Датотечни системи

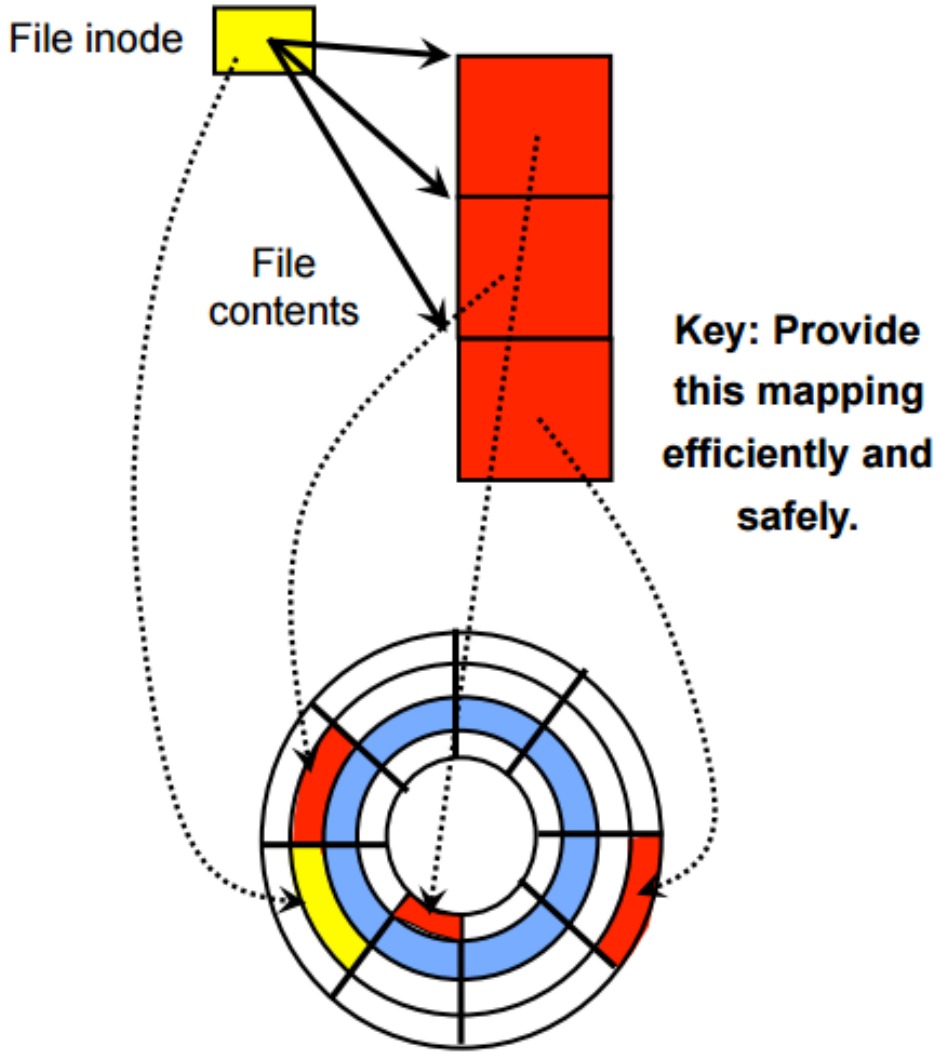
Оперативни системи 2024

проф. д-р Димитар Трајанов,
проф. д-р Невена Ацковска,
проф. д-р Боро Јакимовски,
проф д-р Весна Димитрова,
проф. д-р Игор Мишковски,
проф. д-р Сашо Граматиков,
вонр. проф. д-р Милош Јовановиќ,
вонр. проф. д-р Ристе Стојанов,
доц. д-р Костадин Мишев

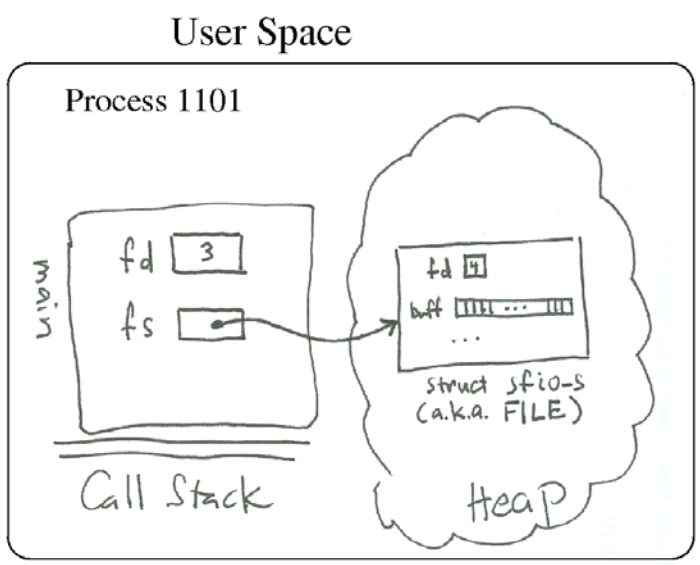
Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

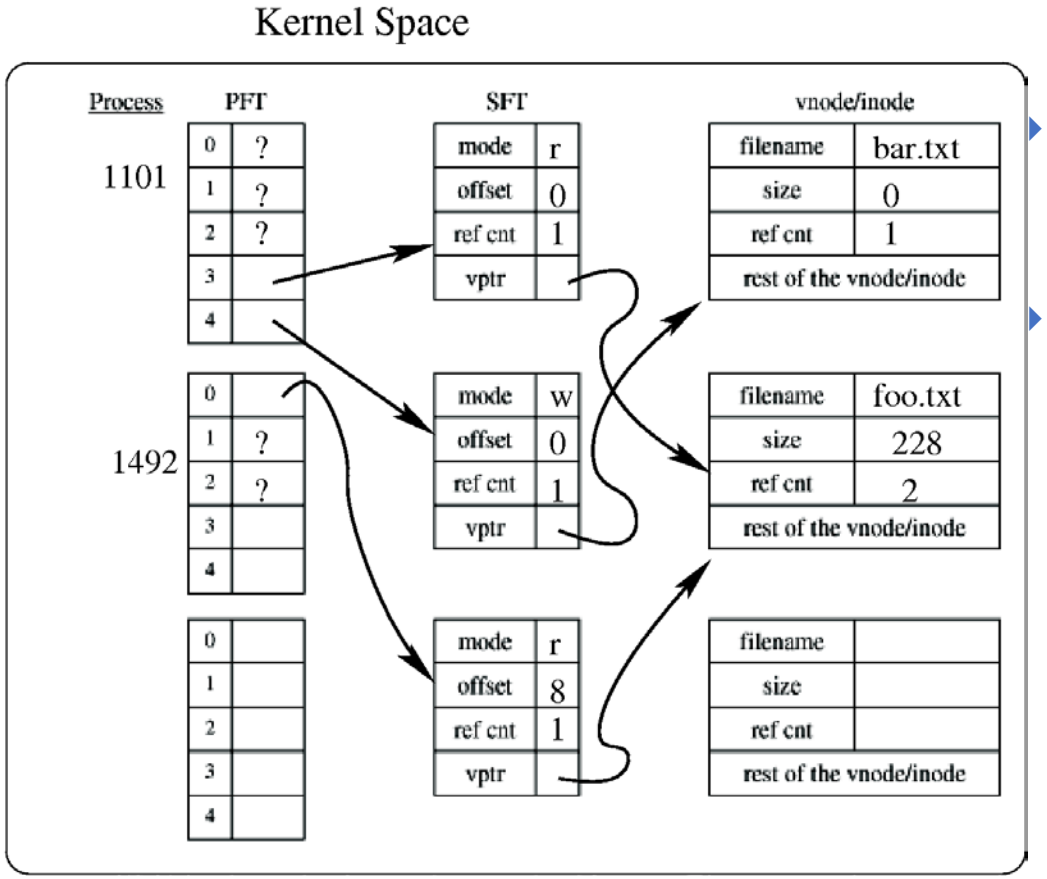
Податочни структури на датотечен систем

- Јадрени (in-mem) структури
  - Global (system wide) open file table (SFT)
  - Per-process open file table (PFT)
  - Free inode list
  - File buffer cache: Cached disk blocks
  - Inode cache
  - Name cache
- Структури на диск
  - Boot control block (per volume)
  - Superblock(per volume): File system format inf
  - Directory structure (per file system)
  - File: Collection of blocks/bytes
  - File descriptor (per file) (inode): File metadata
  - Directory: Special kind of file
  - Free block/inode maps



Податочни структури на датотечниот систем

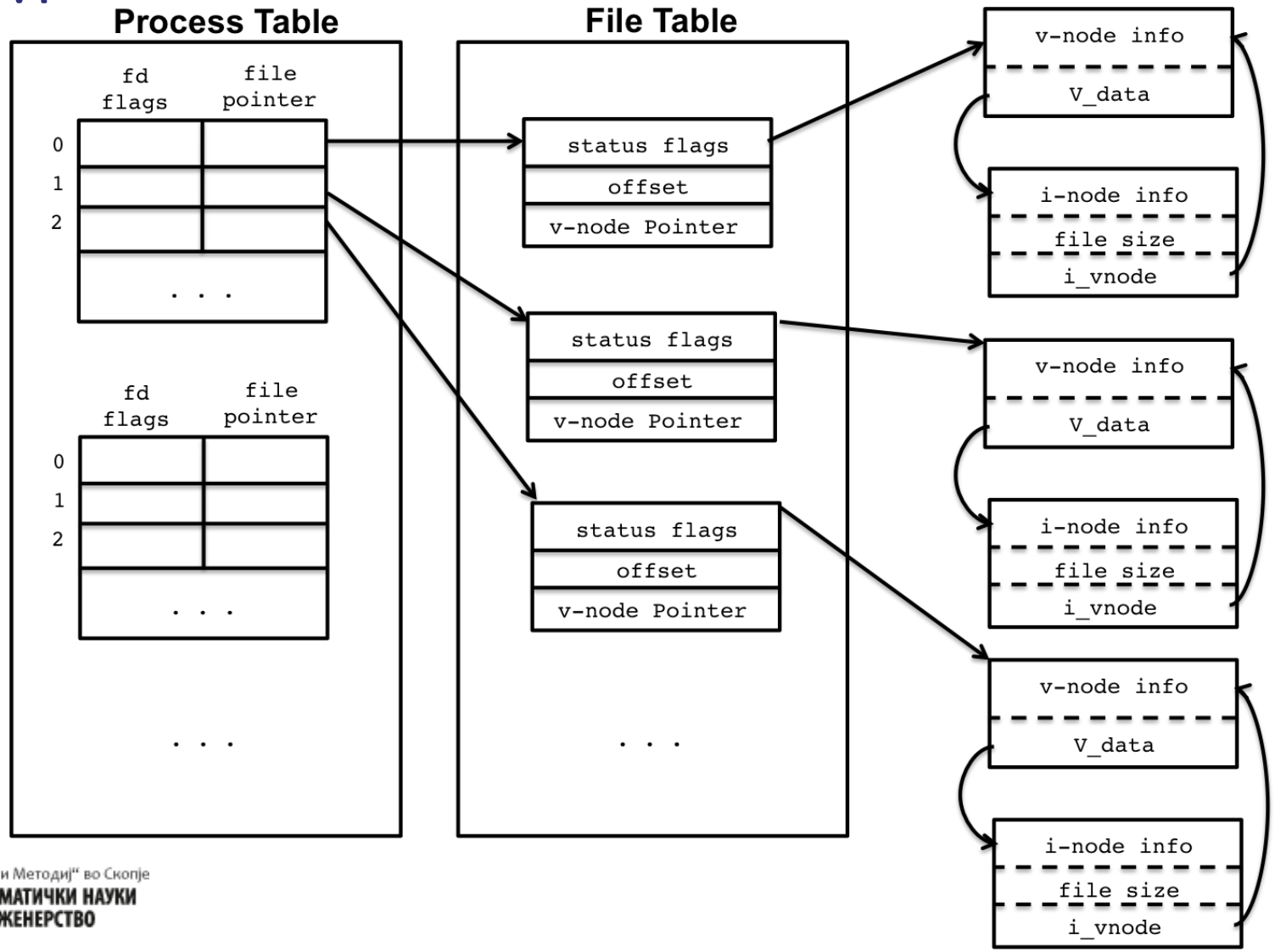




- PFT – Process File Table
  - Листа од покажувачи кон ставки од SFT за датотеки отворени од процесот
- SFT – System File Table
  - Листа од структури со информации за отворени датотеки од процесите
  - При секој повик на open се креира нова ставка
  - Содржи покажувач кон структура со информации за датотеката (на сликата v-node, виртуелен i-node)

https://cseweb.ucsd.edu/classes/sp16/cse120-a/applications/ln/lecture15.html
https://www.usna.edu/Users/cs/wcbrown/courses/IC221/classes/L09/Class.html

Податочни структури на датотечниот систем



Главни податочни структури во меморија

- Табела на отворени датотеки (SFT)
  - Ја делат сите процеси со отворена датотека
  - Секоја ставка содржи
    - Тековна позиција во датотеката (“seek/offset” покажувач)
    - Режим на пристап (read, write, read-write)
    - Покажувач до inode/vnode на датотеката
    - Број на отворени датотеки (најчесто е 1)
    - Локација на блоковите на датотеката во кеш баферот на датотеката (подолу)
  - За секој процес има посебна ставка, дури и ако ист процес два пати отвори датотека
    - Во секоја ставка има различна позиција за seek/offset, режим на пристап)
    - Се зголемува бројот на отворени датотеки само при fork() бидејќи детето ги наследува сите отворени датотеки, па наместо нова ставка во SFT, за секоја отворена датотека на детето, се зголемува бројот на отворени датотеки во SFT ставката на родителот

Главни податочни структури во меморија

- Датотечна табела по процес (PFT): приватна за секој процес
  - Покажувач до влез во глобалната табела на отворени датотеки
  - Првите 3 ставки се резервирани за стандарден влез, излез и грешка
- File buffer cache: кеш на датотечни блокови
  - Индексиран по file-blocknum парови (hash структура)
  - Се користи да се намали ефективното време на достап на диск операциите
  - Може да содржи блокови од датотеки на корисници, директориуми, метаподатоци на датотечниот систем (inodes)

Податочни структури во меморија

- Чекори кога процес повикува системски повик open() за некоја датотека
  - Се пребарува i-node за датотеката низ дрвото на датотечниот систем според името на датотеката
  - Се пребарува дали i-node за датотеката се наоѓа во RAM меморијата (табела на индексни јазли). Ако не се наоѓа во RAM, оди на чекор (A), во спротивно, оди на (B)
  - [A] Се чита i-node од дискот и се внесува во нов влез во табелата на индексни јазли за отворените датотеки.
  - [B] Се креира нов влез во системската табела на отворени датотеки (SFT) во кој се внесува покажувач кон влезот во табелата индексни јазли

Податочни структури во меморија

- Се креира нов влез во табелата на отворени датотеки (PFT) за процесот (пристап има само процесот) која содржи покажувач кон влезот во системската табела на отворени датотеки (SFT).
- Индексот на влезот (file descriptor) во оваа табела се враќа како резултат на системскиот повик open()
- **Ако една датотека е отворена n пати од ист или различни процеси, се креираат n влезови во системската табела на отворени датотеки.**

Податочни структури во меморија

- При повик на fork()
  - Се прави копија на табелата на отворени датотеки на родителот и се запишува во PCB на детето
  - За секоја отворена датотека на родителот, се зголемува бројот на референци во системската табела на отворени датотеки!
    - Бидејќи и детето ги наследува отворените датотеки

Пример за табели за управување со датотеки

Датотеката **/root/output.txt** има i-node со број 9015, а **/root/messages.txt** има i-node со број 2016.
Процесот со PID=21 го извршува следниот код:

```
int fd1=open("/root/output.txt", O_WRONLY); //open for writing
write(fd1,buffer1,100); //write 100 bytes from buffer1
int fd2=open("/root/messages.txt",O_RDONLY);
read(fd2, buffer2, 200); //read 200 bytes in buffer2
```

Процесот со PID=22 го извршува следниот код:

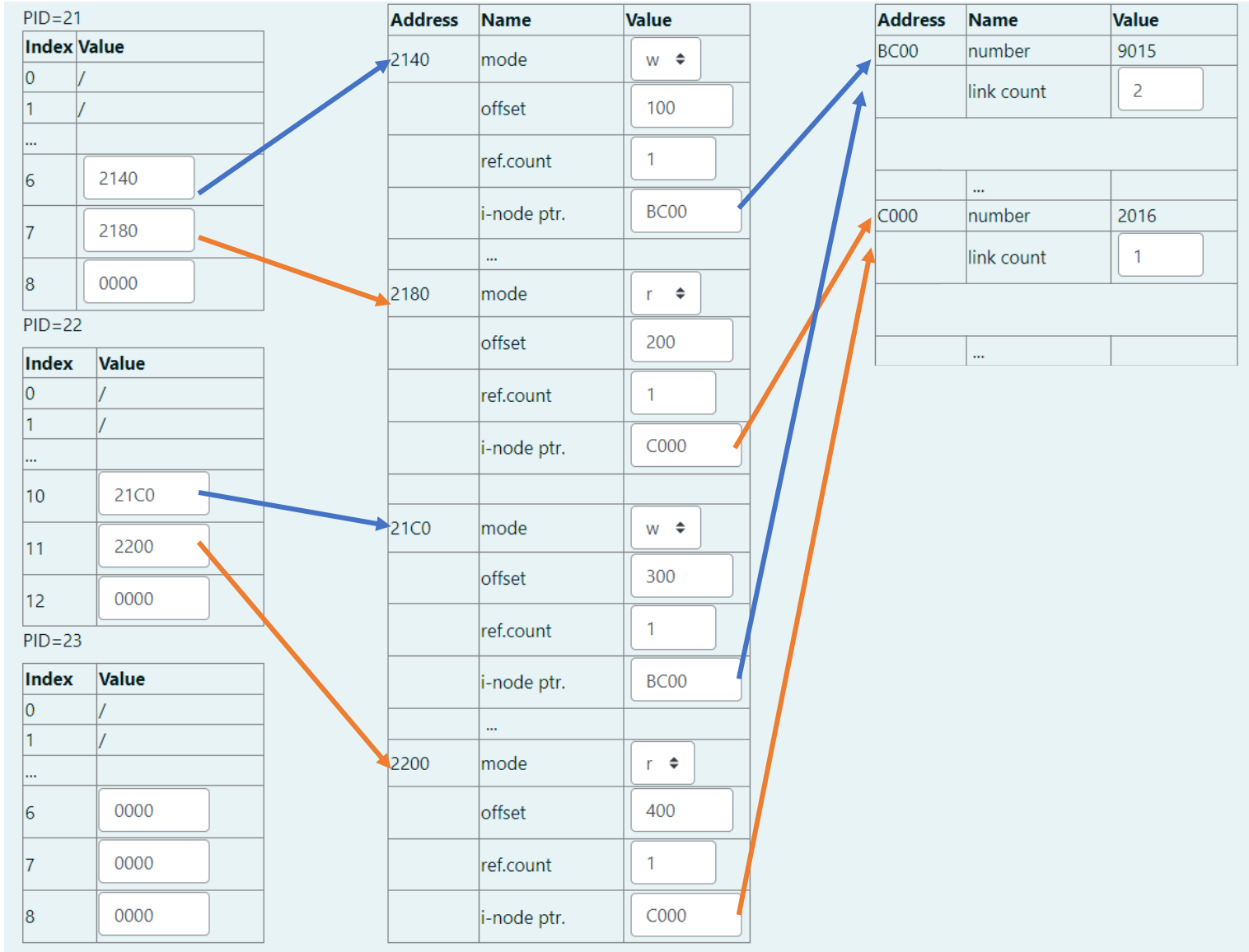
```
int fd3=open("/root/output.txt",O_WRONLY); //open for writing
write(fd3,buffer3,300); //write 300 bytes from buffer3
int fd4=open("/root/messages.txt",O_RDONLY);
read(fd4, buffer4, 400); //read 400 bytes in buffer4
```

Процесот со PID=23 го извршува следниот код:

```
ln /root/output.txt /home/output.txt //create hard link
```

Пополнете ја содржината на дадените табели по извршување на кодот од процесите. Ако некои од полињата не се потребни, оставете ги празни.

Пример за табели за управување со датотеки



Вредноста на fd1, fd2, fd3 и fd4 ќе биде

6 , 7 , 10 и 11

Главни податочни структури во меморија

- Кеш на имиња: кеш на претходни пребарувања на имиња (lookup)
  - Индексиран по целите имиња на датотеките (hash структура)
  - Се користи да се елиминираат пребарувањата на директориумите за дадено име
- Битмапа на слободни блокови
  - Кои блокови на дискот се слободни
- Битмапа на слободни inode
  - Кои индексни јазли на дискот се слободни

Главни податочни структури на диск

- File descriptors (“inodes”)
  - Бројач на врски
  - Сигурносни атрибути: UID, GID, …
  - Големина
  - Време на пристап/модификација
  - “Покажувачи” до блоковите
- Directory file: array of…
  - Име на датотека (фиксна или променлива големина)
  - Број на Inode
  - Должина на влезот во директориумот
- Битмапа на слободни блокови
- Битмапа на слободни inode
- Superblock - мета-податоци кои го опишуваат дискот
  - Физички карактеристики: size of disk, size of blocks, …
  - Локација на слободен простор и битмапи на слободни inode
  - Локација на inodes
  - Се чува во повеќе копии - редунданција

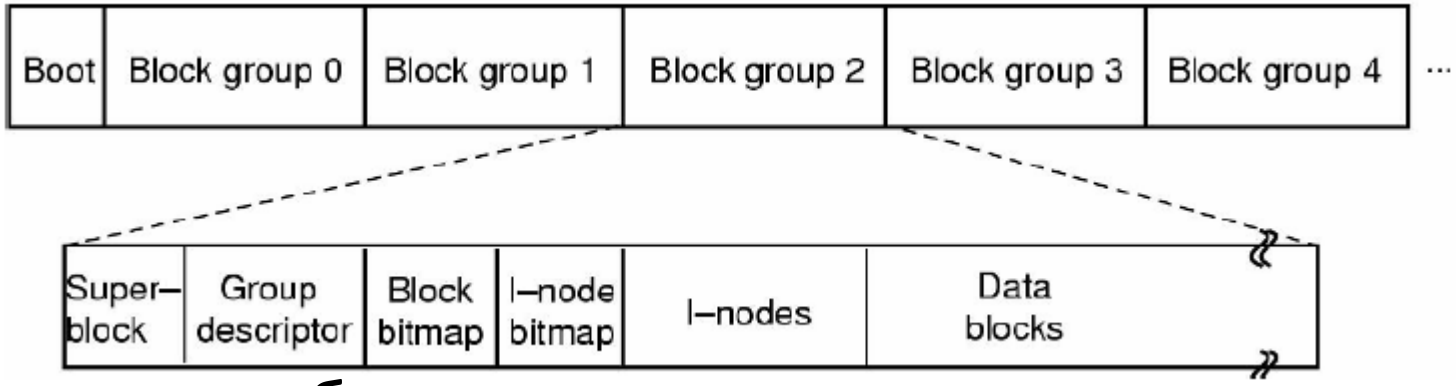
File descriptor (inode):

ulong links;
uid_t uid;
gid_t gid;
ulong size;
time_t access_time;
time_t modified_time;
addr_t blocklist…;

Directory file:

Filename	inode#
Filename	inode#
REALLYLONGFILENAME	
inode#	Filename
inode#	Short inode#

Case study: Ext2

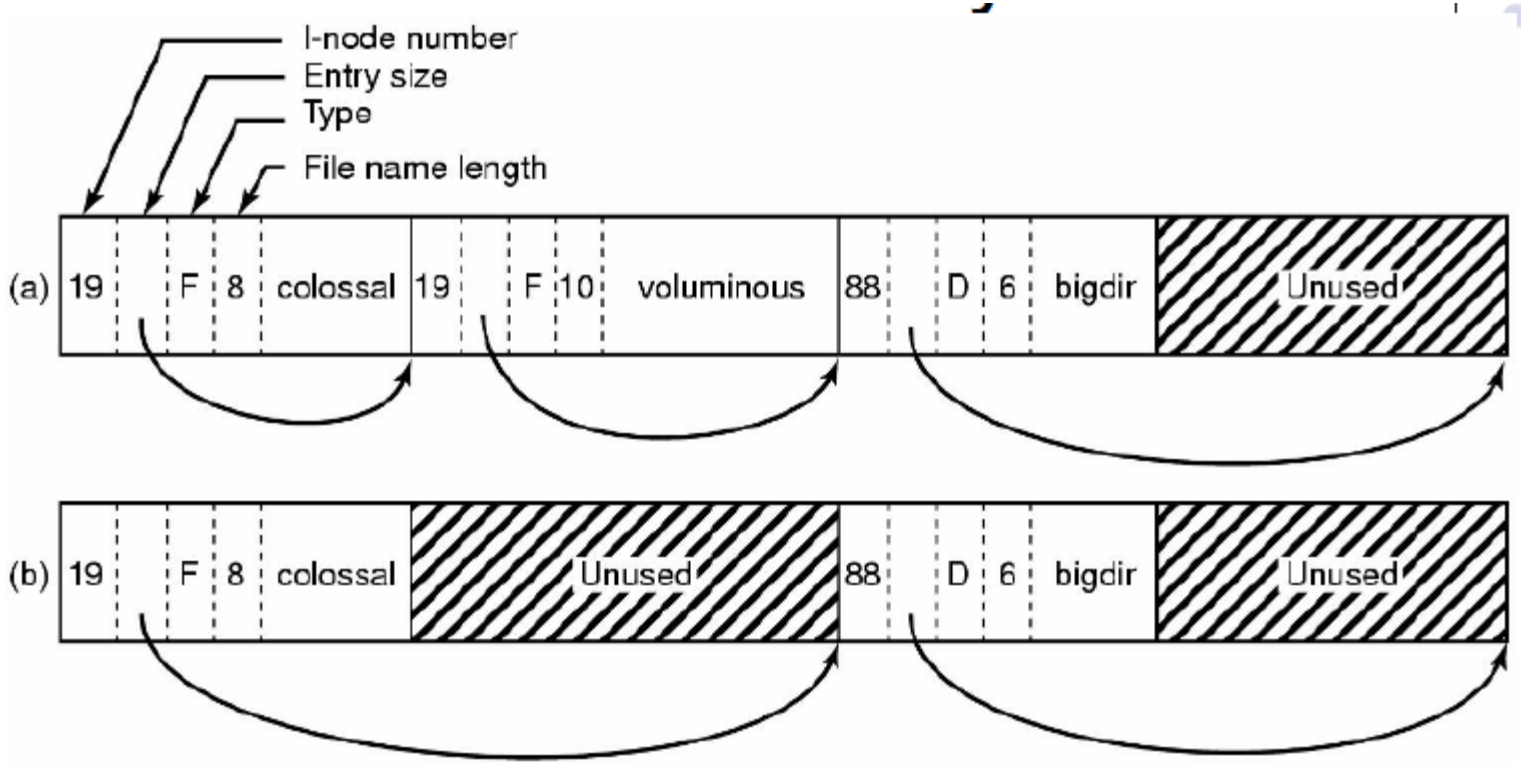


- Диск партицијата се дели на групи од блокови
- Секоја група содржи:
  - Супер блок (колку блокови и индексни јазли постојат, колкава е големината на блокови итн.)
  - Group descriptor (локација на битмапите, бројот на слободни блокови, бројот на индексни јазли и именици во групата)
  - Битмапа за слободни блокови и Битмапа за индексни јазли
  - Индексни јазли (128B секој, опишува една датотека, stat, 12 директни и 3 индиректни адреси)

Главни On-Disk Податочни структури

- **Case study: Ext2**
  - I-nodes за имениците се расфрлани низ различни групи од блокови на дискот
  - Датотеките се сместуваат во групата каде се наоѓаа нивниот родител-именик
  - Податочните блокови се сместуваат во истата група каде се наоѓа и соодветниот I-node
  - Се користи бит-мапа за слободните блокови
  - Се преалоцираат 8 блокови за секоја датотека – минимизација на фрагментација

Датотека на именик (директориум) во Linux



inode number: 4 bytes (assuming a 32-bit system)

record length: 2 bytes (length of this directory entry)

name length: 1 byte (length of the filename)

file type: 1 byte (type of file, e.g., regular file, directory)

filename: Variable length (actual name of the file)

Пребарување на датотеки

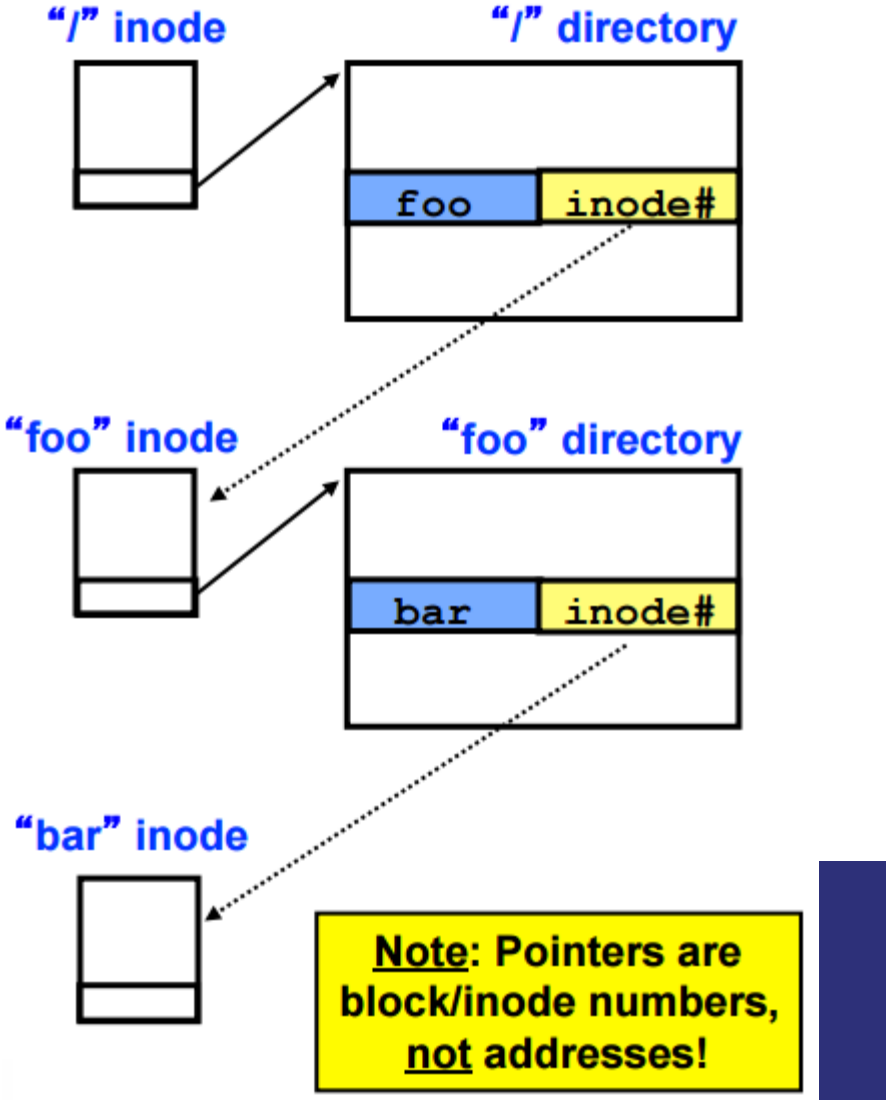
- Линеарно пребарување
- За долги директориум линеарното пребарување е многу бавно
  - Кеширање на пребарувањата
    - Работи добро кога имаме мал број датотеки кои постојано се побаруваат
  - Користење на HASH табела (комплексна администрација)
    - Се исплати само кога имаме илјадници датотеки

Пребарување именици

- Како да лоцираме дескриптор за “/foo/bar”?
  - Подели го името на компоненти (пр., “/” , “foo”, и “bar”).
  - Рекурзивно намалувај ја хиерархијата на директориумот и во секој чекор:
    - Прочитај датотечен дескриптор (i-node) за “следната” директориумска датотека
    - Искористи ги податоците од i-node да ја лоцираш и преземеш (load) содржината на директориумот (блок од диск)
    - Скенирај ја содржината на блокот за бараното име на датотеката или следната компонента од името
    - Ако имаш совпаѓање -> земи го бројот на i-node (name, i-node)
    - Ако немаш совпаѓање -> lookup failure
- Како да се забрза овој процес?
  - Кеш за имиња (се кешира името на датотеката и нејзиниот i-node)
  - Ако се најде совпаѓање на /foo/bar во кешот, се избегнува процесот на пребарување низ секој именик од патеката

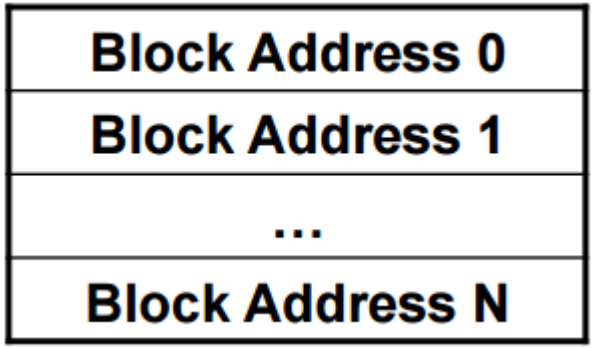
Наоѓање на Inode на датотеката на диск

- Најди inode за/foo/bar:
  1. Најди го inode за “/”
     - Секогаш на позната локација
  2. Прочитај го именикот “/” во меморија
  3. Најди ставка “foo” во именикот
     - Ако не успееш, врати грешка
  4. Прочитај го inode за “foo” од диск
  5. Провери привилегии за пристап
     - Ако нема привилегии, врати грешка
  6. Прочитај ги блоковите за именикот “foo”
  7. Најди ставка “bar”
     - Ако не успееш, врати грешка
  8. Прочитај го inode за “bar” од диск
  9. Провери привилегии за пристап
     - Ако нема привилегии, врати грешка



Наоѓање на блоковите на датотеката на диск

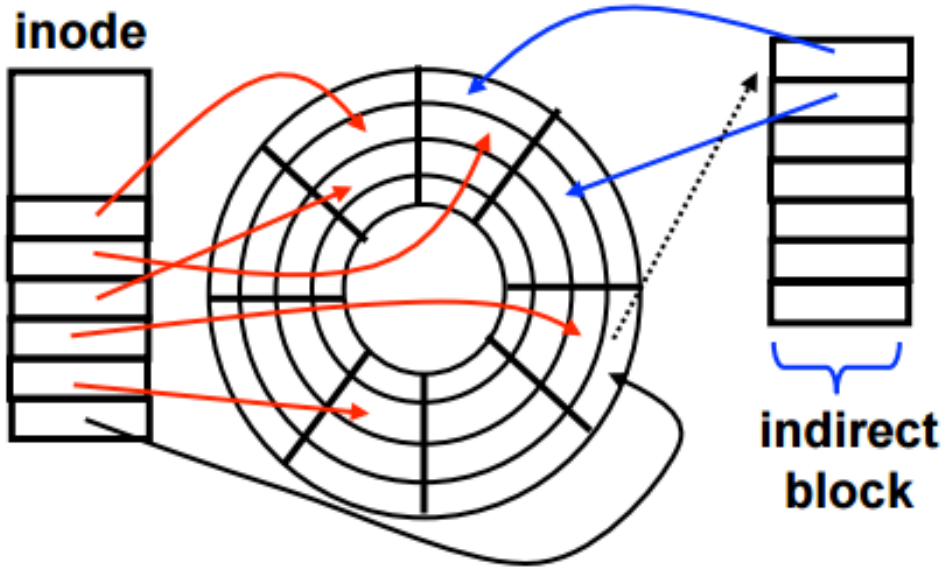
- Концептуално, inode треба да ја содржи табелата:
  - Една ставка за секој блок од датотеката
  - Секоја ставка ја содржи физичката адреса на блокот на диск
    - (пр., плоча 3, цилиндар 1, сектор 26)
  - За да се прочита податок со offset X,
    - Прочитај ја ставката (блокот) X / block_size



- Прашање “Како физички да се имплементира табелава?
  - Најголем број од датотеките се мали
  - Најголем дел од дискот е зафатен од (релативно малку) големи датотеки
  - Треба ефикасно да се поддржи и секвенционалниот и случајниот достап
  - Да има едноставни inode lookup и механизми на управување

Inodes (блоковите)

- Inode содржи:
  - Низа со фиксна големина од директни блокови
  - Мала низа од индиректни блокови
  - (опционално) двојни/тројни индиректни блокови
- Индиректни блокови:
  - Блокови со адреси од блокови
  - Двоен индиректен блок: блок со адреси на индиректни блокови



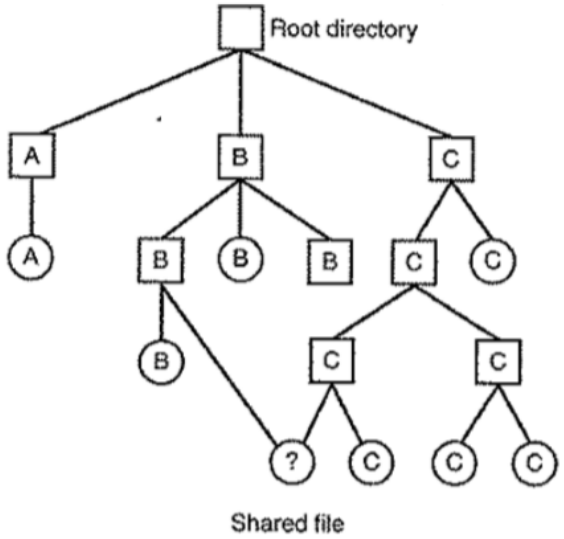
Inodes

- Добри страни
  - Едноставен offset -> блоковско пресметување за секвенцијален или случаен достап
  - Овозможува инкрементално растење/ намалување
  - Фиксна големина (мали) inodes
  - Многу брз пристап до (најчесто) малите датотеки
- Лоши страни
  - Пристапот користејќи индиректни блокови го оптоварува случајниот пристап до големите датотеки
  - Блоковите може да се расфрлени по целиот диск -> повеќе seeks

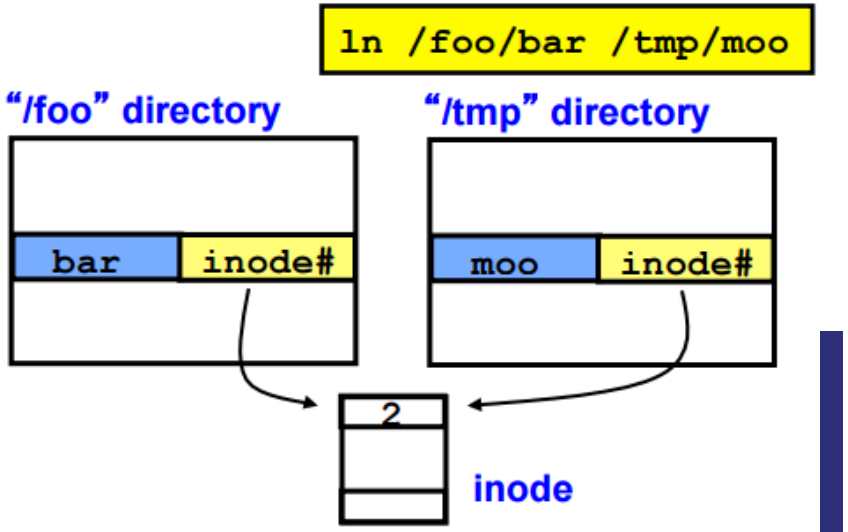
Inodes

- Пример Ext2 File System
  - Inode содржи 12 директни блок адреси
  - Inode содржи 1 адреса на индиректен блок
  - Inode содржи 1 адреса на двоен индиректен блок
  - Inode содржи 1 адреса на троен индиректен блок
- Ако адресите на блоковите се 4-bytes и блоковите се 1024-bytes, која е максималната големина на датотека?
  - Број на блок адреси по блок = 1024/4 = 256
  - Број на блокови мапирани со директни блокови -> 12
  - Број на блокови мапирани со индиректен блок -> 256
  - Број на блокови мапирани со двоен индиректен блок -> 256^2 = 65536
  - Број на блокови мапирани со троен индиректен блок -> 256^3 = 16777216
  - Максимална големина на датотека: (12 + 256 + 65536 + 16777216) * 1024 = 16,1 GB

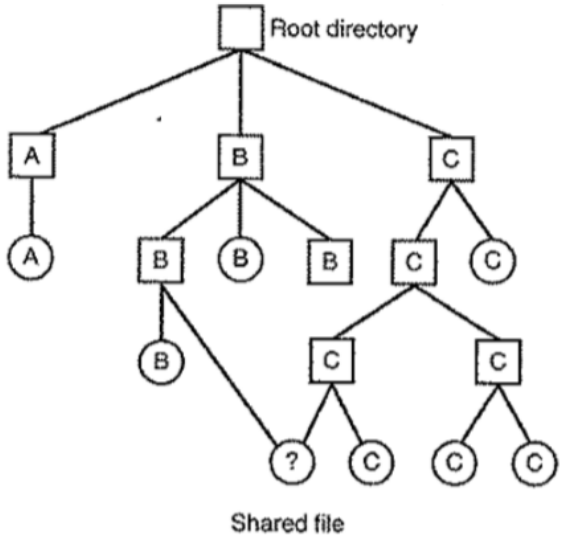
Врски до споделени датотеки



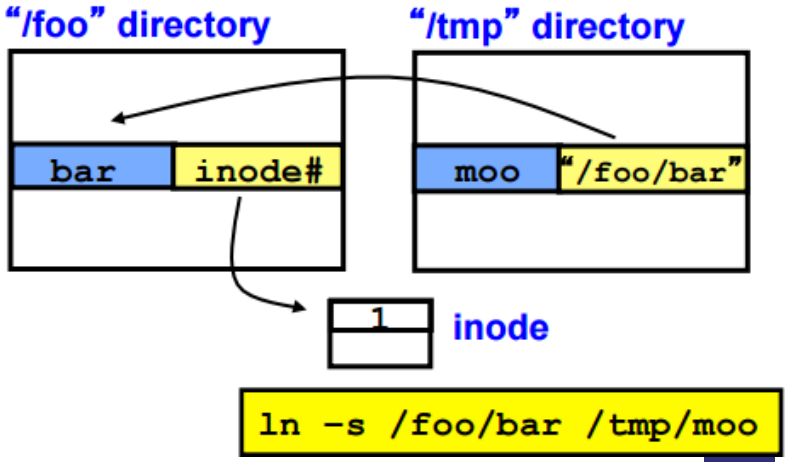
- Врските ни овозможуваат да имаме повеќе имиња до иста датотека
  - Два типа: тврда и мека
- Тврда врска:
  - За секоја врска се креира влез во именикот со име на врската и иста вредност на i-node
  - Два (или повеќе) влеза покажуваат на ист i-node
  - Кога се креира тврда врска се зголемува бројот на врски во i-node за датотеката
  - Кога се брише датотека или тврда врска
    - Се намалува бројот на врски во i-node
    - Датотеката се брише ако бројот на врски во i-node е 0 (кога и последната врска е избришана)



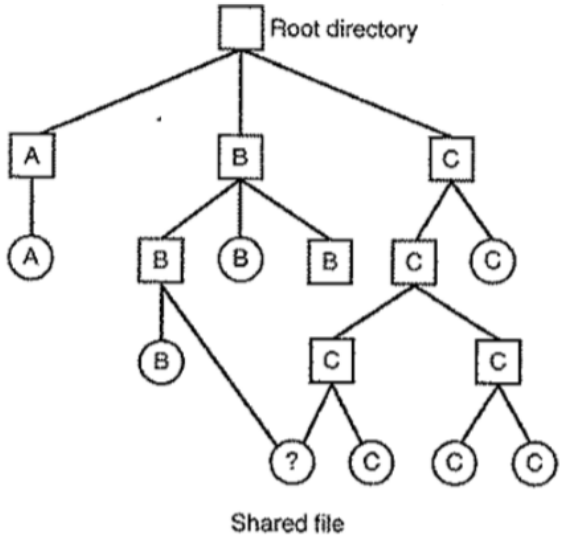
Врски до споделени датотеки



- Симболичка/мека (Soft) врска:
  - За секоја врска се креира влез во именикот со име на врската и симболички “покажувач” кон датотеката
  - Влезот во именикот **специјално се означува** (за да се разликува од обичните датотеки)
  - Има само една “реална” врска до датотеката (пар име – i-node)
  - При додавање на нова врска или бришење на постоечка **не се менува бројот на врски во i-node**
  - Датотеката ќе се избрише кога креаторот ќе ја избрише
    - Постоечките меки врски кон избришаната датотека остануваат во системот, но покажуваат кон невалидна локација (broken links)
    - Пристапот до датотека преку врска фрла грешка



Врски до споделени датотеки



- Приказ на содржина на именик, вклучувајќи ги и врските

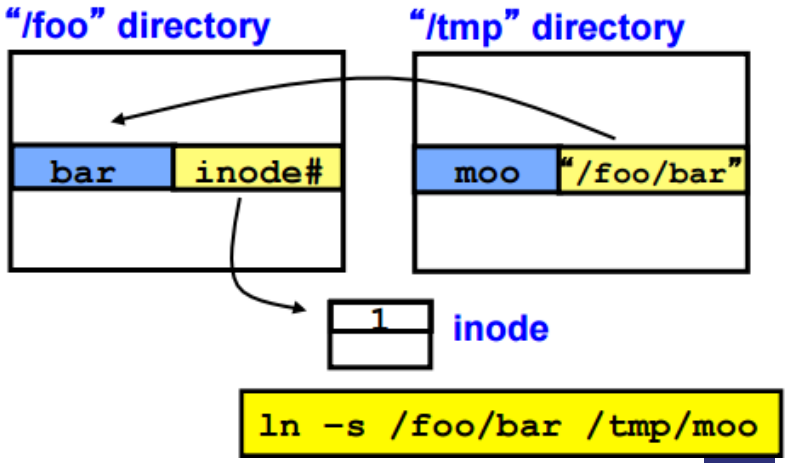
```
ls -l
```

- Приказ на содржина на именик, вклучувајќи го и бројот на i-node

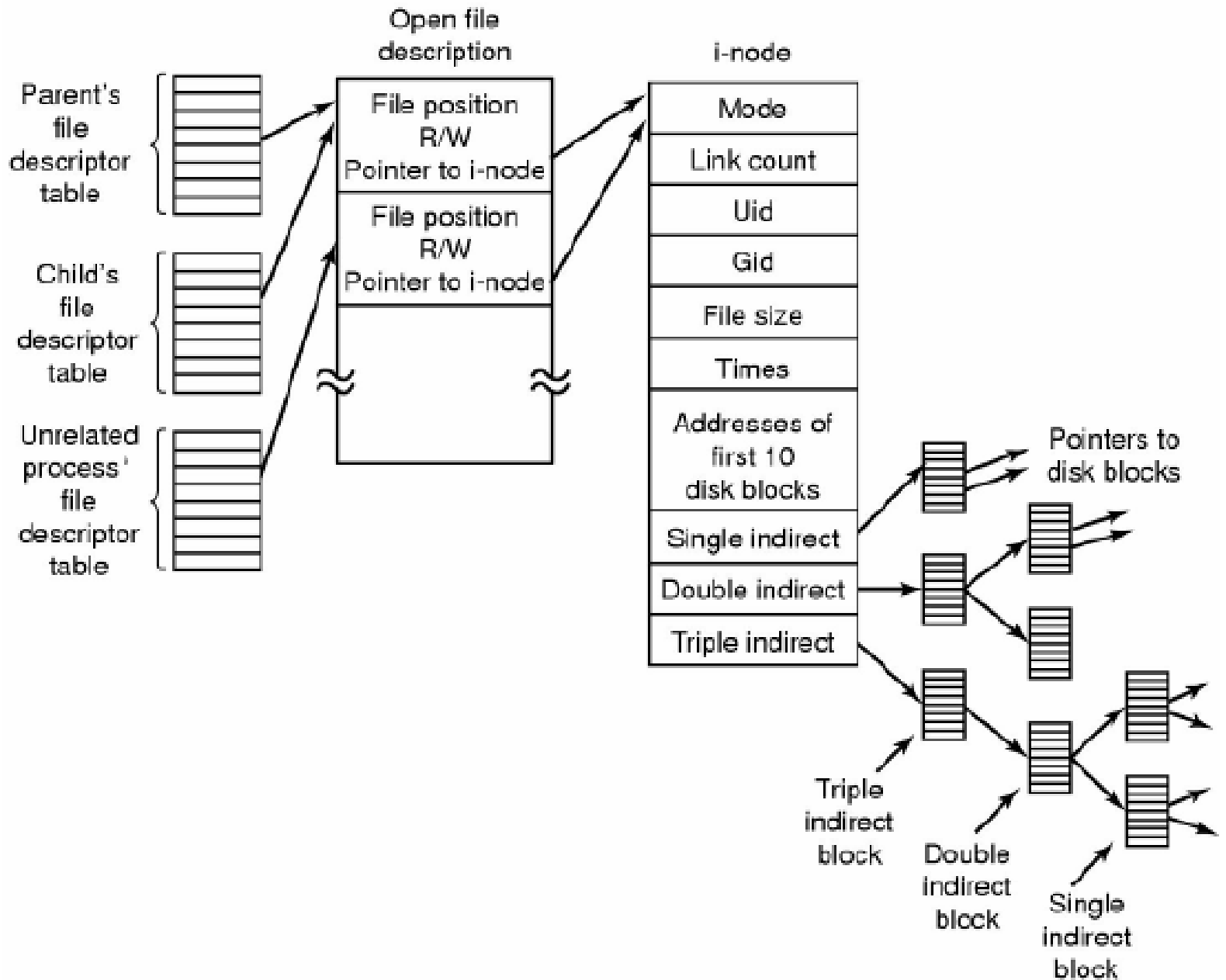
```
ls -i
```

- За примерот на сликата

```
> ls -l /tmp
> lrwxrwxrwx  1 user group moo->/foo/bar
```



Ext2 – Големата слика



Journaling датотечен систем

- Се чува дневник (log) која акција треба да ја направи оперативниот систем
- Операции при отстранување на датотека
  - Се отстранува датотеката од именикот (влезот име i-node)
  - Се ослободува i-node во базенот на слободни i-nodes
  - Сите блокови на дискот се враќаат во базенот на слободни блокови
- Во присуство на падови, редоследот е битен!

Journaling датотечен систем

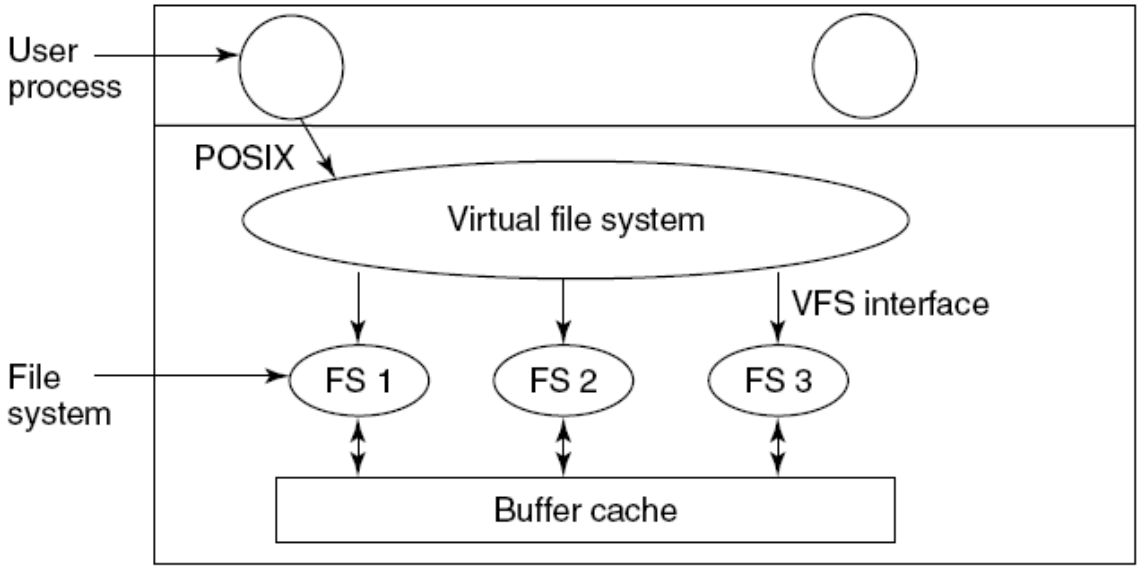
- Journaling датотечниот систем ги запишува трите акции во дневник, пред да ги изврши
  - Ако системот падне, акциите се повторуваат по неговото подигање
- Операциите во дневникот се идемпотентни (можат да се извршат повеќе пати без да се наруши конзистентноста на податоците)
- Се воведува концепт на атомична трансакција
  - Група акции ограничени со почетна и крајна операција
  - Мора цела група наеднаш да се изврши
- NTFS, ReiserFS, ext3, ext4 користат journaling

Виртуелни датотечни системи (1)

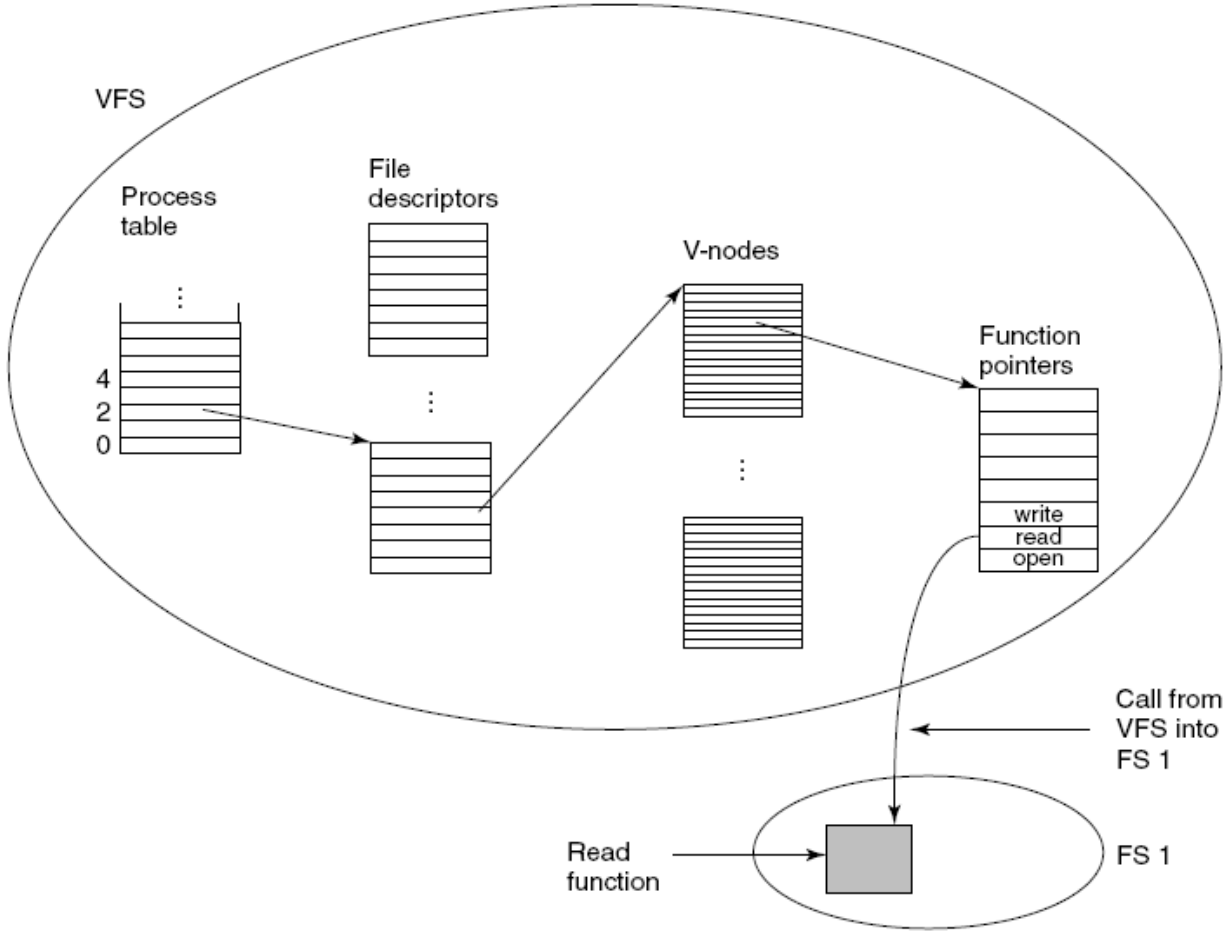
- Интеграција на различни датотечни системи
  - Во еден систем можно е истовремено користење на повеќе датотечни системи
  - USB и HDD имаат различни датотечни системи
- Корисникот не е потребно да е запознаен каков датотечен систем користи
  - Сè што е потребно е било кој датотечен систем да обезбеди имплементација на основните системски повици за манипулација со датотеки
- Идеја: апстракција на датотечните системи со основи системски повици од ОС и поставување на кодот (драјверот) во посебен слој кој е задолжен за соодветната имплементација на вистинскиот датотечен систем

Виртуелни датотечни системи (2)

- FS1, FS2, …
  - Имплементираат потребен интерфејс од VFS
  - Задолжени за менаџирање на конкретните партиции каде датотеките се организирани на начин специфичен за соодветниот ДС
  - NTFS, FAT, FAT32, …
  - EXT2, EXT3, EXT4, …
  - Може и мрежен ДС
    - NFSv3, NFSv4,
    - Windows sharing (SMB)



Виртуелни датотечни системи (3)



Слика 4-19. Упростен приказ на податочните структури и кодот кој го користи VFS и реалниот FS

Управување со датотечни системи и оптимизација

Големина на алокациска единица

- Скоро сите датотечни системи ги делат датотеките на делчиња со фиксна големина и тие не мора да се последователно поставени
- Големината на алокациската единица е важна
  - Ако е голема, малите датотеки зафаќаат голем простор
  - Ако е мала, датотеките ќе се протегаат на повеќе блокови, што значи повеќе seek операции и ротациони задоцнувања
- Типични големини
  - 512B, 1KB, 2KB, 4KB, 8KB
- Најчесто, најголемиот број на датотеки се од редот на неколку KB

Големина на алокациска единица

- Кој ја определува големината на блокот?
  - Производителот на дискот
    - Физичка големина на самиот диск
  - Датотечниот систем
    - При форматирање на дискот, може да се постави различна големина на блокот (мултипл од полседователни физички блокови)
    - Секое читање/пишување на блок кое стигнува до драјверот се конвертира во читање/пишување на последователен број на физички блокови
  - Оперативниот систем
    - Го поставува корисникот
    - Секое барање за работа со блок, оперативниот систем го пресликува во работа со поставената големина

Управување со дисков простор (1)

- Со големина на блок од 1KB само од 30% до 50% ги собира во еден блок
- Со големина од 4KB процентот е 60% до 70%
  - 93% од дисковите блокови се искористени за 10% од најголемите фајлови

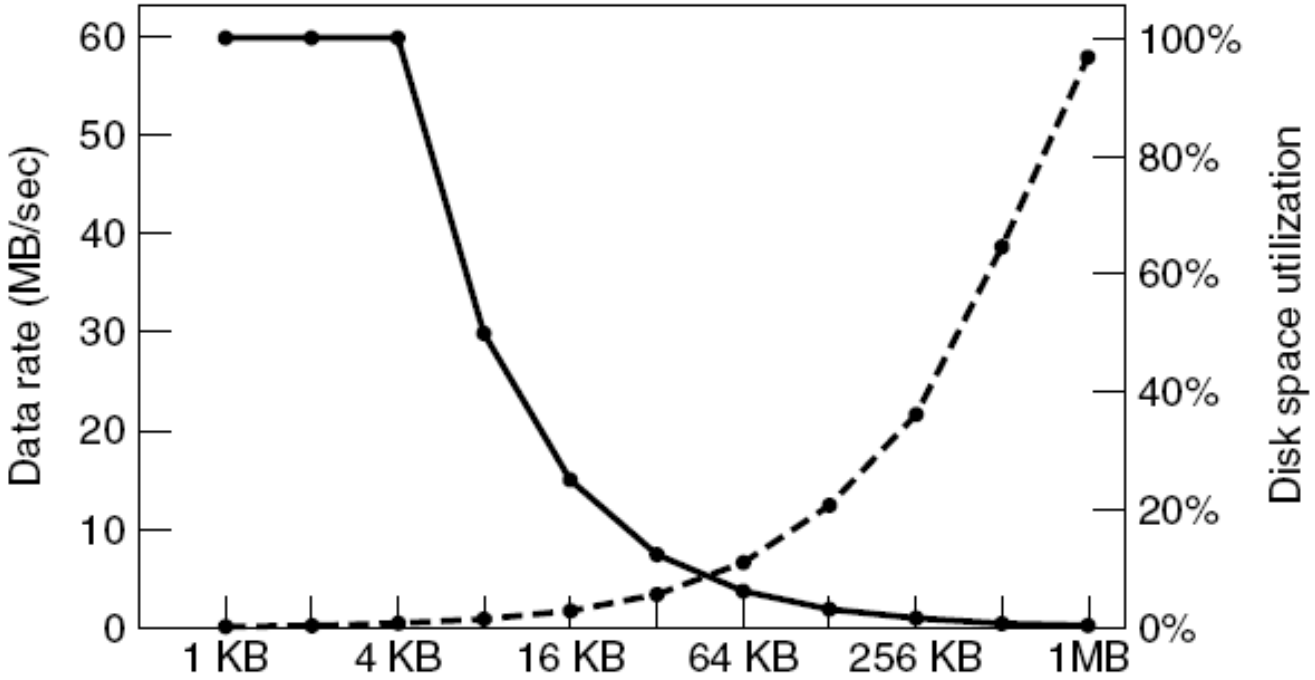
Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Управување со дисков простор (2)

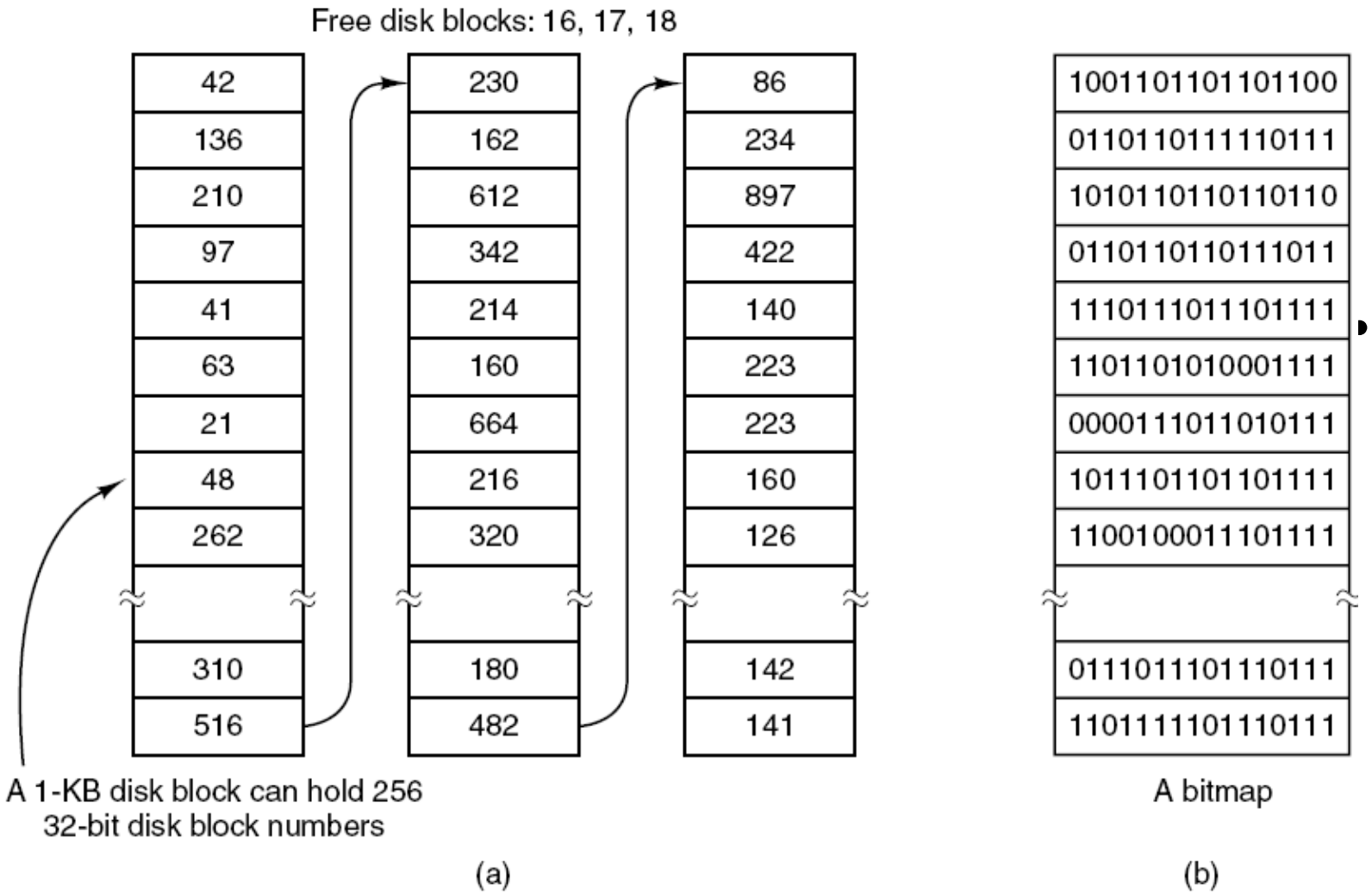
Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Управување со дисков простор (3)

- Полна линија дава ефикасност на искористеност на дисковиот простор
  - согласно претходните табели
- Испрекинатата линија дава податочна рата на диск.
  - поголема ако одеднаш читаме големи блокови



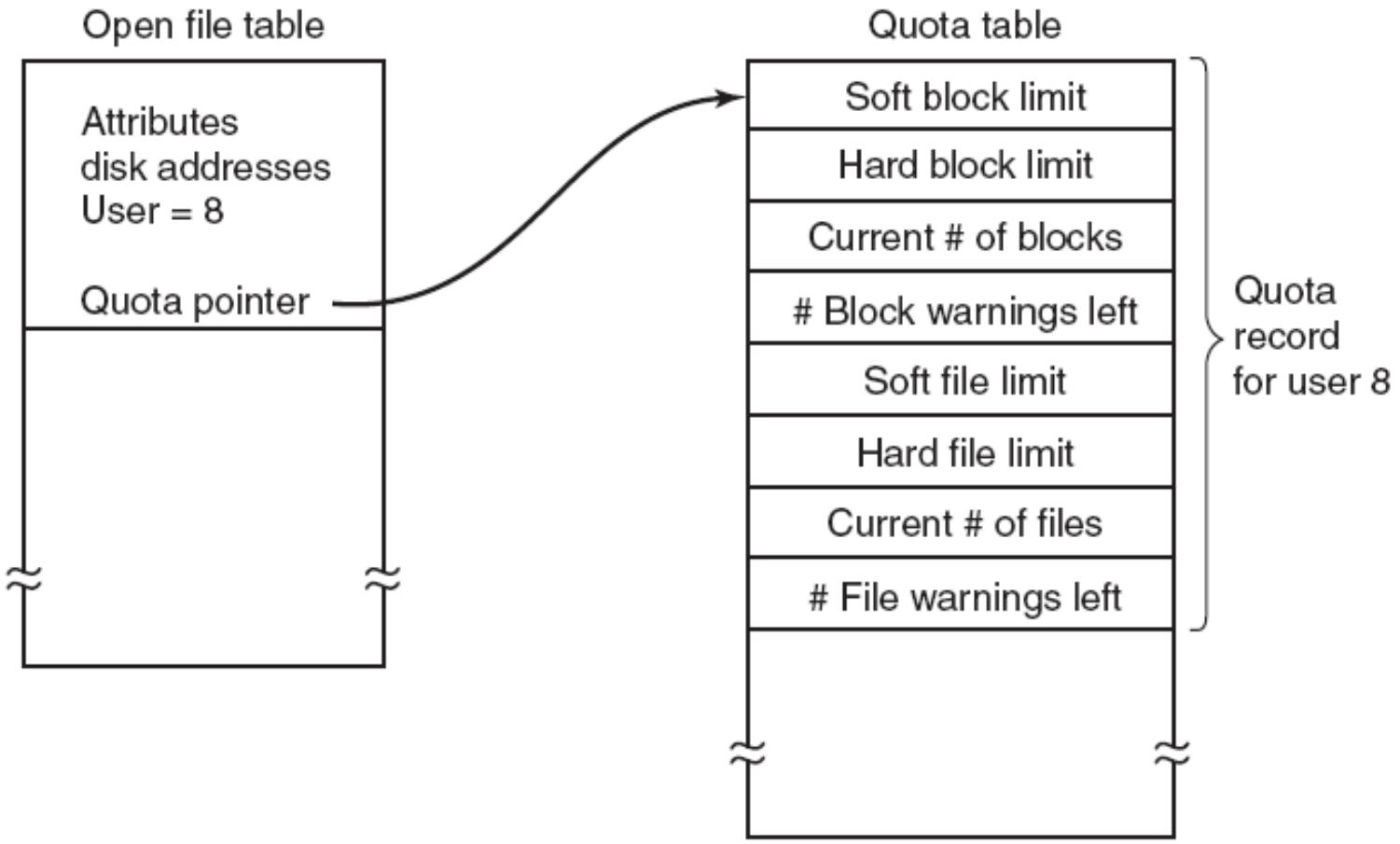
Евиденција за слободни блокови (1)



- Чување на слободни блокови во листа
  - За 500 GB диск, 488 милиони блокови, 32 битен број на блок, потребни се 1.9 милиони блока за да се чуваат слободните блокови
- Чување во бит-мапа
  - Потребни се 60000 блока
  - Ext4 и NTFS

Диск квоти

- Покрај слободниот простор, некои датотечни системи (ext4, ntfs) овозможуваат ограничување (квота) на количината на податоците по корисник



Backup на датотечниот систем (1)

- Backup се врши на магнетна лента (евтин и компактен медиум) во случај на:
  - Оporавување од катастрофа
    - корупција на дел од ДС поради баг или хардверски проблем
  - Оporавување од глупост
    - несакано бришење или пребришување на податоци
- Backup најчесто се врши само на специфични датотеки и именици
  - Но може и на цели, па дури и системски партиции

Backup на датотечниот систем (2)

- Целосен или инкрементален backup
- Компресија на податоци пред да се изврши снимање на backup на ленти
- Можни се две стратегии за backup на цели партиции со датотечен систем:
  - Физички dump
    - празни блокови, оштетени блокови
    - едноставен
    - нема можност за селективен и инкрементален backup
  - Логички dump

Проблеми со логички dump

- Листата на слободни блокови не е датотека
  - мора да биде реконструирана од постоечките зафатени
- Датотека се наоѓа како линк во два или повеќе именици
  - При restore мора само еднаш да ја има датоката и двата именици да покажуваат на неа
- Специјалните датотеки, на пр. цевки, не треба да се зачувуваат

Конзистентност на датотечен систем

- Ако не се запишат сите блокови кои треба да се запишат (поради пад на системот), се добива неконзистентна состојба
  - Посебно е важно за i-nodes, директориумски блокови и оние кои содржат листа на слободни блокови
- Услужните програми за обезбедување конзистенција можат да проверуваат блокови или датотеки

Конзистентност на датотечен систем (fsck и scandisk)

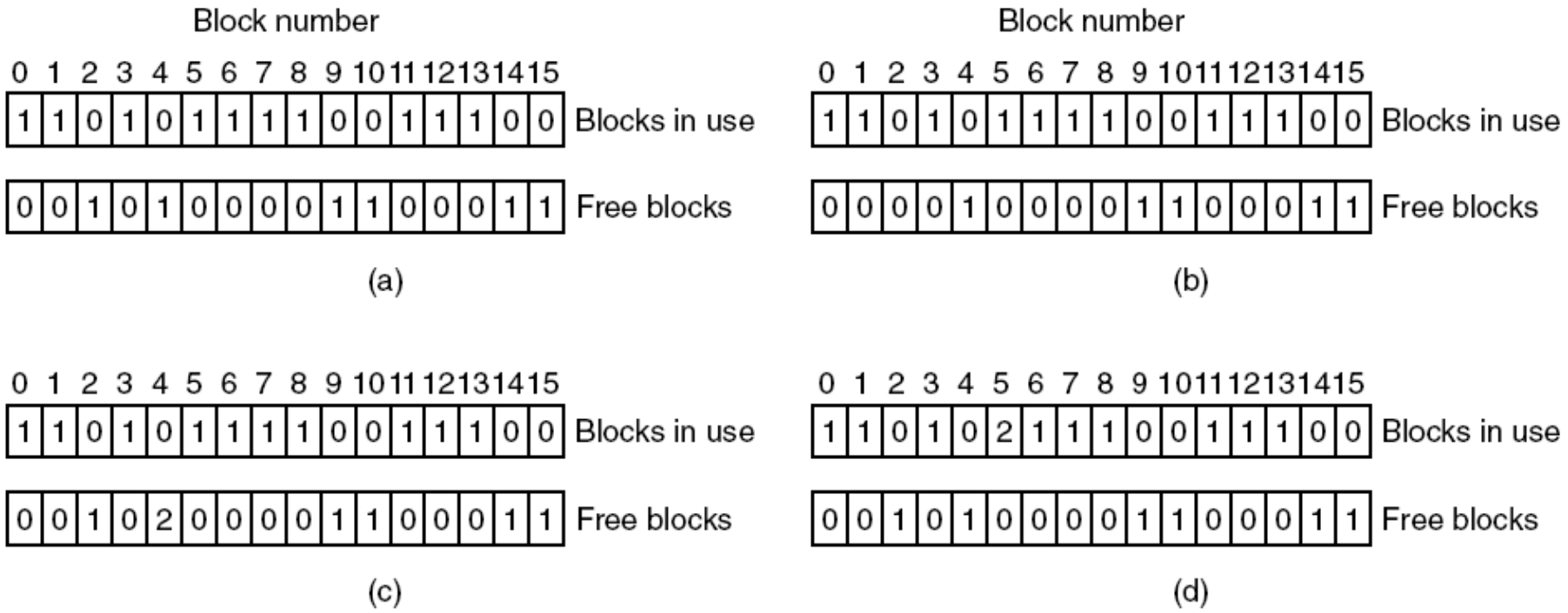


Figure 4-27. Состојби на датотечен систем. (a) Конзистентна. (b) Недостасува блок. (c) Дуплициран блок во листа на слободни (d) Дуплиран податочен блок.

Disk Buffer Кеш(1)

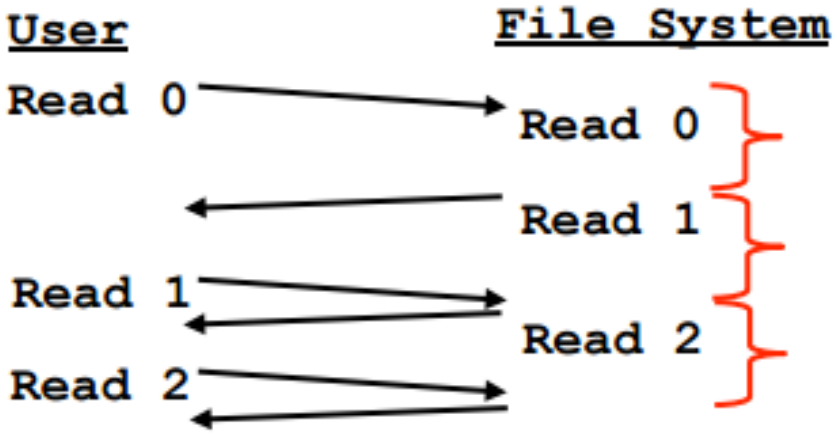
- Идеја: Чувај ги фреквентно користените блокови во меморијата на јадрото
- Процес чита од датотека:
  - Ако блоковите не се во кешот
    - Алоцирај место во диск бафер кешот
    - Започни процес на читање од диск
    - Блокирај го процесот додека диск операцијата не е комплетна
  - Копирај податоци од кешот во меморијата на процесот
  - Врати се на системскиот повик
- Најчесто процесот не го гледа баферот директно
- mmap() мапира страници од кешот во RAM на процесот

Disk Buffer Кеш(2)

- Процесот запишува во датотека:
  - Ако блоковите не се во кеш баферот
    - Алоцирај ги потребните страници
    - Започни читање од диск
    - Блокирај го процесот додека не заврши диск операцијата
  - Копирај ги податоците од RAM на процесот во кеш баферот
- Најчесто: операциите на запишување креираат страници кои треба да се запишат (dirty pages) во кешот, а потоа се враќа системскиот повик
  - Податоците се запишуваат на уредот во позадина

Prefetching

- Идеја: Прочитај ги потребните блокови од дискот пред вистинската потреба на корисникот
- Цел: Да се намали бројот на seeks видливи за корисникот
  - – Ако блокот е прочитан пред барањето -> хит во кеш баферот



- Проблем: кои блокови претходно да ги преземеме (prefetch)?
  - Лесно: Детектирај секвенцијален достап и преземи однапред N блока
  - Потешко: Детектирај периодичен/ предвидлив “случаен” достап

Пример за операција: Open /tmp/foo

- Open `/tmp/foo‘
  1. Види дали веќе го знаеме inode за:
     - /tmp/foo (goto `B')
     - /tmp (goto `A')
  2. Види дали root i-node е во i-node кеш (инаку прочитај го root i-node)
  3. Провери пермисии за корисникот за root именикот
  4. Одреди ја локацијата на блоковите кои го содржат root именикот
  5. Види дали секој од тие блокови е во buffer cache
  6. Преземи ги оние кои не се и постави ги во cache
  7. Пребарај го root именикот за влезот `tmp' и добиј го неговиот i-node број

Пример за операција: Open /tmp/foo

- Open `/tmp/foo‘
  1. [A] види дали i-node на /tmp е во i-node cache (инаку преземи го)
  2. Провери пермисии на корисникот за /tmp
  3. Одреди ја локацијата на блоковите кои го содржат /tmp именикот
  4. Види дали секој од блоковите е во buffer cache
  5. Преземи ги оние кои не се и постави ги во cache
  6. Пребарај го именикот за влез кој одговара на `foo' и добиј го неговиот i-node број

Пример за операција: Open /tmp/foo

- Open `/tmp/foo‘
  1. [B] види дали i-node е во i-node cache (инаку прочитај го)
  2. Провери ги пермисиите на корисникот за /tmp/foo
  3. Искористи го i-node бројот за да одредиш дали /tmp/foo е веќе отворена (т.е., има влез во SFT):
     - Ако не, алоцирај влез во SFT, постави го потребниот inode број, постави врска до inode-от во јадрото за /tmp/foo
  4. Најди слободно место во PFT
     - Врати грешка ако нема место
     - инаку, иницијализирај влез, додади врска до влезот во SFT
  5. Иницијализирај влез
  6. Врати индекс за влезот од PFT како “fd”

NTFS Структура

NTFS Структура

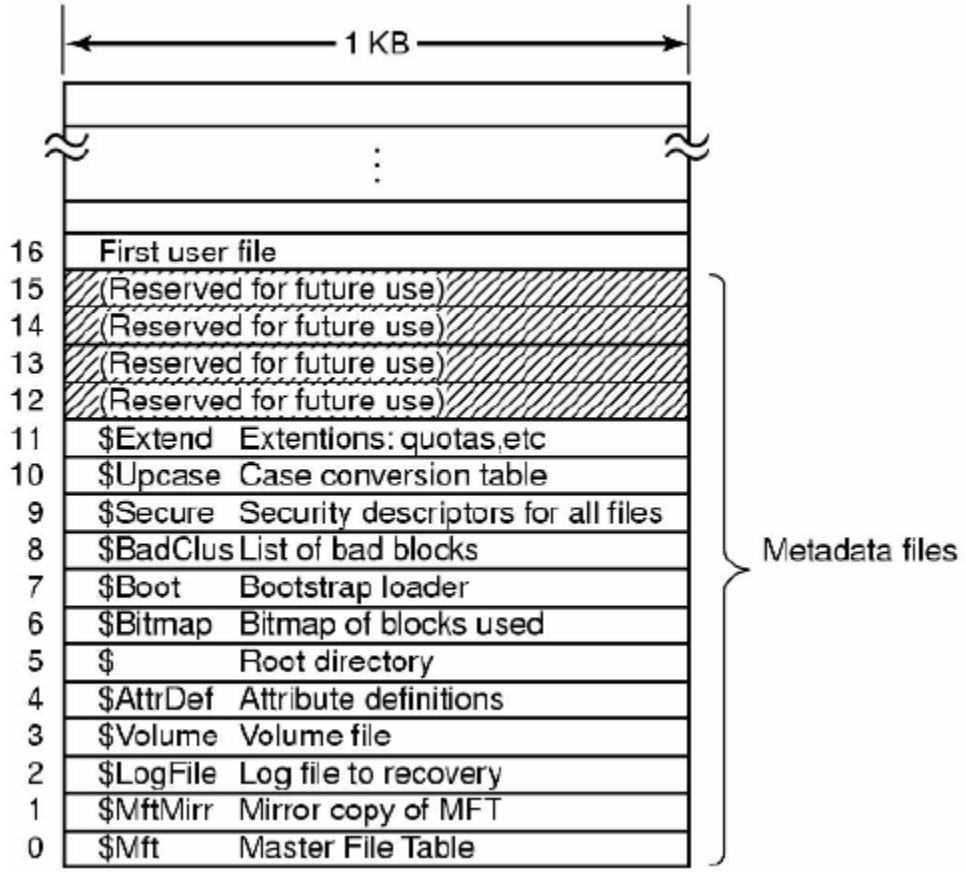
- Секоја NTFS партиција содржи датотеки, именици, битмапи и други податочни структури.
- Партицијата е организирана во блокови со големина од 512 бајти до 64 KB во зависност од големината на партицијата.
- Најчесто 4KB – **КОМПРОМИС**
- Референцирање на блок
  - преку 64 битен offset од почетокот на партицијата.

Структура на NTFS

- MFT (Master File Table) – главна податочна структура, организирана како линеарна секвенца од записи со големина од 1KB.
  - Секој запис опишува датотека или директориум.
- Записот содржи:
  - атрибути (име и временски жиг).
  - листа од дискови адреси.
- Големи датотеки може да користат 2 или повеќе записи (extents).
- Се користи битмапа за слободни записи
- MFT е датотека која може да биде лоцирана каде било на дискот и има максимум 2^{48} записи

Master File Table

- Секој запис се состои од секвенца од парови (заглавие на атрибут и вредност)
- Ако вредноста на атрибутот е голема во MFT се сместува покажувач на новата адреса на атрибутот.
- Првите 16 записи се резервирани за metadata датотеки
- Секој запис почнува со заглавие на записот



Master File Table

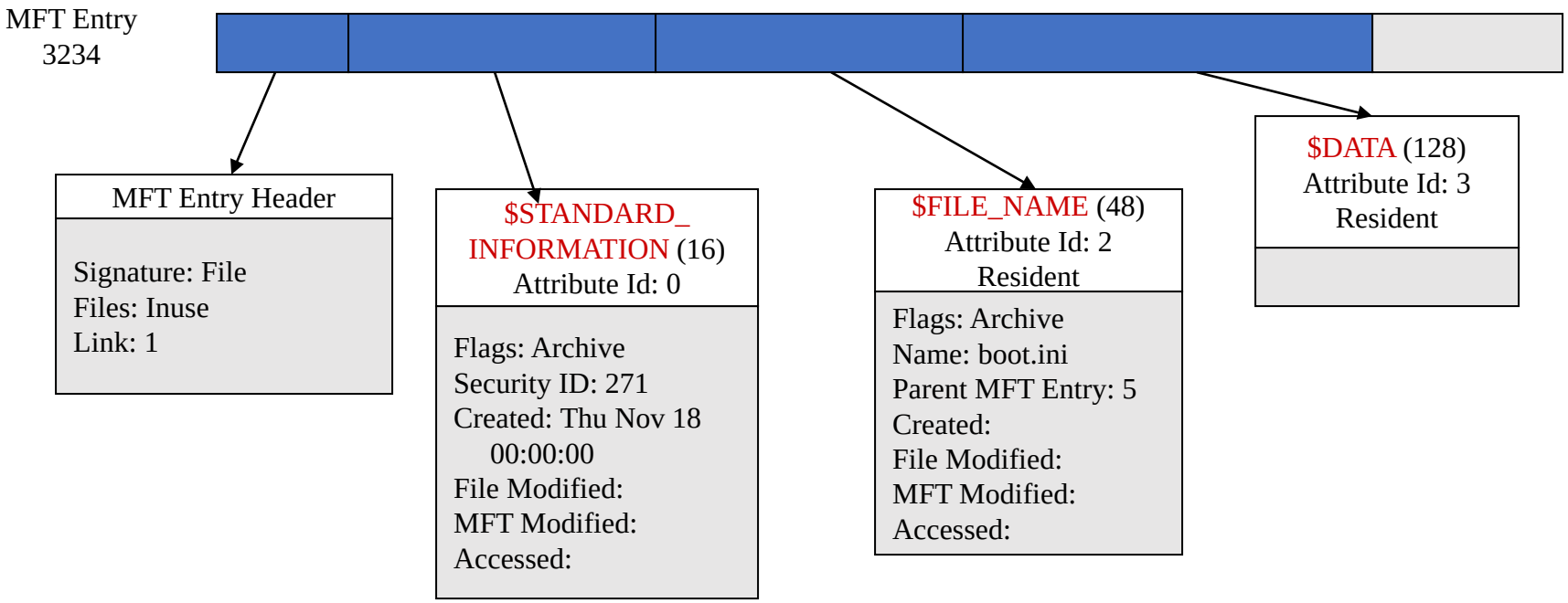
- Постојат 13 заглавија на атрибути (24B) што може да се појават во MFT табелата

Attribute	Description
Standard information	Flag bits, timestamps, etc.
File name	File name in Unicode; may be repeated for MS-DOS name
Security descriptor	Obsolete. Security information is now in $Extend$Secure
Attribute list	Location of additional MFT records, if needed
Object ID	64-bit file identifier unique to this volume
Reparse point	Used for mounting and symbolic links
Volume name	Name of this volume (used only in $Volume)
Volume information	Volume version (used only in $Volume)
Index root	Used for directories
Index allocation	Used for very large directories
Bitmap	Used for very large directories
Logged utility stream	Controls logging to $LogFile
Data	Stream data; may be repeated

Податочен атрибут

- Директни (Immediate) датотеки (неколку стотина бајтови)– податоците се директно сместени во MFT записот.
- Најчесто овој атрибут е нерезидентен и тогаш треба да се „скока“ од еден на друг блок.
  - Аргументи: Почетните блокови на кои се „скока“ и бројот на блокови од наведениот почетен блок (run length)

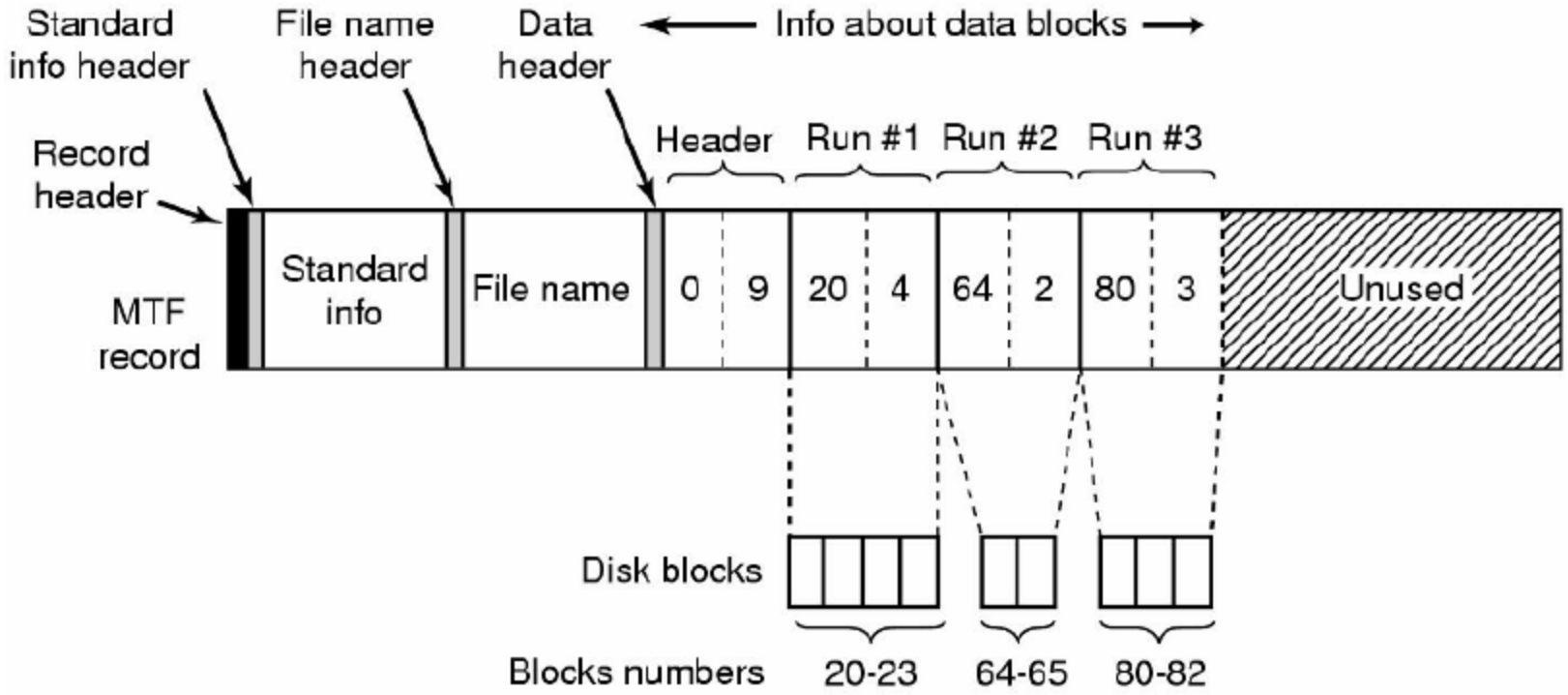
Metadata Category



Типична датотека ги има следните 3 атрибути:

- $STANDARD_INFORMATION
- $FILE_NAME
- $DATA

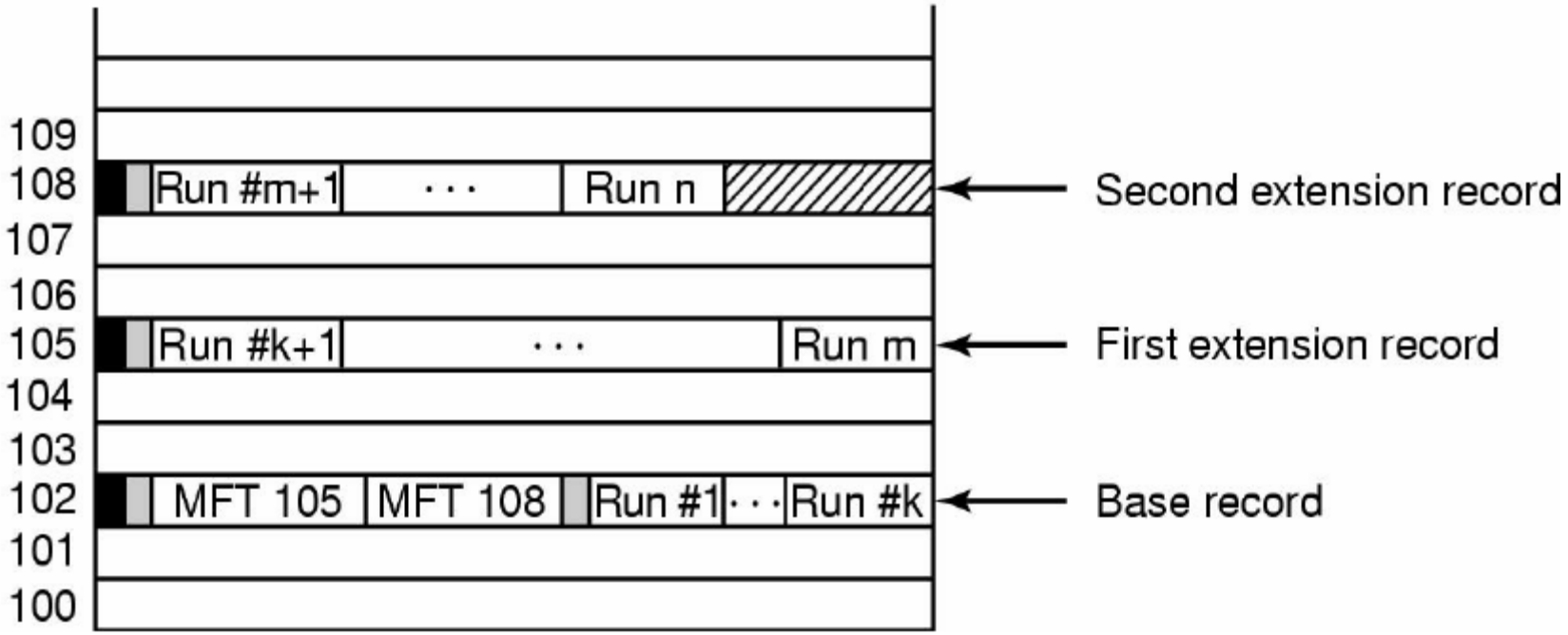
Пример


Standard info header
File name header
Data header
Info about data blocks
Record header
Header
Run #1
Run #2
Run #3
MTF record
Standard info
File name
0
9
20
4
64
2
80
3
Unused
Disk blocks
Blocks numbers
20-23
64-65
80-82

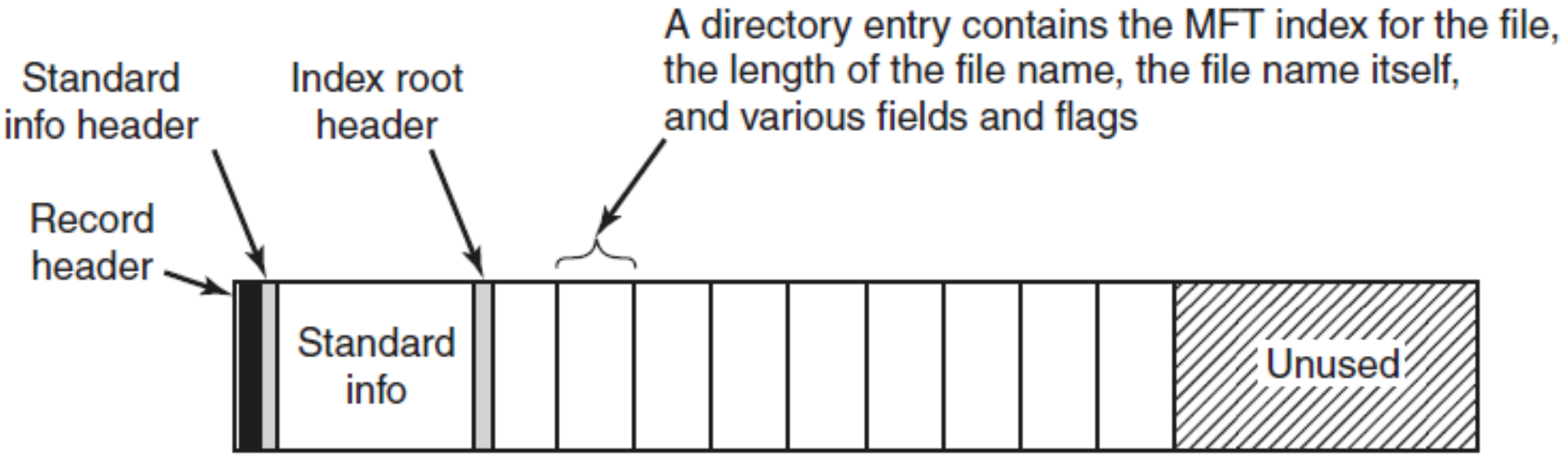
Податочен атрибут

- Бројот на вакви RUN-ови зависи од ефикасноста на Disk Block Allocator.
- За датотека со n блокови бројот на вакви run-ови се движи од 1 до n.
- Коментар:
  - Датотека од 20 GB која се состои од 20 run-ови од 1 милион блокови со големина од 1KB целосно ја собира во еден запис од MFT табелата.
  - Датотека од 60 KB распрскана на 60 изолирани блокови не ја собира во еден запис од MFT табелата.

Пример

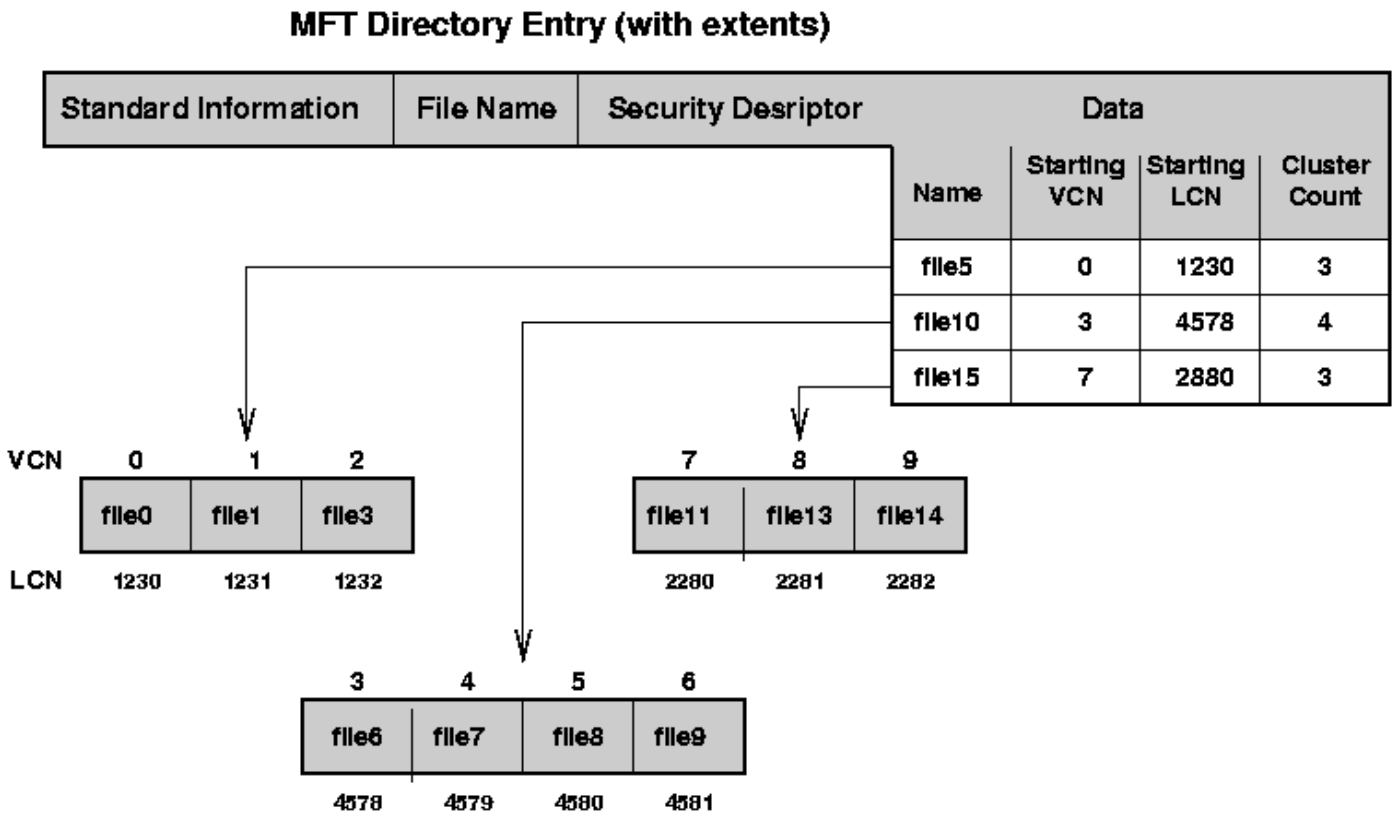

109
108
107
106
105
104
103
102
101
100
Run #m+1
· · ·
Run n
Run #k+1
· · ·
Run m
MFT 105
MFT 108
Run #1
· · ·
Run #k
Second extension record
First extension record
Base record

MFT запис за мал именик



MTF запис за голем именик

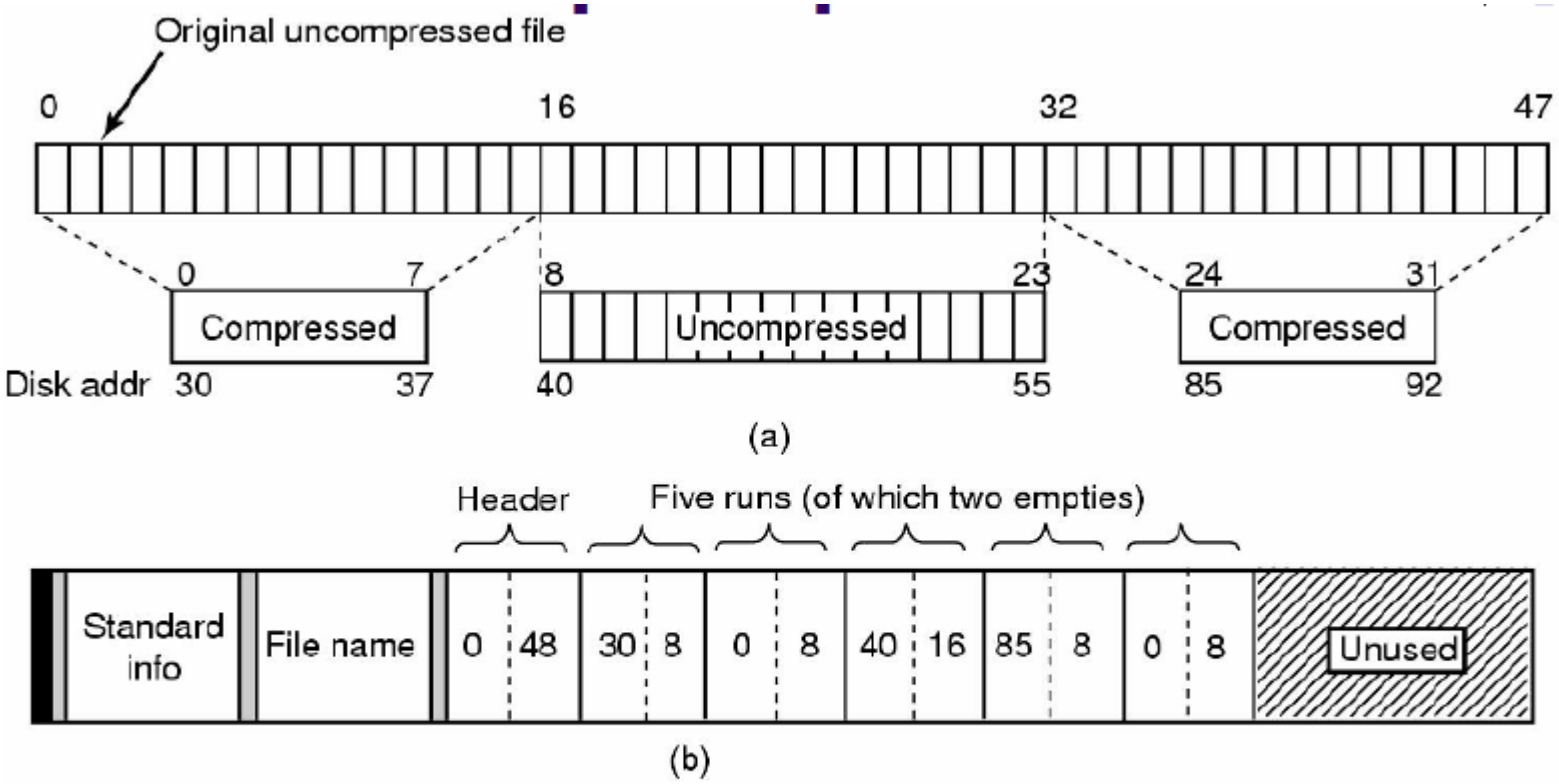
- Се користи B+ стебло



Компресирање на датотеки

- Транспарентна автоматска компресија/декомпресија на датотеки.
- Начин на работа:
  - Се проверуваат првите 16 блока од датотеката и се стартува компресирачки алгоритам.
  - Ако резултантните податоци може да се сместат во <= 15 блокови, тогаш компримирани се сместуваат на диск, во спротивно се запишуваат неспакувани.
- Се повторува постапката за наредните 16 блокови

Пример



Зошто по 16 блока?

КОМПРОМИС помеѓу ефективност на компресијата и цената на случаен пристап до блок од датотеката.

Пример

Дадена следнава MFT табела и првиот MFT запис за датотеката 'os.txt' е 100.

105			105	i	115	j	120	k	135	l	140	m
104												
103			50	d	70	e	90	f	95	g	100	h
102			250	34	340	100	660	1	700	5	800	9
101												
100			MFT103		MFT105		10	a	17	b	23	c

Притоа, со зелена боја се дадени дополнителните MFT записи за датотеката, додека со жолта боја дадени се почетните блокови од датотеката, додека со буквите од a до m означени се бројот на блокови од наведениот почетен блок (односно run length), вредностите за run length се дадени во следнава табела:

a b c d e f g h i j k l m

1 2 10 10 2 1 2 3 4 2 11 5 10

Според дадената информација одговорете на следниве прашања:

- Станува збор за нерезидентен запис.
- Колку записи од MFT табелата се резервирани за os.txt датотеката. 3 записи (внесете цел број).
- Доколку големината на блокот 1 KB, тогаш големината на датотеката 63 KB.