Adattárolás és fájlrendszerek (jegyzet)

Tuza Zoltán, Uhlár László, Bérci Norbert

2015. október 1-i óra anyaga

Tartalomjegyzék

1. Bevezetés

Egy program futása során az elvégzett számításokból (rész)eredmények keletkeznek, amit az illékony[1] memória [volatile memory] tárol. Ha ezeket az eredményeket el akarjuk tárolni a számítógép két bekapcsolása között vagy egy program két futtatása között, akkor szükséges, hogy a tárolás elektromos áram nélkül is biztosítható legyen. Erre kezdetben nyomtatót, illetve lyukkártyákat használtak (ez utóbbi annyival volt szerencsésebb, hogy egy lyukkártya-olvasóval könnyen vissza lehet tölteni az információt a memóriába). Később megjelentek a szalagos tárolási módszerek, melyek segítségével nagyobb mennyiségű adatot tudtunk lineárisan elmenteni (emlékezzünk a magnókazettákra, ahol ha egy számot ki szerettünk volna hagyni, azt a szalag gyors tekerésével tudtuk csak megtenni).

[0] $Revision : 51$ ($Date : 2013 - 10 - 0722 : 03 : 42 + 0200(Mon, 07Oct2013)$)

[1] tápfeszültség megszűnésével a benne tárolt adat elveszik

Az áttörést a cserélhető lemezes olvasó jelentette, ahol egy mágnesezhető korongot forgattunk egy mozgatható mágneses olvasó/író fej előtt, szemben a mágneses szalaggal, ahol az a olvasó fej fixen volt tartva. Ezzel a fejet különböző, a korong más és más részén lévő adatsávok fölé tudtuk helyezni. Ezt a módszert - melyben a tárolón lévő adatok bármelyikét a többi adat érintése/átlépése nélkül érhetjük el - hívjuk véletlen hozzáférésnek [random access]. Ezen a ponton két irány indult el, az egyik mentén a cserélhető lemezeket fejlesztették, míg a másik vonalon létrejöttek fix- vagy merevlemezek.

Mivel ezek az eszközök nem közvetlenül vannak a számítógép alaplapjára építve, hanem valamilyen csatlakozón keresztül kapcsolódnak hozzá, így adattároló perifériák gyűjtőnévvel hivatkozunk rájuk. Például az egeret és a monitort is perifériának tekintjük, az egyiket adatbeviteli perifériának, míg a másikat adatmegjelenítő perifériának hívjuk. Egy adattároló eszköz legfontosabb három jellemzője a következő:

- (adat)hozzáférési idő [(data) latency]: Az adat megcímzése és az adat kiolvasása között eltelt idő. Mai merevlemezeknél ez néhány ms.
- adatátviteli sebesség [transfer rate]: Egy időegység alatt az eszközre írt vagy onnét kiolvasott adat mennyisége.
- tárolókapacitás [capacity]: az eszközön tárolható bitek/bájtok mennyisége. Fontos tisztázni, hogy ez egy bruttó érték: mivel nem a „nyers" merevlemezt használjuk a nettó érték ettől eltérő lehet hiszen a különböző logikai fájlrendszerek más-más módon építik fel a tárolási adatszerkezeteket - lásd a fájlrendszerek részt. Az Információ- és kódelmélet című tárgyban részletesen tárgyalásra kerülnek az információ tömörítéséhez illetve az adatvesztéssel szembeni részleges rezisztenciához szükséges módszerek.

További fontos jellemző az adattárolás élettartalma, azaz meddig képes egy eszköz a ráírt információt megőrizni. A mágneses elven működő eszközök általában előbb szenvednek mechanikai hibából kifolyó adatvesztést, mint hogy a mágneses elven tárolt adat elveszne. További, külső tényezők miatt is sérülhet az adatintegritás, például mechanikai behatás vagy hőhatás miatt[2]; a Compact Diskek (CD) esetében például a felület elgombásodása jelent veszélyt az információra nézve.

2. Adattároló perifériák

2.1. Merevlemez (Hard Disk Drive - HDD)

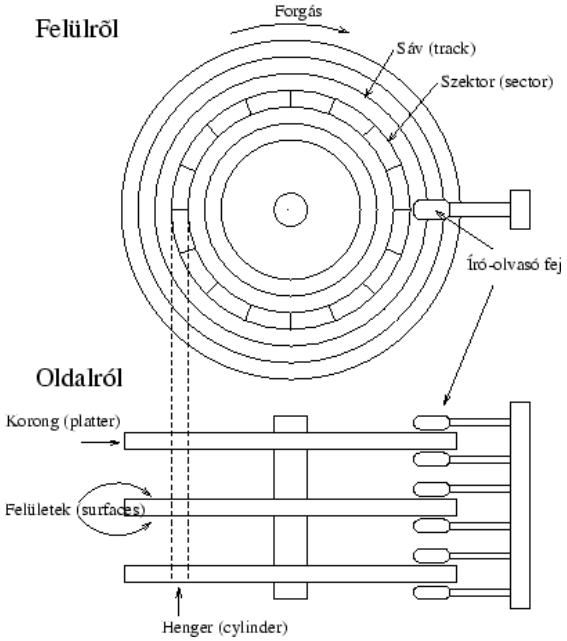
Ahogyan a cserélhető lemezes adattárolást, úgy a „fix" vagy „merevlemezes" adattárolást is az IBM mérnökei találták fel az 1950-es években[3]. Maga az adattárolás elve az elmúlt ötven évben nem sokat változott, egyedül a megbízhatóság és a tárolható adatmennyiség mértéke nőtt drasztikusan. A külső levegőtől elzárt vagy porszűrővel ellátott nyitott ház a következő részeket tartalmazza:

- mindkét oldalán mágnesezhető korongok
- olvasó/író fejek, amelyek a mágneses felület felett pár nanométerrel - légpárnán siklanak[4]
- vezérlő elektronika, amely pozicionálja a fejeket, olvasás esetén elvégzi az analóg mágneses mérés digitalizálását, illetve a csatoló felületnek megfelelő adatátviteli protokollt megvalósítja (IDE, SATA, stb.).
- forgatómotor, jellemzően 5400 illetve 7200 forulat/perc sebességgel forgatja a lemezeket, tehát az olvasó fejek - a sugárirányú pozíciótól függően - kb. 270 km/h-val száguldanak a lemezek felett.

[2] Minden mágneses anyagnak létezik egy úgynevezett Curie-pontja, ezen hőmérséklet felett az anyag elveszti mágneses tulajdonságát

[3] Az első merevlemezt IBM 350 RAMAC néven forgalmazták és 5 megabyte tárolókapacitással rendelkezett, ezt ötven darab 24"-os lemezzel érték el

[4] ebből következik, hogy nincs vákuum a merevlemez belsejében



1. ábra. A merevlemez vázlatos képe

A lemezeket a különböző kerületek mentén sávokra osztják, a sávokat pedig szektorokra, ez a legkisebb címezhető egység egy merevlemezen. Ezek mérete régebben 512 byte volt, jelenleg elérhető 4096 byte is. Mivel az olvasó fejek együtt mozognak, ezért a különböző lemezeken azonos sávon állnak minden pillanatban, ezeket a sávokat együttesen cilindernek nevezzük. Látható tehát, hogy az összetartozó adatokat szomszédos szektorokra ill. azonos cilinderekre érdemes írni. Fontos fogalom még a klaszter, ami az azonos sávon egymás után elhelyezkedő szektorok gyűjtőneve. Lásd az 1. ábrán. A merevlemez hatékony felosztása és az adatok tárolása a merevlemezre telepített fájlrendszer feladata, melyet részletesebben tárgyalunk.

2.2. Compact Disk (CD)

A cserélhető lemezes fejlődési vonalat az utóbbi évtizedig a mágneses tárolási elven működő eszközök határozták meg, de ezek tárolókapacitása nem nőtt és/vagy hozzáférési ideje nem csökkent olyan mértékben, mint a Compact Diskeké, ezért a továbbiakban nem is foglalkozunk velük.

A CD-s adattárolás egy - a merevlemezes tárolástól eltérően - alapvetően optikai elven működő tárolási módszer. Egy lézerdióda által kibocsátott koherens fénysugár tapogatja le a CD lemez felületét, melyen apró gödrök és púpok váltják egymást, melyekről másként verődik vissza a lézersugár - ezzel reprezentálva a bitek értékeit. Gyártás során a CD ROM lemezen - mint a merevlemeznél - sávokat és szektorokat hoznak létre, amelyben a biteket a vájatok reprezentálják. Pl.: ha van mélyedés [pit] akkor az logikai egyes jelent, ha nincs [lane] akkor logikai nullát. Amikor az olvasó fej mindig azonos szögből megvilágítja a felületet, akkor a vájatba beeső lézer fény máshova verődik vissza, mint az lézer fény, mint ami nem esett bele a vájatba. Mivel a lemez fixen síkban forog és az olvasófej is meghatározott szögben világítja meg a felszínt, ezért a várható visszaverődési helyekre fényérzékeny szenzorokat helyeznek el. Értelemszerűen tehát a logikai egyes és nullás értékek más-más szenzorból váltanak ki jelet. Ebből a felépítésből látható, hogy minél fókuszáltabb a lézersugár, illetve minél kisebbek a vájatok a lemez felületén, annál nagyobb az elérhető adatsűrűség (természetesen további fontos paraméterek is vannak - anyagtechnikai jellemzők, a lézer hullámhossza, valamint több adattároló réteggel rendelkező lemezek is léteznek). A CD-ROM elnevezésben a ROM (Read Only Memory/Media) rövidítés arra utal, hogy ezeket az eszközöket csak egyszer lehet írni, utána már csak olvashatóak. Azóta kifejlesztették az újra írható CD lemezeket is, ahol az olvasástól eltérő tulajdonságú lézersugárral visszaállítják az eredeti felületet - természetesen ebben az esetben nem vájatokkal dolgozunk, hanem a felületnek változtatjuk meg a visszaverődési tulajdonságait az írás során.

2.2.1. feladat. Miből adódhat a blu-ray disc (BD) elnevezése?

2.3. Pendrive, Flashdrive, SSD

Mivel ezen eszközök működésének megértéséhez komoly elektronikai háttér tudásra van szükség, ezért a technikai részleteket nem tárgyaljuk. A működési elvről elegendő annyit megjegyeznünk, hogy ezek a tárolók olyan különleges áramkörök, amelyekben lévő tranzisztorok tápfeszültség jelenléte nélkül is képesek megtartani azt az állapotot, amit tápfeszültség jelenlétében beállítottunk. Fontos még megjegyezni, hogy ebből a technikai megvalósításból kifolyólag az ilyen típusú eszközök írási és olvasási sebessége jelentősen eltér egymástól.

A flash memóriák két fő típusát különböztetjük meg: a NAND és a NOR memóriacellából állókat, amik nevüket onnét kapták, hogy a megvalósításuk a logikai NAND (negált ÉS) és NOR (negált VAGY) kapukéra hasonlít. A legfontosabb különbség köztük, hogy a NAND cellák gyorsabban törölhetők és írhatók, kevesebb szilícium területet igényelnek, azaz olcsóbbak is, továbbá a tárolt adatokat csak blokkokban lehet elérni (a NOR flash bájt szintű elérést is lehetővé tesz). Mindebből adódóan napjaink flash memóriái majdnem kizárólag NAND alapúak, és elsődleges feladatuk a háttértárak helyettesítése (mivel azok szintén blokkonként címzettek).

A hardveres megvalósítás iránt érdeklődők a következő kulcsszavak mentén tudnak további információhoz jutni: Flash memory, Floating gate, EEPROM.

2.3.1. Wear levelling

A flash memóriák/tárolók egyik legfontosabb tulajdonsága a már említett blokkonként történő törlés, ami miatt egyetlen bájt megváltoztatása is a teljes blokk törlését követeli meg. Nagyon fontos tehát, hogy flash tárolók esetén több, egymáshoz közel lévő bájt írását egyetlen műveletben végezzük. Sajnos a blokk újraírások számának is van egy felső határa, ami napjainkban (2013) jellemzően százezres nagyságrendű, így a blokkokba szervezett írás sem elegendő, arra is szükség van, hogy a blokkokat lehetőség szerint ugyanannyira használjuk el (ugyanannyiszor töröljük), így megnövelve az eszköz élettartamát. (Ha ezt nem tennénk, akkor lennének olyan blokkok az eszközön, amit olyan sokszor újraírtunk/töröltünk, hogy azok meghibásodnak, és ha ezek olyan helyen helyezkednek el, ami a fájlrendszer szempontjából kritikus, akkor akár a teljes tárolóeszközt is használhatatlanná tennék. A nem kritikus helyen lévő hibás blokkok is bosszúságot okoznak, hiszen ezek azt jelentik, hogy az eszközön tárolt fájl kiolvasott tartalma nem egyezik meg azzal, amit odaírtunk.)

Ez a meghibásodás-elkerülő technika a *wear levelling*, ami egy közbülső réteget képez a flash memória fizikai blokkjai és a felsőbb rétegek (driver, operációs rendszer) által megcímzett logikai blokkok között. Így lehetősége nyílik arra, hogy egy logikailag ugyanarra a blokkra irányuló írás/törlés műveletet más fizikai blokkra irányítson, azaz a fizikai blokkok írásainak/törléseinek számát közel azonos szinten tartsa, így a tárolóeszköz élettartamát nagyságrendekkel meghosszabbítsa (azaz az élettartam ne a legtöbbet írt/törölt logikai blokktól függjön, hanem a teljes eszköz blokkjai gyakorlatilag közel egy időben romoljanak el, az írási/törlési műveletek szétterítésének, egyenletesebbé tételének köszönhetően).

A wear levelling technikában megkülönböztetünk dinamikus és statikus módszert: a dinamikus módszer csak a törlések/írások során végzi el a logikai és a fizikai blokkok összerendelésének megváltoztatását az adott blokk törlési számának figyelembe vételével, míg a statikus módszer a nem írt/törölt blokkokra is kiterjeszti ezt. A különbség tehát, hogy amíg a dinamikus esetben azok a logikai blokkok, amelyeket nagyon ritkán írunk, a helyükön maradnak (és jó állapotban vannak, hiszen csak ritkán írtuk felül), addig a gyakran írt blokkok gyakran cserélődnek, de egyre rosszabb állapotú blokkokon foglalnak helyet (bár ezen blokkok egyenletesen rosszak). A statikus esetben a nem írt/törölt blokkokat is fizikailag áthelyezi a flash vezérlő, így az összes blokk közel ugyanazon az elhasználtsági fokon van. Ha a teljes tárat tekintjük, ez nagy élettartam növekedéshez vezethet (nyilván attól függően, hogy felsőbb szintről a tár mekkora részét írjuk felül/töröljük: például ha egy flash tárolót úgy használunk, hogy a rá másolt adatokat azok felhasználása után töröljük és úgy írunk rá újabb adatot, akkor nincs számottevő különbség a statikus és a dinamikus wear levelling között, viszont ha nagy részén fixen ugyanaz az adat található, és csak kis részét írjuk újra-és-újra, akkor óriási a különbség a statikus és a dinamikus wear

levelling által elért élettartam hosszabbodás között.) A statikus módszer egyben azt is jelenti, hogy akkor is törlést/írást kell végezni, amikor azokat nem a felsőbb rétegek kezdeményezik, emiatt a teljesítménye elvileg kevesebb, mint a dinamikus módszerrel működő háttértáré, ugyanakkor megfelelő ütemezéssel ez a különbség számottevően csökkenthető. Ugyanezen okok miatt a statikus wear levelling komplexebb algoritmust, így komplexebb hardveres implementációt igényel, azaz drágább.

Fontos kiemelni, hogy az ilyen típusú meghibásodások oka kizárólag az írás/törlés műveletek száma, nem pedig az utolsó írás óta eltelt idő, azaz egy csak olvasásra használt flash tároló élettartama nagyságrendekkel hosszabb, mint egy írásra is használté.

A meghibásodások elkerülése érdekében végzett wear levelling mellett szükséges, hogy a már meghibásodott blokkokat is nyilván tartsuk, hogy az arra történő írást elkerüljük. Ekkor viszont arra is lehetőség van, hogy a gyártás során eleve hibás blokkokat szintén megjelöljük, ami hatékonyabb, kevesebb selejttel történő (azaz olcsóbb) gyártást jelent, mivel a jelenlegi technológia nem garantálja a 100%-os hibamentességet.

A wear levellinget és a hibás blokkok nyilvántartását az USB (pen)drive-ok illetve az SSD-k hardveres megvalósításban tartalmazzák (*flash vezérlő*), ezeket az eszközöket úgy kell használni, mintha hagyományos háttértárak lennének, minden fentebb említett feladatot elvégez a hardver. Költség és funkcionalitás szempontjából a pendriveok általában dinamikus- míg az SSD háttértárak statikus wear levellinget tartalmaznak.

3. Partíciók

Lehetőségünk van arra, hogy a háttértáron lévő területet felosszuk és különböző méretű, de összetartozó területeket hozzunk létre. Egy ilyen területet partíciónak hívunk. Minden partíció külön kezelhető a többitől: törölhető, formázható, másolható, és saját fájlrendszerrel rendelkezik. Ezért fizikailag egy lemezen tárolhatunk különböző operációs rendszereket, anélkül, hogy zavarnák egymást.

A partíciók MS Windows alatt betűvel címkézve jelennek meg, pl. C:, D:. - fontos megjegyezni, hogy ha fizikailag másik lemezen van egy partíció az a meghajtó betű jeléből nem deríthető ki (pl. elképzelhető, hogy a C: meghajtó egy partíció, amely a teljes merevlemezt elfoglalja, míg a D: illetve E: meghajtók fizikailag a C: -től különböző, de ugyanazon a lemezen helyezkednek el: valamilyen arányban megosztják a lemez területét).

GNU/Linux alatt már tisztább a helyzet, a *dev* könyvtár tartalmazza a számítógéphez csatolt perifériák eszközfájljait, így a merevlemezekét is. Az IDE csatolóval rendelkező lemezeket `hda`, `hdb`, `hdc`, `hdd`, ... névvel találjuk a könyvtárban, míg a SATA vagy SCSI csatolóval rendelkezőek `sda`, `sdb`, `sdc` stb névvel érhetőek el. Általában két IDE csatlakozó van egy alaplapon, amelyre két-két eszközt lehet csatlakoztatni, ezért a `hda` az elsőleges IDE csatlakozó master eszköze, a `hdb` ugyanezen kábelen lévő slave eszköz. A `hdc`, `hdd` a másodlagos IDE csatlakozóra felfűzött eszközöket jelzik. Ha a `hda` lemez partíciókat tartalmaz, akkor az elsőleges partíció `hda0` néven, míg a második partíció `hda1` néven fog szerepelni a `/dev` könyvtárban. SATA/SCSI eszközök esetén az `sd` után következő `a,b,c,d` betűjelek és az 1,2,3,4 számok ugyanezt jelentik (például: `/dev/sda0` vagy `/dev/sda1`).

Természetesen az operációs rendszert informálni kell arról, hogy milyen partíciók léteznek az adott lemezen, amit a Master Boot Record (MBR) tartalmaz. Ebben találhatók a partíciók méretei, kezdő és vég értékeik valamint az, hogy melyik partíció tartalmaz operációs rendszer elindításához szükséges adatokat - ezt/ezeket a partíciókat hívjuk bootolható partíciónak.

A MBR-t az 1980-as években találták ki. Továbbfejlesztése a GUID Partion Table (GPT), amely számos kiterjesztést tartalmaz az MBR-hez képest, például az MBR esetében a legnagyobb partíció mérete maximum 2 TiB lehet, míg a GPT esetén ez 8 ZiB (ZiB = Zebi Byte = 1024^7 Byte), továbbá GPT partícionálás esetén a partíciók száma is sokkal több lehet.

4. Fájlrendszerek

A fájlrendszer feladata, hogy az eltárolandó fájlokat és könyvtárakat a háttértár egy partícióján a megfelelő helyen elhelyezze, garantálja annak visszaolvashatóságát, valamint a változásokat

adminisztrálja. Tehát a fájlrendszer funkciója kettős: egyrészt tárolja egy adott partíción lévő adatok (fájlok) helyét, másrészt kezeli az ezekhez kapcsolódó metaadatokat[5]. Minden, a fájlrendszerben tárolt adathoz (egy adott fájl fizikai elhelyezkedése a lemezen) tartozik egy metaadat bejegyzés is, ez tartalmazza a fájl vagy könyvtár nevét, a létrehozás, módosítás, utolsó hozzáférés dátumát, a tulajdonos adatait, valamint a hozzáférési jogosultságokat. A könyvtárak a fájlrendszer adatbázisában lévő bejegyzésként vannak tárolva (tehát amennyiben egy lemezről elveszítjük a fájlrendszert leíró adatbázist (a metaadatokat), úgy a nyers adatok (fájlok tartalma) visszanyerhetőek, de: 1) nem fogjuk tudni, hogy melyik fájl hol kezdődött, hogyan következnek egymás után a fájl tartalmát alkotó blokkok és hol van vége 2) nem fogjuk tudni rekonstruálni a könyvtárrendszert.) Látható, hogy a fájlrendszer helyes működése létfontosságú a tárolt adatok használhatóságának szempontjából, éppen ezért a fájlrendszer adatbázisa a lemezen általában több példányban, sokszorosítva kerül eltárolásra, csökkentve a megsérülés valószínűségét.

Mivel a számítógépek számos különböző feladatra használhatóak (családi személyi számítógép, bankszámlakezelő rendszer, egy tőzsdei kereskedőrendszer vagy egy milliós forgalmú webkiszolgáló), ezért az adatok tároláskor is különböző igények léphetnek fel (mind teljesítmény, mind biztonság tekintetében). Ezen igényekre kielégítésére rengeteg fájlrendszer megvalósítás létezik más-és-más tulajdonságokkal, teljesítménnyel, funkcionalitással. Ebben a jegyzetben részletesebben a FAT, illetve az ext2/3/4 fájlrendszerrel fogunk megismerkedni.

Mivel fájlrendszer szükséges magának az operációs rendszernek az installálásához is, a fájlrendszereket az operációs rendszerrel együtt fejlesztik: például MS Windows operációs rendszerhez három fájlrendszer érhető el: a FAT, az NTFS, és az exFAT. Linux alatt a legelterjedtebb az ext3/4 fájlrendszer, de számos egyedi igényt kielégítő megvalósítás is létezik (XFS, JFS, btrfs, Reiserfs, stb).

Léteznek hálózati fájlrendszerek is (Google Drive, NFS, sshfs, stb.), amelyek az operációs rendszer szempontjából átlagos partíciónak tűnnek, de fizikailag az adatok nem az adat számítógép háttértárán tárolódnak, hanem egy távoli szerveren. Például az egyetemen lévő tárhelyeinket (turdus, users) is felcsatolhatjuk meghajtóként az általunk használt operációs rendszerben (sőt, valójában is ez történik, nézzük meg a `mount` paranccsal).

Fontos megjegyezni, hogy a fájlrendszerek fájl- és könyvtárelhelyezési implementációja nem azonos az általunk megszokott könyvtárstruktúrával! Tehát a felhasználók számára látható könyvtárstruktúra mögött a fizikai tárolási módja ettől teljesen eltérő, ahogy ezt látni fogjuk.

4.1. Fragmentáció

Egy fájlrendszerben a fájlok blokkokban tárolódnak. Ha ezek a blokkok fizikailag nem egymás utáni területen helyezkednek el a háttértáron, akkor *külső fragmentáció*ról beszélünk. Ez azoknál a háttértáraknál érdekes, amelyeknél az egymás utáni blokkok olvasása sokkal gyorsabb művelet, mint a távoli blokkok olvasása (ilyen háttértár a merevlemez, mivel a fej mozgatásához illetve a lemez megfelelő helyre forgásához idő kell). Annak érdekében, hogy a fájlok olvasását meggyorsítsuk, a blokkokat egymás után helyezhetjük, amit *defragmentáció*nak nevezünk.

Mivel a fájlokat blokkokban tároljuk, és blokknál kisebb egység lefoglalására nincs lehetőség[6], a fájlt tartalmazó blokkok közül az utolsó majdnem mindig tartalmaz szabad helyet. Ezt *belső töredezettség*nek nevezzük, mivel a blokkokon belül van kihasználatlan hely. Mindebből az is következik, hogy ha N darab 1 bájtot tartalmazó állományt hozunk létre, akkor a fájlrendszerben minden fájlhoz egy blokk lefoglalódik, azaz a lemezen lévő tárhely nem $N \cdot 1$ bájttal csökken, hanem $N \cdot$ blokkméret bájttal.

A blokkméret megválasztásával a belső fragmentáció csökkenthető (hiszen ekkor az utolsó blokk mérete is kisebb, így kevesebb kihasználatlan hely lehet benne), de a blokkok száma növekszik (hiszen kisebb blokkból több jön létre, mivel a teljes partíciót fel kell osztani blokkokra), ami nagyobb lehetőséget ad a külső fragmentációra. Az általánosan használt blokkméret 1-4-64 kiB között változik a partíció méretétől függően, mert ez vállalható kompromisszumot jelent a külső és belső fragmentáció között.

[5] adatok tulajdonságait leíró adatok

[6] az újabb fájlrendszerek már adnak erre lehetőséget, pontosan ezt a problémát megoldandó

4.2. Fájlrendszer implementációk

4.2.1. Flash fájlrendszerek

A flash tárolókra optimalizált fájlrendszerek figyelembe veszik a flash tárolók fentebb tárgyalt sajátosságait, ugyanakkor nagyon fontos, hogy ezek a fájlrendszerek kizárólag közvetlenül a flash memórián használandók, azaz az USB (pen)drive, SSD háttértárak esetében szükségtelen ilyen fájlrendszereket használni, mert a flash speciális kezelését a hardver (a fentebb említett flash vezérlő) elvégzi. Flash fájlrendszerekre példa: JFFS(2), YAFFS.

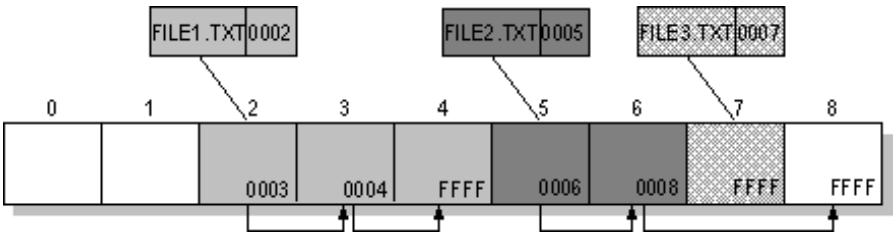
4.2.2. A FAT fájlrendszer

A File Allocation Table (FAT) fájlrendszert a Microsoft fejlesztette, és a Windows XP megjelenéséig ez volt a Windows operációs rendszerek által kizárólagosan használt fájlrendszer. A FAT fájlrendszer klasztereket tart számon, és a különböző FAT verziók főként abban különböznek, hogy hány biten tárolják a klaszterek sorszámait (FAT12, FAT16, FAT32). Ez bitszám határozza meg, hogy összesen mekkora lehet egy FAT partíció mérete. Az operációs rendszert is tartalmazó FAT partíciók tartalmaznak boot sectort is, ez a szektor kerül beolvasásra a memóriába az operációs rendszer bootolásának első lépéseként.

Az alábbiakban részletesen megnézzük, hogyan tárolja a fájlokat a FAT fájlrendszer. A 2 ábrán látható a partíció egy darabja: a tárolt fájlok, valamint a hozzájuk tartozó allokációs tábla-beli bejegyzések. Az allokációs tábla - többek között - tartalmazza a fájl nevét és annak a klaszternek a számát, ahol a fájl kezdődik. Minden klaszter végén található egy cím amely a fájl többi darabját tároló klaszterre mutat vagy egy 0xFFF(F...F) jelzés, ami azt jelenti, hogy ez az utolsó klaszter, amiben a fájl részlete volt eltárolva. Fontos észrevennünk, hogy a rendszer nem követeli meg, hogy egy nagyobb méretű fájl egymás utáni klaszterek sorozataként legyen a lemezen: az operációs rendszer utasításaitól függően akár rengeteg, a lemez különböző pontjain elhelyezkedő klaszterbe is kerülhet a fájl egy-egy darabja. A klaszterméret az esetek túlnyomó többségében 2-32 KiB közötti.

FAT fájlrendszert használnak a pendrive-ok.

4.2.1. feladat. Egyes pendrive-ok esetében a háttértár első néhány kilobájtja speciálisan kialakított. Mi lehet ennek az oka?



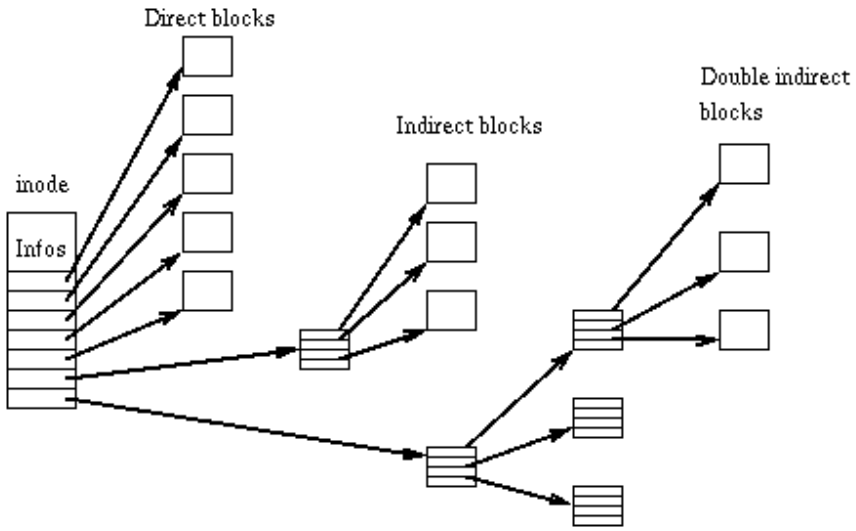
2. ábra. Egy FAT partíció darabja. Forrás:
http://www.ntfs.com/images/recover-FAT-structure.gif

4.2.3. Az Ext2 fájlrendszer

Az ext2 fájlrendszer alapegysége a blokk, amelynek mérete tipikusan 1-8 kiB-ig terjed (ez a fájlrendszer létrehozásakor beállítható érték, de később nem változtatható és minden blokk ekkora méretű lesz). Így ha létrehozunk egy fájlt amiben elhelyezünk 2 karaktert az minimum 1 kiB-ot (vagy épp 8 kiB-ot, ha akkora a blokkméret) fog elfoglalni a lemezen. A szuperblokk a partíció elején helyezkedik el és az operációs rendszer bootolásához szükséges adatokat, illetve magáról a fájlrendszerről egyéb információkat tartalmazz. A blokkokat csoportokban tárolják, ezzel is csökkentve a töredezettség mértékét. Ezeket a blokkokból álló csoportokat *extents*-nek nevezzük.

A ext2 fájlrendszerben minden fájlt és könyvtárat egy úgynevezett inode ír le. Az inode tartalmazza a fájllal vagy könyvtárral kapcsolatos adminisztratív információkat: fájlnevét, létrehozás-,

módosítás dátumát, tulajdonost, jogosultságokat, stb. Az inode többi része 12-15 linket (blokk címeket) tartalmaz, amely egy csoportot címez meg, ezek a direkt blokkok (lásd a 3. ábrán). Amennyiben a fájl mérete meghaladja a direkt blokkokban tárolható adatmennyiséget, akkor az utolsó link helyére nem egy direkt blokk címet helyezünk az inode-ban, hanem egy másik csoportleírót, ami további blokkokra vagy csoportleírókra mutat. Ezzel a módszerrel a legnagyobb tárolható fájl mérete 1 kiB-os blokk méretnél 16 GiB, míg 8 kiB blokkméret esetén 2 TiB.



3. ábra. Az ext2 inode felépíése Forrás:
http://upload.wikimedia.org/wikipedia/commons/a/a2/Ext2-inode.gif

A fájlok és könyvtárak mellett létezik egy másik típusú inode bejegyzés-típus is, ez a link. A link nem más, mint egy bejegyzés a fájlrendszerben, amely egy másik fájlrendszer-beli bejegyzésre hivatkozik. Önmagában tehát nem tárol adatot, hanem az őt megnyitó programot továbbirányítja az általa mutatott fájlra (ennek egyszerűbb változata az MS Windows-beli parancsikon). Két típusát különböztetjük meg, az egyik a soft link a másik a hard link.

A hard link esetében a könyvtárbejegyzésben szereplő inode bejegyzés egy már létező i-node-ra mutat. A hard linkek tehát pontosan ugyanúgy néznek ki, mint az adott fájl első könyvtárbejegyzése, azaz a hard linkek egyenrangúak! Ezzel szemben a soft link egy speciális fájl, amely annak a fájlnak az elérési útját tartalmazza, amire mutat. Ebből következően a hard linknél a mutatott fájl vagy könyvtár addig nem törölhető, amíg létezik rá mutató link (ezt a link számlálóból tudja - lásd feladatok). Hard link létrehozására az `ln` parancs használható. Szintaktikája: `ln régi új` ahol `régi` jelenti azt a már meglévő fájlt, amire linket akarunk létrehozni, és `új` jelenti a létrehozandó linket. Az `ls -i` paranccsal kilistázhatók az i-node számok is, így ellenőrizhető, hogy a hard link valóban ugyanarra az i-node-ra mutat. Fontos, hogy hard link csak fájlra hozható létre (azaz könyvtárra nem)! Soft linket az `ln -s régi új` paranccsal hozhatunk létre.

A soft linknél a link az elérési utat tárolja, így a mutatott fájl vagy könyvtár nem tudja, hogy létezik olyan hivatkozás, amely reá mutat. Éppen ezért ha letöröljük a hivatkozott fájlt, a link „célpont” nélkül marad, és „törött” link [dangling / broken link] jön létre. Másik fontos különbség, hogy hard linket csak partíción belül lehet létrehozni, mivel az inode-ra mutat, aminek a számozása partíción belül egyedi. Ezzel szemben a soft link elérési utat tárol (ahogy azt már említettük egy partíció a könyvtárstruktúra tetszőleges pontjára becsatolható), így a soft link mutathat másik partíción elhelyezkedő fájlra is.

4.2.4. SWAP fájlrendszer

Egy adott pillanatban nem minden programot használunk, amit elindítottunk a számítógépünkön, illetve az adott programnak sem használjuk minden részét. Ebből kifolyólag a nem aktív programokat, valamint programrészeket az operációs rendszer nem a viszonylag szűkös memóri-

ában tartja, hanem a háttértáron, az úgynevezett swap területen. A tárolás olyan formátumban történik, hogy a kiírt memórialapokat szükség esetén külön keresés-konvertálás nélkül a memóriába tudja visszatölteni. (Például: amikor a tálcára leteszünk egy programot és sokáig nem foglalkozunk vele, majd később elővesszük azt tapasztaljuk, hogy elég lassan reagál a kéréseinkre, és a háttértár nagy tempóban dolgozik: ekkor kerülnek vissza a swap területről a memóriába az adott programhoz tartozó adatok és program részletek). MS Windows alatt a fájlrendszerben egy fájlként jelenik meg a swap terület, amit Pagefile-nak hív a rendszer. GNU/Linux rendszereknél a swap tárterület fájl mellett egy linux-swap típusú fájlrendszerrel rendelkező külön partíció is lehet.

5. Könyvtárstruktúra és a fájlrendszer adminisztrációjának manipulációja

Linux alatt a BASH shell segítségével lehetőségünk van a parancssori utasítások segítségével fájlok és könyvtárak létrehozására, módosítására, valamint törlésére, tehát a könyvtárstruktúra módosítására. Továbbá lehetőségünk van a fájlrendszer adminisztrációs információk megjelenítésére, megfelelő jogosultság esetén módosítására.

5.1. Alapvető parancsok

A `cd`, `pwd`, `mkdir`, `rmdir`, `ls`, `rm`, `mv`, `cp` parancsokat az első óra anyaga tartalmazta. A `cat` paranccsal egy fájl tartalma jeleníthető meg, a `touch` paranccsal egy üres fájl hozható létre.

5.2. Jogosultságok

A linux disztribúciókban található egy kitüntetett felhasználó, a rendszergazda, ami a telepítés során jön létre, a neve: root. Neki mindenhez joga van, bármit törölhet, bármit megnyithat, létrehozhat felhasználót, stb. Az ő általa indított programok az ő jogaival futnak, egy szándékosan vagy véletlenül megváltoztatott program a root jogaival futva komoly károkat tud okozni. Ezért a legtöbb disztribúcióban létre kell hozni már a telepítéskor egy korlátozott jogú felhasználót, akinek az adataival belépve korlátozott jogokkal tudunk dolgozni. Ez így biztonságos!

5.2.1. példa. Írjuk be: `cat /etc/passwd` A kapott hosszú lista első oszlopa a rendszerünkön lévő felhasználók neveit tartalmazza, a sajátunkat is ott kell látnunk. (Talán kiderült már: a cat utasítással szöveges fájlok tartalmát lehet kilistázni.)

A lentebb bemutatásra kerülő jogoknak igazi jelentősége a több felhasználó által használt rendszerek esetében van (pl.: users és turdus szerverek), ha egy gépet csak egyedül mi használunk, a jogosultságok állítgatása nem lesz annyira fontos.

Minden felhasználó valamilyen csoportnak is tagja (akár többnek is), mindenkinek van egy alapértelmezett csoportja (elsődleges csoport), ez Debian rendszeren megegyezik a felhasználó nevével, a felhasználó létrehozásakor jön létre, az új felhasználó egyből belekerül.

5.2.2. feladat. Adjuk ki a következő utasítást: `cat /etc/group` A kapott lista első oszlopa a rendszerünkön lévő csoportok neveit tartalmazza.

A Linux fájlrendszere tárolja a fájl tulajdonosának azonosítóját a fájlhoz tartozó csoportot és a hozzáférési jogosultságot is. A hozzáférési jogosultságok ábrázolásához egy három részből álló kódot használ, amit fájlmodnak nevezünk.

- Első rész a saját (user) jogot
- Második rész a csoport (group) jogot
- Harmadik rész mindenki más (others) jogait rögzíti

A saját jog alatt a fájl tulajdonosának jogait értjük, legtöbb esetben ő az adott fájl vagy könyvtár létrehozója is. Mindegyik rész a következő komponensekből áll:

- r (Read): olvasási jog (vagyis az adott fájl ezáltal olvasható)
- w (Write): írási jog (az adott fájl ezáltal válik írhatóvá)
- x (eXecute): végrehajtási jog (futtatási jog)

5.3. Jogosultságok megváltoztatása

Egy fájl tulajdonosi (hozzáférési) jogait csak a fájl tulajdonosa, vagy a rendszergazda tudja megváltoztatni a következő paranccsal: `chmod +|-<mód> <fájlnév>` Meg kell határozni az alábbiakat: adunk vagy elveszünk jogot (+: adunk, -: elveszünk), kinek/kitől (saját, csoport, mindenki más (ugo)), milyen jogot adunk (r w x / 4 2 1).

5.3.1. példa. Saját magunknak írási jog: `chmod u+w munka.tar.gz`

5.3.2. példa. Másoknak futtatási jog: `chmod o+x munka.tar.gz`

5.3.3. példa. Egyszerre több jogot is meg lehet változtatni: `chmod o+x,u+w munka.tar.gz`

5.3.4. példa. Mindenkinek minden jog: `chmod 777 munka.tar.gz` ugyanezt a funkciót valósítja meg a `chmod a+rwx munka.tar.gz`

5.3.5. példa. Csak nekem legyen jogom mindenhez: `chmod 700 munka.tar.gz` ugyanaz mint: `chmod u+rwx,g-rwx,o-rwx munka.tar.gz`

Fájlok esetében a végrehajtási jognak csak a futtatható fájloknál van jelentőségük (bináris állományok, scriptek). Könyvtárak esetén az olvasási jog azt jelenti, hogy elolvashatja a fájlok neveit az adott könyvtárban, az írási jog jelenti, hogy a könyvtárban állományt, könyvtárat hozhatunk létre, míg a futtatási jog megengedi a belépést a könyvtárba.

5.3.6. feladat. Hozzunk létre egy könyvtárat, és változtassuk meg a jogosultságait úgy, hogy aki ismeri a könyvtárban lévő fájlneveket, alkönyvtárakat, az el tudja ezeket olvasni, de más nem!

5.4. Alapértelmezett jogok

Amikor egy fájlt létrehozunk, akkor az a jogosultságok alapértelmezett értékével fog rendelkezni. Pl.: Létrehozunk egy üres fájlt:

```
$ touch akarmi
$ ls -la akarmi
-rw-r--r--  1 bnorbert  staff  0 Okt 31 06:14 akarmi
```

A létrehozáson kívül, alapértelmezés szerint írási és olvasási joggal, a csoportba tartozók és mindenki más pedig csak olvasási joggal rendelkeznek. Ennek az az oka, hogy az operációs rendszer a fájl létrehozásakor a 022 maszkot alkalmazza. Egy állomány létrehozásakor alapértelmezésben senki sem kap futtatási jogot. Az alapértelmezett maszk lekérdezhető a következő paranccsal:

```
$ umask
022
```

Könyvtárak létrehozása esetén a 777-ből vonódik ki a mask, azaz alapértelmezetten egy könyvtár 755 jogokkal jön létre. Fájloknál a 666-ból vonódik ki a mask, így 644 jogokkal jönnek létre a fájljaink.

5.5. Tulajdonos megváltoztatása

Az egyes bejegyzések (fájlok és könyvtárak) tulajdonosának megváltoztatása a chown parancs segítségével történik, valamint felhasználó csoport váltása a chgrp parancs segítségével lehetséges. A jogosultságokat is tartalmazó részletes listát a `ls -l` paranccsal kaphatunk (illetve kombinálhatjuk a már megismert `-i` kapcsolóval is, hogy az i-node azonosítók is láthatóak legyenek: `ls -li`

5.6. Fájlrendszerrel kapcsolatos parancsok

- Az fsck parancs segítségével lehet ellenőrizni, hogy a háttértár tartalma megegyezik-e az adminsztrációs fájlok által leírt állapottal, azaz a fájlrendszer koherens állapotban van-e. Ilyen például akkor fordulhat elő, amikor hirtelen kikapcsol a számítógép (pl. áramszünet esetén) és valamilyen lemezművelet félbeszakad. Szintén problematikus eset a fájlrendszer koherenciájának szempontjából, ha akkor távolítunk el egy cserélhető eszközt, amikor még nem fejeződött be a lemezre írás művelet.
- Fájlrendszer egy üres partícióra a mkfs parancs segítségével tudunk létrehozni, a parancs lefutása létrehozza az összes adminisztrációs állományt, ami szükséges a fájlrendszer menedzseléséhez. Hasonlóan, ha egy fájlrendszerrel rendelkező partíciót leformázunk, akkor a formázás létrehozza az üres adminisztrációs fájlokat (fontos, hogy ezzel még az előző fájlrendszerben tárolt adatok megmaradnak, csak nem tartozik hozzájuk adminisztrációs állomány).
- Az érvényes, hibamentes fájlrendszereket tartalmazó partíciókat használat előtt fel kell csatolnunk a könyvtárstruktúrába. Általában az mnt könyvtárban van egy - partícióhoz tartozó - üres könyvtár, ahová a mount paranccsal tudjuk becsatolni a partíciót.

5.7. Feladatok

- Mit csinál a df parancs? Keressük meg a man oldalán, hogy mit csinál a -T kapcsoló és futtassuk a df -T parancsot.
- Nézzük meg a stat parancs man oldalát, próbáljuk ki a következőekre: sima fájl, könyvtár, eszközfájl, soft link, hard link.
- Nézzük meg az ls -i parancsot, keressünk egy fájl a könyvtárból és nézzük meg, hogy az ls -i parancsban megadott inode szám egyezik-e a stat parancs kimenetével.
- Hozzunk létre soft és hard linkeket, fájlra, könyvtárra, figyelünk a link counter értékének változására. Töröljük azt a fájlt amire a link mutat mit tapasztalunk szoft illetve hard link esetén?
- nézzük meg a dumpe2fs parancsot és futtassuk az egyik partícióra. A grep parancs segítségével (grep -i superblock) nézzük meg hány példányban tárolódik a lemezen a szuperblokk.

6. Könyvtárszerkezet

Linux alatt fa gráfba van szervezve a teljes könyvtárszerkezet, (azaz ne számítsunk C, D, ... meghajtókra!) Mindennek az alapja a `/` jellel jelölt gyökérkönyvtár, más néven root. Ez minden fájlrendszer alapja, ebből ágaztatható le a teljes szerkezet.

6.0.1. példa. Adjuk ki a következő utasítást: `ls /` Hasonló listát kell látnunk:

```
bin
boot
cdrom
dev
etc
home
lib
lost+found
media
mnt
opt
proc
root
```

```
sbin
sys
tmp
usr
var
vmlinuz
```

Ezek a főkönyvtárak majdnem minden Linuxban változatlanul megvannak, leszámítva talán a /cdrom-ot és /media-t. A /media egy újabb „találmány", ide kerülnek a cserélhető médiák. Nézzük, melyikben mi található:

bin, sbin: A bin könyvtárakban futtatható állományok vannak. Több bin könyvtár is található ezen kívül, például a /usr/bin és a /usr/sbin. Bár ez nem törvényszerű, de általában a bin könyvtárakban a minden felhasználó által elérhető programok kerülnek az sbin könyvtárakba pedig olyan rendszereszközök, melyeket általában rendszergazdák használnak. A /bin és /sbin az alaprendszerhez, a boot folyamathoz szükséges programokat tartalmazza, a felhasználói programok a /usr/bin /usr/sbin alá kerülnek.

boot: a boot könyvtárban találhatók a bootnál fontos fájlok: általában a rendszermag (kernel), illetve Grub rendszerbetöltő esetén annak konfigurációs állománya is.

cdrom:Ez alá csatolódik be a CD meghajtó egység.

dev: Linux alatt fájlokon keresztül érünk el mindent, a CD-vel kezdve, a hangon át, az egérig. Ezek a speciális eszközfájlok találhatók ebben a mappában.

etc: Az etc könyvtár a gyűjtőhelye a különböző programok globális konfigurációs fájljainak. Ellentétben a Windowsos registry megoldással, Linux alatt minden konfigurációs állomány egyszerű szövegfájlba van mentve, aminek nagy előnye, hogy az állományok akkor is egyszerűen elérhetők, ha a rendszer egyébként használhatatlan. Természetesen emellett az egyes programok felhasználó specifikus beállításokkal is rendelkeznek, ezeket a home könyvtárakban tárolja a rendszer, rejtett mappákban.

home: ez alatt a könyvtár alatt találhatók a felhasználói könyvtárak, az adott könyvtár alatt a felhasználónak teljes dúlási joga van, ezen az egy könyvtáron kívül azonban leginkább csak olvasási joga van alapból.

lib: a lib könyvtár alatt már a rendszer részei lapulnak: library fájlok, kernel modulok, stb.

lost+found: egy speciális könyvtár, jelen esetben egy ext3 típusú fájlrendszerrel szerelt partícióról van szó, ez a könyvtár nem is a Linux, mint inkább a fájlrendszer része: a fájlrendszer javításakor előkerült, névvel nem rendelkező fájl darabokat helyezi el itt a rendszer.

media: rendszerfüggő, általában a /media könyvtár alá kerülnek befűzésre a CD/DVD eszközök, pendrive illetve a floppy. Röviden: a cserélhető médiák.

mnt: a másik becsatolásra használt könyvtár. Ez alá a könyvtár alá kerülnek (általában) csatolásra a fix partíciók. Mivel ebben a könyvtárstruktúrában nincs kiemelt helye/neve egy egy meghajtónak, mint Windows alatt a C:, D:, stb., így egy-egy eszközt tetszőleges helyre befűzhetünk a fájlrendszerbe. Különösen praktikus ez például a home könyvtár estén: ha kinőjük az e célra fenntartott partíciót, és veszünk egy új vincsesztert, egyszerűen csak rámásoljuk anyagainkat, letöröljük az eredeti példányt, majd befűzzük a /home könyvtár alá az új adathordozót.

opt: a hivatalos leírás szerint külsős programok települnek ebbe a könyvtárba, de a rendszerek nagy részén üresen áll...

proc: Itt találhatóak az éppen futó folyamatokkal kapcsolatos metaadatok, illetve információk a rendszerről: processzorról, memóriáról, stb. Nagy mennyiségű hasznos információt talál itt az avatott kéz.

root: A rendszergazda (root) felhasználó home könyvtára

tmp: Az egyes programoknak szükségük van/lehet átmeneti fájlokra. Ezek kerülnek ide. Ez a másik olyan könyvtár, amely alapértelmezetten írható minden felhasználó számára.

usr: Ez alatt a könyvtár alatt található minden. Persze ez így kicsit túlzónak hat, de majdnem igaz: az usr könyvtár alatt található a telepített programok nagy része, hagyományból ide szoktunk forrásokat pakolni (/usr/src), és azt lefordítani. Itt találhatók a dokumentációk, itt találhatók az ikonok nagy része, stb...

var: Szintén számos szolgáltatás gyűjtőkönyvtára. Itt találhatók a naplófájlok, egyes programok hosszabb ideig tárolt, mégis átmeneti fájljai, alapértelmezetten a felhasználói postaládák, stb.

6.0.2. feladat. Nézzünk bele az egyes könyvtárakba: adjuk ki a következő utasítást (utánuk ENTER): `ls /bin` (aztán `ls /boot`, `ls /home`,...)

6.0.3. feladat. Gépeljük be, majd nyomjunk ENTER-t: `cat /proc/meminfo`

7. Egyéb parancsok

Néhány gyakran használt, fontosabb parancs:

date: kiírja az aktuális dátumot.

df: disk free, egy kis statisztikát jelenít meg az egyes partíciók foglaltságáról. pl.: `df -h`

du: disk usage, az egyes állományok, könyvtárak méretéről készít kis statisztikát. pl.: `du -hs ./` (a man alapján próbáljuk meg értelmezni az egyes kapcsolókat, paramétereket!)

ncal: calendar, egy kis naptár program. pl.: `ncal 2011`

Természetesen a listát még hosszasan lehetne sorolni, aki további parancsokkal szeretne megismerkedni, használja ki az internet lehetőségeit! Bármely kereső a „linux parancsok” kifejezésre több jól használható oldalt is ajánl.

8. Feladatok

8.0.1. feladat. Nézz utána, hogy mit csinál az ncal parancs!

8.0.2. feladat. A hét milyen napján születtél?

8.0.3. feladat. Mekkora helyet foglalsz a users.itk.ppke.hu szerveren?

8.0.4. feladat. Hozd létre a következő könyvtárstruktúrát a saját könyvtáradon belül!

```
./szulok/apa
./szulok/anya
```

8.0.5. feladat. Hozz létre egy fájlt (akár üreset is lehet) az apa alkönyvtáron belül! (touch, esetleg nano, esetleg cat,. . .)

8.0.6. feladat. Másold át az anya alkönytárba!

8.0.7. feladat. Írasd ki egy fájlba az elmúlt 10 percben módosított fálok neveit a munkakönyvtáradon belül! (find parancs)

8.0.8. feladat. Fűzd hozzá a fájl végéhez az aktuális dátumot! (date és átirányítás)

8.0.9. feladat. Módosítsd az előző fájl jogait, hogy neked csak írási jogod, másoknak (csoport, egyéb) pedig semmilyen joga ne legyen!

8.0.10. feladat. Próbáld meg a tartalmát kilistázni! (pl.: cat)

8.0.11. feladat. Szerezz információkat az od programról! (man, keresők,. . .)

8.0.12. feladat. Add ki a következő utasítást:
verb=echo ő | od -t x1
Értelmezd az eredményt!