Operačné systémy / Súborový systém

Dušan Bernát
bernat@fmph.uniba.sk

Prehľad

- Súbory a typy súborov,
- adresárová štruktúra,
- prístupové práva,
- atribúty,
- i-uzol,
- štruktúra súborového systému *ext2* (superblok, mapa voľných blokov, mapa voľných i-uzlov, tabuľka i-uzlov, dátové bloky),
- *inotify*.

Čo je súbor?

Súborový systém

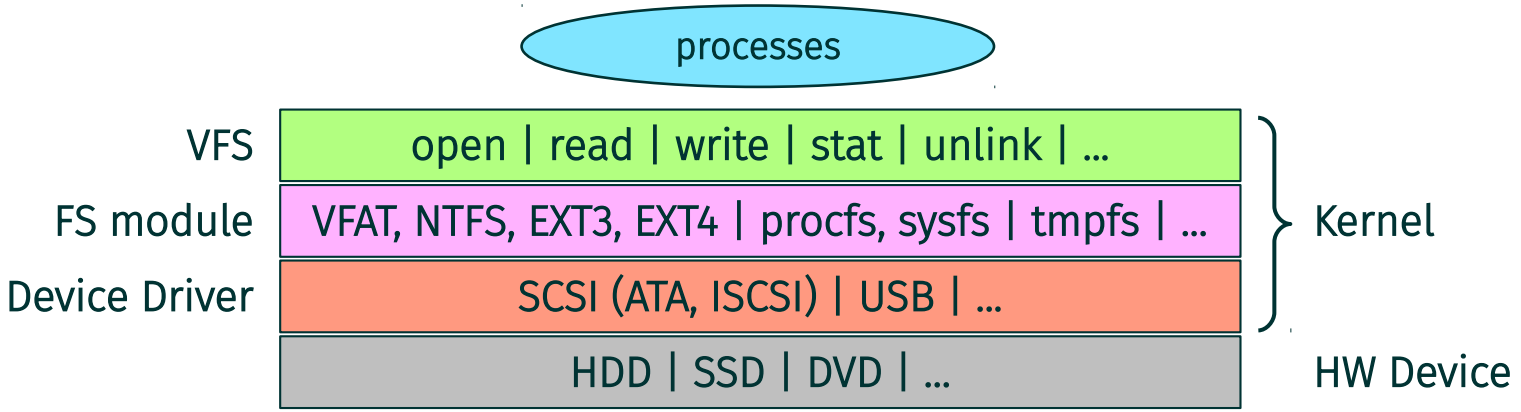
- Poskytuje abstraktný pohľad na súbor ako **postupnosť bajtov**.
  - Táto abstrakcia umožňuje pomocou súboru reprezentovať nie len dáta na disku, ale aj sieťové spojenia, vstup a výstup procesu, zariadenia (terminál), ...
- Transformuje požiadavky na operácie so súbormi na operácie s diskovými blokmi.
- Organizuje:
  - bloky do súborov, sektory do blokov, bloky do skupín (fyzická organizácia),
  - súbory do adresárov (logická organizácia).
- Spravuje informácie o súboroch.
- Riadi prístup k súborom.

Vrstvy súborového systému

- Operačný systém poskytuje procesom jednotné rozhranie pre prácu so súbormi, bez ohľadu na konkrétnu organizáciu.
  - Túto abstrakciu zabezpečuje vrstva **VFS** (*Virtual File System*).
  - open, read, write, seek, unlink, stat, mkdir, rmdir, lock, ...
  - Prevádza názvy súborov na potrebné údaje, spravuje otvorené súbory, riadi prístup.
- Súborových systémov, teda spôsobov **organizácie** súborov do blokov, môže byť mnoho (FAT, NTFS, ZFS, EXT4, UFS, ...).
  - Dátové bloky sú číslované, konvertuje číslo bloku na sektor.
  - Spravuje voľné bloky, alokuje bloky, udržuje potrebné štruktúry.
- Dáta súborov môžu byť uložené na rôznych fyzických médiách (harddisky, SSD, DVD, sieťové disky, ...).
  - Prístup ku konkrétnemu hardvéru zabezpečujú ovládače zariadení.

Hierarchia súborového systému OS

- Vrstvu VFS má implementovanú viacero operačných systémov.
- Poskytuje operácie rovnaké pre všetky FS.



Štruktúra súborového systému

- Implementácia súborov – súborový systém musí vedieť, ktoré dátové bloky patria k súboru.
- Využíva k tomu štruktúry, tiež uložené na médiu ktoré je rozdelené na bloky (číslované od 0).
- Možných spôsobov usporiadania je viacero, napríklad:
  1) Súvislé oblasti po sebe idúcich blokov.
  2) Zoznam blokov.
  3) Alokácia založená na i-uzloch.

1) Súvislá alokácia

- Dáta súboru sú umiestňované v po sebe idúcich blokoch.
- Implementácia je jednoduchá. Pre každý súbor stačí poznať číslo prvého bloku a veľkosť (počet blokov).
  - Môže byť uchovávané v adresári s názvom súboru.
- Čítanie je rýchle. Najmä sekvenčné, z HDD. Po nastavení hlavičky idú dáta za sebou.
- Súbory majú rôznu veľkosť → Ako vznikajú a zanikajú, voľné miesto podlieha fragmentácii.
- Pri vytváraní súboru je potrebné nájsť vhodné miesto. Konečná veľkosť súboru však nemusí byť pri vytváraní známa.
- Dnes sa používa pre súborové systémy určené pre RO (respektíve *Write-once)* médiá, napr. CD, DVD a pod.
  - Veľkosti všetkých súborov sú známe. Odpadá aj problém s fragmentáciou.

2) Zoznam blokov

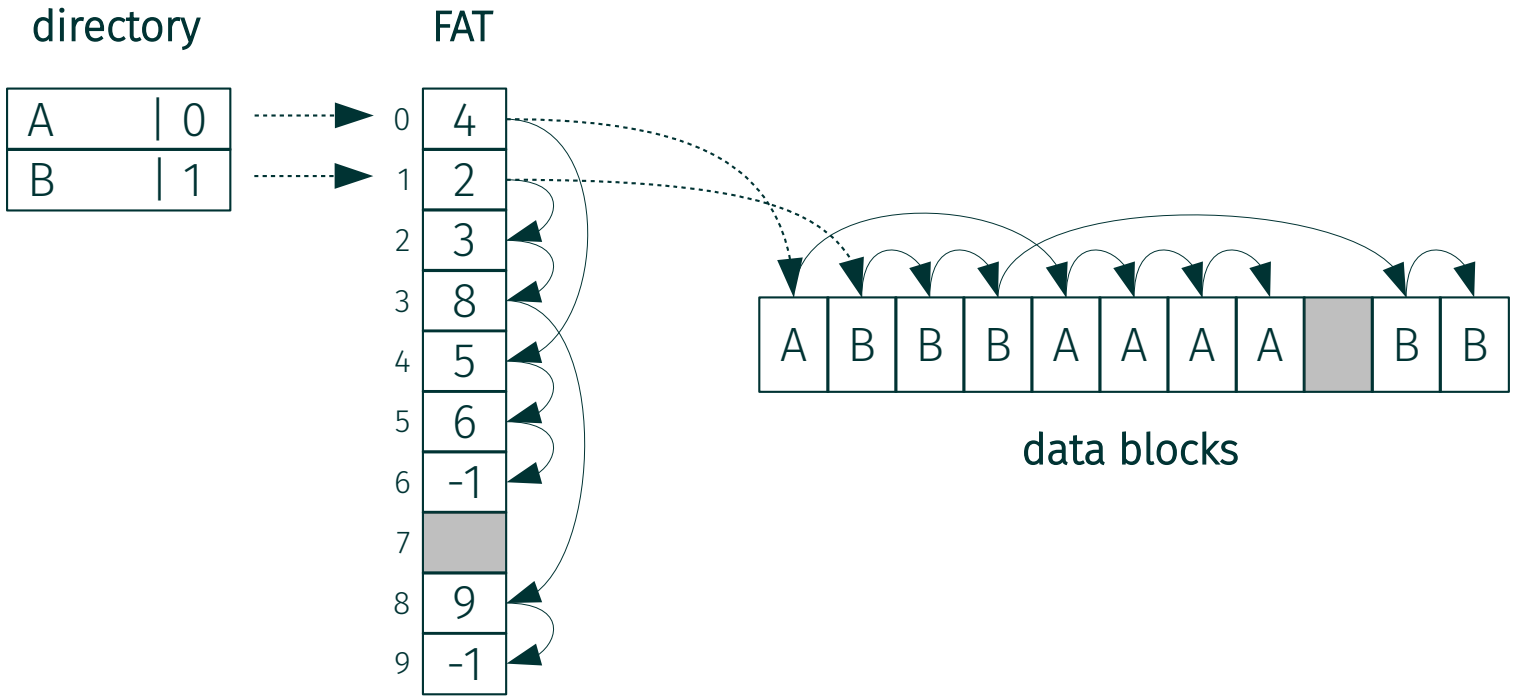
- Každý dátový blok súboru obsahuje odkaz na nasledujúci blok.
  - Riadiace informácie zaberajú aj časti dátových blokov (napr. prvé slovo).
  - Veľkosť miesta pre dáta v bloku už nemusí byť mocninou dvoch.
- Sekvenčný prístup je jednoduchý a rýchly.
- Náhodný (priamy) prístup k dátam súboru nie je možné realizovať bez prečítania všetkých predchádzajúcich dátových blokov.
  - Veľmi neefektívne.
- Zoznamy blokov tvoriacich súbor môžu byť umiestnené v samostatnej tabuľke, mimo dátové bloky.
  - Tým sa uvedené nevýhody odstránia.
  - Tabuľka sa nazýva FAT (*File Allocation Table*).

FAT (File Allocation Table)

- Každému bloku zodpovedá jedna položka v tabuľke FAT.
  - Bez ohľadu na to, či je blok využitý.
- Položka obsahuje číslo nasledujúceho bloku (položky), alebo značku konca súboru (neplatné číslo bloku).
- Položky blokov jedného súboru sú roptýlené po celej tabuľke. Pre prístup k jednému súboru je teda potrebná celá tabuľka.
- Pre veľké disky (a veľký počet blokov) sa tabuľka stáva príliš veľkou.
  - Nepoužíva sa pre disky väčšie než 200 MB.
- Napríklad:
  - Veľkosť disku $S = 128\ GB$, veľkosť bloku $BS = 4\ kB$, veľkosť položky $S_e = 4\ B$.
  - Počet blokov: $n = S / BS = 128 \cdot 2^{30} / 4 \cdot 2^{10} = 32 \cdot 2^{20} = 32\ M$
  - Veľkosť FAT (réžia): $S_{FAT} = n \cdot S_e = 32 \cdot 2^{20} \cdot 4\ B = 128\ MB$

FAT (File Allocation Table)

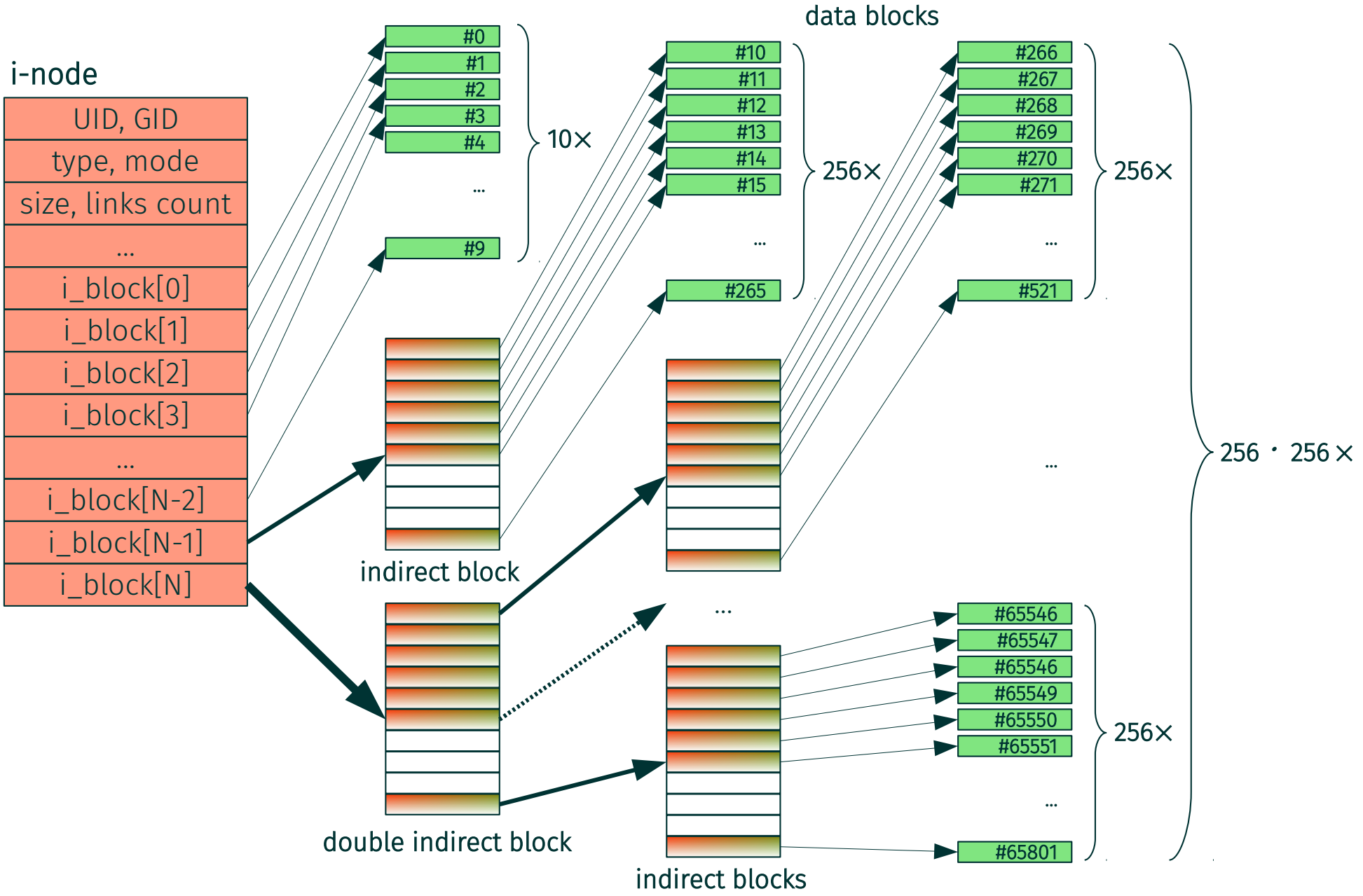
- FAT sa používala v MSDOS, FAT12 (32MB), FAT16 (2GB), FAT32 (2TB) vo Windows. Linux poskytuje RW podporu cez FS VFAT.
- Názvy súborov boli obmedzené na 8.3, resp. 255 znakov.
- Súborový systém obsahoval dve identické kópie FAT.



3) Alokácia založená na i-uzloch

- Tzv. i-uzol (*i-node*) je dátová štruktúra reprezentujúca súbor.
  - Každému súboru prislúcha práve jeden i-uzol.
  - Obsahuje všetky metadáta súboru:
    - prístupové práva, vlastníka, skupinu, príznaky,
    - veľkosť, počet blokov, počet liniek,
    - časy modifikácie obsahu súboru, modifikácie i-uzla, prístupu, zmazania, ...
  - Neobsahuje meno súboru.
- Obsahuje zoznam odkazov na bloky s dátami súboru.
  - Niekoľko priamych a zopár nepriamych, dvojitonepriamych, ...
  - Nepriame bloky sa nealokujú, pokiaľ nie sú potrebné (pre menšie súbory).
- Pre prístup k súboru postačujú informácie z jeho i-uzla.
  - Pri otvorení súboru môže byť skopírovaný do pamäte pre urýchlenie ďalších operácií.
  - Pri prístupe k väčším súborom je potrebné načítať aj bloky ukazovateľov.
- Tento spôsob využívajú všetky unixové OS, tiež Windows NTFS.

Štruktúra (nepriamych) odkazov na bloky



Parametre systému i-uzlov

- Aká je maximálna veľkosť súboru?
- Koľko miesta zaberajú odkazy?
- Aký vplyv na dobu prístupu majú nepriame bloky?
- Koľko by malo byť jednotlivých typov odkazov v i-uzle?
- Koľko by malo byť i-uzlov?
- Aký je vzťah veľkosti súborového systému a veľkosti čísla bloku?

Alokácia blokov

Alokácia dátových blokov súboru

- Aby bolo čítanie súboru čo najrýchlejšie, je výhodné alokovať dátové bloky v súvislých skupinách.
  - Fragmentácia dát zvyšuje dobu prístupu.
  - Platí to najmä pre klasické harddisky, pre SSD to nemusí byť tak.
- Veľkosť súboru nemusí byť vopred známa. Ako dosiahnuť súvislú alokáciu blokov?
- Prealokácia – pri alokácii bloku sa priamo v i-uzle vyhradí aj niekoľko (8) nasledujúcich. Využijú sa neskôr, pri zväčšovaní súboru.
  - Využívalo sa v ext2, nebolo kompatibilné so žurnálmi ext3.
- Rezervačné okno – pri alokácii blokov jadro vyhradí v pamäti súvislý rozsah blokov z ktorého prideľuje len jednému i-uzlu. Okná rôznych i-uzlov sa neprekrývajú.

Oneskorený zápis

- Bloky určené na zápis do súboru sa nezapíšu, až kým nie je potrebné uvoľniť pamäť (*allocate on flush*).
- Zvyšuje sa tým šanca, že v čase skutočného zápisu bude možné alokovať na jedekrát už všetky potrebné bloky.
  - Znižuje sa potreba viacnásobnej alokácie pre jeden súbor.
- Dočasné súbory sa možno vôbec nedostanú na disk, čo môže podstatne šetriť čas.
- Problém so stratou dát pri páde.

Súvislé úseky blokov (*extents*)

- Objem metadát potrebných na uchovanie zoznamu všetkých dátových blokov patriacich súboru môže byť pre veľké súbory značný.
  - Tiež veľká réžia pri odstraňovaní veľkých súborov.
- V prípade, že súbor má alokované súvislé oblasti blokov, môže byť výhodnejšie uchovávať pre každý úsek číslo začiatočného a koncového bloku úseku.
- Takýto súvislý blok sa nazýva *extent*.
  - Napr. ext4 ich využíva, čo je indikované atribútom “e” súboru.
  - *mount option* **`EXT4_FEATURE_INCOMPAT_EXTENTS`**
- Pri veľkej fragmentácii súboru môžu byť tiež potrebné dodatočné bloky pre zoznam úsekov.

Fragmentácia voľného miesta

- Nie je možné alokovať menej než jeden blok.
- Posledný blok súboru typicky nebude obsadený celý. Malé súbory nezaplnia ani jeden blok.
- Čím väčšia veľkosť bloku, tým väčšia interná fragmentácia.
- *Tail-merging* – spájanie koncov súborov.
  - Koncové časti viacerých súborov sú umiestnené v jednom bloku.
  - Zvyšuje to efektívnosť využitia diskového priestoru (znižuje to fragmentáciu voľného miesta), zároveň zvyšuje fragmentáciu dát, čo môže viesť k potrebe viacerých čítaní pri prístupe k dátam.
  - Je to výhodné najmä pre skupiny malých súborov. V tomto prípade je pravdepodobné, že *read-ahead* stratégia prenesie blok s koncami pri jednom čítaní spolu s predchádzajúcimi blokmi súborov.

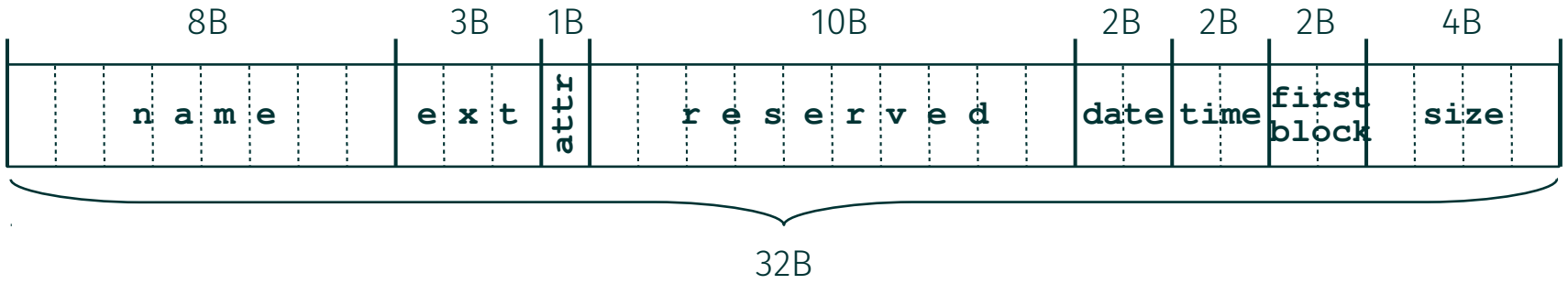
Príklad: Štruktúra i-uzla v EXT2

```
#include <ext2fs/ext2_fs.h>
struct ext2_inode {
 __u16 i_mode;          /* File mode */
 __u16 i_uid;           /* Low 16 bits of Owner Uid */
 __u32 i_size;          /* Size in bytes */
 __u32 i_atime;         /* Access time */
 __u32 i_ctime;         /* Creation time */
 __u32 i_mtime;         /* Modification time */
 __u32 i_dtime;         /* Deletion time */
 __u16 i_gid;           /* Low 16 bits of Group Id */
 __u16 i_links_count;  /* Links count */
 __u32 i_blocks;          /* Blocks count */
 __u32 i_flags;         /* File flags */
 __u32 i_block[EXT2_N_BLOCKS];    /* Pointers to blocks */
 __u32 i_generation;   /* File version (for NFS) */
 __u32 i_file_acl;     /* File ACL */
 __u32 i_dir_acl;      /* Directory ACL */
 __u32 i_faddr;         /* Fragment address */
   …                      …
}
```

Adresárová štruktúra

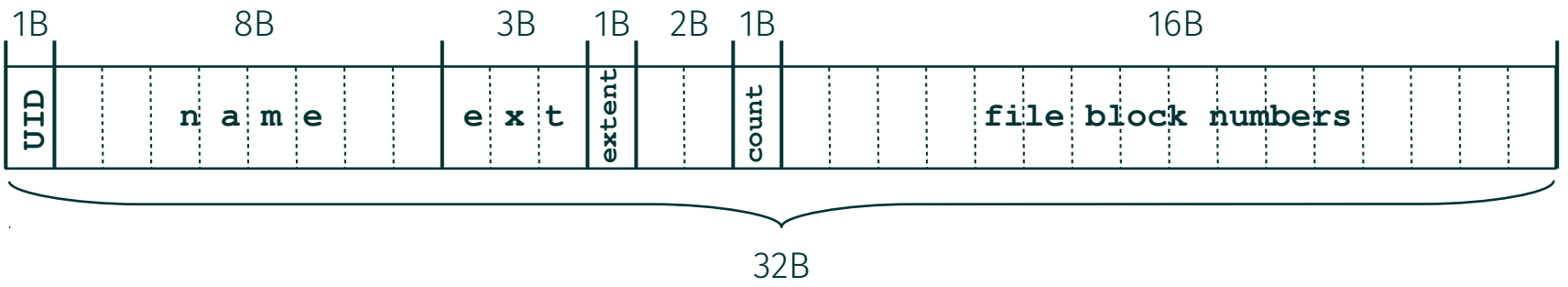
Logická organizácia súborov

- Adresár združuje viacero súborov.
  - Obsahuje názvy súborov, ktoré do neho patria.
- Spája tiež názov súboru s jeho umiestnením na disku.
  - Priraďuje názvu súboru pozíciu vo FAT, alebo číslo i-uzla.
- Adresáre môžu byť usporiadané, väčšinou tvoria strom.
- Položka adresára v MS DOS:
  - Konštantná veľkosť, obmedzený názov súboru.
  - Bloky sú alokované súvislo.



Tabuľka súborov so zoznamom blokov

- Položka adresára v CP/M:
  - V systéme bol len jeden adresár (nájsť súbor bolo jednoduché).
  - Konštantná veľkosť, obmedzený názov súboru.
  - Obsahuje zoznam blokov tvoriacich súbor.
  - Ak bol súbor väčší, alokovalo sa viacero položiek. Položky toho istého súboru sa usporiadavali cez pole **`Extent`**.



Adresáre v unixových systémoch

- Pôvodný System V Unix mal 16 bajtovú položku adresára. Obsahovala len 2B pre číslo i-uzla a 14B pre názov súboru.
- Dnešné systémy umožňujú premenlivú dĺžku názvu súboru.
  - Položka nemá konštantnú veľkosť. Adresár je pole štruktúr (položiek).
- Keď sa položka vymaže, zlúči sa s predchádzajúcou (zväčší sa jej dĺžka).
- Adresár je tiež súbor (má vlastný i-uzol aj dátové bloky). Ide však o súbor zvláštneho typu. Jeho štruktúra je daná.

Názov a cesta k súboru v adresárovej štruktúre

- Adreserárová štruktúra je strom. Má jeden koreň, označuje sa /.
- Názov súboru nemôže obsahovať znak / (s výnimkou koreňového adresára) a znak s kódom 0.
  - Znak / sa používa ako oddeľovač adresárov v ceste, 0 názov vždy ukončí (OS je naprogramovaný v C).
- Absolútna cesta k súboru sa začína vždy od koreňového adresára. Je jednoznačná.
  - Napr: **`/etc/sysconfig/network-scripts/ifcfg-eth0`**
- Relatívna cesta sa začína v aktuálnom adresári. Každý proces má vždy práve jeden aktuálny adresár (*current working directory*).
  - Aktuálny adresár je súčasťou PCB ((task_struct) current → fs → pwd).
  - Nastavuje sa systémovým volaním **`chdir()`**, dá sa zistiť cez **`getcwd()`**.

Počet odkazov na i-uzol

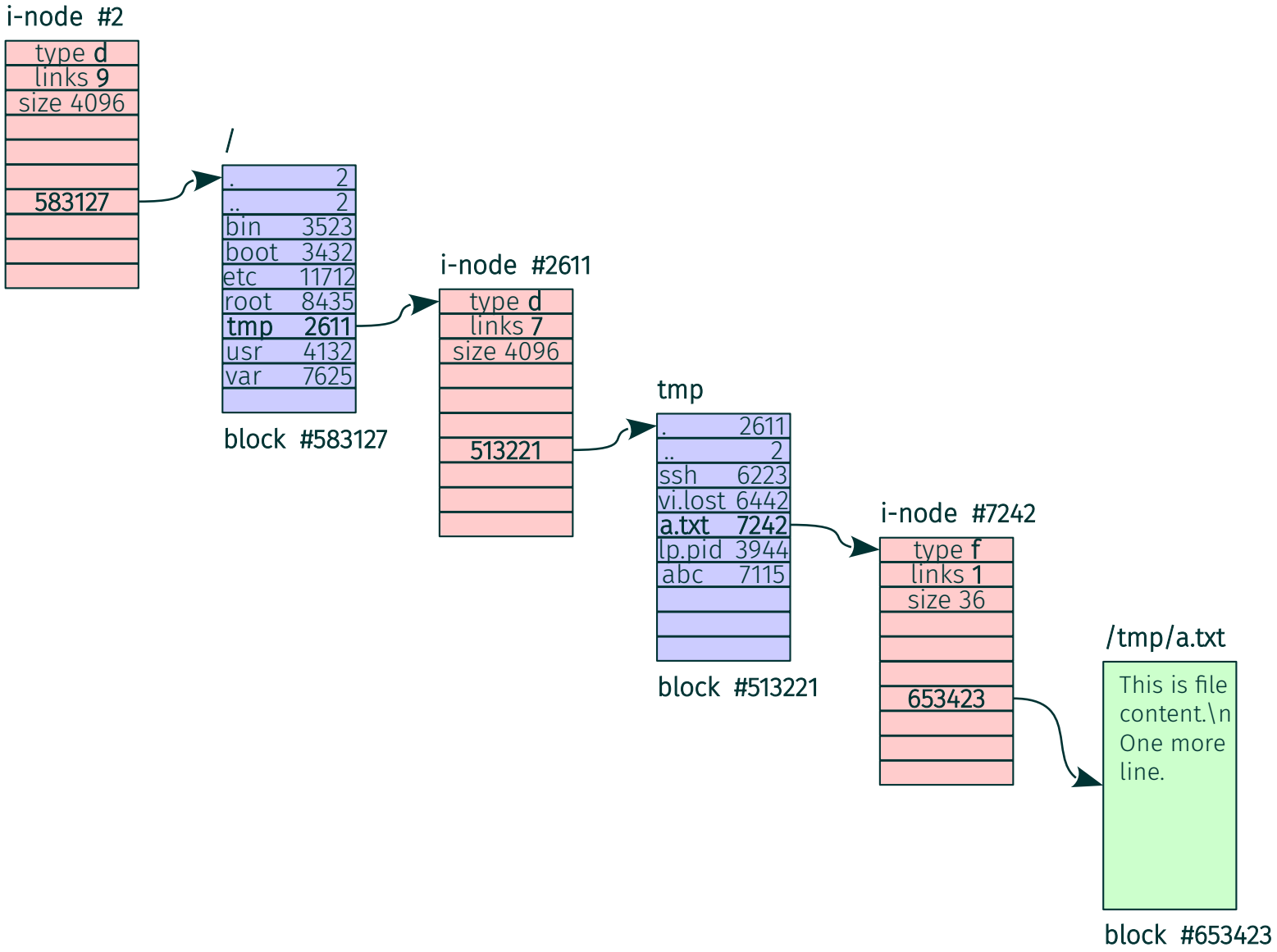
- Priradenie názvu súboru k číslu i-uzla v adresári sa nazýva linka (*hard link*).
  - Na jeden súbor, čiže jeden i-uzol, môžu odkazovať aj viaceré názvy z toho istého, alebo aj rôznych adresárov. Jeden súbor tak môže mať viacero rovnocenných názvov, obsah je len jeden.
  - Počet odkazov (počet liniek) je súčasťou i-uzla. Volania **`link()`** a **`unlink()`**.
  - Keď počet liniek klesne na nulu (teda keď sa zmaže aj posledný názov tohto súboru) a súbor sa nepoužíva, dátové bloky sa uvoľnia a i-uzol sa označí ako voľný.
- Každý adresár, aj prázdny, obsahuje minimáne dve položky.
  - Jedna ukazuje na seba "." a druhá na rodičovský adresár "..", teda na adresár, ktorý obsahuje tento adresár. Výnimkou je koreňový adresár, kde sú obe položky zhodné.
  - Napr: **`./a.out`**, **`../../etc/hosts`**, **`/etc/../etc/sysconfig/network-scripts/`** alebo **`/etc/./sysconfig/./../network-scripts/`**
- **`i_links_count`** má 16 bitov, čo limituje maximálny počet podadresárov.

Špeciálne (vyhradené) i-uzly

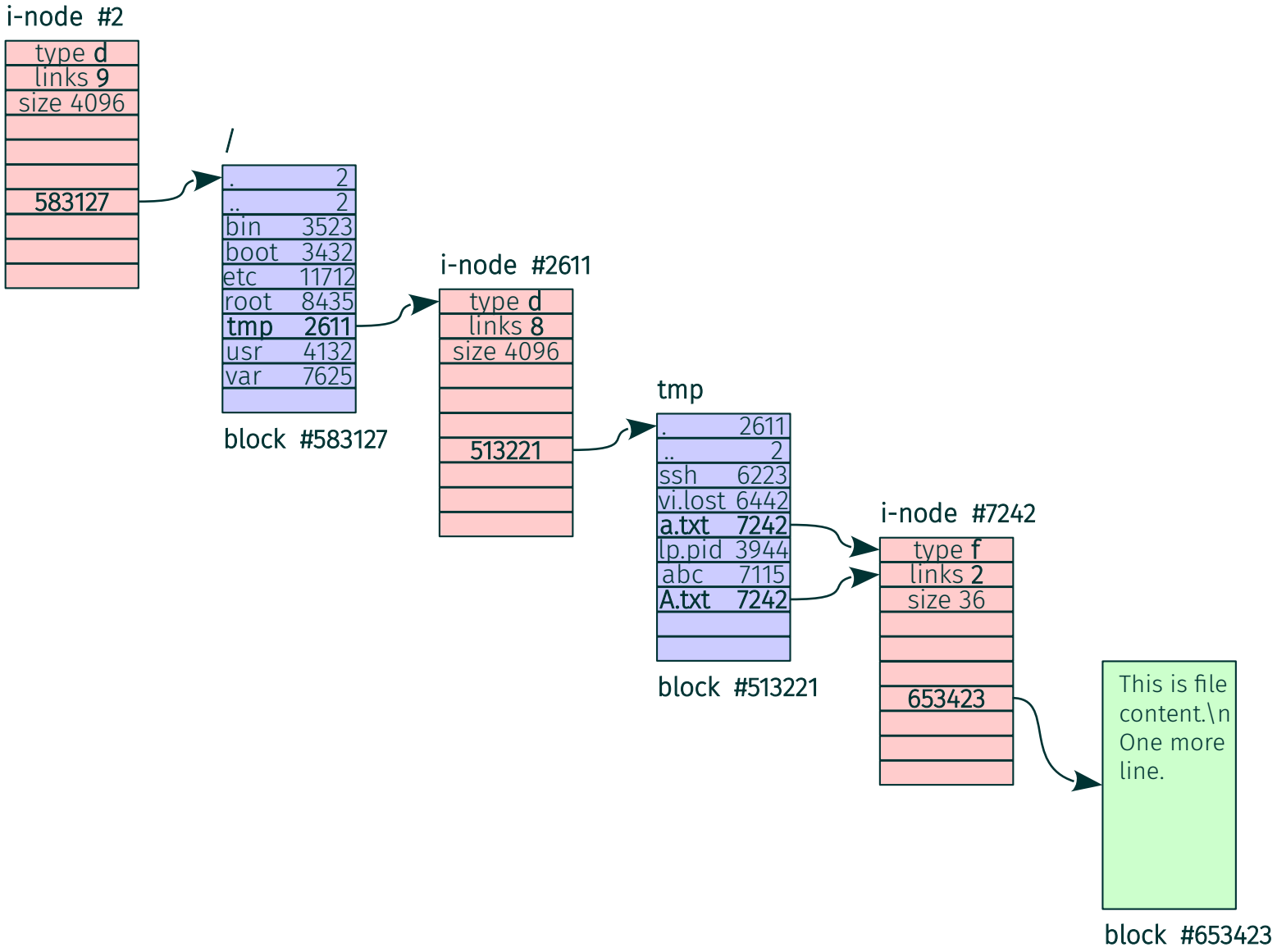
- V súborovom systéme je zvyčajne niekoľko i-uzlov obsadených bez toho, že by mali priradené meno.
- Číslo prvého použiteľného i-uzla je v superbloku (`__u32 s_first_ino`), typicky 11.

```
#include <ext2fs/ext2_fs.h>
#define EXT2_BAD_INO          1      /* Bad blocks inode */
#define EXT2_ROOT_INO         2      /* Root inode */
#define EXT2_ACL_IDX_INO      3      /* ACL inode */
#define EXT2_ACL_DATA_INO     4      /* ACL inode */
#define EXT2_BOOT_LOADER_INO  5      /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO    6      /* Undelete directory
                                        inode */
#define EXT2_RESIZE_INO       7      /* Reserved group
                                        descriptors inode */
#define EXT2_JOURNAL_INO      8      /* Journal inode */
```

Štruktúra adresárov a i-uzol



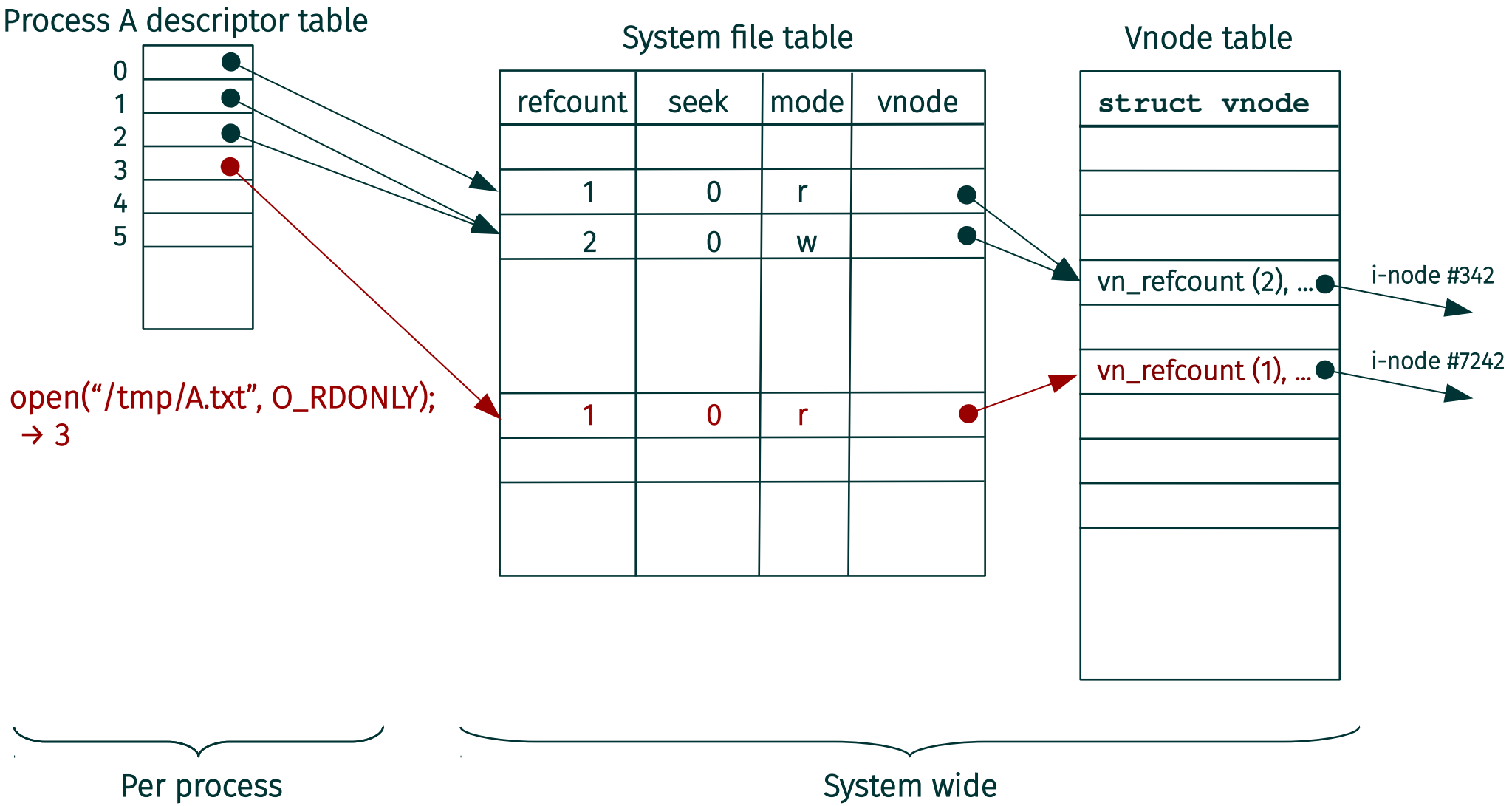
Hard linka


i-node #2
type d
links 9
size 4096
583127
/
. 2
.. 2
bin 3523
boot 3432
etc 11712
root 8435
tmp 2611
usr 4132
var 7625
block #583127
i-node #2611
type d
links 8
size 4096
513221
tmp
. 2611
.. 2
ssh 6223
vi.lost 6442
a.txt 7242
lp.pid 3944
abc 7115
A.txt 7242
block #513221
i-node #7242
type f
links 2
size 36
653423
This is file content.\n One more line.
block #653423

Implementácia VFS

- Jadro OS udržuje tabuľku používaných v-uzlov (**`vnode`**).
- Štruktúra **`vnode`**, na rozdiel od i-uzla, je nezávislá od konkrétneho súborového systému.
  - Napríklad FS na MS DOSe nemal žiadne i-uzly.
  - Štruktúra **`vnode`** reprezentuje súbor ľubovoľného súborového systému.
  - Štruktúra **`vnode`** existuje vždy len v pamäti, i-uzol na disku.
- Štruktúra **`vnode`** obsahuje:
  - Počet odkazov na tento vnode (**`vn_refcount`**), koľkokrát je práve otvorený (**`vn_opencount`**), zoznam zámkov pre výlučný prístup (**`vn_countlock`**), ukazovateľ na súborový systém ktorý súbor obsahuje (**`vn_fs`**), metadáta súboru konkrétneho FS (**`vn_data`**, napr. i-uzol), operácie nad týmto súborom (**`vn_ops`**).
- Operácie (open, read, write, ...) môžu byť špecifické pre daný súborový systém.

Vnode table


Process A descriptor table
0
1
2
3
4
5
System file table
refcount
seek
mode
vnode
1
0
r
2
0
w
1
0
r
Vnode table
struct vnode
vn_refcount (2), ...
i-node #342
vn_refcount (1), ...
i-node #7242
open("/tmp/A.txt", O_RDONLY);
→ 3
Per process
System wide

Typy súborov

Typy súborov

- Položka `i_mode`, (16 bitov), horné 4 bity.
- Určuje typ súboru z pohľadu OS (nie formát dát používateľa).

```
#include <ext2fs/ext2fs.h>
#define LINUX_S_IFSOCK  0140000 // UNIX domain socket (s)
#define LINUX_S_IFLNK   0120000 // symbolická linka (l)
#define LINUX_S_IFREG   0100000 // obyčajný súbor s dátami (f)
#define LINUX_S_IFBLK   0060000 // blokové zariadenie (b)
#define LINUX_S_IFDIR   0040000 // adresár (d)
#define LINUX_S_IFCHR   0020000 // znakové zariadenie (c)
#define LINUX_S_IFIFO   0010000 // pomenovaný dátovod/named pipe (p)
…
#define LINUX_S_ISREG(m) (((m) & LINUX_S_IFMT) == LINUX_S_IFREG)
#define LINUX_S_ISDIR(m) (((m) & LINUX_S_IFMT) == LINUX_S_IFDIR)
```

Obyčajný súbor (*regular file*)

- Bežný súbor, jeho úlohou je v dátových blokoch na disku uchovávať používateľské dáta.
- Obyčajný súbor obsahuje dáta ktorých význam pozná len používateľ, resp. aplikácia. Pre OS nemajú význam.
- Názov súboru taktiež nemá pre OS žiadny význam.
- Tzv. prípona je dobrovoľná. V názve môže byť aj viacero bodiek.
  - Niektoré aplikácie považujú názvy začínajúce bodkou za skryté a nezobrazujú ich.
  - Príkaz `file` sa snaží odhadnúť formát dát (text, zdrojový kód, obrázok, zvuk, ...) podľa databázy známych signatúr. Teda z obsahu, nie podľa názvu.
- Typ súboru z hľadiska OS uvedený v i-uzle nemá nič spoločné s typom, resp. formátom dát ktoré súbor obsahuje.

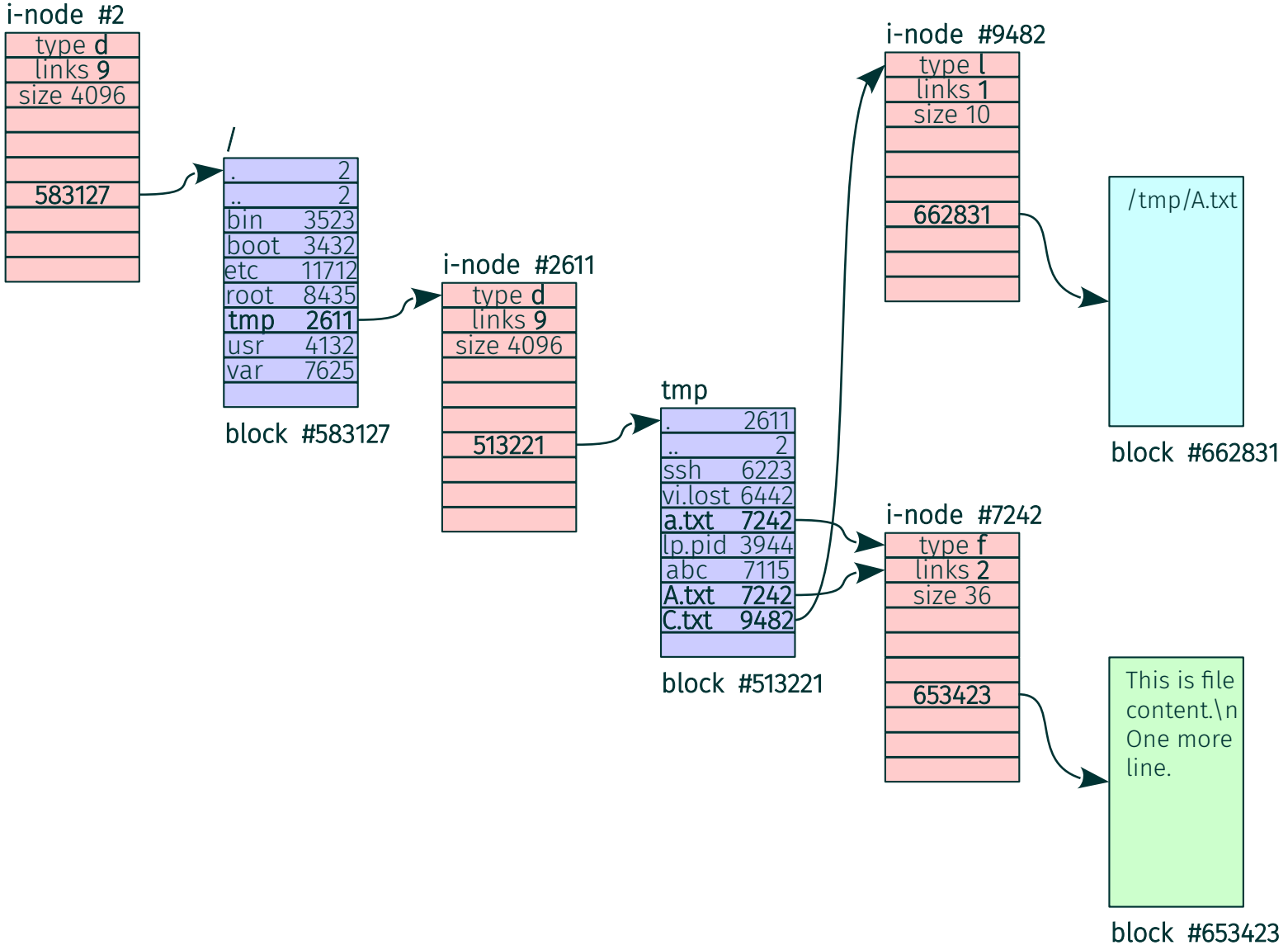
Adresár

- Adrresár je súbor.
- Obsah súboru typu adresár (d) má význam pre OS.
  - Dátové bloky obsahujú pole štruktúr adresárových položiek (**`dirent`**).
- Na manipuláciu s obsahom súboru typu “d” nie je možné použiť volania **`read`** a **`write`**.
- Narába sa s nimi pomocou zvláštnych volaní: mkdir, rmdir, opendir, closedir, readdir, getdents, seekdir, telldir, rewinddir, ...
- Optimalizácia:
  - Položka adresára môže obsahovať aj typ súboru. Ušetrí sa tým čítanie i-uzla v prípade, že operácia závisí od typu.
  - *mount option* **`EXT3_FEATURE_INCOMPAT_FILETYPE`**.

Symbolická linka

- Symbolická linka je súbor.
  - Má svoj názov a má svoj obsah. (A má typ “l”.)
  - V dátovom bloku obsahuje názov (iného) súboru.
- Vytvorí sa volaním **`symlink()`**.
- Operácia požadovaná pre názov súboru typu “l” sa namiesto toho vykoná so súborom ktorého názov linka obsahuje.
- Prístupové práva symbolickej linky sa neberú do úvahy.
- Optimalizácia: Pokiaľ je obsah linky krátky a zmestí sa na miesto odkazov na bloky v i-uzle, ušetrí sa pri prístupe jedno čítanie z disku.
- Hard linka je vlastnosť štruktúry súborového systému. Symbolická linka je súbor.

Symbolická linka



Ostatné typy

- Súbory ostatných typov (b, c, p, s) nevyužívajú dátové bloky.
- Odkazy v i-uzle môžu využívať iným spôsobom.
  - Špeciálne súbory blokových a znakových zariadení v nich uchovávajú hlavné a vedľajšie číslo zariadenia (*major/minor device number*).
  - *Inline data*, do 60 bajtov, ak to FS podporuje (*mount option*).
- Súbor daného typu (okrem d a l) je možné vytvoriť volaním **`mknod()`**. Pomenovaný dátovod tiež **`mkfifo()`**.

Prístupové práva

Vlastník a skupina

- Prístupové práva k súboru sa viažu na používateľa, ktorý operáciu vykonáva.
- Štruktúra i-uzla obsahuje identifikáciu vlastníka a vlastniacej skupiny súboru.
- Sú uložené v položkách:

```
__u16   i_uid;  /* Low 16 bits of Owner Uid */
__u16   l_i_uid_high;
__u16   i_gid;  /* Low 16 bits of Group Id */
__u16   l_i_gid_high;
```

- Pre vlastníka (UID), skupinu (GID) a ostatných má každý súbor v i-uzle určené práva prístupu na čítanie, zápis a vykonávanie.

Prístupové práva

- Položka **`i_mode`**, (16 bitov), dolných 9 bitov.

```
#include <ext2fs/ext2fs.h>

#define    LINUX_S_IRWXU  00700   // všetky práva pre vlastníka
#define    LINUX_S_IRUSR  00400   // právo čítať pre vlastníka
#define    LINUX_S_IWUSR  00200   // právo zapisovať pre vlastníka
#define    LINUX_S_IXUSR  00100   // právo vykonávať pre vlastníka

#define    LINUX_S_IRWXG  00070   // všetky práva pre skupinu
#define    LINUX_S_IRGRP  00040   // Read pre skupinu
#define    LINUX_S_IWGRP  00020   // Write pre skupinu
#define    LINUX_S_IXGRP  00010   // eXecute pre skupinu

#define    LINUX_S_IRWXO  00007   // všetky práva pre ostatných
#define    LINUX_S_IROTH  00004   // Read pre ostatných
#define    LINUX_S_IWOTH  00002   // Write pre ostatných
#define    LINUX_S_IXOTH  00001   // eXecute pre ostatných
```

Zvláštne oprávnenia

- Položka `i_mode`, bity 9–11.

```
#include <ext2fs/ext2fs.h>
#define LINUX_S_ISUID  0004000     // Set UID
#define LINUX_S_ISGID  0002000     // Set GID
#define LINUX_S_ISVTX  0001000     // Sticky bit
```

- SUID a SGID:
  - Na súbore spôsobia, že program sa bude vykonávať s efektívnymi právami vlastníka, resp. skupiny súboru.
  - Na adresári spôsobia, že nové súbory v ňom vytvárané, budú mať nastaveného vlastníka, resp. skupinu ako tento adresár.
- *Sticky bit*:
  - Na súbore sa väčšinou ignoruje.
  - Na adresári spôsobí, že súbor v ňom môže vymazať len vlastník (používa sa v `/tmp`).

Príznaky súborov

Špecifické vlastnosti súborov (1)

- Položka **`i_flags`** (32 bitov) určuje, ako má jadro pristupovať k tomuto i-uzlu.
- Určuje jeho špecifické vlastnosti.
- Mnohé sú implementačne závislé, nie všetky sú podporované.

- Secure deletion (s), `EXT2_SECRM_FL 0x00000001`
  - Po odstránení súboru sa jeho dátové bloky prepíšu náhodným (nulovým) obsahom.
- Undelete (u), `EXT2_UNRM_FL 0x00000002`
  - Po odstránení bude súbor presunutý do zvláštneho adresára s vymazanými súbormi.
- Compress file (c), `EXT2_COMPR_FL 0x00000004`
  - Transparentná kompresia obsahu súboru.
  - Obsah súboru je komprimovaný. Použitý kompresný algoritmus je nastavený v superbloku **`s_algo_bitmap`**.

Špecifické vlastnosti súborov (2)

- Synchronous Updates (S), `EXT2_SYNC_FL 0x00000008`
  - Všetky zmeny sa zapíšu do súboru hneď ako aplikácia robí operáciu **`write`** (synchrónne).
  - Vhodné pre súbory ktorých obsah by nemal byť stratený ani pri páde.
  - Zvyšuje pravdepodobnosť zachovania integrity obsahu súboru.
  - Celý súborový systém je možné pripojiť s voľbou **`sync`**.
- Immutable (i), `EXT2_IMMUTABLE_FL 0x00000010`
  - Systém nedovolí zmeny v i-uzle (žiadne nové linky, žiadne zmeny v dátach). Navyše súbor nemôže byť premenovaný ani zmazaný.
  - Tento atribút môže meniť len *root*.
  - Bloky súboru by sa nemali presúvať ani pri defragmentácii. Vhodné pre systémové súbory (napr. Jadro, bootloader).
  - Ak je *securelevel* jadra zvýšená, tento príznak je možné odstrániť len v "*single-user* móde".

Špecifické vlastnosti súborov (3)

- Append Only (a), `EXT2_APPEND_FL 0x00000020`
  - Obsah súboru už nemôže byť odstránený. Súbor je možné otvoriť len v režime *append*.
  - K doterajšiemu obsahu súboru je možné len pridávať .
  - Ak ide o adresár, súbory je možné vytvárať , nie však mazať.
  - Vhodné napríklad pre súbory log-ov.
  - Navyše ak je *securelevel* jadra zvýšená, tento príznak je možné odstrániť len v "*single-user* móde".
- No Dump (d), `EXT2_NODUMP_FL 0x00000040`
  - Program **dump** pre vytváranie záloh súborového systému bude tento i-uzol ignorovať.
  - Vhodné pre dočasné súbory s nepodstatným obsahom.

Špecifické vlastnosti súborov (4)

- No atime (A), `EXT2_NOATIME_FL 0x00000080`
  - Pre tento súbor sa nebude aktualizovať **`a_time`**.
  - Môže znížiť réžiu, následne možno aj spotrebu.
- No Tail (t), `EXT2_NOTAIL_FL 0x00008000`
  - Koniec súboru, ktorý nezaberá celý blok, sa nebude pripájať ku koncu iného súboru (ktorý má v poslednom bloku miesto).
  - Vhodné napr. pre súbory bootloader-a, ktorý má implementovanú podporu *ext2*, ale nepozná *tail-merging*.
- Journalled (j), `EXT3_JOURNAL_DATA_FL 0x00004000`
  - Obsah súboru by mal byť “žurnálovaný”.

Špecifické vlastnosti súborov (5) – index adresára

- B-tree Format Directory/Hash-indexed directory (**`EXT3_FEATURE_COMPAT_DIR_INDEX`**)
  - Keďže organizovanie súborov v adresároch pomocou zreťazených zoznamov je veľmi neefektívne pri veľkých adresároch, je možné použiť inú organizáciu.
  - Adresár na začiatku obsahuje hlavičku (dĺžka hlavičky, typ indexu, verzia *hash* funkcie, hĺbka stromu), potom nasledujú dvojice (*hash*, blok).
  - Koniec adresára – list stromu – vyzerá ako zvyčajne (kompatibilita).
  - Postup pri hľadaní i-uzla k názvu súboru:
    - Pri prístupe k súboru sa najskôr vypočíta k názvu súboru *hash* hodnota.
    - Binárnym vyhľadávaním sa nájde blok “listu stromu” ktorý by mohol obsahovať hľadané meno súboru.
    - Toto sa opakuje až po najnižšiu úroveň stromu. V tomto bloku (liste stromu) sa hľadá zvyčajným spôsobom – tento blok má aj pôvodnú štruktúru.
    - Ak sa nenájde a ide o kolíziu, prehľadávanie pokračuje v uzle nasledovníka.
  - Vkladanie, rozdeľovanie uzlov (vyvažovanie), kolízie kľúčov, ...

Špecifické vlastnosti súborov (6)

- Synchronous Directory Updates (D), `EXT2_DIRSYNC_FL 0x00010000`
  - Obsah adresára sa zapisuje synchrónne.
- Top of Directory Hierarchies (T), `EXT2_TOPDIR_FL 0x00020000`
  - Využíva sa pri alternatívnom spôsobe alokovania blokov (Orlov).
- Inodes Uses Extents (e), `EXT3_EXTENTS_FL 0x00080000`

- Manipulácia s príznakmi
  - Zobrazenie príznakov súborov: **`lsattr`** – podobne ako **`ls`**
  - Zmena príznakov: **`chattr [-R] +=[AsacDdIijsTtu] files`**

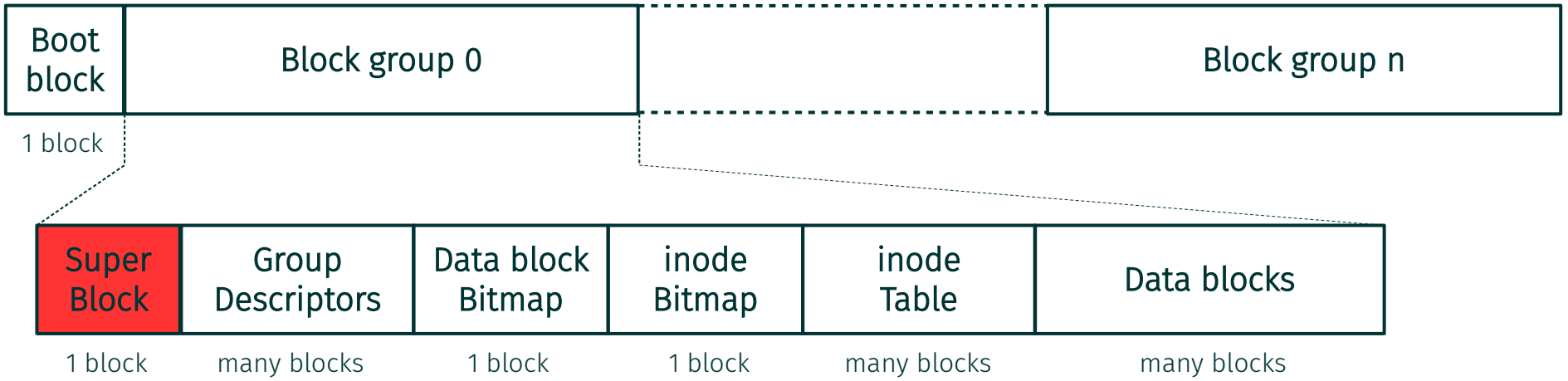
Časové údaje v i-uzle

- **`i_ctime`** – *inode change time*, čas poslednej zmeny obsahu i-uzla (čiže metaúdajov o súbore, nie obsahu súboru),
- **`i_mtime`** – *modification time*, čas poslednej modifikácie obsahu súboru (jeho dátových blokov),
- **`i_atime`** – *access time*, čas posledného prístupu k obsahu súboru (čítanie/zápis dátových blokov),
- **`i_dtime`** – *deletion time*, čas vymazania tohoto i-uzla; ak sa nejedná o vymazaný i-uzol, je nastavený na 0.
- Všetky časy sú uchovávané v počte sekúnd od začiatku roku 1970.
- *ext4* už podporuje nanosekundové časy (**`EXTRA_ISIZE`**).

Štruktúra súborového systému *ext*

Štruktúra *ext2fs*

- Súborový systém je rozdelený na niekoľko skupín blokov.
- Každá sa začína kópiou superbloku a štruktúrami pre evidenciu voľných i-uzlov a dátových blokov.



Superblok

- Štruktúra, nachádzajúca sa na pozícii 1024 bajtov od začiatku média (diskového oddielu). Má veľkosť 1kB.
- Obsahuje informácie potrebné pri interpretácii štruktúry a obsahu FS, okrem iného:
  - verziu FS, (zoznam podporovaných vlastností),
  - veľkosť bloku, počet blokov a i-uzlov v skupine,
  - počet blokov a i-uzlov (všetkých a voľných), veľkosť i-uzla,
  - čas pripojenia, modifikácie a poslednej kontroly,
  - stav, počet pripojení od poslednej kontroly,
  - počete rezervovaných blokov, UID a GID pre koho sú vyhradené,
  - UUID, názov, ...
- Všetky hodnoty majú usporiadanie *little endian*.
- Manipulácia so superblokom: **`tune2fs`**.

Príklad: Ukážka superbloku

```
#include <ext2fs/ext2_fs.h>

struct ext2_super_block {
   __u32 s_inodes_count;       /* Total fs Inodes count */
   __u32 s_blocks_count;       /* Total fs Blocks count */
   __u32 s_free_blocks_count; /* Free blocks count */
   __u32 s_free_inodes_count; /* Free inodes count */
   __u32 s_first_data_block;  /* First Data Block */
                              /* beginning of first block group */
   __u32 s_r_blocks_count;     /* Reserved blocks count */
   __u16 s_def_resuid;         /* Default uid for reserved blocks */
   __u16 s_def_resgid;         /* Default gid for reserved blocks */
      …
```

Veľkosť bloku

- Udáva sa v bajtoch, ako mocnina 2, od 1024, ($block\_size = 1024 \cdot 2^{s\_log\_block\_size}$)
- Prípustné hodnoty teda sú 1024, 2048, 4096, ?
- Ak je priemerná veľkosť súboru väčšia (niekoľko sto bajtov a viac), veľké bloky znižujú počet potrebných I/O operácií a teda aj réžiu.
- Ak sú súbory menšie, výhodné sú menšie bloky, pretože sa tým zmenšuje interná fragmentácia.
- Bloky sa môžu deliť na fragmenty. Menšie časti viacerých súborov sú v jednom bloku.
- Fragment môže byť väčší aj menší než 1024.
- Prakticky sa v *ext2fs* fragmenty nepoužívajú, veľkosť fragmentu je rovnaká ako veľkosť bloku.

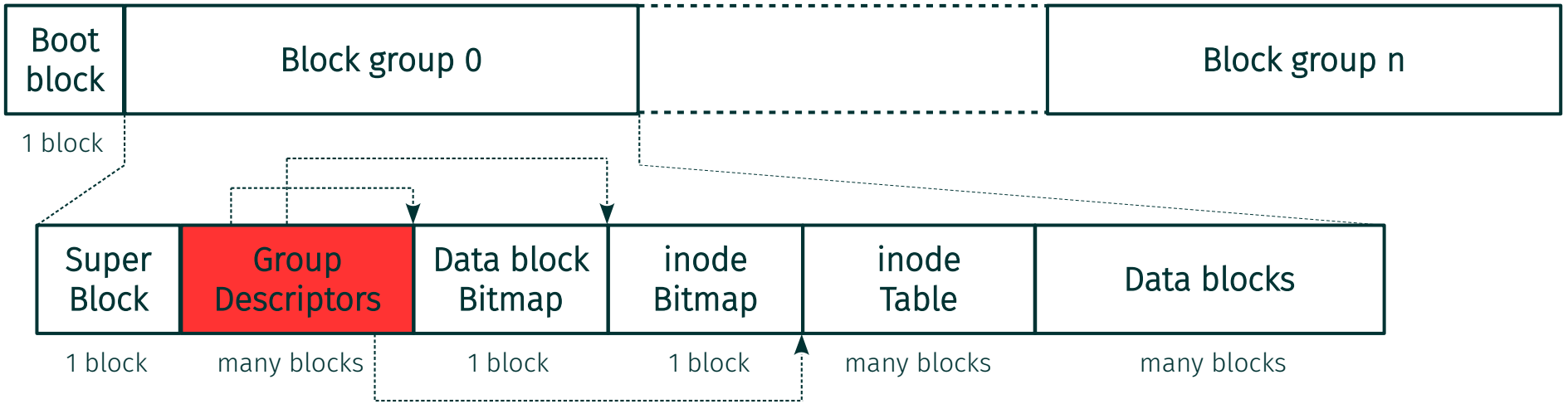
```
__u32 s_log_block_size;  /* Block size */
__s32 s_log_frag_size;   /* Fragment size */
```

Podporované možnosti súborového systému

- Štruktúra súborových systémov založených na *ext2* umožňuje bezpečne pridávať nové vlastnosti bez narušenia kompatibility so staršími verziami.
- Superblok obsahuje tri bitové mapy vlastností, ktoré určujú správanie sa jadra v prípade, že danú vlastnosť nepozná (nemá implenetovanú).
- COMPAT: vlastnosti kompatibilné so staršími verziami. Nemenia pôvodnú štruktúru, len pridávajú nové možnosti, ktoré staršie jadro môže bezpečne ignorovať aj pri R/W prístupe (napr. HAS_JOURNAL, DIR_PREALLOC, IMAGIC_INODES, EXT_ATTR, RESIZE_INODE, DIR_INDEX).
- RO_COMPAT: vlastnosti kompatibilné pri čítaní, ale zapisovanie staršieho jadra by súborový systém poškodilo, preto sa pripája len na čítanie (read-only), (napr. SPARSE_SUPER, LARGE_FILE).
- INCOMPAT: vlastnosti, ktoré menia štruktúru FS tak, že staršie jadro ho ani neprečíta (napr. COMPRESSION, FILETYPE, EXTENTS), preto ak takéto vlastnosti nepozná, tak FS nepripojí.

Skupiny blokov

- Bloky sa zlučujú do skupín, aby sa zmenšila fragmentácia dát a znížil čas prístupu (nastavenia hlavy disku) pri čítaní dlhých sekvencií.
- Metadáta a dáta jedného súboru sú umiestnené v rovnakej skupine.



Štruktúra skupiny blokov

- Počet skupín sa určí z hodnôt v superbloku (počet blokov, fragmentov a i-uzlov v skupine):
  - *bg_nr = s_blocks_count/s_blocks_per_group*, kde *s_blocks_per_group = 8 · block_size*
- Pri vytváraní FS sa počet blokov v skupine zvolí tak, aby sa bitová mapa zmestila do jedného bloku (pri 4kB bloku je to 32768).
- Umiestnenie deskriptora skupiny **`group_num`**:
  - *gd_offset = (1 + sb · s_first_data_block) · block_size + sizeof(struct ext2_group_desc) · group_num*
- Každá skupina blokov sa začína kópiou superbloku (ak nie je SPARSE_SUPER) a kópiou tabuľky deskriptorov (všetkých) skupín.
- Jadro používa informácie len z prvej skupiny. Do ostatných sa zmeny zapíšu len pri kontrole programom **`e2fsck`**. Ak sa údaje v prvej skupine poškodia, z ďalších kópií je ich pravdepodobne možné obnoviť.

Group descriptor

```
struct ext2_group_desc
{
   __u32 bg_block_bitmap;        /* Blocks bitmap block */
   __u32 bg_inode_bitmap;        /* Inodes bitmap block */
   __u32 bg_inode_table;         /* Inodes table block */
   __u16 bg_free_blocks_count;   /* Free blocks count */
   __u16 bg_free_inodes_count;   /* Free inodes count */
   __u16 bg_used_dirs_count;     /* Directories count */
   __u16 bg_pad;
   __u32 bg_reserved[3];
};
```

- Obsahuje čísla blokov s bitmapami voľných dátových blokov a i-uzlov a tabuľkou i-uzlov.
- Ostatné štatistické údaje používa jadro na rovnomerné rozmiestňovanie údajov na disku.

Bitové mapy

- *Data Block Bitmap:* Každý bit takéhoto bloku označuje, či príslušný dátový blok je voľný, alebo alokovaný pre nejaký súbor.
- Hodnota 1 označuje obsadené bloky, 0 voľné. Prvý blok v skupine je opísaný nultým bitom nultého bajtu v bloku bitmapy.
- Analogicky bitmapa i-uzlov (*inode bitmap*) udáva či sú jednotlivé i-uzly z tabuľky i-uzlov v danej skupine blokov voľné pre nové súbory, alebo už obsadené existujúcimi súbormi.
- Tabuľku i-uzlov (*inode table*) tvorí pole štruktúr i-uzlov umiestnené v súvislej skupine blokov.
- Každému i-uzlu zodpovedá jeden bit v bitovej mape.
- Pozíciu i-uzla je možné vypočítať z jeho čísla a parametrov súborového systému.

Počet i-uzlov

- Celkový počet i-uzlov a teda aj súborov v súborovom systéme *ext2fs* je daný napevno, v čase jeho tvorby.
- Zadávajú sa v pomere počtu bajtov na i-uzol (čiže na súbor) – priemerná veľkosť súboru.
- Napríklad:
  - FS má 2GB, hustota i-uzlov je 4096 B/i-uzol,
  - Počet i-uzlov: $2G/4096 = 524288$ i-uzlov,
  - pri 128B i-uzle to je: $524288 \cdot 128 = 64MB$, čo predstavuje 3.125%.
- Nevýhody:
  - Ak ich vytvoríme veľa, zostanú nevyužité a zbytočne zaberajú miesto,
    - Bežne na zaplnených systémoch sa použije len niekoľko percent i-uzlov, rádovo desiatky až stovky MB zostávajú teda nevyužité (ale robí sa to).
  - Ak ich vytvoríme málo a minú sa, nedajú sa vytvárať ďalšie súbory.
    - Počet i-uzlov nie je možné zvýšiť.
    - Ide o vážne obmedzenie.

Synchronizácia

Konzistentnosť obsahu súborov

- Súborový systém (na rozdiel napríklad od databáz), nie je transakčný. Viaceré procesy môžu mať súčasne otvorený ten istý súbor.
  - Aj na zápis. Poradie v akom sa zmeny zapíšu nie je dané.
- Zápis sa robí na aktuálnu pozíciu (*seek*), ktorá je uložená v globálnej tabuľke otvorených súborov.
  - Ak bol súbor otvorený s príznakom *append* (dopĺňanie), **`write`** najskôr vždy atomicky posunie ukazovateľ na koniec (**`lseek`**/SEEK_END) a potom zapisuje.
  - Ak je súbor otvorený na zápis len ako *append* (všetkými procesmi), žiadne dáta sa neprepíšu. Inak to zaručené nie je.
- Operácia **`write`** nemusí zapísať celý *buffer*. Vráti počet skutočne zapísaných bajtov.
- Dôsledky:
  - Zápisy od rôznych procesov sa môžu vo výsledku rôzne "prekrývať".
  - Zápis súvislého bloku dát (napríklad štruktúry) nie je atomický.
  - Rôzne zápisy môžu prepisovať rovnaké miesto súboru (ak nie je nastavený príznak *append*).
  - Pokiaľ záleží na dodržaní poradia zápisu od rôznych procesov, je potrebná synchronizácia.

Zamykanie súborov bez podpory OS

- Synchronizáciu prístupu k súborom je možné dosiahnuť pomocou tzv. zámkov spojených so súbormi.
  - Súborový systém je “viditeľný” všetkými procesmi (netreba spoločnú pamäť).
  - Väčšina OS dnes takúto možnosť ponúka.
- Ak nie, ako zámok môže slúžiť prázdny súbor so známym názvom.
  - Ak (zámok) existuje, súbor je už používaný, treba čakať alebo skúsiť neskôr.
- Otestovať či súbor existuje, ak nie, vytvoriť ho? → Nie je to atomické!
- Vytvorenie súboru je atomické.
  - Operácia `open` s príznakmi (O_CREAT | O_EXCL) zlyhá (EEXIST), pokiaľ daný súbor už existuje. Ak nie, vytvorí ho.
- Výhoda: prenositeľnosť. Nevýhoda: *polling*.

Súborové zámky

- Pre deskriptor otvoreného súboru je možné nastaviť zámok.
- Volanie: `int flock(int fd, int operation);`
- Operácie so zámkami (*advisory lock*):
  - LOCK_SH - zdieľaný zámok, viacero procesov môže mať takýto zámok súčasne pre ten istý súbor (na čítanie),
  - LOCK_EX - exkluzívny, pre daný súbor ho súčasne môže mať len jeden súbor (na zápis),
  - LOCK_UN - uvoľnenie zámku.
- Ak bude v type operácie nastavený bit LOCK_NB, žiadosť o uzamknutie už zamknutého súboru sa nezablokuje (vráti EWOULDBLOCK).
- Prístupové práva sa pri tomto volaní nekontrolujú.

Zdieľané a exkluzívne zámky

- **Shared lock**
  - Viaceré procesy môžu získať zdieľaný zámok súčasne.
  - Ak má niektorý proces takýto zámok, exkluzívny zámok už nemôže byť získaný.
  - Čítacie procesy pred prístupom musia získať zdieľaný zámok.
- **Exclusive lock**
  - Ak má niektorý proces takýto zámok, žiadny proces nemôže získať ani exkluzívny, ani zdieľaný zámok.
  - Zapisujúci proces pred prístupom musí získať exkluzívny zámok.

Odporúčacie a povinné zámky

- **Advisory lock** - tieto zámky majú len “odporúčací” charakter.
  - Implementuje ich volanie `flock()`.
  - Síce umožňujú zabezpečiť konzistentný prístup k súborom, ale nezaručujú ho.
  - Ak niektorý proces zámky ignoruje (nepoužije vôbec funkciu `flock`), k súboru pristupovať môže.
- **Mandatory lock** - stav týchto zámkov kontroluje OS pri každej operácii so súborom (open, read, write, ...) pre všetky procesy.
  - Pri zamknutom zámku operácia môže byť zablokovaná, alebo vráti EAGAIN.
  - Musí ich podporovať súborový systém.
  - Linux: Mount option: mand, kombinácia bitov v prístupových právach súboru (g+s, g-x). Zámky nemajú vplyv na volania `rename()` a `unlink()`. Nespoľahlivé (obmedzená implementácia, možný súbeh).
- Rozhranie `fcntl()` môže byť použité pre oba druhy zámkov.

Operácie s deskriptormi

- Okrem prístupu k dátam je možné so súbormi robiť aj ďalšie operácie.
- Volanie: `int fcntl(int fd, int cmd, /* arg */ );`
- Operácie:
  - Duplikácia deskriptora na prvý voľný.
  - Uzamknutie časti súboru.
  - Notifikácia o zmene obsahu adresára alebo súborov v ňom (dnotify).
  - Čítanie a nastavenie príznakov (FD_CLOEXEC) a stavu deskriptora.
  - Notifikácia o prístupe iného procesu k súboru (leases).
  - Zmena kapacity dátovodu.

Zamknutie časti súboru

- Operácia F_SETLK umožňuje procesu uzamknúť na čítanie alebo zápis (*l_type*) istý rozsah súboru (*l_whence, l_start, l_len*).
  - Toto volanie je neblokujúce a v prípade, že zámok nie je možné získať (napríklad ak daný rozsah je už zamknutý iným procesom), vráti chybový kód.
- Operácia F_SETLKW bude čakať, až kým nebude možné zámok získať.
- Samostatné rozhranie pre zámky POSIX (ktoré môže využívať **`fcntl`**) poskytuje **`lockf()`**.
- Informácie o aktuálnych zámkoch (Linux):
  - **`/proc/locks`**
  - FLOCK/POSIX, ADVISORY/MANDATORY, READ/WRITE, PID, MAJOR:MINOR:INODE, START, END/EOF
  - Názov súboru pre INODE sa dá zistiť napríklad v **`/proc/PID/fd`**.

To je zatiaľ všetko