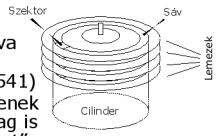
Operációs rendszerek: Fájlrendszerek

Balogh Ádám
Lőrentey Károly
Eötvös Loránd Tudományegyetem
Informatikai Kar
Algoritmusok és Alkalmazásaik Tanszék

Tartalomjegyzék

Lemezegységek felépítése

- Logikai felépítés: blokkok egydimenziós tömbje
  - 1 blokk általában 256 vagy 512 bájt
  - Le kell képezni a fizikai felépítésre
- Fizikai felépítés: cilinder, sáv, szektor
  - a hardver firmware-je elfedheti a valós felépítést
  - a lemez közepe felé haladva a sávon belüli szektorok száma csökkenhet (pl. C1541)
  - CD, DVD lemezeken nincsenek sávok; a szektorok fizikailag is egy spirális alakba „feltekert” egydimenziós tömbben helyezkednek el



Fizikai és logikai formázás

- Fizikai formázás
  - A sávok és szektorok kialakítása az adathordozó felületen
  - Általában már a gyárban elvégzik, de szükség esetén a felhasználó megismételheti (pl. szektorméret-változtatás céljából)
  - A szektoroknak „láthatatlan” fej- és láblécük van, benne a szektor számával és a hibajavító kóddal
- Logikai formázás
  - A felhasználás előkészítése
  - Két lépésből áll:
    1. A nyers lemezterület felszeletelése elkülönített részekre (partíciókra)
    2. Az egyes partíciókon a felhasználáshoz szükséges adatszerkezetek felépítése (a fájlrendszer létrehozása)

Boot blokk

- A lemez első blokkját a rendszer speciális célra tartja fenn, neve *boot blokk*
- Egy rövid programot tartalmaz, mely a memóriába olvassa az operációs rendszer magját, és ráadva a vezérlést elindítja a rendszert
- A rendszer bekapcsolásakor a ROM-ba huzalozott rendszerindító rutin olvassa be és indítja el a boot blokkot
- A rendszerindítás általában többszintű, bonyolult folyamat

Tartalomjegyzék

Bevitel/kivitel ütemezés (1)

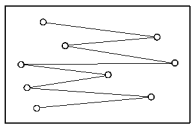
- Alacsonyszintű IO kérések szerkezete:
  - kérés fajtája (olvasás vagy írás)
  - a kért blokk száma (vagy fizikai cím)
  - pufferterület címe a memóriában
  - mozgatandó bájtok száma
- Általában egy lemezegységet egyszerre több folyamat is használni akar
  - Több IO kérés is kiszolgálásra várakozik
  - Melyiket hajtsuk végre először?

Bevitel/kivitel ütemezés (2)

- Az író olvasó fej mozgatása sokáig tart
  - Fejmozgási idő (seek time)
  - Egy cilinderen belül nem kell mozogni
- Ha már a megfelelő cilinderen állunk, meg kell várni, míg a megfelelő szektor a fej alá pörög
  - Elfordulási idő (rotational latency)
- Nem mindegy, hogy milyen sorrendben olvassuk, írjuk a blokkokat
- Az IO ütemező feladata a kérések kiszolgálási sorrendjének „jó” megválasztása
  - Fejmozgások, elfordulási idő minimalizálása
  - Átlagos válaszidő csökkentése, sávszélesség növelése
  - Cserébe nő a CPU igény (overhead)

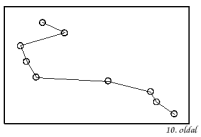
Sorrendi ütemezés (FCFS)

- First Came, First Served
  - A kéréseket egyszerűen a beérkezés sorrendjében szolgáljuk ki
  - Hasonló a FIFO ütemezéshez
- Nem törődik a fej mozgásával
  - Hosszú válaszidő, kis sávszélesség
  - Cserébe a válaszidő szórása kicsi
  - Igazságos ütemezés, kiéheztetés nem fordulhat elő



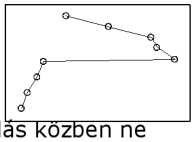
Lusta ütemezés (SSTF)

- Shortest Seek Time First
  - A kéréseket a kiszolgáláshoz szükséges fejmozdulás sorrendjében szolgálja ki
  - Lásd SJF ütemezés
- A fejmozdulások töredékükre csökkennek
  - Sávszélesség látványosan megnő
  - Válaszidő szórása nagy
  - Fennáll a kiéheztetés veszélye, főleg hosszú várakozás sor esetén



Lift ütemezés (SCAN)

- Pásztázó algoritmus
  - A fej a diszk egyik szélétől a másikig ide-oda pásztáz
  - Az éppen útba eső kérést szolgálja ki
  - A liftek működéséhez hasonlít
- Jó kompromisszum
  - Sávszélesség nagy
  - Kiéheztetés szinte kizárva
  - Várakozási idő szórása viszonylag nagy
- Javítás: a visszafelé haladás közben ne szolgáljunk ki kéréseket (C-SCAN)



Előlegező ütemezés

- *Anticipatory I/O scheduling*
- A lift ütemezés heurisztikus javítása
- Gyakran egymás utáni blokkokat olvasnak a folyamataink (szekvenciális olvasás)
- Egy ilyen olvasás után várakozzunk egy kicsit mielőtt más kéréseket kiszolgálunk
  - *Egy kicsit*: néhány milliszekundum
  - Ha tudjuk, hogy a folyamat nem szekvenciálisan olvas, ne várakozzunk
  - Más információ híján várakozzunk mindig
- Párhuzamos szekvenciális olvasások esetén megspórolhatjuk az ide-oda pásztázást

Bevitel/kivitel ütemezés (3)

- Melyik ütemező algoritmust válasszuk?
- Sorrendi ütemezés
  - Triviális implementáció, nincs is ütemező
  - Ha egyszerre csak egy kérés van, csak egyféleképpen lehet ütemezni
  - Egyfelhasználós rendszerek
- Lusta ütemezés
  - Egyszerű, természetes, hatékony
  - Túl nagy terhelés esetén fellép a kiéheztetés
- Lift ütemezés
  - Nagy IO terhelésű rendszerek
  - Fair ütemezés, megelőzi a kiéheztetést

Bevitel/kivitel ütemezés (4)

- Az elfordulási idő is késlekedést okoz
  - Modern lemezegységeken a fejmozdulás ideje alig haladja meg az elfordulás miatti késedelmet
  - A fenti algoritmusok csak a fejmozdulási időt veszik figyelembe

Bevitel/kivitel ütemezés (5)

- Beépített ütemezés
  - A lemezegység saját ütemezőt tartalmazhat (pl. SCSI-2 tagged queuing)
  - Az adott modellhez illeszkedő, elfordulási időt is figyelembe vevő ütemezés
  - Az OS a kéréseket ömlesztve továbbítja a lemezvezérlőnek, rábízza az ütemezést
  - Az OS feladatszintű IO ütemezést is végez (lapozás vs. egyéb, írás vs. olvasás, stb.)

Kiszolgálási idő csökkentése

- Ügyes szervezéssel hatékonyabbá tehetjük rendszerünket
  - Az összetartozó adatok legyenek egymás mellett a lemezen
  - A sávszélesség a lemez szélén a legnagyobb
  - A leggyakrabban használt adatok legyenek a lemez közepén, vagy tároljuk őket több példányban
  - Olvassunk/írjunk egyszerre több blokkot
  - A szabad memóriát használjuk fel lemez-gyorsítótárnak
  - Adattömörítéssel csökkentsük az IO műveletek számát (a CPU igény rovására)

Tartalomjegyzék

Partíciók

- A partícionálással a lemezt független szeletekre osztjuk
- A partíciók az alkalmazások és az OS magasabb rétegei számára általában a lemezegységekhez hasonló eszközként látszanak
- Az egyes partíciókat különböző célokra használhatjuk
  - Nyers partíciók (pl. adatbáziskezelőknek)
  - Virtuális memóriaterület (*swap*)
  - Fájlrendszer

RAID (1)

- Redundant Array of Inexpensive Disks (olcsó lemezegységek redundáns tömbje)
  - Ha egy diszk átlagosan 100 000 üzemóra (kb. 11 év) után mondja fel a szolgálatot, akkor egy 100 diszkből álló rendszerből kb. 42 naponta egy diszk kiesik!
  - Megoldás: az adatainkat tároljuk egyszerre több diszken
  - A redundancia megvalósítását rejtsük egy virtuális lemezegység mögé (nincs szükség új interfészre)
  - Különböző RAID szintek közül választhatunk (RAID-0-6)

RAID (2)

- Szoftverből és hardverből is megvalósítható
  - Hardver-RAID esetén általában egész diszkeket kötünk össze, az OS szemszögéből az eredmény egy szokásos lemezegységnek látszik
  - Szoftver-RAID-et az OS valósítja meg, így partíciók felett is működhet
  - A hardver megvalósítás drágább, de hatékonyabb

RAID 0 (Striping)

- Néhány diszk tárterületének összefűzésével megsokszorozhatjuk az összefüggő(nek látszó) tárkapacitást
- A logikai diszk blokkjait általában felváltva osztjuk szét a fizikai diszkek szektorai között (*striping*)
- Az IO műveletek párhuzamosításával nő a teljesítmény
- Nincs redundancia!
- Az adatvesztés esélye nem csökken, hanem nő
- Általában a blokknál nagyobb egységeket kezelünk (*stripe*), de akár bitszintű szétosztás is lehetséges

RAID 1 (tükrözés)

- Minden adatot két független diszken tárolunk
- A tárolókapacitás a felére csökken
- Olvasási teljesítmény nőhet, írás nem változik, vagy kissé csökken
- Diszkhibából eredő adatvesztés esélye jelentősen csökken
- Egyszerű, de drága megoldás
  - Nagyon kicsi processzorigény
  - 1 GiB adat tárolásához 2 Gib diszkterület szükséges

RAID 2 (ECC)

- A gépi memóriánál megszokott hibajavító kódok használata (ECC memória)
- Az adatbitek mellett néhány extra bitet is tárolunk (lásd Hamming kódok)
- A bájt bitjeit és a hibajavító biteket tároljuk különböző diszkeken
- Az egyik diszk meghibásodása esetén a paritásbitekből helyreállítható a hiányzó bit
- pl. 4 diszk kapacitáshoz elég 7 fizikai diszk
- A gyakorlatban ritkán használt

RAID 2 (ECC) – Példa (1)

- Négy adatbit, három paritásbit
  - Háromféle paritást számolunk, a hét bit három különböző, átlapolt részhalmazára:
    - 1. paritás: 1., 3., 5., 7. bitek összege mod 2
    - 2. paritás: 2., 3., 6., 7. bitek összege mod 2
    - 3. paritás: 4., 5., 6., 7. bitek összege mod 2
  - A paritásbiteket úgy választjuk meg, hogy az összeg 0 legyen (páros paritás)
  - Kiosztás: p_1, p_2, d_1, p_3, d_2, d_3, d_4
  - Ellenőrzés: ha van nem nulla paritás, az hiba
  - A számolt paritásokból közvetlenül előáll a hibás bit sorszáma

RAID 2 (ECC) – Példa (2)

- Példa:
  - 1011 elkódolása: 0110011
  - Mivel:
    - $c_1 = 0 + 1 + 0 + 1 \bmod 2 = 0$
    - $c_2 = 1 + 1 + 1 + 1 \bmod 2 = 0$
    - $c_3 = 0 + 0 + 1 + 1 \bmod 2 = 0$
  - Hiba az ötödik biten: 0110111
  - Számolt paritások:
    - $c_1 = 0 + 1 + 1 + 1 \bmod 2 = 1$
    - $c_2 = 1 + 1 + 1 + 1 \bmod 2 = 0$
    - $c_3 = 0 + 1 + 1 + 1 \bmod 2 = 1$
  - $c_1c_2c_3 = 101_2 = 5_{10}$, tehát az ötödik bit hibás

RAID 3 (paritásbitek)

- A memóriával ellentétben a lemezegységek jelzik, ha hiba történik
- Nincs szükség a teljes hibajavító kódra, elég egyszerűen a paritásbitet tárolni (XOR)
- Az ismert pozíciójú hibás bit ebből helyreállítható
- Előnyök:
  - Olcsó: n diszk kapacitáshoz elég n+1 diszk
  - Egy blokk írása/olvasása szétosztódik a diszkek között, tehát felgyorsul
- Hátrányok:
  - Magasabb CPU igény
  - I/O műveletekben az összes diszk részt vesz, a párhuzamos teljesítmény romlik

RAID 4 (paritásblokkok) (1)

- A RAID 0 megoldást egészítsük ki egy paritásdiszkkel
- Nincs szükség a bájtok felszabdalására, a paritás független blokkokra is számolható
- Egy diszk kiesése esetén a paritásdiszk és a többi diszk blokkjaiból helyreállíthatók az adatok

RAID 4 (paritásblokkok) (2)

- Előnyök:
  - A RAID 3-hoz hasonlóan olcsó
  - Egy blokk beolvasásához elég egyetlen diszk, így a független olvasások párhuzamosíthatóak
- Hátrányok:
  - Az egyedi olvasásműveletek sebessége csökken
  - Az írások nem párhuzamosíthatóak (a paritásdiszket minden írás használja)
  - A diszkek igénybevétele nem egyforma

RAID 5 (elosztott paritásblokkok)

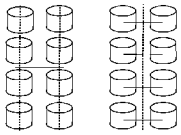
- A RAID 4 javítása: a paritásblokkokat keverjük az adatblokkok közé
- Például egy 5 diszkből álló tömbben az n. blokkhoz tartozó paritásblokkot tároljuk az (n mod 5)+1. diszken, a többi diszk n. blokkjai tárolják az adatokat
- A diszkek igénybevétele kiegyenlítődik
- Az írások némileg párhuzamosíthatók, azonban még mindig jóval lassabbak az egyszerű tükrözésnél

RAID 6 (P+Q)

- Paritásblokk (P) mellett hibajavító kódok (Q, Reed-Solomon)
- n+2 diszk költséggel n diszknyi kapacitást nyújt, és bármely két diszk kiesését elviseli
- Matematikai háttér: Galois-terek
- Jelentős, a RAID 5-nél is magasabb CPU igény
- Elvileg általánosítható kettőnél több diszk kiesésére, a gyakorlatban általában nem éri meg
- A P és Q blokkokat célszerű itt is az adatblokkok közé keverni

RAID 0+1, RAID 1+0

- A RAID 0 teljesítményét ötvözhetjük a RAID 1 megbízhatóságával, ha kombináljuk a kettőt
- Szerencsés esetben a tömb akár egyszerre több diszk kiesését is elviseli
- A RAID 1+0 némileg megbízhatóbb
- A RAID 1-hez hasonlóan drága megoldás



RAID 0+1 RAID 1+0

RAID összefoglalás (1)

- A gyakorlatban csak a 0., 1., 5. és 6. szinteket alkalmazzák, illetve az 1+0, 0+1 kombinált megoldásokat
- A komponens diszkek méretének egyformának kell lennie
- Új diszkek menet közbeni hozzáadásával nem lehet növelni a tárkapacitást, újra létre kell hozni a tömböt
- A 6. illetve szerencsés esetben az 1+0, 0+1 kivételével valamennyi szint csak egyetlen diszk kiesését viseli el

RAID összefoglalás (2)

- Választási szempontok:
  - Magas megbízhatóság: 1, 5, 6, 1+0, 0+1
  - Nagy teljesítmény: 0, 1, 1+0, 0+1
  - Alacsony költség: 0, 5, 6
  - Ezek közül bármelyik kettőt teljesíthetjük
- Minden rendszernél külön kell mérlegelni, melyik a legmegfelelőbb megoldás; gyakran több különböző RAID szintű és méretű tömböt definiálunk

RAID összefoglalás (3)

- Általában lehetőség van készenléti diszkek definiálására, melyeket a rendszer automatikusan üzembe állít, ha egy diszk kiesik
  - Az új diszk szinkronizációja időbe telik, ezalatt a tömb teljesítménye csökken
  - A szinkronizáció közbeni új diszkhiba végzetes (a RAID-6 ez ellen védelmet nyújt)

RAID összefoglalás (4)

- A redundáns tömbök nem nyújtanak védelmet minden hibalehetőség ellen
  - Emberi tévedések
  - Programhibák
  - Az egész rendszert érintő hibák
    - Váratlan leállások
    - Túlfeszültség
    - Természeti katasztrófák

RAID összefoglalás (5)

- A fejlett operációs rendszerek kötetkezelő rendszerekkel (volume manager) könnyítik meg a hibatűrő logikai diszkek létrehozását és üzemeltetését
  - Új indirekciós szint a logikai blokkok és a RAID tömbök fizikai blokkjai között
  - A kötetkezelő rendszertől a partíciókhoz hasonló, de rugalmasan átméretezhető, hibatűrő logikai tárterületek (kötetek) igényelhetők
  - A rábízott fizikai partíciók tárterületével a kötetkezelő automatikusan gazdálkodik

Összefoglalás

- A logikai lemezterületek az operációs rendszer magasabb szintjei számára a fizikai lemezekhez hasonlóan kezelhető objektumokként látszódnak
- Rögzített méretű blokkok lineáris tömbje
  - Túlságosan alacsonyszintű
  - Megoldás: új absztrakciós szintek
    - fájlok (adatállományok)
    - könyvtárak (mappák)
    - fájlrendszerek

Tartalomjegyzék

Fájlrendszerek (1)

- *Fájlrendszer*: adatállományok a rendelkezésre álló lemezterületen történő tárolásának és elrendezésének módszere
  - Absztrakt adatszerkezetek, műveletek a lemezterület blokkjainak kiosztására
  - A fájlokat könyvtárak (mappákba) rendezhetjük
  - Lemezterület nélkül is lehet fájlrendszer! (hálózati fájlrendszerek, „virtuális” fájlrendszerek stb.)

Fájlrendszerek (2)

- A megoldandó problémák hasonlítanak a memóriakezelés feladataira
  - Az ismert allokációs algoritmusok itt is használhatók (First/Next/Best/Worst Fit)
  - A rögzített méretű blokkok a lapokra emlékeztetnek
  - A diszk fizikai sajátosságai (sebesség, elérési idők) azonban alapvetően más megoldásokhoz vezetnek

Fájlok (1)

- A perzisztens információtárolás egysége
- Legegyszerűbb megközelítésben a fájlok olyan bitsorozatok, melyek tartalma a folyamat befejeződése vagy a rendszer újraindítása után sem vész el
- A fájlok logikai szerkezete lehet
  - Bájtsorozat (leggyakoribb)
  - Fix méretű rekordok sorozata
  - Változó méretű rekordok sorozata (pl. szövegsorok)
  - Egyéb (pl. szegmensekre osztott programfájlok stb.)
- Fizikai szerkezet: blokksorozat
  - Nem biztos, hogy egymás utáni blokkok (külső töredezés)
  - Az utolsó blokk általában töredékblokk

Fájlok (2)

- Ha a rekordméret nem egész osztója a blokkméretnek, fellép a *belső töredezettség*
  - Nem célszerű, ha a rekordok túlnyúlnak a blokkhatárokon
  - A fizikai blokkok végén kihasználatlan maradékterület marad
  - Ha a fájlba sok beszúrás történik, célszerű az első rekordokat „levegősen” elhelyezni, szándékosan hosszú maradékterületek hagyásával (később így nem kell mindig új blokkokat beszúrni)
  - Az utolsó töredékblokk is egyfajta belső töredezettséget okoz

Fájlok (3)

- A tulajdonképpeni információtartalom mellett a fájlt magát leíró ún. meta-adatokat, vagy attribútumokat is tárolunk:
  - A fájl neve, gépi azonosítója, típusa, létrehozója
  - A fájl elhelyezkedése a diszken (összetevő blokkok sorszámai)
  - A fájl mérete (nem biztos, hogy a blokkméret többszöröse)
  - A fájl tulajdonosa(i), elérési jogosultságok
  - Elérési, módosítási időbélyegzők
  - stb.
- A meta-adatokat a fájl tartalmától elkülönítve, önálló fájlleíró blokkokban tároljuk a lemezen
  - UNIX terminológiában ezek neve index node, vagy information node; röviden i-node

Fájlok (4)

- Egyes rendszereken a fájlok több, egymástól független bitsorozatra bonthatóak
  - Apple Macintosh: erőforrás- és adatszegmens (resource fork, data fork)
    - Az erőforrás szegmensben a felhasználó által módosítható segédinformációk vannak
  - Microsoft NTFS: tetszőleges számú, névvel ellátott attribútum, független bájtsorozat
    - Jelölés: fájlnév:attribútumnév
    - Alapértelmezett bájtsorozat: unnamed
    - Felhasználás: pl. képfájlok előnézetének tárolására

Fájlok (5)

- A fájlokat elérésük módja szerint is csoportosíthatjuk
- Szekvenciális elérés: a fájlt elejétől végéig
- Közvetlen elérés: bármely rekordot, tetszőleges sorrendben beolvashatunk vagy kiírhatunk
  - Relatív elérés (a rekord fájlon belüli sorszáma alapján)
  - Indexelt elérés (a rekord tartalma (kulcsérték) alapján)
    - Egyszintű/többszintű rendezett indexek (pl. IBM ISAM)
    - B-fák, hasítótáblák stb.

Fájlok (6)

- A fájl típusa közli, hogy a fájl tartalma hogyan értelmezhető
  - Explicit meta-attribútumokban (pl. Macintosh)
  - A fájlnévbe építve, kiterjesztés formájában (pl. Windows)
  - A fájl tartalmában elhelyezett speciális kódokkal (pl. UNIX)
- Fájlműveletek:
  - Új, üres fájl létrehozása
  - Egy létező fájl törlése
  - Írás (átírás, hozzáírás, beszúrás)
  - Olvasás (szekvenciális vagy közvetlen)

Fájlok (7)

- Fájlműveletek (folytatás):
  - Újrapozícionálás (szekvenciális írás/olvasás közben)
  - Adott pont utáni rekordok törlése (nyesés, truncate)
  - Meta-adatok lekérdezése, módosítása; pl. átnevezés
  - Egyebek; pl. zárolás, memóriába képezés stb.
- A fájlrendszer feladata a fenti műveletek hatékony implementációja

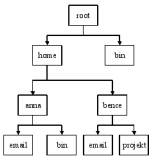
Könyvtárak (1)

- A fájlrendszeren tárolt fájlokról jegyzéket vezetünk: ezeket hívjuk könyvtárnak
- Legegyszerűbb megoldás: egyetlen könyvtár, mely az összes fájlt felsorolja
  - Korai rendszerek, legegyszerűbb fájlrendszerek
  - Nehezen átlátható
  - A névütközések problémát okoznak, különösen több felhasználó esetén

Könyvtárak (2)

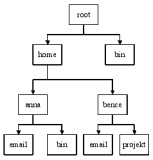
- Javítás: minden felhasználónak hozzunk létre egy-egy saját könyvtárat
  - Kétszintű rendszer: felhasználók jegyzéke + fájlok jegyzéke
  - A folyamatok a felhasználó könyvtárában keresik a fájlokat
    - Izolálja a felhasználókat, fájlmegosztást kizárja
    - A rendszerfájlokat mindenkinek el kell tudnia érni
  - Be kell vezetni a más könyvtárakra hivatkozás lehetőségét

Könyvtárak (3)



- Kézenfekvő általánosítás a szabadon egymásba ágyazott könyvtárak megengedése:
  - Tetszőleges könyvtárnak lehetnek alkönyvtárai
  - A gyökérkönyvtár kivételével minden könyvtárnak pontosan egy szülőkönyvtára van
  - Faszerkezetű, hierarchikus elrendezés
  - A rendszer a folyamatokhoz aktuális könyvtárat rendel, amely könyvtárban lévő fájlokkal a folyamat éppen dolgozik

Könyvtárak (4)



- Szabadon egymásba ágyazott könyvtárak (folytatás):
  - A faszerkezetben szerteszét szórt programfájlok keresésére keresési listákat szoktak alkalmazni ($PATH)
  - Egyes rendszerek általánosabb gráfszerkezetet is megengednek
    - Körmentes gráfok (könyvtárak megosztása)
    - Általános gráfok
    - A könyvtárak törlése bonyolultabb ellenőrzéseket igényel (hivatkozásszámlálás, ill. általános szemétgyűjtés)

Könyvtárak (5)

- Egy fájl elérési útja a gyökérkönyvtárt a fájlt tartalmazó könyvtárral összekötő út a könyvárszerkezetben
  - Az érintett könyvtárak felsorolásával lehet megadni
  - Az elérési utat általában a fájlnév részeként kezeljük
  - Ha nem adunk meg elérési utat, a rendszer a fájlt aktuális könyvtárban keresi
  - Az aktuális könyvtárhoz relatív elérési utakat is megadhatunk

Könyvtárak (6)

- Elérési út (folytatás):
  - Egyes rendszereken a fájlrendszer nevét is külön meg kell adni
  - Ahány rendszer, annyiféle hivatkozási szintaxis, konvenció:

```
/home/lorentey/work/os/fajlrendszerek.sxi
C:\Documents And Settings\Lőrentey
 Károly\Work\OS\Fájlrendszerek.sxi
USER$DISK:[PROGMAT.LORENTEY.WORK.OS]FAJLRENDSZ
 EREK.SXI;1
HD:Lorentey:Work:OS:Fájlrendszerek
```


Könyvtárak (7)

- A könyvtáraknak is vannak meta-adatai:
  - A könyvtár neve, hossza
  - Tulajdonos, elérési jogosultságok beállításai
  - Időbélyegzők
- Könyvtárműveletek:
  - Új, üres könyvtár létrehozása
  - Létező könyvtár törlése
  - Listázás, bejárás, keresés
  - Meta-adatok lekérdezése, módosítása (pl. átnevezés)

Könyvtárak (8)

- A könyvtárakat felfoghatjuk speciális célú, rekordszerkezetű fájlokként
  - Rekordok egy-egy fájlbejegyzést tartalmaznak
  - Egyes rendszereken (pl. UNIX) a könyvtárak ténylegesen fájlokként viselkednek
- A fájlbejegyzések tartalmazhatják a fájl összes meta-adatát, vagy csupán a fájl nevét és az inode-ja azonosítóját
  - A könyvtár szülőkönyvtárára speciális bejegyzés hivatkozik
  - Független inode-ok esetén ugyanaz a fájl akár több néven, különböző könyvtárakban is elérhető lehet (hard link)

Könyvtárak (9)

- Néhány rendszeren a hardver eszközöket speciális fájlbejegyzések formájában teszik elérhetővé (UNIX)
  - Az eszközvédelmet a fájlvédelemre bízzák
  - Az eszközök maguk is többé-kevésbé fájlként viselkednek

Fájlhivatkozások (1)

- Fájlhivatkozások (szimbolikus linkek): egy másik bejegyzésre hivatkozó speciális könyvtárbejegyzések
- A hivatkozás létrehozásakor meg kell adni a hivatkozott fájl nevét
- A fájlhivatkozás írása/olvasásakor az operációs rendszer automatikusan a hivatkozott fájlon végzi el a műveletet
  - A hard linkekhez hasonlóan, az alkalmazások számára transzparensen működnek
  - Akár hivatkozásláncokat is létrehozhatunk

Fájlhivatkozások (2)

- Könyvtárakra is hivatkozhatunk
  - Gráfszerkezetű könyvtárrendszert szimulálhatunk szigorúan faszerkezetű fájlrendszeren
  - Akár köröket is létrehozhatunk
- A hivatkozás törlésekor a hivatkozott objektum nem kerül törlésre, és viszont
  - A szemétgyűjtési problémákat így elkerüljük
  - Előfordulhat, hogy a hivatkozás célja nem létezik (dangling symlinks)

Egyéb fájlrendszerek (1)

- RAM diszkek: a memóriaterület egy részét lemezegységként kezeljük
  - Viszonylag kicsi méret, nagy sebesség
  - Nem maradandó tároló, kikapcsolás után elvész
- Flash memória
  - Félvezető alapú maradandó adattárolás
  - Mozgó alkatrész nélküli megoldás
  - Rövid (konstans) elérési idő, jó sávszélesség
  - Egyelőre véges élettartam
  - Gyorsan fejlődő, egyre népszerűbb technológia
- Optikai (CD, DVD), magnetooptikai (WORM)

Egyéb fájlrendszerek (2)

- Hálózati fájlrendszerek, osztott fájlrendszerek
  - NFS, AFS, SMB/CIFS, Coda, Intermezzo, stb.
  - A fizikai diszkeket is csatlakoztathatjuk hálózaton keresztül (Storage Area Networks, SAN)
  - Nem tárgyaljuk

Egyéb fájlrendszerek (3)

- Speciális célú pszeudo-fájlrendszerek
  - Az operációs rendszer belső állapotának kényelmes lekérdezésére, módosítására szolgálnak
    - Üzemi paraméterek
    - Futó folyamatok tulajdonságai, paraméterei
    - A rendszer hardverkörnyezetének leírása, stb.
  - A fájlok tartalmát a rendszer automatikusan generálja
  - A fájlok módosításakor közvetlenül a paramétereket állítjuk
  - Nincs szükség speciális rendszerprogramokra, a megszokott fájlműveletek, eszközök használhatóak

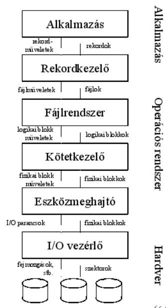
Egyéb fájlrendszerek (4)

- Pszeudo-fájlrendszerek (folytatás)
  - Példa: UNIX ill. Linux /proc fájlrendszere
  - Plan 9 operációs rendszer: a hálózati és folyamatok közti kommunikációtól kezdve a grafikus ablakozó rendszeren át minden virtuális fájlokon keresztül történik
- Egyéb, alkalmazásszintű pszeudo-fájlrendszerek
  - Archívumok (.zip, .tar.gz stb.) egyszerű olvasására/írására
  - Verziókezelésre (pl. ClearCase)
  - Egyéb alkalmazásszintű feladatokra

Tartalomjegyzék

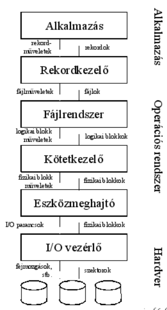
Áttekintés (1)

- A fájlkezelés alrendszerét rétegre oszthatjuk
  - Az egyes rétegek csak a közvetlenül alattuk és fölöttük lévő rétegekkel kommunikálnak
    - Az alsó réteg szolgáltatásait használják
    - A felső réteg számára szolgáltatásokat nyújtanak



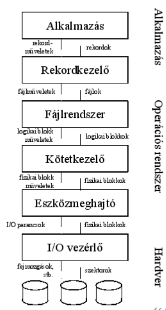
Áttekintés (2)

- A gyakorlatban az egyes szintek nincsenek mindig szigorúan szétválasztva
  - Hatékonysági megfontolások
  - Egyes műveletek elvégzéséhez keresztül kell nyúlnunk a szintfelosztáson
- I/O vezérlő feladata a lemezegység fizikai vezérlése
  - A hardverbe épített
  - Nem foglalkozunk vele



Áttekintés (3)

- Eszközmeghajtó a fizikai blokkokra vonatkozó kéréseket lefordítja az adatátviteli interfész és a lemezegység típusának megfelelő alacsonyszintű parancsokra
- Kötetkezelő valósítja meg a tárterület-szervezést
  - Partíciók, RAID tömbök, kötetek
  - Logikai blokkokról fizikai blokkokra képez



Áttekintés (4)

- A fájlrendszer feladata a fájl- és könyvtárműveletek megvalósítása
  - Az adott fájlhoz használt fájlrendszert a VFS választja ki
  - Blokkok foglalása, felszabadítása, könyvtárrendszer, meta-adatok, stb.
  - A továbbiakban ezzel fogunk foglalkozni

Áttekintés (5)

- A rekordkezelő a fájlok belső szerkezetét valósítja meg
  - Rekordokról fájlon belüli bájt- vagy blokkpozícióra képez
  - Az OS-ből gyakran hiányzik, ilyenkor az alkalmazásokon belül kell megvalósítani, ha szükséges

Több fájlrendszer-típus támogatása (1)

- Általában az OS nem csak egyféle fájlrendszer-típust támogat
  - Más fájlrendszer való a winchesterre, más a CD, DVD lemezekre
  - A fejlett rendszereken gazdag fájlrendszer-választék áll rendelkezésre, melyből kiválaszthatjuk a saját céljainkra legmegfelelőbbet
    - Üzembiztonság
    - Az egyes fájlműveletek gyors végrehajtása
    - Helytakarékosság
    - Kompatibilitás más operációs rendszerekkel
  - Szélsőséges példa: Linux 2.6: több mint 50(!) fájlrendszer-típus

Több fájlrendszer-típus támogatása (2)

- Virtuális fájlrendszer-kapcsoló (VFS): az alkalmazások I/O műveleteiről eldönti, hogy melyik konkrét implementációnak kell továbbadni
  - Objektum-orientált szemlélet, absztrakt fájlrendszer
  - Az ismétlődő részeket a VFS-ben összefoghatjuk, nem kell újra és újra megírni

Fájlrendszerek csatolása (1)

- A fájlrendszert a használatba vétel előtt csatolni (*mount*) kell a rendszerünkhöz
  - Ez a superblock beolvasását és a fájlrendszer-táblázatban egy új bejegyzés létrehozását jelenti
  - A csatolás eredményeképp a fájlrendszer fájlai elérhetővé válnak
- A csatolás történhet
  - Automatikusan, amint a fájlrendszer rendelkezésre áll (pl. hordozható háttértárak, Windows), vagy
  - Automatikusan, amint hivatkozás történik rá (pl. hálózati fájlrendszerek), vagy
  - Rendszergazdai közreműködés hatására (UNIX)

Fájlrendszerek csatolása (2)

- Egyes rendszerek minden fájlrendszert külön névtérként kezelnek (VMS, Windows)
- Más rendszerek egyetlen közös könyvtárrendszert alkotnak, és a csatolás után a fájlrendszer tartalma a (tetszőlegesen) megadott könyvtárból lesz elérhető

Fájlrendszerek megvalósítása (1)

- A lemezen tárolt adatszerkezetek:
  - Boot blokk
  - Fájlrendszer-leíró blokk vagy blokkok (superblock)
  - Könyvtárszerkezet a fájlok szervezéséhez
  - Fájlleíró blokkok (inode) a fájlok meta-adatainak tárolására
  - Adatblokkok a fájlok tartalmának tárolására
- A fájlrendszer-leíró blokk a fájlrendszerről magáról tartalmaz adatokat:
  - Fájlok száma
  - Blokkok száma és mérete, szabad blokkok száma

Fájlrendszerek megvalósítása (2)

- Fájlrendszer-leíró blokk (folytatás):
  - Fájlrendszer állapota (tiszta, gyanús, hibás)
  - Időbélyegzők:
    - Utolsó csatolás (mount)
    - Utolsó ellenőrzés
    - Utolsó módosítás
  - A fájlrendszer címkéje (neve)
- A superblock nélkül a fájlrendszer nem használható, ezért több példányban is el szokás tárolni
- ----------

Fájlrendszerek megvalósítása (3)

- Memóriában tárolt adatszerkezetek:
  - Csatolt fájlrendszerek táblázata
  - Nemrégiben használt könyvtárak tartalma
  - Rendszerben megnyitott összes fájl leírása
    - Inode másolata, és egyéb információk
  - Folyamatonkénti táblázat a folyamat megnyitott fájlairól
    - Megnyitott fájl azonosítója (fájlrendszer azonosító és inode szám)
    - Megnyitás módja (írás, olvasás, vegyes) és egyéb beállítások
    - Fájlon belüli aktuális pozíció (szekvenciális olvasáshoz, íráshoz)
    - Ugyanazt a fájlt egyszerre egynél több folyamat is megnyithatja
  - Blokk-gyorsítótár

Könyvtárak megvalósítása (1)

- A könyvtárakat általában speciális tartalmú adatállományokkal reprezentáljuk
  - Tárolásuk a hétköznapi fájlokhoz hasonlóan történik
- A könyvtárak fájlneveket képeznek a tulajdonképpeni fájlokra, így a bejegyzések tekinthetők egyszerű (fájlnév, inode azonosító) pároknak
  - Gyakran a könnyebb elérés érdekében az inode-ból néhány meta-adatot (pl. fájltípus) a könyvtárbejegyzésbe is bemásolnak
  - Az egyszerűbb fájlrendszerekben (pl. FAT) nincs is külön inode struktúra, a könyvtár tartalmazza a fájlok minden meta-adatát

Könyvtárak megvalósítása (2)

- A könyvtárak szemantikailag rekordokra osztott fájlokra emlékeztetnek
  - A könyvtár bejegyzései felelnek meg a rekordoknak
  - A bejegyzések lehetnek rögzített, vagy változó hosszúságúak (a fájlnév lehetséges hosszától függően)
  - Általában a fájlműveletek nem használhatók könyvtárakon (kivétel: klasszikus UNIX, UFS)

Könyvtárak megvalósítása (3)

- A bejegyzéseket legegyszerűbb egymás után, egy láncolt listában elhelyezni
  - Egyszerű, könnyen megvalósítható módszer
  - Változó bejegyzésméret nem zavarja
  - Sok fájl esetén a keresés ideje túlságosan megnőhet (lineáris számú diszkművelet!)
  - A lista rendezésével a keresési időt csökkenthetjük (logker), de cserébe a beszúrás/törlés ideje nő meg

Könyvtárak megvalósítása (4)

- Sok fájl egy könyvtárban való tárolását könnyíti meg a hasítótáblázatot használó fájlrendszer
  - Általában konstans műveletigény
  - A táblázat telítődésével a hatékonyság jelentősen romlik
  - Ha az előre kialakított táblázat betelt, a tábla növelése jelentős számú műveletet igényel
    - Minden bejegyzést át kell pakolni
    - Ezt elkerülhetjük, ha láncoljuk a hasítótáblákat
- Természetesen a faszerkezetű kereső adatszerkezetek (b-fák, kiegyensúlyozott keresőfák stb.) is használatosak

Blokkfoglalás

- A fájlok tartalmát adatblokkokban tároljuk
  - Nyilván kell tartanunk, hogy mely blokkok tartoznak egy fájlhoz (és milyen sorrendben)
  - Tudnunk kell új blokkokat adni egy már létező fájlhoz
- A blokkok nyilvántartását is a meta-adatok között tároljuk
  - Magában az inode-ban, vagy erre a célra elkülönített diszkterületen

Folytonos blokkfoglalás (1)

- Legyenek a fájlok blokkjai együtt, egy folytonos diszkterületen
  - Legkézenfekvőbb megoldás
  - A memóriakezelésnél (dinamikus partíciók) tanult algoritmusok itt is használhatók
  - Egy fájl blokkjainak nyilvántartásához elég a kezdőblokkot és a fájl hosszát tárolni – nem pazarolja a tárterületet
  - Nagyon hatékony megoldás, hiszen kizárja a fájl töredezettségéből eredő fejmozgásokat

Folytonos blokkfoglalás (2)

- Hátrányok:
  - A szabad terület elaprózódása miatt felmerül a külső töredezettség problémája
  - Mivel a végső fájlméretet előre meg kell adni, a lassan növő fájlok jelentős belső töredezettséget eredményeznek
  - A fájlok mérete utólag nagyon nehézkesen növelhető
- Csak olvasható háttértárakon (CD, DVD, stb.), illetve igen nagy IO teljesítményigény esetén (nagyszámítógépek) éri meg használni
- A lemezterület partícionálása egyfajta folytonos foglalás-típusnak tekinthető

Extens alapú blokkfoglalás (1)

- Javítsunk a megoldásunkon
  - Engedjük meg, hogy a fájlméret növekedése esetén a szükséges új tárterületet a diszk egy másik összefüggő területe (*extense*) adja
  - A nyilvántartás (kezdőblokk, hossz) párok sorozata
  - Az extensek méretét célszerű egységesnek választani
  - Ekkor sem kell egyedi blokkokat nyilvántartani, így ez is tárhatékony megoldás
  - Az extensméretet a fájl létrehozásakor meg kell adni

Extens alapú blokkfoglalás (2)

- Általános célra is jól használható módszer
  - Pl: Veritas File System
  - Ha az extensméret túl nagy, jelentős lehet a belső töredezettség
  - Ha többféle extensméretet használunk, fellép a külső töredezettség is

Láncolt listás blokkfoglalás (1)

- Ötlet: minden adatblokk végére tegyük be a következő adatblokk számát
  - Az inode csupán az első adatblokk számát tartalmazza
  - Láncolt lista adatszerkezet
  - Csak szekvenciális IO műveletek esetén elfogadható
  - A blokkmutatók tárolása miatt a használható tárkapacitás százalékokban kifejezhető mértékben csökken
  - Ha a blokkmutatók (bármilyen okból) elromolnak, súlyos adatvesztés következhet be

Láncolt listás blokkfoglalás (2)

- A gyakorlatban a fenti „tiszta” láncolt listás megoldást ritkán használják
  - Több blokkot fürtbe (*cluster*) fogva javítható a viselkedés, de cserébe erősödik a belső töredezettség
  - Jellemző a blokkmutatók kigyűjtése egy külön táblázatba (pl. FAT)

Indexelt blokkfoglalás (1)

- Ötlet: a fájl adatblokkjainak sorszámát soroljuk fel egy külön e célra fenntartott indexblokkban
  - Az inode-ban erre az indexblokkra hivatkozunk
  - Minden fájlhoz külön indexblokkot rendelünk
  - A szekvenciális és közvetlen elérést egyaránt jól támogatja
  - Rövid fájlok esetén az utolsó adatblokk mellett az indexblokk is csak részlegesen lesz kihasználva
  - Az indexblokkokat általában a memóriában megőrizzük

Indexelt blokkfoglalás (2)

- Mi történik, ha betelik az indexblokk?
  - Új blokkot láncolhatunk hozzá (láncolt listás index)
  - Második szintű indexblokkot definiálunk
    - (Közvetlen) indexblokkok számait sorolja fel
    - Tetszőleges szintig kiterjeszthető
  - A két megoldást kombináljuk
    - Az inode-ban felsorolunk néhány adatblokkot, néhány közvetlen indexblokkot, és egy-egy 2, ill. 3 szintű indexet
    - Hatékony, gazdaságos megoldás
    - Nagyon elterjedt (klasszikus UNIX UFS, Linux ext2, stb.)

Szabad terület nyilvántartása (1)

- Ahhoz, hogy új blokkokat csatolhassunk egy fájlhoz, tudnunk kell, hogy mely blokkok nem tartoznak egyetlen fájlhoz sem
- Bittérképes megoldás
  - A tárterület egy elkülönített részén tárolt bittérképen jelöljük, hogy melyek a szabad blokkok
    - Minden blokknak egy bit felel meg
    - Ha a bit értéke 1, akkor a blokk szabad

Szabad terület nyilvántartása (2)

- Bittérképes megoldás (folytatás)
  - A hatékonyság érdekében célszerű a térképet a memóriában is megőrizni
  - Népszerű, viszonylag olcsó megoldás
  - Ügyes algoritmussal elkerülhető a fájltöredezés
- Láncolt listás megoldás
  - Az adatblokkoknál látott módszerrel fűzzük listába a szabad blokkokat
  - Általában csak egy új blokkra van szükség

Szabad terület nyilvántartása (3)

- Láncolt listás megoldás (folytatás)
  - Pl. FAT fájlrendszer (az allokációs algoritmusokat használja)
  - Nem teszi lehetővé a fájltöredezettség elkerülését
- Szabad blokkok kigyűjtése
  - Az első szabad blokkban felsoroljuk a többi szabad blokk sorszámát
  - Ha betelik, az indexet megtoldjuk egy következő indexblokk hozzáláncolásával
  - Nem igényel extra tárterületet, a nyilvántartás „helyben” történik

Szabad terület nyilvántartása (4)

- Szabad blokkok kigyűjtése (folytatás)
  - Gyorsan lehet egyszerre sok szabad blokkot foglalni
  - A fájltöredezettség kivédése itt is nehézkes
- Számláló, extens alapú nyilvántartás
  - Az adatblokkok közötti „lyukakat” tartsuk nyilván (kezdőblokk,blokkszám) párokkal
  - Azonos számú szabad blokk nyilvántartásának tárigénye jelentősen csökkenhet
  - Folytonos foglalás esetén jól jön az összefüggő szabad tárterületek méretének közvetlen eltárolása

Naplózott fájlrendszerek (1)

- A fájlrendszer módosításai gyakran egynél több I/O műveletet igényelnek
  - Meta-adatokat (blokknyilvántartás stb.) is módosítani kell
  - Váratlan hiba (pl. áramszünet) miatt a rendszer két művelet között is leállhat
  - A fájlrendszer inkonzisztens állapotba kerülhet
- Példa: könyvtármozgatás
  - (Legalább) két művelet: törlés a szülőkönyvtárból, beszúrás a célkönyvtárba
  - Ha az egyik művelet elvész, a fájlrendszer sérül
    - Megsérül a szigorú faszerkezet, vagy
    - Könyvtárszerkezeten kívüli, elveszett könyvtár jön létre

Naplózott fájlrendszerek (2)

- Váratlan leállások utáni rendszerindításkor konzisztencia-ellenőrzést kell végezni
  - Az egész fájlrendszert meg kell vizsgálni
  - Hosszú ideig, akár órákig eltarthat
  - Gyakran a helyreállítás nem lehetséges, vagy emberi közbeavatkozást igényel
- Az egyetlen fájlrendszer-módosításhoz tartozó műveletek összességét tranzakciónak nevezzük

Naplózott fájlrendszerek (3)

- Ötlet: az adatbáziskezelő rendszerek módosításnaplójának (redo log) átvétele
  - A lemezterület egy arra kijelölt részét módosításnaplónak használjuk
    - (Meta-)adatmódosítások ciklikus listája
    - A tranzakciók műveleteinek leírását tartalmazza
  - A módosítást elvégzettnek tekintjük (committed), ha a naplóba bekerült a tranzakció összes művelete
  - Egy speciális rendszerfolyamat a háttérben folyamatosan átvezeti az elvégzett tranzakciók műveleteit a naplóból a tulajdonképpeni fájlrendszerre

Naplózott fájlrendszerek (4)

- Váratlan rendszerleállás után csak át kell vezetni a napló elvégzett tranzakcióit a fájlrendszerre
  - A napló végén lévő esetleges befejezetlen tranzakció nem okoz inkonzisztenciát
  - Nincs szükség költséges teljes ellenőrzésre
- Hatékonyabb, ha a naplót külön diszken vezetjük
  - A fejmozgatásból eredő időveszteség csökken
  - A napló diszkjének meghibásodásakor a fájlrendszert ellenőrizni kell

Naplózott fájlrendszerek (5)

- Az üzembiztonság elvész, ha a lemezegység firmware-je szabadon átrendezheti az I/O műveleteket
  - A napló bejegyzéseit szigorúan egymás után, azonnal kell lemezre írni (synchronous I/O)
  - A napló szekvenciális jellege ellensúlyozza a I/O ütemezés és a writeback cache elvesztése miatti hátrányokat

Naplózott fájlrendszerek (6)

- Az alkalmazások többségében a naplózás némileg rontja a teljesítményt
  - A fájlrendszer-módosításokon felül a naplót is vezetni kell
  - Az üzembiztonság általában fontosabb szempont
  - A hatékonyságnövekedésre is akad példa:
    - A módosítások nem kerülnek átvezetésre a fájlrendszerre, ha közben jön egy újabb változás (pl. ideiglenes fájl létrehozása, majd szinte azonnali törlése)
    - Hirtelen feltorlódó, de általában kevés módosításigényt a naplózás jobban elviselhet, hiszen a módosítások átvezetésével „szétkeni" a terhelést

FAT fájlrendszer (1)

- Az adatblokkok nyilvántartását láncolt listás megoldás szerint, egy elkülönített lemezterületen lévő táblázatban végzi
  - Fájl-Allokációs Táblázat, File Allocation Table, FAT
  - Blokksorszámokról blokksorszámokra képező tömb
  - Az i. sorszámú blokkhoz tartozó táblaérték megadja, hogy melyik blokk következik az i. blokk után a fájlban
  - A FAT méretének csökkentése érdekében a blokkokat fürtökbe tömöríti, és ezeket kezeli egységnek
    - Fürtméret: 512 bájttól akár 256 kilobájtig
    - Jelentős (akár 50%) lehet a belső töredezettségből eredő kapacitáscsökkenés

FAT fájlrendszer (2)

- Egyszerű szekvenciális könyvtárak
  - A gyökérkönyvtárra a superblock hivatkozik
  - A könyvtári bejegyzések egységesen 32 bájtosak
    - Nincs külön inode struktúra
    - Fájlnév (8+3), védelmi és egyéb attribútumok
    - Utolsó módosítás ideje
    - Az első fürt sorszáma, a fájl mérete
  - Az újabb változatok hosszú fájlneveket is megengednek
- A FAT hordozható eszközökön ma is igen elterjedt
  - Egyszerű, könnyen megvalósítható
  - A flash alapú tárolóegységek gyakori fájlrendszere

Linux ext2fs

- Egyszerű általános célú fájlrendszer a GNU/Linux operációs rendszerhez
- 1993 eleje, Rémy Card, Wayne Davidson és mások
- A BSD FFS-hez (Fast File System) hasonló klasszikus, blokkszerkezetű UNIX fájlrendszer
- Népszerű megoldás, gyakran ez az alapértelmezett fájlrendszer
- Létezik naplózó kiterjesztése (ext3), mely a meta-adatok mellett a fájlok tartalmának védelmére is használható
- Van kvóta- és ACL támogatása

Ext2fs blokkcsoportok (1)

- A lemezterületet az ext2 fájlrendszer rögzített méretű blokkcsoportokra (block group) osztja
- A blokkcsoportok biztosítják az adatok lokalitását
  - A fájlok adatblokkjai egymáshoz közel helyezkednek el
  - A metaadatokat az adatblokkok közelébe kerülnek

Ext2fs blokkcsoportok (2)

- A blokkcsoportok tartalma:
  - A fájlrendszer-leíró superblock másolata (1024 bájt)
  - Blokkcsoport-leíró (4 bájt)
    - A további részek blokkszámai
    - Csoporton belüli szabad inode-ok és blokkok száma
    - A csoportban tárolt könyvtárak száma
  - Foglalt blokkok bittérképe (1 blokk) – n. bit 1, ha az n. blokk foglalt
  - Foglalt inode-ok bittérképe (1 blokk)
  - Inode lista (általában 128 bájt inode-onként)
  - Adatblokkok (fájlok, könyvtárak tartalma és indirekt blokkok)

Ext2fs blokkcsoportok (3)

- A blokkcsoportok méretét az egy blokk hosszúságú bittérképek korlátozzák
  - 1024 bájtos blokkméret esetén maximum 8192 blokk lehet egy blokkcsoportban
  - Nagyméretű fájlrendszereken nagy blokkméretet kell választani, különben szükségtelenül sok blokkcsoport jön létre
- Az inode lista egyszerűen azonos méretű inode-ok felsorolása
  - Az inode bittérkép ezen belül adja meg a szabad inode-ok pozícióját

Ext2fs inode-ok

- A fájlleíró inode-ok tartalma:
  - Típus (fájl, könyvtár, fájlhivatkozás stb.)
  - Fájlméret
  - Időbélyegzők: elérés, módosítás, létrehozás, törlés
  - A fájl tulajdonosa, védelmi beállításai
  - 12 közvetlen blokkmutató
  - 1 egyszeresen indirekt blokkmutató
  - 1 kétszeresen indirekt blokkmutató
  - 1 háromszorosan indirekt blokkmutató
- A fájlok nevét nem az inode tárolja!
- Speciális inode számok:
  - 1: hardverhibás blokkokból álló speciális fájl
  - 2: a gyökérkönyvtárhoz tartozó inode

Ext2fs blokkmutatók (1)

- A blokkmutatók adják meg, hogy a fájl adatait mely blokkok tartalmazzák
  - Ha a fájl rövidebb 12 blokknál, akkor az inode-ban található közvetlen blokkmutatók elégségesek a felsoroláshoz
  - Nagyobb fájlok esetében egyszeresen, kétszeresen, végül háromszorosan indirekt blokkmutatók állnak rendelkezésre
    - Olyan blokkokra mutatnak, melyek kizárólag adatblokkok (vagy további indirekt blokkok) sorszámaiból állnak
    - A rövidebb fájlok kevesebb diszkműveletet igényelnek
  - Nagyon rövid fájlokat (szimbolikus linkek) magában az inode-ban, a blokkmutatók helyére is el lehet tárolni

Ext2fs blokkmutatók (2)

- A fájlok végén az utolsó blokk általában csak részben van feltöltve (belső töredezettség)
  - Sok apró, egy blokknál rövidebb fájl esetén jelentős lehet a tárterület-veszteség
  - Minél nagyobb a blokkméret, annál komolyabb probléma lehet

Ext2fs könyvtárak (1)

- A könyvtárakat az ext2fs speciális fájlként kezeli
- Tartalom: változó hosszúságú könyvtárbejegyzések egyszerű felsorolása
- A bejegyzések szerkezete:
  - Inode szám (4 bájt)
  - A bejegyzés szóhatárra felkerekített összhossza (2 bájt)
  - A fájl típusa (1 bájt, az inode-ból kimásolva)
  - A bejegyzés neve (fájlnév, 1-255 bájt)
- A lista nem rendezett!

Ext2fs könyvtárak (2)

- A könyvtár első bejegyzése („.”) önmagára hivatkozik, a második bejegyzés („..”) pedig a szülőkönyvtárra
- Egy bejegyzés törlésekor az inode számot nullára kell állítani, és a törölt bejegyzést össze kell vonni a megelőző bejegyzéssel
- Új bejegyzés beszúrása a legelső elégséges szabad „rés” megtalálásával történik
- Több ezer fájlból álló könyvtárak esetén a lista bejárása időigényes lehet

Ext2fs blokkfoglalás (1)

- Új blokk lefoglalásához a blokk bittérkép használható
- A töredezettség elkerülése érdekében az ext2fs speciális allokációs eljárásokat alkalmaz
  - Minden inode tartalmaz egy célblokkszámot *(target block),* mely a következő foglalás optimális helyét tartalmazza
  - Új blokk foglalásakor a szabad blokkok keresése az alábbi sorrendben történik:
    1. A célblokk 32 blokknyi környezetében
    2. Az adott blokkcsoporton belül
    3. A fájlrendszer többi blokkcsoportjában
  - Az eljárás a fájl blokkjait igyekszik minél közelebb tartani egymáshoz

Ext2fs blokkfoglalás (2)

- A lefoglalt blokkot közvetlenül követő blokkokat fenntartjuk az adott fájl további kiterjesztései számára
  - Alapértelmezésben 8 blokk
  - A fájl lezárásakor a nem használt blokkok szabaddá válnak

ReiserFS fájlrendszer

- Hans Reiser, Linux
- Naplózott
- A fájlrendszer egyetlen nagy, kiegyensúlyozott keresőfa (B*)
  - Az adatbáziskezelő rendszerekben megszokott adatszerkezet
- A fájl végén lévő kihasználatlan maradék blokkokat összevonja
  - Megszünteti a belső töredezettséget
  - Sok apró fájl esetén jelentős megtakarításokhoz vezet

XFS fájlrendszer

- Silicon Graphics (IRIX), 1993
- Naplózott
- Elől: EFS, Extent Filesystem
- Nagy múltú, megbízható fájlrendszer
- Nagy kapacitású háttértárak, nagy fájlok esetén ajánlott
- Extens alapú blokknyilvántartás
- Szabad blokkokat B+ fába illeszti
- Nagy IO kapacitásigényű realtime alkalmazásoknak speciális szolgáltatásokat nyújt
  - Többé-kevésbé garantált sávszélesség
  - 4 kilobájttól 1 gigabájtig terjedő blokkméret

PalmOS fájlrendszer (1)

- Kéziszámítógépek (PDA) operációs rendszere
- A fájlrendszert nem diszken, hanem a memóriában tárolja
  - Nincsenek blokkok, minden bájt közvetlenül címezhető
  - Nincs szükség adatátvitelre
  - Az adatok közvetlenül elérhetők, módosíthatók
  - Az alkalmazásokban nincs „Mentés” funkció, minden változtatás azonnal megtörténik

PalmOS fájlrendszer (2)

- Fontos szempont az adatszinkronizáció megkönnyítése a kéziszámítógép és az asztali számítógép között
  - A változtatások kétirányú átvitele
  - A legutóbbi szinkronizáció óta módosított adatokat az operációs rendszer nyilvántartja
- Tipikus adatok:
  - Naptárbejegyzések
  - Címtár
  - Jegyzetek

PalmOS fájlrendszer (3)

- Egyszintű könyvtárrendszer
  - Nincsenek alkönyvtárak, az összes fájl egyetlen listában szerepel
  - A fájlokat a Palm terminológiában adatbázisnak nevezik
  - A fájlok meta-adatai:
    - Fájlnév (max. 31 karakter), típus
    - A fájlt létrehozó program egyedi azonosítója
    - Verziószám
    - Módosítások száma
    - A rekordok azonosítóinak felsorolása

PalmOS fájlrendszer (3)

- A fájlok listázására használat közben általában nincs szükség
  - Az alkalmazások „behuzalozott” fájlneveket használnak
  - Az egyszerű könyvtárrendszer így nem teszi átláthatatlanná az adatokat
- Fájlszerkezet: változó hosszúságú rekordok
  - A rekordok létrehozása, törlése, átméretezése a dinamikus memóriakezelés eszközeit használva történik
  - A rekordok szétszórva helyezkednek el a memóriában, nem cél a töredezettség elkerülése

PalmOS fájlrendszer (4)

- Minden rekord rendelkezik egy szabványos formátumú leíróval:
  - Rekordazonosító (3 bájt)
  - Kategóriaszám (a rekordok csoportokba sorolásához) (4 bit)
  - Jelzőbitek (törölt, módosított, zárolt, ill. titkos rekordok jelzésére)
- Az adatrész maximum 64KB lehet, a tartalma természetesen az alkalmazástól függ
  - Ennél nagyobb fájlokat szekvenciálisan kezelhetünk
- A perifériaként kezelt flash memória-kártyákon természetesen a PalmOS is FAT fájlrendszert használ