11. Файлова система. Логическа и физическа организация.

Логическа организация на файлова система (ФС). Имена на файлове. Типове файлове - обикновен файл, специален файл, каталог, символна връзка, програмен канал. Вътрешна структура на файл. Атрибути на файл. Йерархична организация на ФС - абсолютно и относително пълно име на файл, текущ каталог. Физическа организация на ФС. Стратегии за управление на дисковото пространство. Системни структури, съдържащи информация за разпределението на дисковата памет и съхранявани постоянно на диска: за свободните блокове; за блоковете, разпределени за всеки един файл; за общи параметри на ФС. Примери за физическа организация на ФС: UNIX System V; LINUX; MS DOS; NTFS. Забележки: За изпита ще бъдат избрани два от изброените примери за файлова организация.

Файлова система (ФС) това е тази компонента на операционната система, която е предназначена да управлява постоянните обекти данни, т.е. обектите, които съществуват по-дълго отколкото процесите, които ги създават и използват. Постоянните обекти данни се съхраняват на външна памет (диск или друг носител) в единици, наричани файлове.

Файловата система трябва да:

- осигурява операции за манипулиране на отделни файлове, като например create, open, close, read, write, delete и др.
- изгражда пространството от имена на файлове и да предоставя операции за манипулиране на имената в това пространство.

ЛОГИЧЕСКА СТРУКТУРА НА ФАЙЛОВА СИСТЕМА

➢ ИМЕНА И ТИПОВЕ ФАЙЛОВЕ

Най-често името на файл е низ от символи с определена максимална дължина, като в някои системи освен букви и цифри са разрешени и други символи. Много често името се състои от две части, разделени със специален символ, например ".". Втората част се нарича разширение на името и носи информация за типа или формата на данните, съхранявани във файла. Например, следните имена имат разширения, показващи типа на данните във файла.

`file.p` - програма на Паскал
`file.c` - програма на Си
`file.o` - програма в обектен код
`file.exe` - програма в изпълним код
`file.txt` - текстов файл

В някои операционни системи, като UNIX и LINUX, такива разширения представляват съглашения, които се използват от потребителите и някои обслужващи програми (транслатори, свързващи редактори, текстови процесори и др.), но ядрото не ги налага и използва. В други операционни системи се реализира по-строго именуване. Например, няма да бъде зареден и изпълнен файл, ако името му няма разширение "`.exe`" или някое друго.

Какво включва пространството от имена или какви са типовете файлове? Преди всичко то включва имена на ***обикновени файлове*** (***regular files***), съдържащи програми, данни, текстове или каквото друго потребителят пожелае. Но пространството от имена би могло да включва и имена на външни устройства, системни структури и услуги. Външните устройства са специален тип файлове, наречени ***специални файлове*** (***character special и block special device file***) в UNIX, LINUX, MINIX и др. Системните примитиви на файловата система са приложими както към обикновените файлове така и към специалните файлове. Следователно, всяка операция за четене или писане, осъществявана от програмата, е четене или писане във файл. Най-съществения предимство на този подход е, че позволява да се пишат програми, които не зависят от устройствата, тъй като действията, изпълнявани от програмата зависят само от името на файла.

Трябва да се съхранява информация за файловете във файловата система и за тази цел се използват специални системни структури, наричани ***каталог***, ***справочник***, ***директория***, ***directory***, ***folder***, ***VTOC***, които осигуряват връзката между името и данните на файла и реализират организацията на файловете.

Много системи каталогът е тип файл с фиксирана структура на записите и съдържа по един запис за всеки файл.

И така, типовете файлове, поддържани от файловите системи на UNIX, LINUX, MINIX и др. са:
- обикновен файл
- каталог
- специален файл
- програмен канал, FIFO файл
- символни връзки

Типът ***програмен канал*** (***pipe***) и ***FIFO файл*** се използва като механизъм за комуникация между конкурентни процеси. Чрез ***символните връзки*** (***symbolic link, soft link, junction***) един файл може да има няколко имена, евентуално в различни каталози, или както се казва реализират се няколко връзки към файл. Този тип файл е един от механизмите за осигуряване на общи файлове за различните потребители. Друг начин за работа с общи файлове са ***твърдите връзки*** (***hard links***), но те не са тип файл, а са няколко имена на обикновен файл.

➢ ВЪТРЕШНА СТРУКТУРА НА ФАЙЛ

Файлът най-често представлява последователност от обекти данни. Възможни са два вида данни - запис или байт.

Файлът е последователност от записи с определена структура и/или дължина. Основното в този подход е, че всеки системен примитив `read` или `write` чете или пише един запис.

Другата възможност, реализирана в много от съвременните операционни системи, UNIX, LINUX, MINIX, MSDOS, Windows и др., е последователност от байтове.

➢ АТРИБУТИ НА ФАЙЛ

Всеки файл има име и данни. В допълнение операционната система може да съхранява и друга информация за файла, която ние ще наричаме ***атрибути*** на файла (наричат ги също така ***метеданни***). Атрибутите, реализирани в различните системи се различават, но един списък от възможни атрибути е показан на
Таблица. 1.

Таблица 1. Някои възможни файлови атрибути

Размер	Текущия размер на файла в байтове, блокове или записи.
Време на създаване	Дата и време на създаване на файла
Време на достъп	Дата и време на последен достъп до файла
Време на изменение	Дата и време на последно изменение на файла
Собственик	Потребителят, който е текущия собственик на файла.
Права на достъп	Кой и по какъв начин може да осъществява достъп до файла.
Парола за достъп	Парола за достъп до файла
Флагове:	
Read-only флаг	1 - само за четене, 0 - за четене и писане
Hidden флаг	1 - не се вижда от командите, 0 - видим за командите
System флаг	1 - системен файл, 0- нормален файл
Archive флаг	1 - трябва да се архивира, 0 - не е променян след архивирането
Secure deletion	При унищожаване на файла блоковете, заемани от него се форматират.

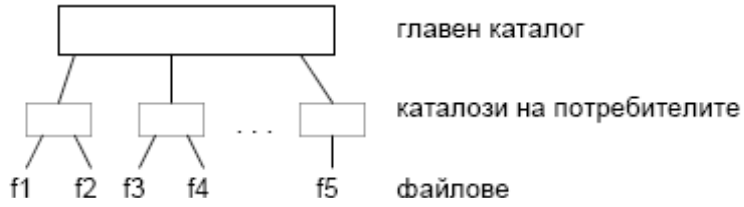
➢ ЙЕРАРХИЧНА ОРГАНИЗАЦИЯ НА ФС

Каталогът съдържа по един запис за всеки файл, който съдържа поне името на файла. Освен това може да съдържа атрибутите на файла и дисковите адреси на данните на файла или указател към друга структура, където се съхраняват дисковите адреси на данните и евентуално атрибутите на файла.

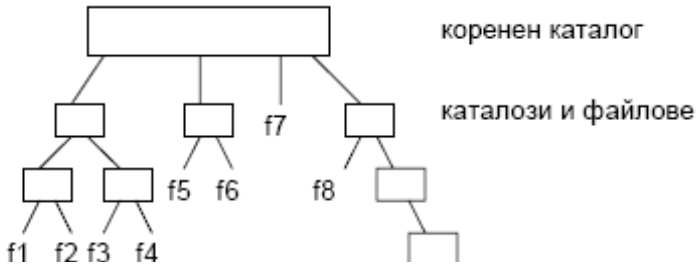
Като правило файловата система е разположена върху няколко носителя и включва файлове, принадлежащи на различни потребители.

Еднокаталогова файлова система - На всеки носител има по един каталог, който съдържа информация за всички файлове върху носителя. Такава е организацията в някои ранни операционни системи или в по-нови, но примитивни. Преимущество на тази организация е простотата и възможността за бързо търсене на файл .

Йехархична структура с фиксиран брой нива - Всеки потребител има свой каталог, който съдържа записи за всички файлове на потребителя. Има главен каталог, който съдържа по един запис за каталозите на потребителите. Тогава имената, които потребителят дава на своите файлове, трябва да са уникални в рамките на неговия каталог



Йерархична структура с произволен брой нива - Вътрешните възли на дървото трябва да са каталози, а листата могат да бъдат каталози, обикновени файлове, специални файлове и други типове фалове, ако се поддържат такива. За потребителя каталогът представлява група от файлове и каталози (подкаталози).



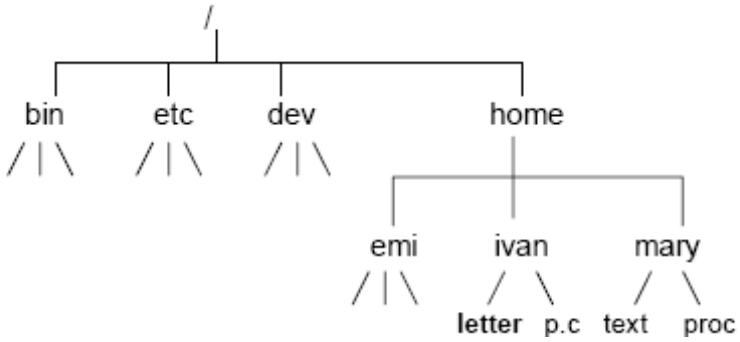
➢ АБСОЛЮТНО И ОТНОСИТЕЛНО ИМЕ НА ФАЙЛ; ТЕКУЩ КАТАЛОГ

Всеки връх в дървото, каталог или друг тип файл, има име, което потребителят избира да е уникално в рамките на съдържащия го каталог и което ние ще наричаме ***собствено име***. Тогава всеки файл ще има име, което уникално го идентифицира в рамките на цялата йерархична структура и ние ще го наричаме ***пълно име***. Използват се два начина за формиране на пълно име:

• ***абсолютно пълно име*** (***absolute path name***) - Всеки файл или каталог притежава едно абсолютно пълно име, което съответства на единствения път в дървото от корена до съответния файл или каталог.
• ***относително пълно име*** (***relative path name***) - Този начин за формиране на пълно име е свързан с понятието ***текущ или работен каталог*** (***current/working directory***). Във всеки един момент от работата на потребителя със системата, той е позициониран в един от каталозите на дървото. Относителното пълно име на файл или каталог съответства на пътя в дървото от текущия в даден

момент каталог до съответния файл или каталог. Като в този случай е разрешено и движение нагоре по дървото.

Пълното име на файл, абсолютно или относително, се състои от компоненти - собствените имена на каталозите в пътя и името на самия файл, разделени със специален символ-разделител, като напр. ">", "\", "/". Движението нагоре по дървото се записва чрез специално име, което обикновено е "..". Ако първият символ в пълното име е символа-разделител, това означава, че името е абсолютно. Всяко име, което не започва със символа-разделител, се приема за относително. За илюстрация да разгледаме една типична йерархия на файлова система в UNIX:



Абсолютно пълно име на файл е /home/ivan/letter. Ако текущ каталог е /home, то относителното пълно име на същия файл е ivan/letter. Ако текущ каталог е /home/emi, то относителното пълно име на същия файл е ../ivan/letter

ФИЗИЧЕСКА ОРГАНИЗАЦИЯ НА ФАЙЛОВА СИСТЕМА

➢ СТРАТЕГИИ ЗА УПРАВЛЕНИЕ НА ДИСКОВАТА ПАМЕТ

Стратегията за управление на дисковата памет трябва да даде отговор на два въпроса.

1. *Кога се разпределя дискова памет за файл?* Едната възможност е това да се прави статично (предварително) при създаването на файла, а другата е да се прави динамично при нарастване на файла.
2. *Колко непрекъснати дискови области може да заема един файл*? Дали за файла се разпределя една (или малко на брой) непрекъсната дискова област с нужния размер или за файла могат да се разпределят много порции дискова памет, които не е задължително да са физически съседни, а броят им най-често се ограничава от капацитета на диска.

Най-често реализираните стратегии са две:

Статично и непрекъснато разпределение

За файл с размер n байта се отделя една непрекъсната област от диска, в която могат да се съхранят всичките n байта и това се прави при създаването на файла. За тази цел обаче, системата трябва да има предварителна информация за максималния размер на файла, който той може да достигне по време на съществуването си. Естествено тази информация може и трябва да се осигури от потребителя по време на създаване на файла и тогава системата ще разпредели дискова памет за файла. Преимуществото, което този метод дава, е високата производителност на системата при последователна обработка на файла, тъй като последователните байтове на файла са разположени в съседни сектори, писти и цилиндри на диска.

Проблем може да възникне при **нарастване** на файла, което е нещо обичайно, ако размерът на файла надмине разпределената му предварително памет. Едно възможно решение на този проблем е разпределяне на нова непрекъсната област с по-голям размер и преместване на файла в новата област. Това обаче не е добро решение, тъй като операцията може да се окаже бавна и скъпа при големи файлове. Друго решение е да се направи компромис при искането за непрекъснатост на дисковата памет, т.е. един файл да може да заема не една, а няколко, но малко на брой непрекъснати области от диска с размери, заявени от потребителя.

Друг недостатък на тази стратегия е известен като **фрагментация** на свободната дискова памет. Това означава съществуване на достатъчно свободни участъци дисковата памет, които обаче не са съседни и поради това не може да бъде удоволетворена заявка за първично или вторично разпределение на памет за файл, тъй като се изисква непрекъснатост.

Тази стратегия се оказа приложима и в съвремието при CD-ROM, където размера на файла е предварително известен.

Динамично и поблоково разпределение

Файлът се дели на части с фиксирана дължина, които ще наричаме **блокове**. Последователните блокове на файла при записването им на диска не е задължително да са физически съседни. Това дава възможност дискова памет за файл да се разпределя по блокове тогава, когато е необходима, т.е. динамично при нарастване на файла. Така се решава и проблема с нарастване на файла и проблема с фрагментацията на свободната дискова памет, тъй като памет се разпределя динамично по блокове, които са с еднакъв фиксиран размер.

При прилагане на тази стратегия трябва се избере **размер на блока -** който най-често е 1К байта, 2К байта, 4К байта. При по-нататъшното изложение ще предполагаме, че се използва динамично и поблоково разпределение на дискова памет, както е в повечето съвременни системи. Единицата за разпределение на дискова памет ще наричаме блок. *Следователно, за файловата система дисковото пространство е последователност от блокове с адреси (номера) 0,1,2, ...N.*

- СИСТЕМНИ СТРУКТУРИ, СЪДЪРЖАЩИ ИНФОРМАЦИА ЗА РАЗПРЕДЕЛЕНИЕТО НА ДИСКОВАТА ПАМЕТ, т.е. за СВОБОДНИТЕ БЛОКОВЕ, БЛОКОВЕТЕ РАЗПРЕДЕЛЕНИ ЗА ВСЕКИ ЕДИН ФАЙЛ И ОБЩИ ПАРАМЕТРИ НА ФС

Файловата система трябва да съхранява информация за разпределението на дисковата памет, а именно за *свободните блокове* и за *блоковете разпределени за всеки един файл*. Под системни структури тук ще разбираме структури, съхраняващи такава информация постоянно на диска.

Информация за всички свободни блокове трябва да бъде съхранявана, тъй като само по съдържанието на блока не може да се отличи свободния от заетия блок. Някои възможни структури данни, използвани за тази цел са следните.

Свързан списък на свободните блокове

Всички свободни блокове са организирани в едносвързан списък, т.е. първата дума от всеки свободен блок съдържа адрес на следващ свободен блок. Недостатък на този метод е, че системната структура, т.е. свързаният списък, е пръсната по всички свободни блокове, което крие опасности за повредата й при системни сривове.

Свързан списък на блокове с номера на свободни блокове

Свободните блокове се групират и номерата на блоковете от една група се записват в първия свободен блок. Следователно информацията се съхранява в едносвързан списък от дискови блокове, които съдържат номера на свободни блокове. Самите блокове от списъка вече не са свободни за разлика от предходната структура. Такава структура е по-компактна, големината й е пропорционална на свободното дисково пространство и позволява разработката на ефективни алгоритми за разпределение на блокове. Този подход е използван във файловата система на UNIX System V (s5fs).

Карта или таблица

Структурата представлява масив от елементи, като всеки елемент съответства позиционно на блок от диска, т.е. на блок с номер равен на индекса на елемента в масива. Съдържанието на всеки елемент описва състоянието на блока. В най-простия случай може да се помнят две състояния - свободен и зает. Тогава елемент от картата може да е просто бит. Ако битът е 1, то съответният блок е свободен, а ако е 0 е зает (разпределен за файл или друга структура на диска) или обратното. Такава системна струтура се нарича ***битова карта (bit map).*** Преимущество на битовата карта е, че е компактна и с фиксиран размер, а при разпределяне на памет позволява да се отчита физическото съседство на блоковете. Битова карта е използвана във файловите системи на MINIX, LINUX , OS/2 (HPFS) и Windows (NTFS).

Друг тип информация, която трябва да бъде съхранявана, е за дисковата памет, разпределена за всеки един файл. Всеки файл от гледна точка на физическото му представяне представлява последователност от блокове, съдържащи последователните му данни, като последователните блоковете на файла може да не са физически последователни. Следователно файловата система трябва да съхранява информация за блоковете, разпределени за всеки един файл в съответната логическа последователност. Ще разглеждаме някои възможни структури за тази цел.

Свързан списък на блоковете на файла

Всеки блок на файла ще съдържа в първата си дума номер на следващ блок на файла и данни в останалата част. Основният недостатък на тази структура е неефективната реализация на произволен достъп до файла. За да се прочете произволен байт от файла системата трябва да прочете всички предходни блокове в списъка докато стигне до данните, искани от програмата. Друг недостатък е, че адресната информация е пръсната по

диска, а това прави файловата система неустойчива при повреди. И още един недостатък, който в някои случаи е критичен, е че броят на байтовете за данни в блока вече не е степен на 2.

Карта или таблица

Същността на проблема при предходната реализация се състои в това, че адресите и данните се съхраняват заедно. Идеята на картата (таблицата) на файловете е свързаните списъци от номера на блокове за всички файлове на диска да се съхраняват в една структура отделно от данните. В същност това може да е същатата системна структура карта, в която се съхранява информация за свободните блокове и която описахме по-горе, като елемент на масива е достатъчно голям, така че да позволи съхраняване на следните състояния на блок - свободен, повреден, а ако е зает да съдържа номера на следващия блок на файла. Тази структура е използвана в MSDOS, където се нарича таблицата ***FAT (File Allocation Table).*** По-нататък по-подробно ще разгледаме физическото представяне на файловата система в MSDOS.

Индекси

Основният недостатък на предишния подход е, че информацията за всички файлове се съхранява в една структура, което при големи дискове създава проблеми. При големи дискове естествено голяма става и картата на файловете, а тя трябва цялата да се зарежда и съхранява в оперативната памет по време на работа дори ако е отворен само един файл. Следователно по-добре би било ако списъците на различните файлове се съхраняват в отделни структури и тогава в паметта ще се зарежда само информацията за отворените файлове.

Структурата, в която се съхранява информацията за блоковете, разпределени за определен файл се нарича индекс. Всеки индекс съдържа адресите на дисковите блокове, разпределени за един файл, като наредбата на адресите отразява логическата наредба на блоковете във файла.

Една възможна реализация на индекса е ***индексен списък*** в XINU. Индексният списък е едносвързан списък от индексни блокове. Индексните блокове са с размер, различен от този на блоковете за данни и се намират в област на диска, отделна от областта на блоковете за данни, т.е. адресното пространство за индексните блокове е различно от това на дисковите блокове. Всеки индексен блок, освен адрес на следващ индексен блок, съдържа и определен брой адреси на дискови блокове с данни, като наредбата на адресите отразява логическата наредба на блоковете с данни във файла.

Друга възможна реализация е чрез дърво.Такъв подход е използван в UNIX, LINUX и MINIX, където структурата се нарича ***индексен описател*** (***i-node*** от ***index node***) и при големи файлове е корен на дърво, включващо и косвени блокове.

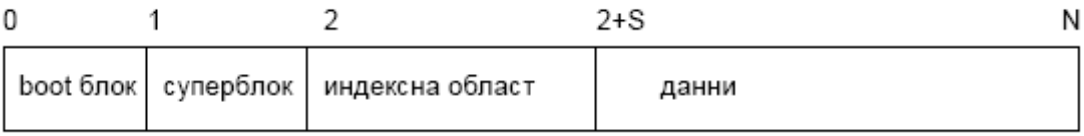
Друг тип информация, която трябва се съхранява, са *общи параметри на физическата структура на файловата система*. Обикновено системната структура се нарича ***суперблок***. В суперблока се съхраняват общи параметри, като:

1- размер на файловата система (максимален номер на блок на тома)
2- размер на блок
3- размери и адреси на различни области от диска, съдържащи системни структури, напр. размер на индексната област, на битовата карта, на FAT
4- общ брой свободни блокове на диска и други ресурси, напр. индексни описатели.
5

ПРИМЕРИ ЗА ФИЗИЧЕСКА ОРГАНИЗАЦИЯ НА ФАЙЛОВИ СИСТЕМИ

- UNIX System V (s5fs)

Всеки диск се състои от един или няколко дяла (partitions). Дяловете се разглеждат като независими устройства (на всеки дял съответства различен специален файл). За файловата система дисковото пространство на всеки диск или дял от диск е последователност от блокове с фиксиран размер и адресирани с номера 0, 1, 2,... N. При създаване на празна файлова система с командата mkfs, дисковото пространство на всеки дял се разделя на четири области



Блок с номер 0, наричан boot блок, съдържа програма за първоначално зареждане на операционната система. Използва се само в коренната файлова система, но заради еднотипността присъства и в другите файлови системи.

Блок с номер 1 е наречен суперблок, тъй като съдържа общи параметри на физическата структура на файловата система.

От блок 2 започва индексната област, където се съхраняват индексните описатели (i-node) на всички файлове. Размерът на тази област S блока е функция от общия размер на файловата система N и трябва внимателно да бъде изчислен така, че да не ограничава броя на файловете на диска. Този размер се задава като параметър при изграждане на празна файлова система с командата `mkfs` и не може да бъде променян след това.

В последната област - данни, се съхраняват блоковете на всички файлове и каталози, косвените блокове, блоковете от списъка на свободните блокове и евентуално някои блокове са свободни.

Индексни описатели

Всеки файл от произволен тип има точно един индексен описател, независимо от това в кой и колко каталога е включен и под какви имена. Индексните описатели се номерират от 1 и номерът съответства на позицията му в индексната област. Индексният описател е с размер 64 байта и има следната структура:

```
struct dinode
{
ushort di_mode; /* mode and type of file */
short di_nlink; /* number of links to file */
ushort di_uid; /* owner's user id */
ushort di_gid; /* owner's group id */
off_t di_size; /* number of bytes in file */
char di_addr[40]; /* disk block addresses */
time_t di_atime; /* time last accessed */
time_t di_mtime; /* time last modified */
time_t di_ctime; /* time last changed */
};
```

В полето `di_mode` се съхранява типа на файла в старшите 4 бита и кода на защита на файла в останалите 12 бита.

Полето `di_nlink` съдържа броя на твърдите връзките или имената на файла, т.е. колко пъти в записи на каталози е цитиран този номер на индексен описател.

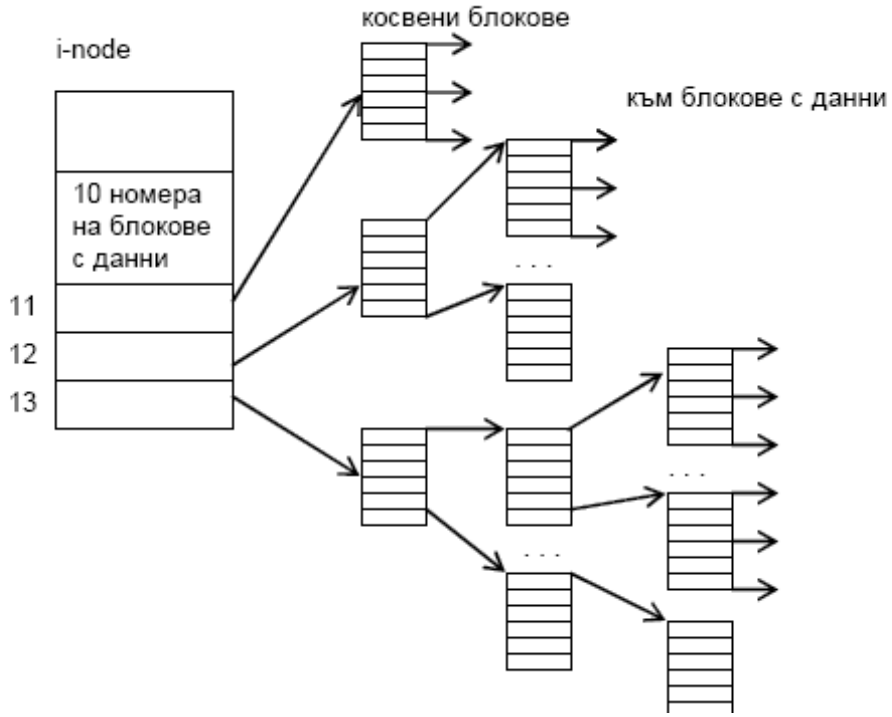
Полетата `di_uid` и `di_gid` определят собственика и групата на файла.

Полето `di_size` при обикновени файлове и каталози съхранява размера на файла в брой байтове.

В полетата `di_atime, di_mtime и di_ctime` се записват дата и време, съответно на последен достъп, последно изменение и последно изменение на i-node на файла.

Ако файлът е специален в полето `di_addr` се съхранява номера на устройството, на което съответства специалния файл.

За другите типове файлове в полето di_addr се съхраняват 13 дискови адреса (номера на блокове от областта за данни) всеки в 3 байта. Първите 10 адреса са директни адреси на първите 10 блока с данни на файла. Ако файлът стане по-голям от 10 блока, тогава се използват косвени блокове. Косвените блокове се намират в областта за данни, но съдържат номера на блокове, а не данни на файла. Единадесетият адрес съдържа номер на косвен блок, който съдържа номерата на следващите блокове с данни на файла. Това се нарича единична косвена адресация. Чрез дванадесетия адрес се реализира двойна косвена адресация, т.е. там е записан номер на косвен блок, който съдържа номера на косвени блокове, които вече съдържат номера на блокове с данни. За представяне на много големи файлове се използва тринадесетия адрес и тройна косвена адресация



Представянето на файловете в UNIX системите съчетава следните предимства: малък индексен описател с възможност за представяне на големи файлове при осигуряване на бърз достъп до произволен байт от файла. И при най-големи файлове за достъп до произволно място във файла са необходими най-много три допълнителни достъпа до диска за трите нива на косвени блокове (индексният описател се зарежда в паметта при отваряне на файла и се съхранява там до затваряне).

Суперблок

Суперблокът съдържа изключително важна информация, характеризираща физическата структура на файловата система. Основните данни, съдържащи се в него, са описани в следната структура:

```
struct filsys
{
      ushort s_isize; /* size in blocks of i-list */
      daddr_t s_fsize; /* size in blocks of entire volume */
      short s_nfree; /* number of addresses in s_free */
      daddr_t s_free[NICFREE]; /* free block list */
      short s_ninode; /* number of i-nodes in s_inode */
      ushort s_inode[NICNODE]; /* free i-node list */
      char s_flock; /*lock during free list manipulation*/
      char s_ilock; /* lock during i-list manipulation */
      char s_fmod; /* super block modified flag */
      char s_ronly; /* mounted read only */
      . . .
      daddr_t s_tfree; /* total free blocks */
      ushort s_tinode; /* total free i-nodes */
      . . .
};
```

Полето `s_isize` съдържа дължината на индексната област в блокове, която се задава и зарежда в полето при създаване на файловата система (параметъра S)

Полето `s_fsize` съдържа максималния номер на блок във файловата система и също както `s_isize` се задава и зарежда при изграждане на файловата система (параметъра N)

Полето `s_nfree` съдържа брой свободни блокове, чиито номера са записани в масива `s_free`.

Полето `s_ninode` съдържа брой свободни индексни описатели, чиито номера са записани в масива `s_inode`.

Полетата `s_flock` и `s_ilock` се използват като флагове за заключване на достъпа до списъка на свободните блокове и свободните индексни описатели по време на манипулирането им с цел избягване на състезание между конкурентни процеси.

Полетата `s_tfree` и `s_tinode` съдържат общия брой свободни блокове и свободни индексни описатели на диска.

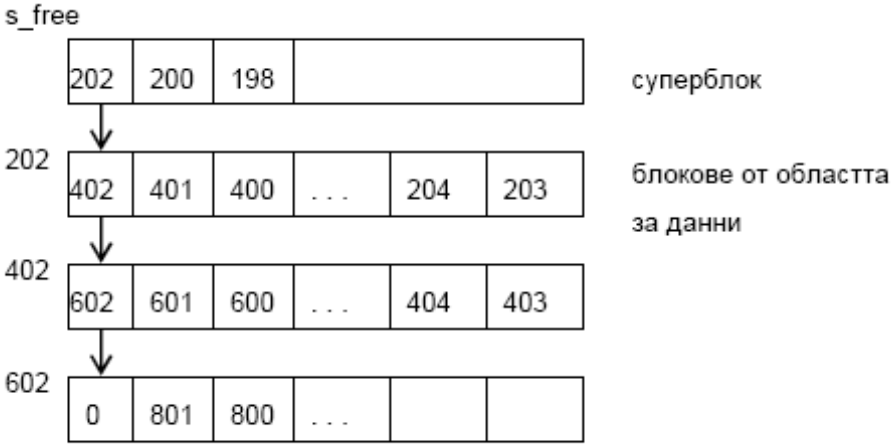
Суперблоковете на коренната и всички монтирани файлови системи се зареждат в оперативната памет в таблицата на суперблоковете и остават там през цялото време на работа на системата. Това осигурява бърз достъп до най-важните характеристики на файловите системи и се използва от алгоритмите за разпределяне и освобождаване на ресурсите - блокове и индексни описатели.

Заб: От анотацията на въпроса не става ясно дали трябва да се включват въпорсните алгоритми за разпределение и освобождаване на блокове и индексни описатели.Алгоритмите и отбелязаните по-долу структури трябва ли да се включват?

Запължителнолли

Списък на свободните блокове

Всички блокове от областта за данни, които не се разпределени за файлове, каталози, за косвени блокове, т.е не съдържат данни на файловата система са свободни. Но файловата система не е в състояние да отличи свободния от заетия блок само по съдържанието му. Затова информацията за всички свободни блокове се съхранява в специална структура - *едносвързан списък от блокове*, които съдържат номера на свободни блокове. Масивът s_free в суперблока представлява глава на списъка. Останалите елементи от списъка на свободните блокове, които са дискови блокове, са разположени в областа за данни. На Фиг. 8 е изобразен примерен списък на свободните блокове.



Фиг. 8. Списък на свободните блокове в s5fs на UNIX

Списък на свободните индексни описатели

Другият ресурс, който файловата система трябва да управлява, са индексните описатели и те са разположени в отделно пространство - индексната област. При създаване на файл за него трябва да се разпредели свободен индескен описател, а при унищожаване на файл индексния му описател трябва да се отбележи като свободен.

В суперблока е разположен масив s_inode, в който са записани известен брой номера на свободни индексни описатели. За разлика от управлението на блоковете, този масив не продължава в свързан списък от блокове. Това означава, че освен номерата в масива s_inode в индексната област може да има и други свободни индексни описатели. Това не създава проблеми при управлението им, тъй като ядрото може да различи свободния от заетия i-node по полето di_mode - битовете за тип са 0 в свободен i-node. В същност би могло и без масива s_inode, т.е. когато е необходим свободен i-node ще се търси в индексната област. Но такъв алгоритъм би бил неефективен, ако всеки път когато се създава файл се обхождат блоковете на индексната област в търсене на свободен i-node. Затова е въведен масива s_inode, в който са кеширани известен брой номера на свободни индексни описатели, но структурата не продължава в пълен списък.

Каталози

Каталозите са тип файлове, които осигуряват връзката между името и данните на файл и йерархичната структура на файловата система. В разглежданата s5fs файлова система записът в каталога има следната структура:

```
struct direct
{
ushort d_ino;
char d_name[DIRSIZE];
};
```

Полето d_name съдържа собственото име на файл или подкаталог, а полето d_ino съдържа номера на индексния му описател. Във всеки каталог има два стандартни записа, в единия полето d_name съдържа "..", а d_ino номера за родителския каталог, в другия полето d_name съдържа ".", а d_ino номера за самия каталог. Тези два записа присъстват и в празен каталог и са част от представянето на дървовидната структура на файловата система. Някои записи може да са празни. Номер 0 в полето d_ino означава, че записът е освободен при унищожаване на файл.

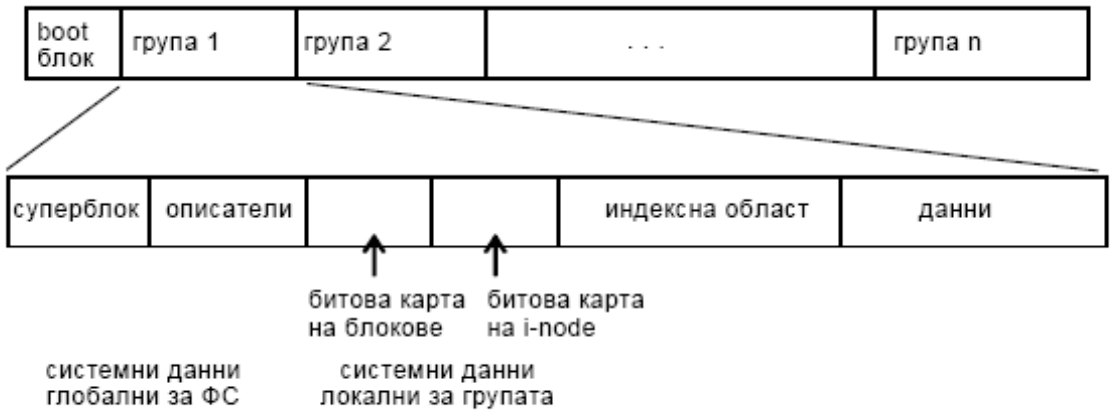
Първите няколко индексни описатели са резервирани за някои важни файлове, като коренния каталог, файла на лошите блокове и др. Така номерът на i-node на коренния каталог е известен, а самия каталог е разположен в произволни блокове от областта за данни

Заб: Трябва ли да включа по-подробно описание за каталози – в лекциите има пример? Също така трябва ли да се включват разяснения за твърдите и симлволните връзки?

Разгледаната файлова система s5fs има много преимущества, които отбелязахме в хода на разглеждането й, но има и слаби места. От гледна точка на надеждността слабо място е суперблока, за който се съхранява само едно копие. За производителността на системата е критично, че индексните описатели са разположени в началото на файловата система и са далеч от данните на файла. Индексната област е с фиксиран размер, който се задава при създаване на файловата система, и неподходящия му избор може да доведе до липса на свободни индексни описатели при наличие на свободно дисково пространство.

➢ LINUX

Основната файлова система в LINUX е Extended File System-version 2 (ext2), а вече и ext3. При управлението на дисковото пространство се използват блокове с фиксиран размер - 1KB, 2KB или 4KB. Това е и адресуемата единица на диска, единна за всички файлови системи. Фиг.13 представя областите, в които е организирано дисковото пространство.



Фиг.13. Разпределение на дисковото пространство в LINUX

Всяка група съдържа част от файловата система и копие на глобални системни структури, критични за цялостността на системата - суперблока и описателите на групите. Ядрото работи със суперблока и описателите от първата група. Когато се извършва проверка на непротиворечивостта на файловата система (изпълнява се командата fsck), ако няма повреди, то двете системни структури от първата група се копират в останалите групи. Ако се открие повреда, тогава се използва старо копие на системните структури от друга група и обикновено това позволява да се ремонтира файловата система.

Битови карти

Битовите карти описват свободните ресурси - блокове и индексни описатели в съответната група. Значение 0 означава свободен, а 1 използван блок или индексен описател. Размерът на групите е фиксиран и зависи от размера на блок, като стремежът е битовата карта на блоковете да се събира в един блок.

Индексни описатели

Индексните описатели са разпределени равномерно във всички групи, но се адресират в рамките на файловата система. Структурата на индексния описател в ext2 и ext3 е разширение на i-node от UNIX с нови полета. Размерът е 128 байта. Адресните полета са 12+1+1+1, т.е. използва се косвена адресация на три нива, но броят на директните адреси е увеличен с два, т.е. е 12. Освен това адресните полета вместо по 3 байта са по 4 байта

Добавени са нови атрибути на файл, например:
1- размер на файла в брой блокове по 512 байта (i_blocks)

2- още едно четвърто поле за дата и време (i_dtime)
3- флагове:

immutable - Файлът не може да се изменя, унищожава, преименува и да се създават нови връзки към него (дори от администратора).

append only - Писането във файла е винаги добавяне в края му.

synchronous write - Системният примитив write завършва след като данните са записани на диска.

secure deletion - При унищожаване на файла блоковете му се форматират.

undelete - При унищожаване на файла съдържанието му се съхранява за евентуално възстановяване впоследствие.

compress file - При съхраняване на файла ядрото автоматично го компресира.

Някои от добавените атрибути са за бъдещо планирано развитие на файловата система.

В този индексен описател има две полета за размер на файл i_size и i_blocks. Файлът се съхранява в цяло число блокове и сл., в общия случай i_size<= 512*i_blocks. Но е възможно и обратното - i_size> 512*i_blocks, ако файлът съдържа „дупка".

Описатели на групи блокове (Group descriptors)

Всяка група е описана чрез запис, с размер 32 байта съдържащ:
1- адрес на блок с битовата карта на блоковете за групата;
2- адрес на блок с битовата карта на индексните описатели за групата;
3- адрес на първи блок на индексната област в групата;
4- брой свободни блокове в групата;
5- брой свободни i-node в групата;
6- брой каталози в групата;
7- резервирано поле от 14 байта.

Описателите на всички групи са събрани в областта описатели, копие на която се съдържа във всяка група.

Суперблок

Суперблокът съдържа общите параметри на физическата структура на файловата система. Някои от основните данни, съдържащи се в него, са следните:
1- общ брой блокове - размер на файловата система;
2- общ брой индексни описатели;
3- брой блокове резервирани за администратора (обикновено е 5% от общия брой блокове). Тези резервирани блокове позволяват на администратора да продължи работа, дори когато няма свободни блокове за другите потребители.
4- общ брой свободни блокове и индексни описатели;
5- размер на блок;
6- брой блокове в група;
7- брой i-node в група;
8- полета използвани за автоматична проверка на файловата система (fsck) при boot, напр., дата и време на последен fsck, брой монтирания след последния fsck, максимален брой монтирания преди следващ fsck, максимален интервал време между две изпълнения на fsck.

Каталог

Ограничението за максимална дължина на името на файл е 255 символа, затова записите в каталога са с променлива дължина и съдържат:
1- номер на i-node;
2- дължина на записа;
3- дължина на името на файла;
4- име на файла, съхранявано в толкова байта, колкото са небходими.
5
И така, новото във физическото представяне на ext2 и ext3 в сравнение с файловата система в UNIX е:

• Използване на битови карти при управление на свободните ресурси - блокове и i-node.

• Разделяне на дисковото пространство на групи блокове.

И двете промени, както и съответните промени в стратегиите и алгоритмите, имат за цел да се постигне по-висока степен на локалност на файловете и системните им структури (да се намали разстоянието между i-node и блоковете на файл), а от там и по-добра производителност.

• За системните структури, съдържащи критична за файловата система информация се съхраняват няколко копия.

➢ MSDOS

За файловата система на MSDOS дисковото пространство е последователност от сектори по 512 байта. Броят им зависи от типа на диска. Твърдите дискове могат да се разделят на дялове (partitions). Във всеки дял от диск или цял диск (ако не е разделян или е дискета) се изгражда независима файлова система, наричана том (volume). На Фиг. 14 е изобразено разпределението на дисковото пространство в една файлова система, изградена на един том.

Фиг.14. Разпределение на дисковото пространство в том на MSDOS

Boot сектор съдържа програмата, която стартира зареждането на операционната система в паметта и параметри на тома, като размер на клъстер, размер на тома, размер и брой копия на FAT, размер на коренния каталог. Тази област присъства на всички томове, дори и да не са системни, но при опит да се стартира системата от несистемен диск се извежда съобщение за грешка.

Следващата област е **FAT** (File Allocation Table) или Таблица за разпределение на дисковото пространство. Тя представлява карта на файловете и съдържа цялата информация за дисковата памет, разпределена за файловете, за свободното дисково пространство и за дефектните сектори. От гледна точка на файловата система това са най-важните и критични данни на диска, затова се пазят две копия на FAT, разположени едно след друго.

Следва областта, в която се съхранява коренния каталог на файловата система. Файловата система в MSDOS е йерархична, но за разлика от UNIX, тук всеки том се разглежда като независима дървовидна структура. По тази причина, може би, коренният каталог на всеки диск е разположен в отделна област, след FAT.

В областта данни се съхраняват всички файлове и каталозите, различни от коренния.

FAT - Таблица за разпределение на дисковото пространство

Дискова памет за файлове се разпределя динамично и единицата за разпределение се нарича клъстер (cluster). Клъстерът е последователност от 1 или повече сектора (обикновено степен на двойката брой сектори) и всички клъстери на един том са с еднакъв размер. За различните томове в една система клъстерите може да са с различни размери. (Заб: Клъстерът е аналога на блок)

FAT съдържа елементи (с дължина 12 или 16 бита), като броят им зависи от размера на тома и от размера на клъстера. Елементи 0 и 1 съдържат код, идентифициращ формата на тома. Всеки от останалите елементи съответства позиционно на клъстер от областта за данни. За удобство номерацията на клъстерите започва от 2, т.е. първият клъстер в областта за данни е с номер 2. Съдържанието на елемент от FAT определя състоянието на съответния клъстер - свободен, разпределен за файл или повреден. Код 0 в елемент означава свободен клъстер. Код 2, 3, 4, ..., N (максимален номер на клъстер) означава, че съответният клъстер е разпределен за файл, а числото в елемента е указател към следващ елемент на FAT (което е и адрес на следващ клъстер на файла), т.е. елементът е част от верига елементи на FAT, представяща клъстерите, разпределени за файл. Максималният код, който може да се запише в елемент, означава край на верига, т.е. съответният клъстер е последен във файл. На Фиг.15 е показано примерно съдържание на FAT. Използвани са следните обозначения: EOF - код за последен клъстер (0xFFFF) и FREE - код за свободен клъстер (0).

FAT

X	0
X	1
EOF	2
10	3
2	4
9	5
8	6
FREE	7
4	8
EOF	9
EOF	10
FREE	11
	. . .

Клъстери, разпределени за файловете A, B и C:

A: 6, 8, 4, 2

B: 5, 9

C: 3, 10

Фиг. 15. Таблица FAT в MSDOS

Каталози

Едно от различията при представяне на каталозите в MSDOS е, че коренният каталог е в отделна област и с фиксиран по време на форматирането размер (до 256 записа). Всички останали каталози са разположени в областта за данни, имат променлива дължина и памет за тях се разпределя динамично, както и при обикновените файлове. Но всички каталози имат еднаква структура. Съдържат записи с дължина по 32 байта, всеки от които описва файл или подкаталог. Структурата на запис от каталога е показана на Фиг. 16.

Фиг.16. Структура на запис от каталог в MSDOS

В полетата име (8 байта) и разширение (3 байта) е записано собственото име на файла. Компонентата разширение на името не е задължителна.

В полето флагове (1 байт) се съхраняват различните атрибути-флагове, които определят характеристики на файла

Таблица 3. Значение на битовете в атрибута флагове

7	6	5	4	3	2	1	0	Предназначение
							1	само за четене
						1		скрит файл
					1			системен файл (от CP/M)
				1				етикет на тома
			1					каталог
		1						бит при архивиране

В полето време (2 байта) се съхранява времето на създаване или последно изменение на файла.

Полето дата (2 байта) съдържа датата на създаване или последно изменение на файла.

В полето първи клъстер (2 байта) е записан номера на първия елемент от FAT, от който започва веригата, представяща разпределените за файла клъстери. Ако за файла още не е разпределена памет, полето съдържа 0.

В полето размер (4 байта) е записан размера на файла в брой байта, въпреки че отделената дискова памет може да е повече, тъй като се разпределя в клъстери. Това ограничава размера на файл до 4GB.

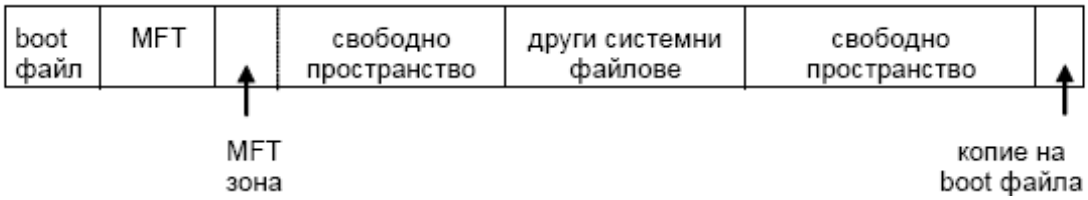
Всички каталози без коренния съдържат двата стандартни записа - с име "." за самия каталог и с име ".." за родителския каталог. Коренният каталог пък съдържа един специален запис за етикета на тома, който е символен низ с дължина до 11 символа и се записва в първите две полета на записа. Това е другата разлика в представянето на коренния и останалите каталози.

➢ NTFS

В NTFS Схемата на разделяне на дялове и изграждане на томове е усъвършенствана. Освен обикновените томове (simple volumes), се реализират многодялови томове (multipartition volumes). Многодялов том означава, че една файлова система се изгражда върху няколко дяла.

Единицата за разпределяне и адресиране на дисковото пространство от NTFS е клъстер. Адресът на клъстер относно началото на тома се нарича **LCN** (Logical cluster number), а адресът в рамките на определен файл се нарича **VCN** (Virtual cluster number).

Една от ключовите новости в NTFS е принципа „всичко на диска е файл": и данните и метаданните се съхраняват в тома като файлове, т.е. на всеки том има обикновени файлове, каталози и системни файлове. Това, че метаданните се съхраняват като файлове, позволява динамично разпределяне на дискова памет при нарастване на метаданните, без да са необходими фиксирани области върху диска за тях. Сърцето на файловата система и главният системен файл е **MFT** (Master File Table).



Файлът MFT представлява индекс на всички файлове на тома. Съдържа записи по 1KB и всеки файл на тома е описан чрез един запис, включително и MFT. Освен MFT има и други файлове с метаданни, чиито имена започват със символа $ и са описани в първите записи на MFT. Самите системни файлове са разположени към средата на тома. В началото на тома е boot файла. Скрито в края на тома е копие на boot файла. За да се намали фрагментирането на MFT файла, се поддържа буфер от свободно дисково пространство - MFT зона, докато останало дисково пространство не се запълни. Размерът на MFT зоната се намалява на половина винаги когато останалата част от тома се запълни.

MFT файл

Всеки файл на тома е описан в поне един запис на MFT файла. Индексът (номерът) на началния MFT запис за всеки файл се използва като идентификатор на файла във файловата система (File reference number). Първите 24 записа са резервирани за системните файлове, т.е. те имат предопределени индекси. Следва списък на част от тези файлове.

Име на файл	Индекс	Описание на съдържанието на файла
$Mft	0	MFT файл
$MftMirr	1	Копие на първите записи от MFT файла
$LogFile	2	Журнал при поддържане на транзакции
$Volume	3	Описание на тома
$AttrDef	4	Дефиниции на атрибутите
\	5	Коренен каталог на тома
$Bitmap	6	Битова карта на тома
$Boot	7	Boot сектори на тома
$BadClus	8	Списък на лошите клъстери на тома

Адресът на MFT файла (на началото му) се намира в Boot файла. Първият запис в MFT файла описва самия файл и следователно съдържа адресна информация, осигуряваща достъп до целия MFT файл, ако той е фрагментиран и заема няколко области на тома. Файлът $MftMirr съдържа копие на първите няколко (16) записа от $Mft. Целта на това дублиране е по-висока надеждност на файловата система. Файлът $LogFile се използва за поддържане на транзакции при манипулиране структурата на файловата система. Всяка

последователност от дискови операции, които реализират една операция на файловата система, като създаване, преименуване, унищожаване на файл и др., представлява транзакция. NTFS създава записи за тези операции в $LogFile. Тези записи се използват за възстановяване на коректността на файловата система след сривове. Действието на всички незавършили транзакции се отменя и файловата система се възстановява до състоянието си преди началото на тези транзакции. Файлът $Volume съдържа информация за тома, като сериен номер и име на тома, версия на NTFS, дата на създаване и др. Файлът $Bitmap представлява битова карта на клъстерите на тома. Причината и boot сектора да е файл, е може би за да се спази правилото всичко на диска е файл. Всички повредени сектори на тома са организирани във файл на лошите клъстери на тома $BadClus.

Атрибути на файл

Всеки файл се съхранява като съвкупност от двойки „атрибут/значение". Един от атрибутите са данните на файла (наричан unnamed data attribute). Други атрибути са име на файл, стандартна информация и други. Всеки атрибут се съхранява като отделен поток от байтове. Обикновен файл може да има и други атрибути данни, наричани named data attribute. Това променя представата ни за файл, като една последователност от байтове, т.е. файлът може да има няколко независими потока данни. Следва списък на част от типовете атрибути (общият им брой е около 14).

Име на тип атрибут	Описание на значението на атрибута
$FILE_NAME	Името на файла. Един файл може да има няколко атрибута от този тип, при твърди връзки или ако се генерира кратко име в MSDOS стил.
$STANDARD_INFORMATION	Атрибути на файл, като флагове, време и дата на създаване и последно изменение, брой твърди връзки.
$DATA	Данните на файла. Всеки файл има един неименован атрибут данни и може да има допълнителни именовани атрибути данни.
$INDEX_ROOT, $INDEX_ALLOCATION, $BITMAP	Три атрибута използвани при реализацията на каталозите.
$ATTRIBUTE_LIST	Този атрибут се използва, когато за файл има повече от един запис в MFT. Съдържа списък от атрибутите на файла и индексите на записите, където се съхраняват.
$VOLUME_NAME, $VOLUME_INFORMATION	Тези атрибути се използват само в файла $Volume. Те съхраняват иформация за тома, като етикет, версия.

Всеки тип атрибут освен име на типа, има и **числов код** на типа. Този код се използва при наредба на атрибутите в MFT запис на файла. MFT записът съдържа числовия код на атрибута (който го идентифицира), значение на атрибута и евентуално име на атрибута. Значението на всеки атрибут е поток от байтове. Един файл може да има няколко атрибута от един тип, например няколко $FILE_NAME или $DATA атрибута.

Атрибутите биват **резидентни** и **нерезидентни**. Резидентен е атрибут, който се съхранява изцяло в MFT записа. Някои атрибути са винаги резидентни, например $FILE_NAME, $STANDARD_INFORMATION, $INDEX_ROOT. Ако значението на атрибут, като данните на голям файл, не може да се съхрани в MFT записа, то за него се разпределят клъстери извън MFT записа. Такива атрибути се наричат нерезидентни. Файловата система решава как да съхранява един атрибут. Нерезидентни могат да бъдат само атрибути, чиито значения могат да нарастват, например $DATA.

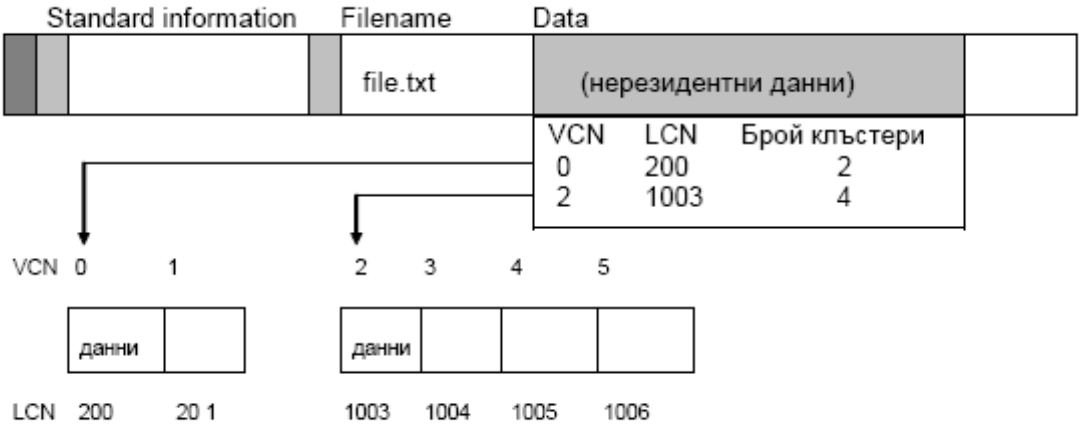
MFT запис

Всеки MFT запис съдържа заглавие на записа **(record header)** и атрибути на файла. Всеки атрибут се съхранява като заглавие на атрибута **(attribute header)** и данни. Заглавието на атрибута съдържа код на типа, име, флагове на атрибута и информация за разположението на данните му. Един атрибут е резидентен ако данните му се поместват в един запис заедно със заглавията на всички атрибути. На Фиг.20 е изобразена структурата на MFT запис за малък файл, т.е. всичи атрибути на файла могат да се съхранят в MFT записа.

Фиг.20. MFT запис за файл само с резидентни атрибути

Заб: Може ли без разясненията за файл с нерезидентни атрибути

Ако един атрибут не е резидентен, заглавието му (което е винаги резидентно) съдържа информация за клъстерите, разпределени за данните му. Адресната информация се съхранява подобно на подхода в HPFS, т.е. представлява последователност от описания на екстенти (run/extent entry). Всеки екстент е непрекъсната последователност от клъстери, разпределени за данните на съответния атрибут и се описва от адрес на началния клъстер и дължина (VCN, LCN, брой клъстери). За разлика от HPFS, тук описанията на екстентите се съхраняват в последователна структура, а не в B+дърво.
На Фиг.21 е изобразена структурата на MFT запис за по-голям файл, т.е. данните на файла се съхранят в два екстента.

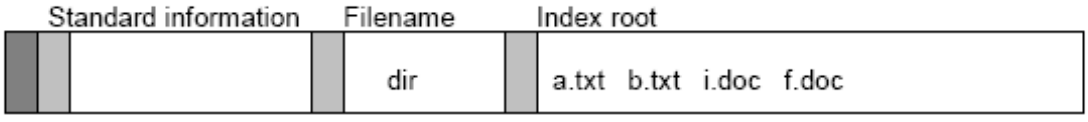


Фиг.21. MFT запис за файл с нерезидентен атрибут данни

Каталози

Каталогът съдържа записи с променлива дължина, всеки от които съответства на файл или подкаталог, в съответния каталог. Всеки запис съдържа името на файла и индексът на основния MFT запис на файла, както и копие на стандартната информация на файла. Това дублиране на стандартната информация изисква две операции писане при изменението ѝ, но ускорява извеждането на справки за съдържанието на каталога.

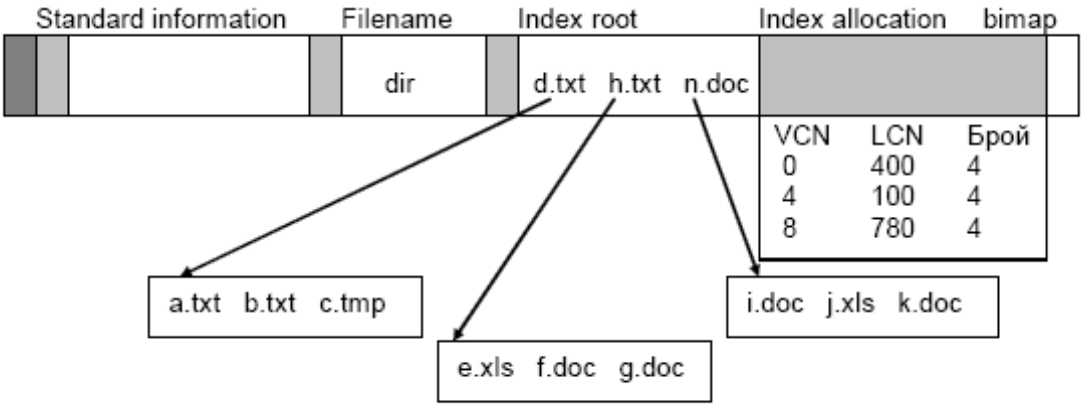
Записите в каталога са сортирани по името на файла и се съхраняват, подобно на HPFS, в структура B дърво. Ако каталогът е малък, тези записи се съхраняват в атрибута $INDEX_ROOT, който е резидентен, т.е. целият каталог се намира в MFT записа си. На Фиг.23 е изобразена структурата на MFT запис за малък каталог.



Фиг.23. MFT запис за малък каталог

Заб: Може ли без разясненията за представяне на голям каталог?

Когато каталогът стане голям, за него се разпределят екстенти с размер 4KB, наречени индексни буфери (index buffers). Атрибутът $INDEX_ROOT и тези екстенти са организирани в B дърво. В този случай каталогът има и атрибут $INDEX_ALLOCATION, който съхранява адресна информация за разположението на екстентите-индексни буфери. Атрибутът $BITMAP е битова карта за използването на клъстерите в индексните буфери. Фиг.24 илюстрира представянето на голям каталог (за простота всеки клъстер съдържа 1 запис).



Фиг.24. Представяне на голям каталог