Súborové systémy

Načo je dobrý súborový systém?

1. Perzistencia údajov (dáta dostupné aj po reštarte OS),
2. organizácia údajov (priečinky, pomenovanie súborov),
3. zdieľanie údajov (medzi aplikáciami a používateľmi).

Aké výzvy predkladá implementácia súborového systému programátorovi?

- Voľba vhodného pomeru medzi výkonom a konkurentnosťou,
- bezpečnosť údajov (integrita, dôvernosť, dostupnosť),
- obnova pri zlyhaní behu systému,
- zdieľanie,
- vyššia úroveň abstrakcie (čo všetko môže byť "súbor": rúry, sieť, zariadenia, "klasický" fs, /proc, ...).

Príklad aplikačného rozhrania pre systémy implementujúce Posix:

- fd = open("adr/sub", "r");
- write(fd, "asd", 3);
- close(fd);
- link("adr/sub", "/data/niekde/inde/subor");
- unlink("adr/sub");

Dôsledky Posix FS API:

1. fd odkazuje na "niečo" (objekt fs), čomu sa nemení obsah, aj keď sa zmení meno súboru alebo je súbor dokonca zmazaný!
2. na obsah súboru môže viesť veľa odkazov ("hard link"), z ktorých ani jeden nemá výnimočné postavenie voči ostatným

Z toho vyplýva:

- FS musí ukladať informácie o súbore niekde inde, nie priamo do adresára, cez ktorý je možné sa k obsahu súboru dostať; informácie o súbore sa ukladajú do štruktúry "inode" na disku;
- FS identifikuje i-uzly číslom, nie názvom!!!
- i-uzol musí obsahovať
    1. počet odkazov (link) na súbor
    2. počet aktuálnych otvorení súboru (fd)
- zrušenie i-uzlu je odložené, pokým sa nezatvoria všetky fd a kým neklesne počet odkazov na 0.

Kam sa ukladajú údaje?

- Na perzistentné médiá,
- najčastejšie používané:
    - HDD (veľmi pomalé, ale lacné a vysoko kapacitné)
    - SSD (veľmi rýchle, ale drahé a s nižšou kapacitou)

HDD (Hard Disk Drive)

- údajové stopy na sústredných kružniciach
- každá stopa (track) je tvorená postupnosťou sektorov (sector)
- veľkosť sektoru je zvyčajne 512B
- časy prístupov
    - náhodný prístup pomalý (cca10ms na jeden prístup)
    - sekvenčný prístup je rýchly (cca 100MB za sekundu)
- každý sektor obsahuje kontrolný súčet na zabezpečenie integrity údajov
- najmenšou údajovou jednotkou disku je sektor, t.j. nemožno čítať ani zapisovať menej než sektor
- preto je zápis menšej časti veľmi "drahý": read - modify - write pre celý sektor, nie iba jeho časť

SSD (Solid State Disk)

- náhodný prístup je rádovo cca 100 mikrosekúnd
- sekvenčný prístup 500 MB za sekundu
- pred prepísaním pamäťovej bunky je potrebné ju najprv vymazať
- fyzikálna hranica počtu prepisov bunky
- za týmto účelom má disk interne rezervný počet buniek, ktoré využíva na "premapovanie" poškodených

Spoločné vlastnosti HDD a SSD:

- sekvenčný prístup je rádovo rýchlejší než náhodný
- zápisy alebo čítania veľkých objemov sú rýchlejšie než po jednotlivých sektoroch
- pri návrhu FS v OS treba počítať s týmito vlastnosťami

Blok disku

- väčšina OS používa bloky (block), ktoré sú celočíselným násobkom sektorov (napr. 4 kB bloky = 8 sektorov)
- OS využíva ako najmenšiu údajovú jednotku blok (je to podobné ako pri stránkovaní: OS používa najmenšiu alokačnú jednotku stránku, hw využíva rámec; stránka je celočíselným násobkom rámca)
- xv6 používa bloky o veľkosti 2 sektory (t.j. 1kB)

Vrstvy FS xv6: **čítaj kapitolu File system z knižky xv6**

1. ovládač hw
2. buffer cache
3. log
4. inode cache
5. inodes
6. operácie nad FD alebo cestami
7. systémové volania

Rozvrhnutie nosiča (disku) v xv6: xv6 považuje disk za pole blokov (ignoruje fyzické vlastnosti nosiča)

- 0: nevyužitý FS, používa sa pri štarte systému (obsahuje zavádzač)
- 1: super blok (obsahuje informácie o veľkosti FS, počte i-uzlov)
- 2: začiatok oblasti pre logovanie transakcií vykonávaných nad FS
- 32: začiatok poľa i-uzlov
- 45: bitmapa využitia blokov (0-voľný, 1-obsadený)
- 46: začiatok blokov obsahujúcich obsah súborov a adresárov
- koniec disku

Čo sú metadáta:

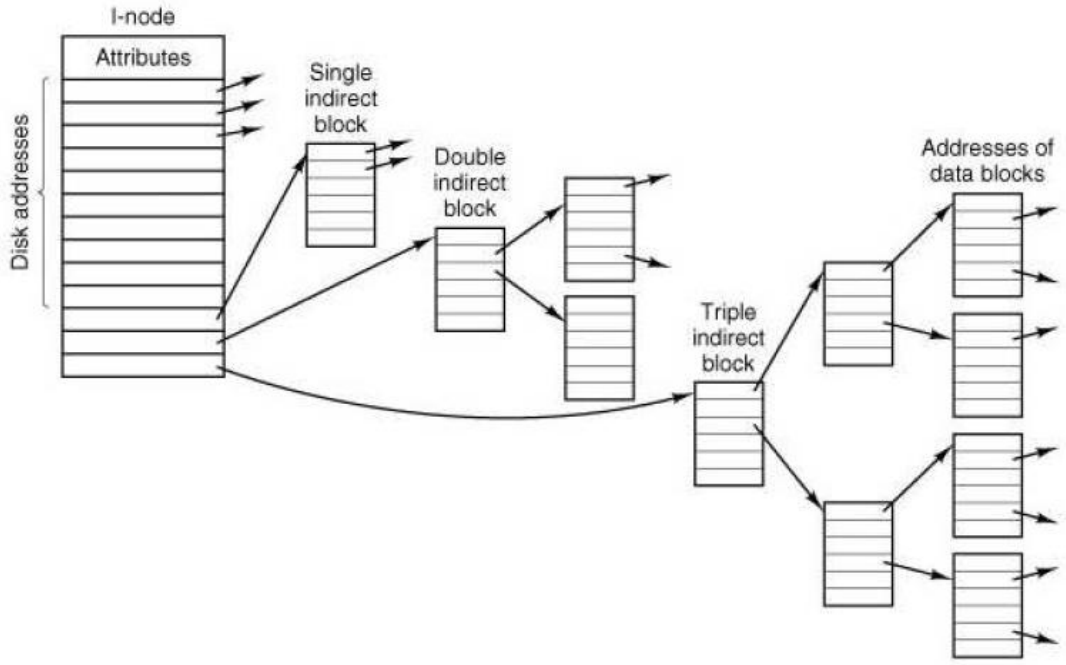
- všetko na disku, čo nie je samotný obsah súborov
- t.j. super blok, i-uzly, bitmapa, obsah adresárov, logy

Obsah i-uzlov na disku:

1. type: typ súboru (voľný, súbor, adresár, zariadenie)
2. nlink: počet odkazov
3. size: veľkosť
4. addrs: 12 priamych a 1 nepriamy blok údajov súboru

Priame a nepriame bloky:

- Čo sú priame a čo nepriame bloky?
- Čo je dvojito nepriamy blok?
- Aká je minimálna a aká je maximálna veľkosť súboru v xv6?
- Bloky údajov súboru nazývame "logické bloky", číslované od 0. V ktorom logickom bloku súboru sa bude nachádzať osem tisíci bajt súboru? (8000 / BSIZE = 8000 / 1024 = 7, takže adresu údajového bloku budeme hľadať v addrs[7])



Obr. 1: Evidencia údajových blokov súboru. Zdroj: Viliam Solčány: Prednáška 12, Operačné systémy 2010/11, FIIT STU Bratislava

Obsah adresára

- adresár je tiež iba súbor, avšak užívateľ nemôže do neho priamo zapisovať
- obsahom tohto súboru je pole štruktúr dirent (dir entry, položka adresára):
  - inum: číslo i-uzlu na disku
  - name: 14 znakové meno súboru
- položka adresára je nevyužitá, ak je inum rovné 0

Ako prebehne v xv6 vytvorenie súboru a zápis do neho? "cat hi > x"

1. create
   - bwrite: block 33 by ialloc (alokovanie i-uzla v bloku i-uzlov č. 33, update i-uzla 33)
   - bwrite: block 46 by writei (zápis do dir entry, pridanie "x" funkciou dirlink())
   - bwrite: block 32 by iupdate (aktualizácia i-uzla dir entry, pretože sa mohol zmeniť)
2. write
   - bwrite: block 45 by balloc (alokovanie bloku v bloku bitmapy č. 45)
   - bwrite: block 764 by bzero (vynulovanie bloku č. 764)
   - bwrite: block 33 by iupdate (aktualizácia i-uzla pre blok č. 764)
   - bwrite: block 764 by writei (zápis "hi" do bloku č. 764)
   - bwrite: block 33 by iupdate (aktualizácia i-uzla pre blok č. 764)

Graf volaní pre open():

```
sys_open()                      sysfile.c
    create()                    sysfile.c
        ialloc()                fs.c
        iupdate()               fs.c
        dirlink()               fs.c
            writei()            fs.c
            iupdate()           fs.c
```

Graf volaní pre write():

```
sys_write()                     sysfile.c
    filewrite()                 file.c
        writei()                fs.c
            bmap()              fs.c
                balloc()                fs.c
                bzero()                 fs.c
            iupdate()           fs.c
```

Čo robí iupdate()? Prečo je potrebný?

Dĺžka súboru, addrs (dátové bloky súboru)...

Ako funguje zmazanie súboru? "rm x"

- bwrite: block 46 by writei (from sys_unlink; directory content)
- bwrite: block 32 by iupdate (from writei of directory content)
- bwrite: block 33 by iupdate (from sys_unlink link count of file; from itrunc zeroed length; from iput marked free)
- bwrite: block 45 by bfree (from itrunc, from iput)

Graf volaní pre unlink():

```
sys_unlink()
        writei()
        iupdate()
        iunlockput()
                iput()
                        itrunc()
                                bfree()
                                iupdate()
                        iupdate()
```

Konkurentné toky vo FS xv6:

1. súčasné čítanie a zápis rôznych súborov
2. súčasný preklad cesty (path lookup)

Ako je riešené súčasné vykonávanie ialloc()?

- viď sekvencia bread() - log_write() - brelse()
- bread() zamkne zámok (s možnosťou čakania), a načítava údaje z disku
- brelse() uvoľňuje zámok

Ako funguje vyrovnávacia pamäť (cache) blokov v bio.c:

- vyrovnávacia pamäť obsahuje iba zopár posledne použitých blokov
- bcache je definovaná na začiatku súboru bio.c

FS volania vyvolávajú bread(), a v rámci nej sa volá bget() v kernel/bio.c:

- bget() zisťuje, či je blok vo vyrovnávacej pamäti
- ak je, uzamkne lock (môže spať) a vráti blok
    - sleeplock je typ zámku, ktorý uspí proces (nebude vyťažovať cpu), pokým sa zámok neuvoľní
    - ak je teda zámok uzamknutý, volanie lock vyvolá uspanie procesu (nebude plánovaný na beh)
- ak nie je, použije nejaký voľný slot vo vyrovnávajúcej pamäti
- zvýšenie počítadla referencií b->refcnt++ chráni slot pred použitím iným procesov počas čakania

Dve úrovne uzamykania si môžeme všimnúť v bget():

1. bcache.lock chráni obsah štruktúry bcache; je to spinlock
2. b->lock chráni obsah konkrétneho slotu vyrovnávacej pamäte; je to sleeplock

Aká politika sa používa na vyraďovanie voľných blokov z vyrovnávacej pamäte?

- brelse() vkladá uvoľnený blok na začiatok zoznamu; nemaže b->dev, b->blockno, b->valid!!!
- bget() hľadá voľný blok z konca zoznamu
- ide o metódu "najdlhšie nepoužitý" (LRU - Least Recently Used); je to však najlepšia metóda?

FS xv6 kopíruje tie isté údaje 2x: z disku do vyrovnávacej pamäte, a z vyrovnávacej pamäte do užívateľského priestoru. Dá sa to zlepšiť tak, aby sa jedno kopírovanie ušetrilo?

Koľko pamäte RAM treba alokovať na diskovú vyrovnávaciu pamäť, aby neprišlo ku zastaveniu behu systému? Viď kód bget() v prípade, že sa nenájde žiaden voľný slot. Čo sa stane v takom prípade?