

OS MMXXII

MIT ;)

<https://pdos.csail.mit.edu/6.828>

Súborový systém

Prečo potrebujeme FS?

- Perzistencia medzi reštartami systému
- Hierarchia, organizácia, pomenovanie dát
- Zdieľanie (medzi aplikáciami a používateľmi)

Čo je na návrhu FS zaujímavé

- Obnova po zlyhaní (*crash recovery*)
- Výkon (konkurentné vykonávanie)
- Zdieľanie
- Bezpečnosť
- Abstrakcia jednotného prístupu
 - /proc, /sys, ...
 - rúry, schránky
 - zariadenia

Príklad systémov typu UNIX

```
fd = open("x/y.txt", ...)
```

```
write(fd, "abc", 3)
```

```
link("x/y.txt", "x/z.txt")
```

```
write(fd, "def", 3)
```

```
close(fd)
```

- Súbor „y.txt” aj „z.txt” (ten istý súbor, dve rôzne cesty k jeho obsahu) obsahuje „abcdef“

UNIX FS API

- Objekty
- Obsah (údaje)
- Pomenovanie
- Organizácia
- Synchronizácia

UNIX FS API

- Objekty: súbory
- Obsah (údaje): pole bajtov
- Pomenovanie: ľudsky čitateľné
- Organizácia: hierarchia mien
- Synchronizácia: nie je

Niektoré dôsledky API

Niektoré dôsledky API

- fd odkazuje na niečo, čo sa zachováva, aj keď
 - sa meno súboru zmení
 - sa meno zmaže (pokým ostáva súbor otvorený)

Niektoré dôsledky API

- fd odkazuje na niečo, čo sa zachováva, aj keď
 - sa meno súboru zmení
 - sa meno zmaže (pokým ostáva súbor otvorený)
- Súbor môže mať viacero liniek
 - T.j. údaje sú dostupné pomocou viacerých ciest
 - Žiadna cesta nemá prioritu

Niektoré dôsledky API

- fd odkazuje na niečo, čo sa zachováva, aj keď
 - sa meno súboru zmení
 - sa meno zmaže (pokým ostáva súbor otvorený)
- Súbor môže mať viacero liniek
 - T.j. údaje sú dostupné pomocou viacerých ciest
 - Žiadna cesta nemá prioritu
- Takže info o súbore musí byť niekde inde ako v adresári

Niektoré dôsledky API

- FS uchováva info o súbore v tzv. i-uzle (*inode*) na disku
- FS odkazuje na i-uzol pomocou i-čísła (*inumber*)
 - ide o niečo podobné ako FD

Niektoré dôsledky API

- FS uchováva info o súbore v tzv. i-uzle (*inode*) na disku
- FS odkazuje na i-uzol pomocou i-čísła (*inumber*) – ide o niečo podobné ako FD
- i-uzol musí obsahovať počítadlo referencií (liniek)
- i-uzol musí udržiavať počet otvorených FD
- Uvoľnenie i-uzla je odložené, pokým počet liniek a počet otvorených FD neklesne na 0

Disk

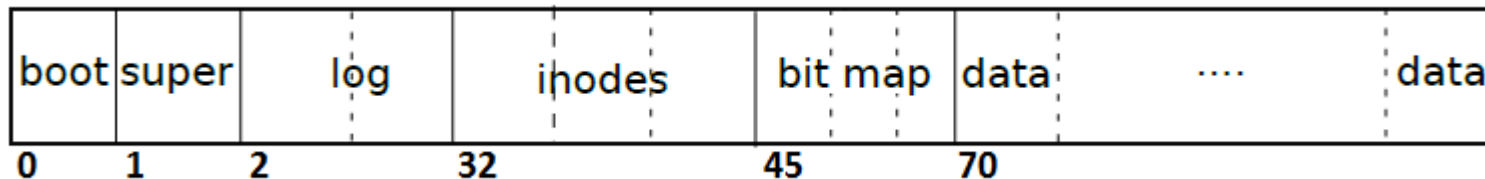
- Údaje sa uchovávajú na perzistentnom médiu
- Najpoužívanéjšie médiá
 - HDD (veľká kapacita, ale pomalé; lacné)
 - SSD (menšia kapacita, ale rýchle; drahé)

Disk

- Údaje sa uchovávajú na perzistentnom médiu
- Najpoužívanejšie médiá
 - HDD (veľká kapacita, ale pomalé; lacné)
 - SSD (menšia kapacita, ale rýchle; drahé)
- Najmenšia adresovateľná jednotka pri práci s diskom je 1 sektor = 512 B (vid' rámeček v RAM)
- OS pracuje s blokmi (násobok sektora)
 - xv6 používa bloky o veľkosti 2 sektorov (1 KiB)

Štruktúra disku xv6

- Blok 0: nepoužitý
- Blok 1: super blok (veľkosť fs, počet i-uzlov)
- Blok 2: začiatok blokov pre transakcie FS
- Blok 32: pole i-uzlov
- Blok 45: blok bitmapy využitia blokov (0=voľný blok, 1=použitý blok)
- Blok 70: začiatok blokov pre obsah súborov a adresárov



Štruktúra disku xv6

- Kto/čo vytvára túto štruktúru?
- Program mkfs (priečínok mkfs)

Štruktúra disku xv6

- Kto/čo vytvára túto štruktúru?
- Program mkfs (priečínok mkfs)
- Čo sú metadáta?
- Všetko na disku okrem samotného obsahu súborov
 - Superblok
 - i-uzly
 - Bitmapy
 - Obsah adresárov

I-uzol na disku

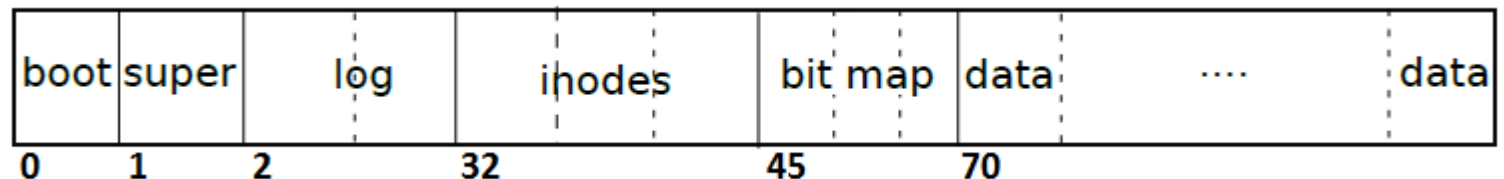
- Čo obsahuje i-uzol na disku?
 - Typ (voľný, súbor, adresár, ...) (`type`)
 - Počet odkazov na súbor z adresárovej štruktúry (`nlink`)
 - Veľkosť súboru (`size`)
 - Odkazy na údaje súboru (`attrs[12+1]`)
- Obrázok priamych, nepriamych, dvojito nepriamych blokov

Výpočet logického bloku

- Ako nájsť adresu bloku disku, v ktorom sa nachádza konkrétny bajt súboru?
- Príklad: 8000-ci bajt súboru
 - Logický blok = $8000/BSIZE = 8000/1024 = 7$
 - Položka v poli adres blokov súboru (addr) s ind. 7
- Príklad: 20134-tý bajt súboru
 - Logický blok = $20134/BSIZE = 19$
 - Kde sa nachádza adresa logického bloku súboru číslo 19?

I-uzol xv6 na disku

- Ako nájsť i-uzol na disku?
- Každý i-uzol má svoje číslo **inum** (index do poľa i-uzlov na disku)
- i-uzol má veľkosť 64 bajtov
- Adresa i-uzla na disku: $32 * BSIZE + 64 * inum$

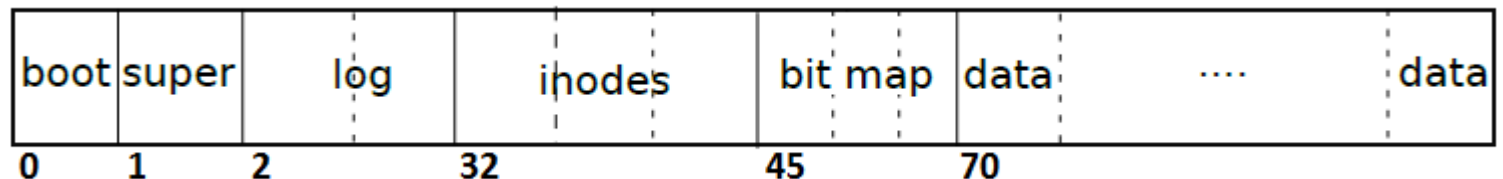


I-uzol xv6 na disku

- Štruktúra i-uzla v xv6: 64 B
- type: typ súboru, 2 B
- major, minor: označenie zariadenia, 2 B a 2 B
- nlink: počet liniek na i-uzol vo fs, 2 B
- size: veľkosť súboru v bajtoch, 4 B
- addrs[NDIRECT+1]: údajové bloky súboru,
4 B*(12+1) = 52 B

Obmedzenia FS v xv6

- Maximálny počet súborov/adresárov
- Maximálna veľkosť súboru/adresára
- Maximálna veľkosť súborového systému (disku)



Obsah adresára

- Adresár je tiež len súbor (t. j. pole bajtov)
 - Ale toto pole bajtov dokáže interpretovať jadro OS
 - K tomuto poľu bajtov nemá priamy prístup používateľ (iba pomocou systémových volaní na prácu s FS)

Obsah adresára

- Adresár je tiež len súbor (t. j. pole bajtov)
 - Ale toto pole bajtov dokáže interpretovať jadro OS
 - K tomuto poľu bajtov nemá priamy prístup používateľ (iba pomocou systémových volaní na prácu s FS)
- Súbor adresára sa v xv6 interpretuje ako pole štruktúr dirent (kernel/fs.h)
 - Veľkosť štruktúry je 16 B
 - Meno položky má iba 14 B
 - 2 B sú pre číslo i-uzla, ktorý popisuje obsah súboru
 - Štruktúra je voľná, ak je inum rovné 0

Spracovanie cesty v xv6

- Tzv. *pathname lookup* (napr. `"/x/y"`)
- `"/` *root* → natvrdo dané číslo *root* i-uzla (1)
- V obsahu súboru, ktorý popisuje i-uzol 1, hľadaj direntry s menom rovným reťazcu `"x"`
- Ak nájdeš položku `"x"`, v obsahu súboru, ktorý popisuje i-uzol tejto položky (v blokoch súboru `"x"`), hľadaj direntry s menom `"y"`
- Ak nájdeš položku `"y"`, našiel si aj číslo i-uzla, ktorý popisuje súbor `"y"`

Ukážka xv6

- Zameriame sa na zápis blokov na disk
- Vytvorenie súboru
- Zapísanie do súboru
- Zmazanie súboru
- `rm fs.img && make qemu`



```
// create
```

```
bwrite: block 33 by ialloc() // alloc inode in inode block 33
```

```
bwrite: block 33 by iupdate() // update nlink in inode
```

```
bwrite: block 70 by writei() // write direntry ('x' by dirlink())
```

```
bwrite: block 32 by iupdate() // update size of dir inode
```

```
bwrite: block 33 by iupdate() // itrunc() new inode
```

```
// write
```

```
bwrite: block 45 by balloc() // alloc a block in bitmap block 45
```

```
bwrite: block 648 by bzero() // zero the allocated block
```

```
bwrite: block 648 by writei() // write 'hi' to it
```

```
bwrite: block 33 by iupdate() // update size in inode
```

```
// write
```

```
bwrite: block 648 by writei() // write '\n' to it
```

```
bwrite: block 33 by iupdate() // update size in inode
```

\$ echo hi > x

Graf volaní

sys_open

sysfile.c

create

sysfile.c

ialloc

fs.c

X

iupdate

fs.c

X

dirlink

fs.c

writei

fs.c

X

iupdate

fs.c

X

itrunc

sysfile.c

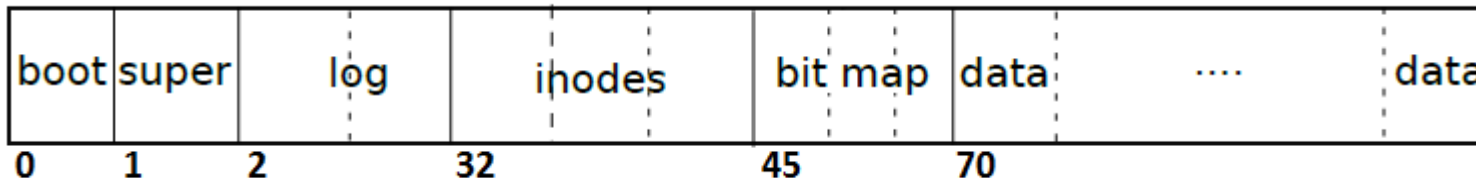
iupdate

fs.c

X

\$ rm x

bwrite: block 70 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
bwrite: block 45 by bfree() // from itrunc; from iput
bwrite: block 33 by iupdate() // from itrunc; zeroed length
bwrite: block 33 by iupdate() // from iput; marked free



Graf volaní

sys_unlink

sysfile.c

writei

fs.c

X

iupdate

fs.c

X

iupdate

fs.c

X

iunlockput

fs.c

iput

fs.c

itrunc

fs.c

bfree

fs.c

X

iupdate

fs.c

X

iupdate

fs.c

X

Domáce čítanie

Chapter 8

File System

xv6: a simple, Unix-like teaching operating system

Okrem kapitol a častí, ktoré hovoria o logovaní
transakcií FS