez tartozó címszót, aztán zárójelben a fejezet számát, végül az oldal rövid leírását kapjuk meg. Tanácsos minden eddig említett és ezután említendő parancsot, s file formátumot megnéznünk a „man"-ban: az itt csak megemlített lehetőségekről részletes magyarázatot, az alkalmazáshoz további példákat találhatunk.

3.7. I/O átirányítás a Unixban

3.7.1. A standard file-leírók

A Unix rendszer file-orientált szervezése a parancsok futtatásakor is megmutatkozik: minden egyes futó parancshoz hozzárendelődik három file:

a standard input (stdin)
a standard output (stdout)
és a standard error (stderr)

Alapértelmezés szerint ezen file-ok hozzárendelése a következő: a standard input a terminál billentyűzetéhez van rendelve, a standard output és a standard error pedig a képernyőhöz. Egy tipikus program pedig a következőképpen fut: olvassa a bemenetét a standard inputról, ebből előállít valamilyen eredményt, amelyet a standard outputra ír, az esetlegesen előforduló hibákat pedig a standard error file-ba írja. Ezeket a standard file-hozzárendeléseket a shell segítségével könnyen megváltoztathatjuk, például ha azt akarjuk, hogy az ls parancs ne a képernyőre, hanem mondjuk a „lista" nevű file-ba írja a kimenetét, használjuk a „ls >lista" parancsot! Ekkor, ha már létezett „lista" nevű file, az felülíródik: ha a kimenetet az eddigi file-hoz hozzá akarjuk fűzni, használjuk a „ls >>lista" formulát. A standard input megváltoztatása hasonlóan egyszerű: a „parancs <bemenet" hatására a „parancs" program a bemenet nevű file-ból fogja venni inputját. A standard hiba átirányítása: „program 2>hiba", ennek hatására a hibakimenet a „hiba" file-ba kerül. Természetesen lehetőség van egyszerre többféle átirányításra is: akár a „parancs <bemenet >>kimenet 2>hiba" utasításnak is van értelme. Sőt, még azt is megcsinálhatjuk, hogy a standard error-t „egybeirányítjuk" a standard outputtal, és a kettőt egy közös file-ba írjuk: „parancs >file_nev 2>&1". Ezek a dolgok (standard file-ok átirányítása,

standard file redirection) már erősen shellfüggő szintaktikájúak: bár mindegyik shellben meg tudjuk ugyanezeket csinálni, az itt ismertetett szintakszis csak a „bash”-ben működik.

Mire lehet ezeket a lehetőségeket használni? Van például a Unixban egy „more” nevű programocska, amely annyit tesz, hogy olvassa a standard inputját, és az olvasottakat képernyőnként tördelve írja ki a kimenetre, minden képernyőváltás között egy space-leütést várva. (A „more”-ból a „q” leütésével léphetünk ki). A „man”-nál már találkozhattunk is vele. Tegyük fel, hogy sok-sok file van a könyvtárunkban, és szeretnénk őket alaposan végignézni, de az ls parancs kimenete mindig kifut a képernyőről.

Tehetjük a következőt:

```
claudius:~$ ls -l >tmpfile
claudius:~$ more <tmpfile
[nézelődés]
claudius:~$ rm tmpfile
```

Egy sorba több parancsot pontosvesszővel elválasztva írhatunk, tehát a fentieket összevonhatjuk:

```
claudius:~$ ls -l >tmpfile ; more <tmpfile ; rm tmpfile
```

Ekkor a shell megvárja az előző parancs lefutását, és utána indítja a pontosvessző utáni parancsot. De ez a megoldás még mindig borzasztóan esetlen, Unixban pedig minden olyan egyszerű és kézenfekvő: biztos lehet ezt egyszerűbben is csinálni!

3.7.2. Programok összekötése

Erre találták fel a „pipe” (csővezeték) intézményét, amely nem csinál mást, mint a pipe bal oldalán álló program standard outputját hozzáköti a pipe jobb oldalán lévő program standard inputjához:

```
claudius:~$ ls -l | more
```

Fontos különbség, hogy itt a shell egyszerre elindítja mind a két programot, tehát nem az történik, mint az előbb, hogy először lefut az ls, és aztán írja ki a valahol eltárolt kimenetet a more: itt az ls futása közben, amint megjelenik valami kimenet, a more rögtön megkapja azt és akcióba lép. A pipe kapcsán még egy hasznos parancsot érdemes megemlítenünk (bár ez már inkább a következő részbe, a szűrők közé tartozik: a „tee” parancsot, amellyel standard output elágaztatást érhetünk el: a tee parancs a standard inputját a standard outputra valamint a paramétereként magadott file-ba másolja. Leginkább programkövetéskor, belövéskor hasznos, amikor nyomon szeretnénk követni a pipe belsejében zajló történéseket.

3.8. A szűrőkről (filters)

A Unix parancsok jó része nem csinál mást (a more-hoz hasonlóan), minthogy olvassa standard inputját (jellemzően szövegsor-orientáltan) a beolvasott inputtal „valamit csinál”, majd az inputot, esetleg annak egy részét, vagy valamely, az inputtól függő mennyiséget, adatot a standard outputra ír. A programok ezen csoportját nevezzük szűrőknek (filters). A szűrők általános tulajdonsága továbbá, hogy első argumentumként egy file-nevet megadva standard inputjukat abból a file-ból veszik, tehát a „more <hosszu_file” és a „more hosszu_file” parancsok tulajdonképpen ekvivalensek: a különbség annyi, hogy az első esetben a more-nak mint programnak nincs egy argumentuma sem, az input-átirányítást a shell végzi, a második

esetben pedig a more parancs tudja, hogy argumentummal indítottuk el, és ő maga intézi úgy, hogy a bemenetet a megadott file-ból olvassa. A továbbiakban néhány hasznos szűrőt nézünk meg, a teljesség igénye nélkül. Közös jellemzőjük, hogy apró, önálló segédprogram mindegyik, amelyekből a shell segítségével bonyolult feladatokra képes programfüzérek rakhatók össze.

A legegyszerűbb szűrő a „cat”, amely nem tesz mást, mint a standard inputját soronként átmásolja a standard outputra. Például:

```
claudius:~$ cat >proba
Ez egy proba.
Igen.
^D
```

Az utolsó sorban szereplő karakter a Control-D, rövid jelöléssel ^D, amely a shell számára file-vég karaktert jelent (hasonlóan a DOS Ctrl-Z karakteréhez).

Meg is nézhetjük újonnan gyártott file-unkat:

```
claudius:~$ cat proba
Ez egy proba.
Igen.
```

Emlékeztetőül: Írhattunk volna „cat <proba”-t is, a végeredmény ugyanaz lenne.

Újabb filter a „wc” (word count), amely az inputján érkező sorokat, szavakat és karaktereket számolja meg, és a végeredményt kiírja:

```
claudius:~$ cat proba | wc
2420
```

(A „wc proba” ugyanezt eredményezte volna.)

Az egyik leghasznosabb szűrő a (később részletesen ismertetendő) „grep”, amellyel minta-illesztést végezhetünk:

```
claudius:~$ cat proba | grep gen
Igen.
```

A grep csak azokat a sorokat engedi tovább kimenetére, amely(ek)ben szerepel a megadott (karakter)minta.

További gyakran használatos szűrő még: a „sort”, amely a standard inputon olvasott sorokat valamilyen szempont szerint rendezve írja ki a kimenetre, opciók nélkül indítva egyszerűen alfabetikusan rendez, a -r hatására pedig megfordítja a rendezési sorrendet:

```
claudius:~$ sort -r proba
Igen.
Ez egy proba.
```

Vagyis a file tartalma soronként fordított abc rendben jelent meg. A sort egyébként képes numerikusan rendezni, vagy akár a sor mezői szerint is: nézzük csak meg a „man”-ban, mennyi mindenre képes.

Sűrűn használják még a „cut”-ot is, amely nevéhez méltóan „horizontális” darabokat képes az inputként adott sorokból kivágni, pl. ha minden sor 3.-9. karakterére vagyunk kíváncsiak:

```
claudius:~$ cut -c3-9 proba
egy pr
en.
```

Nemsokára, a shell még néhány lehetőségének megismerése után majd látni fogjuk, hogyan tudunk ezekből a szűrőkből igazán hatékony programfüzéreket építeni.

3.9. A shell további lehetőségei

3.9.1. Metakarakterek, file-név helyettesítés

File-név helyettesítésre (mint a DOS-ban) Unixban is használhatunk különböző metakarakatereket, sőt, itt sokkal több lehetőségünk van. Egy nagyon lényeges különbség van azonban: a metakarakterek behelyettesítését a shell végzi, a kiadott parancs meghívása előtt, a parancsot már a megfelelően behelyettesített file-nevekkel hívja meg. Szemléletesen láthatjuk ezt, ha meggondoljuk, (és kipróbáljuk), hogy az „echo *” parancs tulajdonképpen az „ls”-sel ekvivalens. Például, ha az aktuális könyvtárban a malac, marha, tehenke file-ok vannak, a „cat m*a*” parancs a cat-ot már két argumentummal hívja meg, úgy mintha a „cat malac marha” parancsot írtuk volna be. A leggyakrabban használt metakarakter (mint láttuk) a *, jelentése: 0 vagy több tetszőleges karakter. A file névben bárhol használható, akár „*a*b*” formában is, aminek jelentése: olyan file-név, amelyben valahol egy „a” betű van, utána pedig valahol egy „b”. A másik gyakori metakarakter a ?, amellyel egy darab, tetszőleges betűt helyettesíthetünk. Megadhatunk halmazokat is: a „?[afu]?” az olyan 3 betűs file-nevekre illeszkedik, amelyek középső karaktere „a” vagy „f” vagy „u „. A „[0-9A-Z]” halmaz olyan karakterre illeszkedik, amely vagy szám vagy nagybetű. A nyitó szögletes zárójel után tett ^ karakter pedig a halmaz komplementumát jelenti: a shell azokra a karakterekre fog illeszteni, amelyek nincsenek a szögletes zárójelek között felsorolva.

3.9.2 Speciális karakterek semlegesítése

Időnként szükség lehet arra, hogy bizonyos karaktereket a shell helyettesítő hatásától megóvjunk: egyes karaktert a karakter elé írt \ (backslash)-szel védhetünk meg. Ha teszem azt valami oknál fogva olyan file-be akarunk írni, amelynek nevében a * karakter szerepel, ezt például a következőképpen tehetjük meg: „cat >ab*c”. Hosszabb sztringet a shell behelyettesítéstől megvédeni a ' egyszeres idézőjellel lehet, ez akkor lehet hasznos, ha például a „grep” szűrővel szóközt is tartalmazó kifejezésre szeretnénk illeszteni. Ha egyszerűen beírnánk a parancssorba a szóközt tartalmazó szöveget, a shell úgy értelmezné, mintha több argumentumot adtunk volna meg. Az „ ls '? *' ” utasítás például az (eléggé extrém) „kérdőjel szóköz csillag” nevű file-ot listázza ki, ha van ilyen az aktuális könyvtárban, nem pedig először az egybetűs file-neveket keresi, aztán pedig az összes file-nevet listázza. Hasonló szerepet tölt be a " (kettős aposztróf vagy dupla idézőjel), azzal a különbséggel, hogy az aposztrófban lévő sztringben a környezeti-változó helyettesítések kivételével véd meg mindent. Egy bonyolult példa: az „ ls "${HOME}/*" ” azt nézi meg, van-e a home könyvtárunkban * nevű file.

A Unix használja a harmadikféle idézőjelet is, a ` -t, (a legtöbb billentyűzeten a bal felső sarokban található), elég érdekes szerepkörben: az ilyen fajta idézőjelbe írt szöveget a shell parancsként értelmezi, a parancsot végrehajtja, és annak kimenetét (standard output) helyettesíti az idézőjeles kifejezés helyére. Az egyéb metakarakterek helyettesítése normálisan történik. Például:

```
claudius:~$
echo Ebben a konyvtarban `ls | wc | cut -c6-12` db file van.
Ebben a konyvtarban 36 db file van.
```

Gondoljuk át alaposan a fenti sor tartalmát! Emlékeztetőül annyit, hogy a „cut” szűrő -c opciója a megadott pozíciójú karaktereket vágja ki a sorból.

3.9.3. Programok csoportosítása zárójelezéssel

Lehetőségünk van futtatandó programjaink csoportosítására a „(shell-parancs)” szintakszis segítségével: ez tulajdonképpen annyit tesz, hogy a zárójelbe tett (leggyakrabban komplex, kettőspontot és pipe-ot tartalmazó) shell parancsot egy utasításként kezelhetjük: például a „ (cat file1 ; ls) | wc ” paranccsal megtudhatjuk a file1-ben lévő sorok és az aktuális könyvtárban lévő file-ok számának összegét. Vagy egy másik alkalmazás: „ cat file1 | (echo „Most tortenik a file iras” ; cat >file2) ” parancs a file2-be történő írás megkezdése előtt kiírja az üzenetet.

3.10. Folyamatok, processzek

3.10.1. Folyamatok a háttérben

Unix alatt egy felhasználó akár több folyamatot is elindíthat - lényegében minden futó program külön folyamatként fut. Egyszerre azonban csak egy folyamattal tudunk kommunikálni (az egy darab billentyűzet miatt), ezt „előtér” (foreground) folyamatnak nevezzük, a többi futó folyamatunkat pedig „háttér” (background) processzeknek. Pl. a Windows-tól eltérően, az előtérben futó folyamat semmilyen szempontból nincs kitüntetett helyzetben, nem élvez nagyobb prioritást - tulajdonképpen a futtató kernel nem is tudja, melyik folyamat van előtérben, hiszen az ő nézőpontjából mindegyik folyamat ugyanolyan file-műveleteket végez. Folyamatot a program neve után írt & jellel indíthatunk a háttérben: az így indított folyamatnak nem lesz standard inputja (megáll és várakozik, ha a program bemenetről akar olvasni), a standard outputja pedig a shell standard outputja (vagyis a képernyőnk) lesz. Próbáljuk is ki:

```
claudius:~$ cat &
[1] 1103
```

A szögletes zárójelben lévő 1-es a folyamat ún. job-azonosítója, a másik szám pedig a folyamat processz-azonosítója. A job azonosítót a shell rendeli az elindított programhoz, azért hogy a felhasználónak ne kelljen a gyakran 5 számjegyű processz-azonosítót (PID) megjegyeznie. Ha ezután kiadunk egy újabb parancsot:

```
claudius:~$ ls valami
valami
[1]+ Stopped (tty input) cat
```

A következő parancs lefutása után a shell észreveszi, hogy az 1-es sorszámú job terminál inputra vár. Háttérfolyamatot előtérbe hozni a

```
claudius:~$ fg %1
cat
```

„fg” paranccsal tudunk. Ekkor a folyamatot úgy használhatjuk, mintha a & jel nélkül indítottuk volna el. Ha egy másik programmal szeretnénk foglalkozni, de azt akarjuk hogy az előtérben lévő folyamat tovább fusson a háttérben, a ^Z (Control-Z) billentyűkombinációval megállíthatjuk (ekkor várakozó, „stopped” állapotba kerül), majd háttérbe helyezni a „bg %job-azonosító” 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 beírjuk a „logout” parancsot, a shell kiirtja a háttérfolyamatokat és kiléptet bennünket a rendszerből. Ha „fg” és „bg” parancsokat argumentum nélkül használjuk, mindig a legutoljára hivatkozott folyamatra vonatkoznak.

Ha több folyamat is fut, amelynek kimenete a képernyő, a shell nem válogatja szét az egyes parancsok outputjait: „ömlesztve” kapunk meg mindent. Ha programfuttatás közben üzenetet kapunk (a rendszertől vagy másik felhasználótól), akkor ez is egyszerűen kiíródik a képernyőre, esetleg jól összekeveredve az éppen futó program kimenetével. Ezért erre mindig érdemes odafigyelni: ha látszólag értelmetlen szöveg jelenik meg a képernyőn, nézzük meg, nincs-e összekeveredve két program kimenete. Az ilyen esetek nagy részét azért elkerülhetjük a háttérben futó programok kimenetének file-ba irányításával.

Unix alatt a folyamatok hierarchikus rendszerbe szervezettek: minden folyamatnak van szülője. Folyamat csak másik folyamat gyermekeként jöhet lére (az egyetlen speciális, rendszerindításkor keletkező „init” kivételével). Gyermekfolyamat létrehozása a szülő folyamat lemásolásával történik, a már megismert copy-on-write mechanizmus használatával. A gyermek folyamat ezért örökli a szülő minden tulajdonságát, jellemzőjét - még nyitott file-jait is! Ez az oka annak, hogy időnként a shell és az általa indított folyamatok (az ő gyermekei) output-ja összekeveredik: ugyanazt a nyitott file-t írják, és közös file-pozíció mutatót használnak. Kilépéskor a gyermek folyamatok szülőjüknek adják vissza visszatérési értéküket.

Egy működő Unix rendszerben sok olyan folyamat fut még, amely nem tartozik egyik felhasználóhoz sem: root-ként futnak, a rendszer indításakor (boot-oláskor) indulnak, és a rendszer működéséért felelősek. Ezeket a folyamatokat daemon processzeknek nevezzük. Tipikus daemon processz például az „update”, amely azért felelős, hogy a buffer cache tartalma bizonyos időközönként (általában 30 másodperc) szinkronizálódjon a diszk fizikai tartalmával. (Magyarul fél percenként kiírja a diszkre az addig még ki nem írt változtatásokat). Ezt egyébként a „sync” paranccsal bármikor mi is megtehetjük. A daemon folyamatok tipikusan valami esemény bekövetkeztére várakoznak, és az esemény bekövetkeztekor aktivizálódnak.

3.10.2. Sokáig futó vagy időzített folyamatok

Lehetőségünk van azonban arra is, hogy egy folyamatot immunissá tegyünk kilépésünkre: hosszan, több óráig, több napig futó programokat a „nohup” paranccsal indíthatunk. Például:

```
claudius:~$ nohup program >outputfile <inputfile &
[1] 88
```

Ekkor a program kilépésünk után is tovább fut, és amint láttuk, gondoskodtunk arról, hogy a programfutás eredményei az „outputfile”-ban megőrződjenek. Egyes shell-ekben a nohup-pal indított parancsok kimenete automatikusan a „nohup.out” file-ba kerül. Ha fut a gépünkön a „cron” nevű program (általában minden Unixon fut) akkor lehetőségünk van időzített program-indításra is az „at” paranccsal:

```
claudius:~$ at 3am program &
[3] 116
```

Ennek hatására a „program” parancs hajnali háromkor elindul. Természetesen itt is érdemes a futási eredményt megfelelő file-ba menteni.

3.10.3. Folyamataink listázása: a ps parancs

A rendszerben aktuálisan lévő folyamatokról a „ps” (processes) paranccsal kérhetünk információt. Alapértelmezés szerint csak az általunk indított folyamatokat listázza ki:

```
claudius:~$ ps
PID TTY STAT TIME COMMAND
79 pp0 S 0:00 -bash
129 pp0 R 0:00 ps
```

Ami azt jelenti, hogy ebben a pillanatban két processzel rendelkezünk: az első a shell, a második pedig maga a ps parancs. Az egyes mezők jelentése:

PID a folyamat azonosítója
TTY a vezérlő terminál azonosítója, jelen esetben ez a ttyp0
STAT a folyamat állapota, bővebben lásd a „man ps” alatt
TIME a processz által eddig elhasznált processzor idő

A rendszerben futó összes folyamatot, a legbővebb információkkal Linux alatt a „ps -aux” opciókkal kérhetjük le. Ekkor az processzekről megtudjuk még tulajdonosukat, az időpontot, amikor elindultak, valamint különféle erőforrás használati információkat (CPU, memória használat, a program mérete a memóriában). Hasznos segédprogram még a „top”, amellyel hasonló információkat kaphatunk, plusz még statisztikákat a rendszer egészéről. A top folyamatosan fut, és 5 másodpercenként frissíti a megjelenített információkat. Kilépni a „q” megnyomásával lehet.

3.10.4. Üzenetek a processzeknek: szignálok

A Unix rendszer a folyamatok vezérlését a folyamatoknak küldött ún. szignálok (signals) segítségével végzi: a ^Z billentyű például egy STOP szignált küld az előtérben futó processznek. Processzt kiölni szintén szignál(ok) segítségével lehet: az előtérben futó program a ^C (Ctrl-c) megnyomására egy INT szignált kap, amely rendszerint a program elhalálozását vonja maga után. Háttérben futó folyamatainkat a „kill” paranccsal irthatjuk ki: alapértelmezés szerint a „kill” egy TERM (terminate) szignált küld a megadott folyamatnak:

```
claudius:~$ sleep 60 &
[1] 332
claudius:~$ ps
```

```
PID TTY STAT TIME COMMAND
310 pp0 S 0:00 -bash
332 pp0 S 0:00 sleep 60
333 pp0 R 0:00 ps
claudius:~$ kill %1
claudius:~$ ps
PID TTY STAT TIME COMMAND
310 pp0 S 0:00 -bash
334 pp0 R 0:00 ps
[1]+ Terminated sleep 60
```

A „sleep” utasítás egyébként várakozik a megadott számú másodpercig („elaltatja” a folyamatot). A „kill %1” helyett természetesen írhattunk volna „kill 332”-t is.

Ha más (nem TERM) szignált akarunk küldeni, a kill parancsot megfelelően paraméterezni kell, például a STOP szignálhoz: „kill -STOP pid”. Ennek ugyanolyan hatása van, mintha az a folyamat az előtérben futna, és a ^Z-t nyomtuk volna meg: a folyamat felfüggesztett állapotba kerül. Folyamatot megölni még a HUP (hangup) és a KILL szignálokkal is lehet. (Az előbb látott nohup parancs ezen HUP szignál ellen teszi immunissá 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 (így van ez az INT szignálnál is). 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 processzel rendelkezhet). Az eddig felsoroltakon kívül még számos egyéb szignál van, megemlítjük még az ALARM szignált: a rendszert megkérhetjük, hogy megadott idő múlva küldjön egyet. Ezt használják időzítési célokra, többek között a „sleep” utasítás is így működik. De szignálokat használ a rendszer sok más egyéb, a folyamatot érintő rendszerinformáció közlésére is, de ezek főleg programozók számára érdekesek.

3.10.5. Folyamatok prioritása, a nice parancs

Már korábban szó volt a folyamatok prioritásáról (egy folyamat prioritását megnézni legegyszerűbben a „top” paranccsal lehet, vagy a „ps” parancs -l opciójával a PRI oszlopban). Ez a prioritás (szokás még ütemezési - scheduling - prioritásnak is nevezni) azt szabja meg, hogy ha több folyamat is van egyszerre futóképes állapotban (több folyamat verseng az egyetlen CPU erőforrásért), 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 negatív is lehet: negatívabb érték magasabb prioritást jelent.

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). A „nice -n növekmény parancs” szolgál arra, hogy a „parancs”-ot a megnövelt nice prioritás értékkel futtassuk. (Vagyis effektíve a folyamat prioritását csökkentsük). 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 tudja folyamatai nice értékét (illetve a visszacsökkentéskor nem tudja az induló érték alá csökkenteni), a root tetszőlegesen állíthat prioritást.

3.11. Érintkezés más felhasználókkal

3.11.1. A többi felhasználó

Azt, hogy jelenleg kik használják rendszerünket, a „who” paranccsal kérdezhetjük le:

```
claudius:~$ who
tiv ttyp2 Jul 21 14:37 (euromath.vma.bme)
```

Ami azt mutatja, hogy a „tiv” nevű felhasználó, a 2-es sorszámú pszeudo-terminálon dolgozik, a bejelentkezés időpontja látható még, illetve az, hogy (pszeudo-terminálról lévén szó) a bejelentkezés hálózaton keresztül történt, és hogy mi a távoli állomás neve. Látható az is, hogy „tiv” jelenleg az egyetlen felhasználó.

Ennél bővebb információt kaphatunk a „w” paranccsal:

```
claudius:~$ w
4:19pm up 6:09, 1 user, load average: 0.00, 0.00, 0.00
User tty from login@ idle JCPU PCPU what
tiv ttyp2 euromath.vma.bme 2:37pm w
```

Ahol a legelső sorban az egész rendszerre vonatkozó információk láthatók, a felhasználóról szóló sorokban pedig az, ami a „who”-nál is látszott, kiegészítve azzal, hogy mennyi ideje nem nyúlt az illető a billentyűzethez (idle time), illetve hogy mennyire CPU igényes munkát folytat (JCPU, PCPU), valamint hogy mely parancsot futtatja éppen (what mező). Még egy hasznos parancs: a „last”. Ezzel a paranccsal megnézhetjük a bejelentkezéseket időben visszafelé haladva, illetve argumentumként egy user-nevet megadva csak annak a felhasználónak a bejelentkezéseit. Opcióként azt adhatjuk meg, hogy hány adatra vagyunk kíváncsiak: a „last -1 user_nev” megadja az adott felhasználó utolsó bejelentkezésének dátumát, azt hogy melyik terminálról jelentkezett be, illetve hogy mennyi időt töltött bejelentkezve.

3.11.2. Kommunikáció a többiekkel: write, talk, mail

Az éppen bejelentkezett felhasználóknak a „write” paranccsal küldhetünk rövid üzenetet: „write [tty] user_nev” után gépelhetjük az üzenetünket, (ha a partner időközben válaszol, az is megjelenik), majd ^D-vel fejezhetjük be. A terminál azonosító megadására akkor van szükség, ha a keresett felhasználó több példányban is be van jelentkezve a rendszerre. (Ez egy grafikus-ablakozós környezetben mint az X Windows könnyen megtörténhet.) Ha nem akarjuk, hogy mások nekünk üzenetet küldjenek, a „mesg n” üzenet használjuk. Üzenetek fogadásának újra-engedélyezéséhez használjuk a „mesg y” parancsot. Mivel a „write” utasítás úgy működik,

hogy üzenetünket a másik felhasználó terminálját reprezentáló file-ba írja, ezek az utasítások tulajdonképpen ennek a speciális file-nak a hozzáférési jogosultságait állítják.

Kulturáltabb kommunikációs lehetőséget jelent a „talk" program: a „talk user_nev [tty]" paranccsal. Erre a felhívott félnek válaszolnia kell (szintén a „talk" paranccsal, csak az user_nev helyébe a hívó nevét írva). A kapcsolat megteremtése után a „talk" program kettéosztja a képernyőt: a felső részben azt láthatjuk, amit mi gépelünk, az alsó részben pedig azt, amit beszélgetőpartnerünk. Hasznos billentyűk: ^L-lel újrafrissíthetjük a képernyő tartalmát (ha valamilyen más program beleírása miatt összezavarodna), és a beszélgetés végén (természetesen az illő búcsúzkodás után) a ^C-vel léphetünk ki. A „mesg" parancs hatása a „talk"-ra is érvényes.

Nem interaktív üzenetet a „mail" paranccsal hagyhatunk valakinek: ennek legegyszerűbb módja a „mail user_nev" parancs: ekkor a „Subject:" kérdésre írjuk be levelünk tárgyát, majd gépeljük be a levelet! Befejezni egy, csak a . karaktert tartalmazó sorral lehet. Az utána megjelenő „Cc:" (Carbon Copy) kérdésre további címeket (felhasználó neveket) adhatunk meg, akik a levélből másolatot kapnak. Ha 'Enter'-t ütünk, a levelet csak az első címzett (akit a „mail" után írtunk) kapja meg.

Ha levelünk érkezett, ezt a rendszer rögtön bejelentkezéskor kiírja: a „You have mail.", vagy „You have new mail." üzenetek valamelyikével, illetve bejelentkezés után a shell is folyamatosan ellenőrzi, hogy érkezett-e új levél, és kiírja, ha érkezett. Leveleinket elolvasni szintén a „mail" programmal lehet:

```
claudius:~$ mail
Mail version 5.5 6/1/90. Type ? for help.
"/var/spool/mail/tiv": 1 message
> 1 VARKONYI@bme-eik.eik Thu Jul 20 09:03 40/1996 "DECnet routing"
&
```

Itt a látható, hogy jelen pillanatban 1 levelünk van: az első mező a levél sorszáma, a második feladó neve, a levél dátuma, majd a sor végén a levél tárgya (subject) jelenik meg. Levelet elolvasni a „mail" program & promptja után írt levélsorszámmal lehet, törölni pedig a „d levelsorszam" paranccsal. Kilépni a „mail" programból a „q" paranccsal lehet. Lehetőség van továbbá a levél file-ba vagy ún. „mail-folder"-be (levél dosszié) mentésére a „s file_nev" paranccsal. Levelek fogadását letiltani nem lehet.

A gyakorlatban ritkán használjál levelezésre a „mail" programot: a legtöbb rendszeren sokkal kényelmesebben használható (teljesképernyős menüvezérelt, állítható szövegszerkesztős) levelezőprogramok is vannak (például az „elm" vagy a „pine") - ezek ismertetésére sajnos terjedelmi okból nincs lehetőségünk.

3.11.3. Hálózatban

Amennyiben gépünk része az Internet hálózatnak, illetve IP (Internet protocol) típusú hálózatba van kötve más gépekkel, ez utóbbi két paranccsal (talk, mail) más gépek felhasználóival is kommunikálhatunk: csupán user-név helyett a user_nev@internet.gep.cim formátumú címet kell megadnunk. Hálózatban másik gépre bejelentkezni a „telnet gep_nev" paranccsal lehet: a „telnet" a hálózati terminál szoftver. Sajnos a hálózati (Internet) szolgáltatások részletes ismertetésére itt nincs bővebb lehetőségem.

3.12. Adatok a rendszer egészéről

Már láttuk, hogy (például a „top”, vagy a „w” parancsokkal) adatokat kaphatunk a rendszer egészéről: nézzük most ezeket meg részletesebben! Például a „top” egy kimenetének felső pár sora:

```
10:38am up 1 day, 27 min, 1 user, load average: 0.00, 0.00, 0.00
20 processes: 19 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 1.3% user, 1.1% system, 0.0% nice, 97.6% idle
Mem: 15176K av, 13484K used, 1692K free, 4484K shrd, 8888K buff
Swap: 16392K av, 0K used, 16392K free
```

A legfelső sorban az idő, aztán a gép ún. uptime értéke (mennyi idő telt el reboot óta), a bejelentkezett felhasználók száma, illetve a három „load average” (átlag terhelés) látható. Ezek jelentése: átlagosan hány futóképes (CPU-t igénylő) folyamat tartózkodik az ütemezőben. A három érték közül az első az utolsó pár (általában 5) másodpercre vonatkoztatott terhelésátlag, a középső az elmúlt pár percre vonatkozik, a harmadik pedig az elmúlt fél órára vetített átlag. Jelen esetben mindhárom érték nulla: a gépen egy felhasználó dolgozik csak, aki leginkább csak a shell-t futtatja: a rendszer ideje legnagyobb részében várakozik.

A következő sorban processz-statisztikát láthatjuk: a rendszeren jelenleg 20 folyamat fut, ebből 19 „alszik” (sleeping - várakozik valamilyen eseményre, általában inputra), 1 fut (running - jelen esetben ez a „top”), 0 zombie (zombie-nek a már halott, de még a rendszerből el nem tűnt folyamatokat nevezik: 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), illetve 0 darab STOP szignállal megállított folyamat van.

A következő sorban (CPU states) a processzor kihasználtságot olvashatjuk le: idejének 1.3%-át tölti a processzor felhasználói folyamatok futtatásával, 1.1%-ot fut a kernel (rendszer-feladatok), 0%-ban foglalkozik csökkentett prioritású folyamatokkal, és idejének 97.6%-át henyéléssel tölti (idle). Ezt az információt (CPU használat statisztika) egyébként a „time” paranccsal kaphatjuk meg egyes programokra: a „time parancs” utasítás először lefuttatja az utasítást, majd sokféle rendszer-statisztikát közöl a futtatásról.

Az utolsó előtti sorban a fizikai memória kihasználtságát láthatjuk: av: összes rendelkezésre álló (itt nem számolja bele a kernel és a rendszeradatok által elfoglaltat), used: használt, free: szabad, shrd: a memóriában lévő osztott könyvtárak (shared libraries) által elfoglalt terület, és buff: a buffer cache aktuális mérete. Az utolsó sorban pedig a virtuális memória adatai vannak: mennyi swap memóriával rendelkezik a rendszer, és ebből mennyi szabad még. Ezeket a (memória foglaltsági) információkat egyébként még a „free” paranccsal is megkaphatjuk.

3.13. Linux specialitások

3.13.1. A virtuális konzol (virtual console)

Azon a problémán, hogy egy géphez csak egy monitor és egy billentyűzet tartozik, úgy próbál a Linux segíteni, hogy több ún. virtuális konzolt használhatunk, amelyek mindegyike úgy viselkedik, mintha a rendszer konzol terminálja lenne. A virtuális konzolok számát a rendszer-gazda konfigurálhatja, közöttük a bal Alt+funkcióbillentyűkkel illetve a bal Alt+jobbra/balra nyíl billentyűkkel válthatunk.

3.13.2. Speciális billentyűzetkombinációk

Nagyon hasznos funkció a Shift+PageUp/PageDown, amellyel - virtuális konzolon - pár oldalnyi szöveget görgethetünk oda-vissza. Az éppen futó processzt a „Print Screen” megnyomásával is kilőhetjük, ha az esetleg a ^C-re nem reagálna. Az operációs rendszert újraindíthatjuk a jól ismert Ctrl-Alt-Del kombinációval (természetesen csak konzolról!) - ezt „shutdown” procedúraként, a gép kikapcsolása előtt is használhatjuk, ugyanis gondoskodik a bufferek diszkre írásáról, a folyamatok kulturált lezárásáról stb.

4. A Unix (Linux) file-szerkezete

4.1. Könyvtár struktúra

Unix alatt az idők során kialakult egy konvencionális könyvtár-struktúra: ez ugyan (kevés kivételtől eltekintve) nem kötelező érvényű, de ha máshogy alakítjuk gépünk könyvtárszerkezetét, általában csak saját dolgunkat nehezítjük vele. Kisebb eltérések persze mindig akadnak: mi, amint eddig is tettük, a „Slackware" nevű Linux disztribúció konvencióit követjük.

Minden a / (root, vagyis gyökér) könyvtárból indul. Első könyvtár a /bin, ahol a rendszer futtatáshoz, a rendszer felállásához legszükségesebb futtatható állományok vannak: a „bash", a „cat", a „ps" és így tovább.

Linux sajátosság a /boot könyvtár, ahol a Linux boot-vezérlő program, a LILO (Linux Loader) adatfile-jai kaptak helyet. Következő a /dev, ahol a rendszer erőforrásait reprezentáló speciális ún. eszköz (device) specifikus file-okat találhatjuk.

A /etc alatt helyezkednek el a rendszer-konfigurációs file-ok, illetve Linux alatt az /etc/rc.d könyvtárban vannak a rendszer indulásakor automatikusan lefutó file-ok. Az /etc/skel könyvtár egy „skeleton" (csontváz) könyvtár a létrehozandó felhasználók számára: ha a standard, Slackware-hez mellékelt programokkal hozunk létre új account-okat a gépen, ennek a könyvtárnak a tartalma másolódik be a home könyvtárukba.

A /home könyvtár alatt találhatók a felhasználók „home" könyvtárai, az eddigi példákban szerepelt /usr/users régi, BSD rendszerekből származó csökevény.

A következő könyvtár a /lib: Linux alatt itt található a dinamikus programkönyvtárak nagy része.

Minden Unix filesystem-en található egy /lost+found alkönyvtár: ha a filesystem megsérül, és a helyreállító program olyan file-okat talál rajta, amelyeknek könyvtárstruktúrába helyezéséhez nincs elég információja, ide teszi őket.

Rendszerint van még egy /mnt alkönyvtár is: általában ide szokás az ideiglenes (például floppy, CD-ROM) filesystem-eket illeszteni (mount). Szintén Linux sajátosság (illetve más, újabb Unix-okban is előfordul már) a /proc alkönyvtár: itt nem „igazi" file-ok vannak, hanem a rendszer, a folyamatok állapotát reprezentáló „álfile-ok". Például a /proc/meminfo kiolvasásával („cat /proc/meminfo") a „free" parancséhoz hasonló kimenetet kaphatunk.

A /root könyvtár Slackware-ben a rendszergazda home-könyvtára. Következő fontos könyvtár a /sbin: ebben a rendszergazda számára fontos, rendszeradminisztráláshoz szükséges futtatható file-okat találjuk.

A /tmp könyvtár az, ahova mindenkinek van írásjoga, ideiglenes file-ok tárolására való. Mivel Unix alatt nagyon sok program használja a /tmp könyvtárat ideiglenes file-jainak tárolására, mindig ügyeljünk arra, hogy itt legyen szabad hely, különben egész alapvető programok nem lesznek képesek futni, és nagyon furcsa dolgok történhetnek.

A /usr könyvtár alatt Unix-okban tradicionálisan a rendszer futtatásához nem feltétlenül szükséges, „minden egyéb" file helyezkedik el: a /usr alatt is van /usr/bin, /usr/lib, /usr/etc, /usr/sbin könyvtár, ahol a különböző installált szoftverek futtatható és adatfile-jai helyezkednek el. (Adatfile-ok rendszerint a /usr/lib alatt.) A /usr/man alatt találhatók a „man" parancs

adatfile-jai, a /usr/include alatt pedig a rendszer C header file-jai kaptak helyet. A /usr/X* könyvtárakban az XFree grafikus rendszer file-jai találhatók. A /usr/local alatt általában megismétlődik egy, a /usr-éhez hasonló könyvtárstruktúra: mivel a /usr/bin-ben lévő programok rendszerint az operációs rendszerrel együtt érkeznek, tágabb értelemben véve annak részeinek tekinthetők, ezért a rendszergazdák nem szívesen keverik közéjük a „lokálisan" (rendszerint forrásból) felinstallált programokat - és a /usr/local alá helyezik őket. Így van ez Linux alatt is. Említést érdemel még a /usr/src könyvtár is: értelemszerűen programforrásokat szokás ide tenni, Linux alatt pedig mindig megtalálható itt a kernel (tehát maga a Linux) forrása is.

A főkönyvtárból nyílik még a /var alkönyvtár, amely mindenféle, a rendszer működésével kapcsolatos „aktuális" file-t tárol: például a /var/adm könyvtárban vannak a különféle „napló" (log) file-ok, amelyek a rendszer eseményeit rögzítik. A /var/spool alatt pedig többek között a nyomtatókezelő és a levelező alrendszer ideiglenes file-jai találhatók meg. A /var alatt lévő file-ok a rendszergazdának (vagy még neki sem, rendszerprogramoknak) szólnak, átlag felhasználó nem sok érdekeset találhat itt.

Végezetül, a gyökérkönyvtárban található még a /vmlinuz, vagy /zImage, vagy /vmunix nevű file, ami a kernelt tartalmazza: innét töltődik be boot-oláskor.

4.2. A /dev alkönyvtár

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

- /dev/audio
  Ha valamilyen hangkártya vagy más zajkeltő szerkezet van a kernelbe konfigurálva, akkor a .au formátumú file-okat ide kiírva meghallgathatjuk őket. Példa: „cat x.au >/dev/audio" Ha a hangkártya digitalizálásra is képes, ez a file olvasható is.
- /dev/cdrom
  Ez általában egy link a bonyolultabb nevű, valódi CD-ROM speciális file-ra. Hasonlóan használható mint egy winchester speciális file.
- /dev/cua*
  A soros vonala(ka)t jelentő speciális file-ok. Írásuk vagy olvasásuk küldést/vételt jelent a megfelelő vonalon.
- /dev/fd*
  A floppy diszkeket reprezentálják. A fd0 kezdetű file-ok az A floppyra vonatkoznak, a fd1 kezdetűek a B-re. Mount-olásnál kell ezeket a neveket használnunk, vagy pedig akkor, ha a floppy-n lévő image-t (összes szektort) egy file-ba akarjuk olvasni. Erre használhatjuk például a „cat floppyimage" parancsot. Blokkos eszközök írása-/olvasásakor érdemesebb a „cat" helyett a „dd" parancsot használni: működése hasonló, csak bővebben paraméterezhető, és blokk-műveletekre optimalizálható. Például: az előző file visszaírása az 1.44MB-s B lemezre: „dd bs=512 /dev/fd1H1440". (bs=block size)

- /dev/hd*
  A rendszerben lévő AT buszos winchesterek: a /dev/hda az első winchestert jelenti, /dev/hdb a másodikat. Ha a gépben két IDE vezérlő vagy egy EIDE vezérlő van, akkor a többi diszkhez a /dev/hd1[ab] néven férhetünk hozzá. Ha ezeket a file-neveket számokkal folytatjuk, az egyes diszkeken lévő partíciókhoz jutunk, például /dev/hda1,/dev/hda2
- /dev/midi, mixer
  Hangkártyához tartozó file-ok, a /dev/midi értelemszerűen midi file-ok kezelésére.
- /dev/mouse, modem
  Általában ezek linkek valamely soros portra.
- /dev/pty*
  Pszeudo-terminál vonalakat reprezentáló speciális file-ok.
- /dev/sd*
  SCSI diszkek.
- /dev/tty*
  A (virtuális) konzol terminálvonalai.
- /dev/ttyS*
  Soros vonali terminálok.
- /dev/null
  Ez egy igen érdekes file: minden beleírt adatot elnyel, és olvasáskor mindig filevége-jelet ad. Akkor hasznos, ha egy parancs kimenetét el akarjuk nyomni. Például, ha nem akarjuk a hibaüzeneteket látni: „parancs 2>/dev/null”.
- /dev/zero
  Az előzőhöz hasonló file, azzal a különbséggel, hogy olvasáskor végtelen sok 0 értékű byte-ot ad vissza. A „dd 10kfile bs=1k count=10” utasítással például létrehozhatunk egy 10Kbyte hosszú, 10kfile nevű csupa 0-ból álló file-t.

4.3. A filesystem-ekről

Mint már korábban szó volt róla, a Linux többféle file-rendszer-formátumot is támogat. Többféle Unix-os filesystem-et:

- minix
  amelyet egy régebbi Unix implementációtól örökölt,
- ext
  (extended) filesystem formátumot, amely az előző továbbfejlesztése
- ext2
  second extended, amely szintén az előző továbbfejlesztése, jelenleg ez a legnagyobb teljesítményű Linuxos filesystem formátum, szinte minden Linux rendszeren ezt használják
- xiafs
  amely egy, Frank Xia által kifejlesztett filesystem: mára az „ext2” már teljesen kiszorította
- /proc amely file-rendszerként van implementálva, az egységesebb kezelés érdekében

Ezek a filesystem formátumok főleg teljesítményben (sebesség, maximális méret, létrehozható maximális file-méret, maximális file-név hossz stb.) térnek el egymástól: jelenleg a legjobb az „ext2”, amelyet Remy Card fejlesztett ki (és fejleszt tovább folyamatosan). További, Linux által támogatott filesystem formátumok:

- msdos
  MS-DOS-os, FAT formátumú partíciók, floppy-k kezelésére
- umsdos
  speciális filesystem, segítségével DOS-os, FAT filesystem-en hozhatunk létre „igazi” Unix-os file-rendszert. (Vagyis biztosítja a file-okhoz tartozó Unix-os plusz információ kezelését.)
- NFS
  Network File System: TCP/IP hálózati környezetben elosztott hálózati file-rendszert hozhatunk létre segítségével.
- ISO9660
  a CD-ROM-ok file formátuma
- HPFS
  (egyelőre csak olvasható) OS/2 HPFS file-rendszer
- SYSV/Coherent
  egy újabb Unix-os file-rendszer.

Az itt felsorolt formátumok a standard Linux kernel részei: kiegészítések (ún. patch-ek) formájában létezik még sok más, ritkábban használt filesystem formátum támogatása is (további Unix-os file-rendszereké, például BSD „ufs”, illetve léteznek Linuxra is röptömörítő szoftverek).

Nem Unix-os file-rendszereket csak látszólag tudunk Unix-osként kezelni: „msdos” filesystem-en nem tudunk például linket létrehozni, file-tulajdonost váltani stb. File-okat írni, olvasni végrehajtani azonban ugyanúgy lehet. „Hosszú” file-neveket szintén nem tudunk annak kezelésére alkalmatlan filesystem-eken létrehozni.

Filesystem-et létrehozni (formázni) a „mkfs” utasítás családdal lehet: „ext2” file-rendszert a „b” winchester második partíciójára például a „mke2fs /dev/hdb2” paranccsal tudunk. A mkfs parancsok nem képesek alacsony szintű (fizikai) formázást végezni: winchesterek esetén ezt a BIOS Setup-ból kell megtennünk, floppy-kat pedig az „fdformat” paranccsal formázhatunk alacsony szinten.

Filesystem-ek beillesztése: ha az előbb létrehozott filesystem-et szeretnénk beilleszteni a /mnt könyvtár alá: „mount -t /dev/hdb2 /mnt”. Az illesztést megszüntetni az „umount /mnt” paranccsal lehet, ez azonban csak akkor lehetséges, ha az összes file le van zárva az adott filesystem-en, és nincs a rendszerben olyan folyamat, amelynek aktuális könyvtára valahova a filesystem területére mutat.

Ha egy filesystem valamilyen okból kifolyólag (általában áramszünet, véletlen kikapcsolás) logikailag meghibásodik, az „fsck” utasítás-család valamely tagjával próbálhatjuk megjavítani („ext2” filesystem esetén használjuk az „e2fsck”- t). Ezt a parancsot mindig használaton kívüli (nem beillesztett, vagy végső esetben „read-only” - csak olvasható - módon beillesztett) filesystem-en futtassuk: az „fsck” direkt diszk-blokkokkal dolgozik és azokat javítja meg, a buffer cache-ben azonban a hibás blokkok marad(hat)nak benn, amelyeket az „update” visszaírhat a diszkre.

4.4. File-elnevezési konvenciók

Bár nem kötelező Unix alatt file-név kiterjesztést (extension) használni, mégis - a könnyebb eligazodás végett - szokás. Futtatható file-oknak általában nem adnak kiterjesztést. A gyakoribb kiterjesztések:

.c,.h	C forrásfile-ok
.cpp	C++ forrás
.s,.S	Assembly forrás
.o	tárgykód (object) file-ok
.a	statikus programkönyvtárak
.conf,.cf	konfigurációs file-ok
.so.*	dinamikus programkönyvtárak
.[1-9]	(szám kiterjesztés): az egyes manual page file-ok
.sh,.csh	(nem mindig) a script file mely shell alá készült
.tar	a „tar” archiválóval készült file
.gz	a „gzip” (GNU zip) tömörítővel tömörített file
.tgz	.tar.gz rövidítése (a „tar” nem tömörít)
.Z	a „compress” (régebbi) tömörítővel pakolt file
.tex	TeX vagy LaTeX forrásfile
.dvi	DVI (Device Independent) szöveges-grafikus file formátum
.ps	PostScript (oldalleíró, nyomtatókezelő nyelv) file
.au,.wav	Hangmintákat tartalmazó file-ok

Ha egy file nevéről nem tudjuk kitalálni, mi is lehet benne, megpróbálkozhatunk a „file file_nev” paranccsal: a „file” utasítás beleolvas és megpróbálja kitalálni, milyen file is az. Az általában ismert file-formátumokra meglepően jól működik.

4.5. az „mtools”: DOS-os floppyk kezelése

Mivel leggyakrabban DOS alá formázott mágneslemezekkel találkozhatunk, létrehoztak Unix alá egy programgyűjteményt, amely ezen floppyk kezelését könnyíti meg: a megszokott DOS szintakszist használhatjuk floppy- műveleteknél. Például egy file bemásolása az A: floppyról az aktuális könyvtárba:

```
claudius:~$ mcopy "a:\file.xxx" .
```

Az összes „m” parancs felsorolásáért nézzük meg a „mtools” man page-t! Figyelmeztetés: használhatunk DOS-os metakaraktereket is a file-nevekben, ezeket azonban mindig érdemes a shelltől idézőjellel megvédeni, az eltérő értelmezés miatt. (Az „mtools” helyesen értelmezi a DOS-os metakaraktereket, csak gondoskodnunk kell arról, hogy ezek a metakarakterek el is jussanak hozzá.)

Vége. Egyelőre...