Università di Udine — Facoltà di Scienze MM.FF.NN.

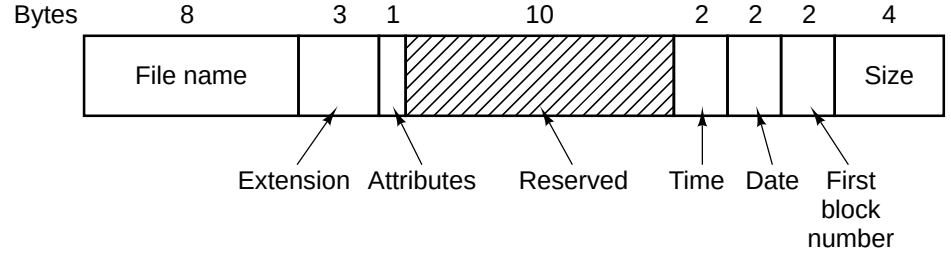
Laurea in Informatica — A.A. 2005/06

Trasparenze del Corso di *Sistemi Operativi*

Ivan Scagnetto

Università di Udine

Directory in MS-DOS



- Lunghezza del nome fissa
- Attributi: read-only, system, archived, hidden
- Reserved: non usati
- Time: ore (5bit), min (6bit), sec (5bit)
- Date: giorno (5bit), mese (4bit), anno-1980 (7bit) (Y2108 BUG!)

FAT12, FAT16, FAT32

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

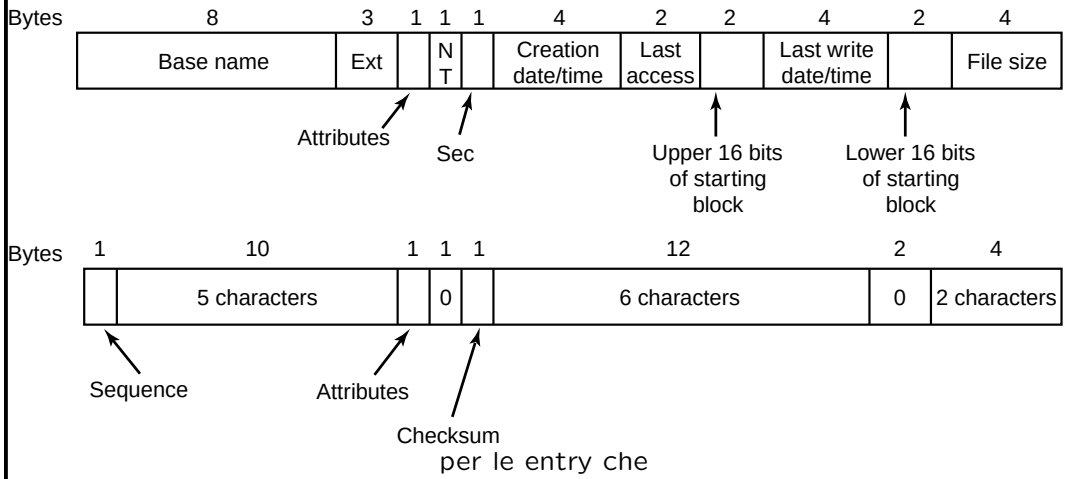
In MS-DOS, tutta la FAT viene caricata in memoria.

Il block size è chiamato da Microsoft *cluster size*

Limite superiore: 2^{32} blocchi da 512 byte = 2TB

Directory in Windows 98

Nomi lunghi ma compatibilità all'indietro con MS-DOS e Windows 3



per le entry che

rappresentano nomi lunghi, attributes=0x0F (valore invalido per MS-DOS)

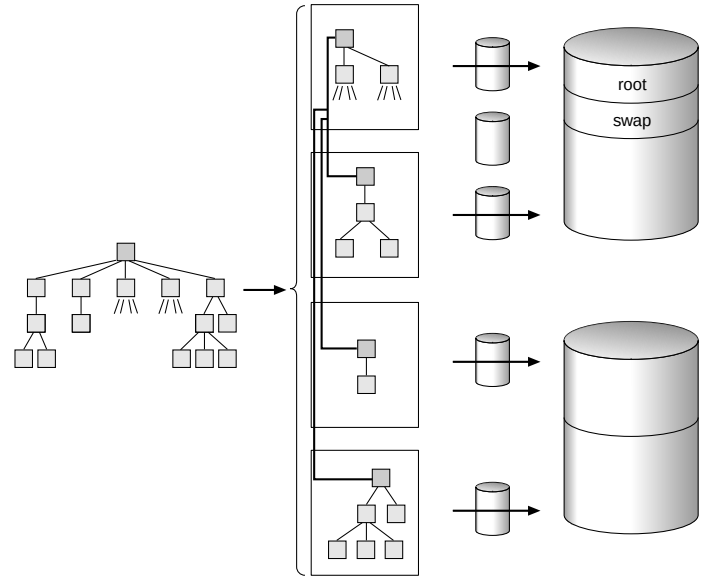
Esempio

```
$ dir
THEQUI~1        749 03-08-2000  15:38 The quick brown fox jumps over the...
```

Bytes	T	H E Q U I ~ 1	A	NT	S	Creation time	Last acc	Upp	Last write	Low	Size
68	d o g		A	0	CK					0	
3	o v e		A	0	CK	t h e			l a	0	z y
2	w n	f o	A	0	CK	x	j u	m	p	0	s
1	T h e	q	A	0	CK	u i c k			b	0	r o

UNIX: Il Virtual File System

Il file system *virtuale* che un utente vede può essere composto in realtà da diversi file system *fisici*, ognuno su un diverso dispositivo logico



Il Virtual File System (cont.)

- Il Virtual File System è composto da più file system fisici, che risiedono in dispositivi logici (*partizioni*), che compongono i dispositivi fisici (dischi)
- Il file system / viene montato al boot dal kernel
- gli altri file system vengono montati secondo la configurazione impostata
- ogni file system fisico può essere diverso o avere parametri diversi
- Il kernel usa una coppia <*logical device number, inode number*> per identificare un file
  - Il logical device number indica su quale file system fisico risiede il file
  - Gli inode di ogni file system sono numerati progressivamente

Il Virtual File System (cont.)

Il kernel si incarica di implementare una visione uniforme tra tutti i file system montati: operare su un file significa

- determinare su quale file system fisico risiede il file
- determinare a quale inode, su tale file system corrisponde il file
- determinare a quale dispositivo appartiene il file system fisico
- richiedere l'operazione di I/O al dispositivo

I File System Fisici di UNIX

- UNIX (Linux in particolare) supporta molti tipi di file system fisici: SYSV, UFS, EFS, EXT2, MSDOS, VFAT, ISO9660, HPFS, HFS, NTFS, ...
- Quelli preferiti sono UFS (Unix File System, aka BSD Fast File System), EXT2 (Extended 2), EFS (Extent File System) e i *journalled file systems* (JFS, XFS, EXT3, ...)
- Il file system fisico di UNIX supporta due oggetti:
  - file "semplici" (plain file) (senza struttura)
  - directory (che sono semplicemente file con un formato speciale)
- La maggior parte di un file system è composta da blocchi dati
  - in EXT2: 1K-4K (configurabile alla creazione)
  - in SYSV: 2K-8K (configurabile alla creazione)

Inodes

- Un file in Unix è rappresentato da un *inode* (nodo indice).
- Gli inodes sono allocati in numero finito alla creazione del file system
- Struttura di un inode in System V:

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Inodes (cont)

- I timestamp sono in POSIX "epoch": n. di secondi dal 01/01/1970, UTC. (Quindi l'epoca degli Unix a 32 bit dura 2^{31} secondi, ossia fino alle 3:14:07 UTC di martedì 19 gennaio 2038).
- Gli indici indiretti vengono allocati su richiesta
- Accesso più veloce per file piccoli
- N. massimo di blocchi indirizzabile: con blocchi da 4K, puntatori da 4byte

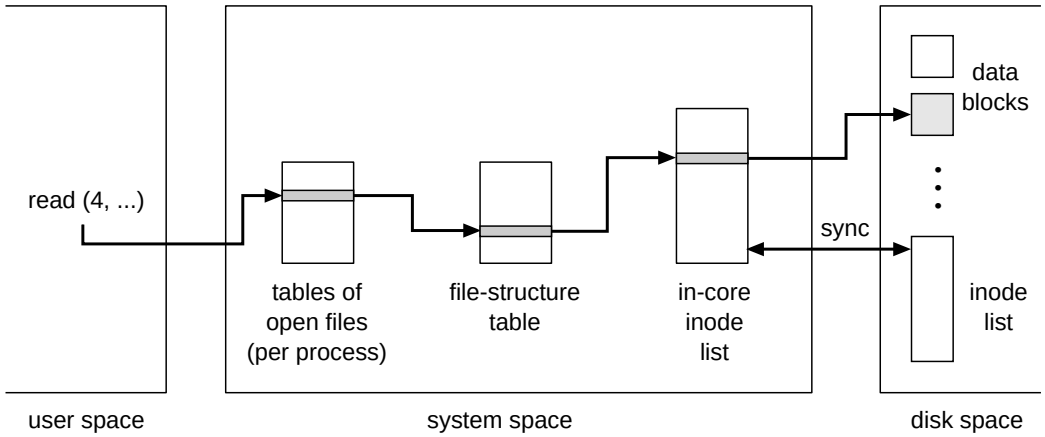
$$\begin{aligned} L_{max} &= 10 + 1024 + 1024^2 + 1024^3 \\ &> 1024^3 = 2^{30} blk \\ &= 2^{42} byte = 4\,TB \end{aligned}$$

  molto oltre le capacità dei sistemi a 32 bit.

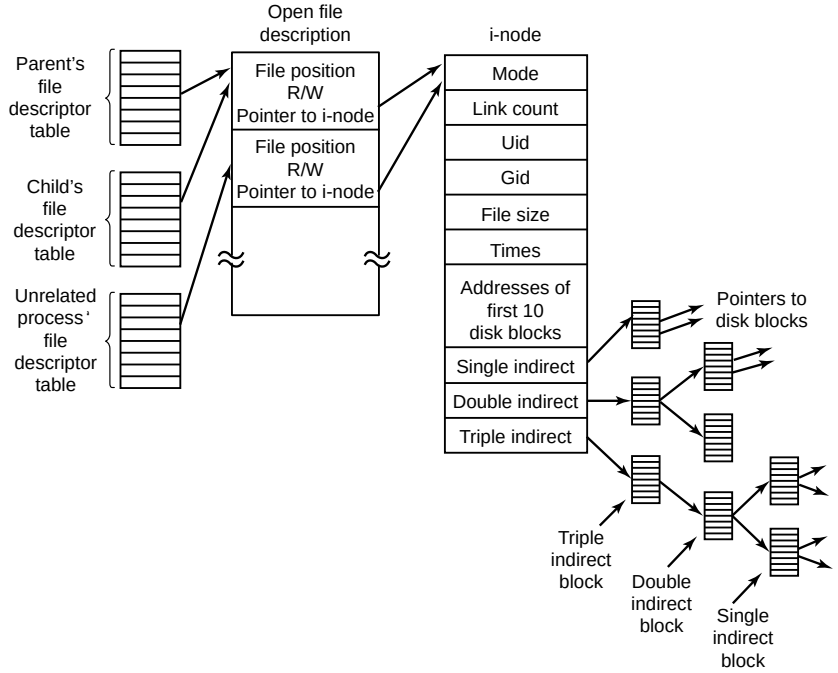
Traduzione da file descriptor a inode

- Le system calls che si riferiscono a file aperti (read, write, close, ...) prendono un *file descriptor* come argomento
- Il file descriptor viene usato dal kernel per entrare in una tabella di file aperti del processo. Risiede nella U-structure.
- Ogni entry della tabella contiene un puntatore ad una *file structure*, di sistema. Ogni file structure punta ad un inode (in un'altra lista), e contiene la posizione nel file.
- Ogni entry nelle tabelle contiene un contatore di utilizzo: quando va a 0, il record viene deallocato

File Descriptor, File Structure e Inode



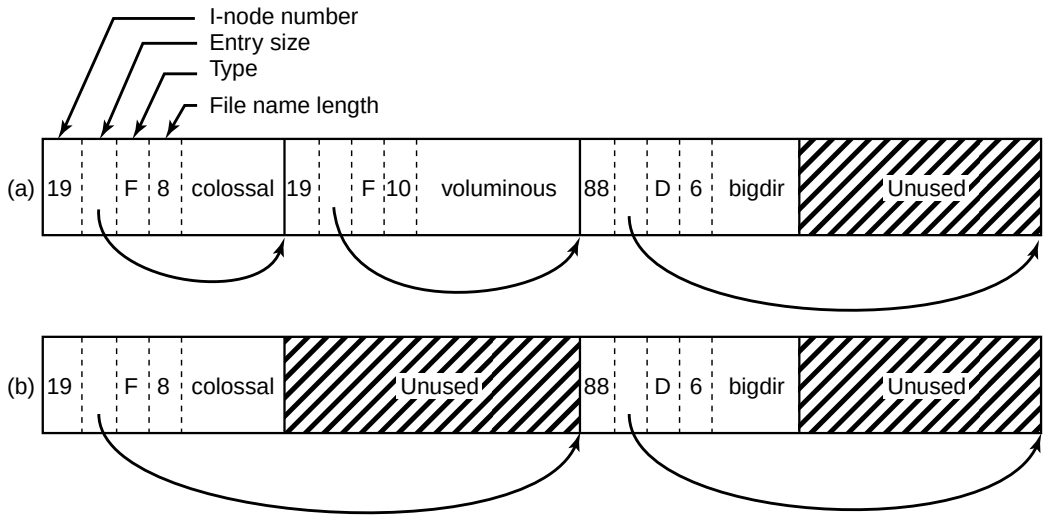
La tabella intermedia è necessaria per la semantica della condivisione dei file tra processi



- Le chiamate di lettura/scrittura e la seek cambiano la posizione nel file
- Ad una *fork*, i figli ereditano (una copia de) la tabella dei file aperti dal padre $\Rightarrow$ condividono la stessa file structure e quindi la posizione nel file
- Processi che hanno aperto indipendentemente lo stesso file hanno copie private di file structure

Directory in UNIX

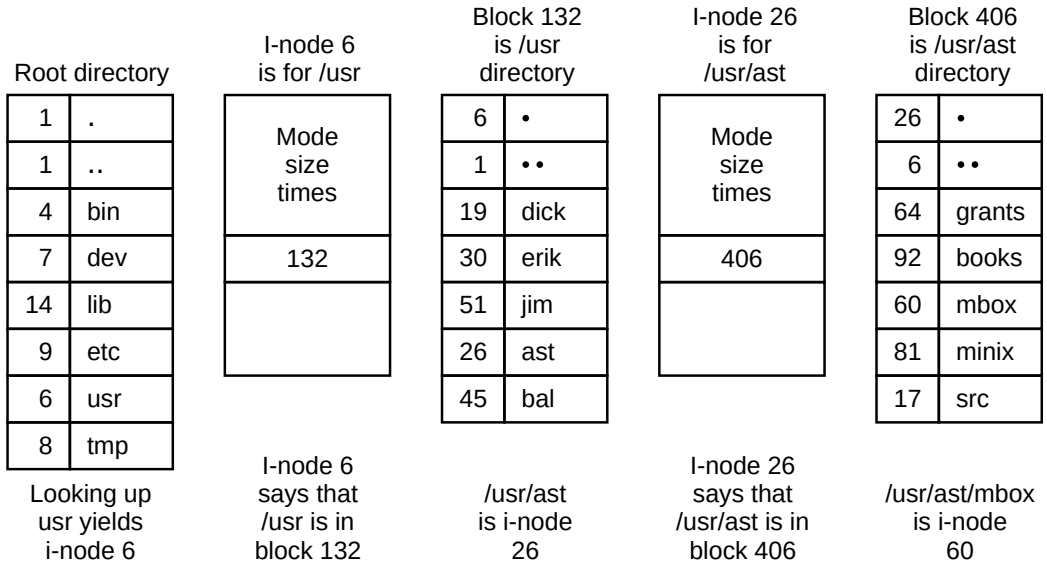
- Il tipo all'interno di un inode distingue tra file semplici e directory
- Una directory è un file con entry di lunghezza variabile. Ogni entry contiene
  - puntatore all'inode del file
  - posizione dell'entry successiva
  - lunghezza del nome del file (1 byte)
  - nome del file (max 255 byte)
- entry differenti possono puntare allo stesso inode (*hard link*)



Traduzione da nome a inode

L'utente usa i nomi (o path), mentre il file system impiega gli inode ⇒ il kernel deve risolvere ogni nome in un inode, usando le directory

- Prima si determina la directory di partenza: se il primo carattere è "/", è la root dir (sempre montata); altrimenti, è la current working dir del processo in esecuzione
- Ogni sezione del path viene risolta leggendo l'inode relativo
- Si ripete finché non si termina il path, o la entry cercata non c'è
- Link simbolici vengono letti e il ciclo di decodifica riparte con le stesse regole. Il numero massimo di link simbolici attraversabili è limitato (8)
- Quando l'inode del file viene trovato, si alloca una *file structure* in memoria, a cui punta il *file descriptor* restituito dalla *open(2)*



Esempio di file system fisico: Unix File System

- In UFS (detto anche Berkeley Fast File System), i blocchi hanno due dimensioni: il *blocco* (4-8K) e il *frammento* (0.5-1K)
  - Tutti i blocchi di un file sono blocchi tranne l'ultimo
  - L'ultima parte del file è tenuta in frammenti, q.b.
  - Es: un file da 18000 byte occupa 2 blocchi da 8K e 1 da 2K (non pieno)
- Riduce la frammentazione interna e aumenta la velocità di I/O
- La dimensione del blocco e del frammento sono impostati alla creazione del file system:
  - se ci saranno molti file piccoli, meglio un fragment piccolo
  - se ci saranno grossi file da trasferire spesso, meglio un blocco grande
  - il rapporto max è 8:1. Tipicamente, 4K:512 oppure 8K:1K.

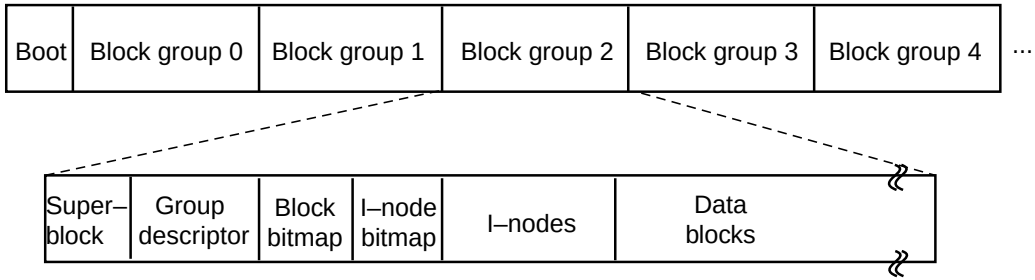
Esempio di file system fisico: Unix File System (Cont)

- Si introduce una cache di directory per aumentare l'efficienza di traduzione
- Suddivisione del disco in *cilindri*, ognuno dei quali con il proprio superblock, tabella degli inode, dati. Quando possibile, si allocano i blocchi nello stesso gruppo dell'inode.

  In questo modo si riduce il tempo di seek dai metadati ai dati.

  - numero di gruppi
  - numero di blocchi liberi e inodes liberi,...
- Ogni gruppo ha una copia del superblock, la propria tabella di inode e tabelle di allocazione blocchi e inode
- Per minimizzare gli spostamenti della testina, si cerca di allocare ad un file blocchi dello stesso gruppo

Esempio di file system fisico: EXT2



- Derivato da UFS, ma con blocchi tutti della stessa dimensione (1K-4K)
- Suddivisione del disco in *gruppi* di 8192 blocchi, ma non secondo la geometria fisica del disco
- Il *superblock* (blocco 0) contiene informazioni vitali sul file system
  - tipo di file system
  - primo inode

NTFS: File System di Windows NT/2K/XP

- Un file è un oggetto strutturato costituito da *attributi*.
- Ogni attributo è una sequenza di byte distinta (*stream*), come in MacOS. Ogni stream è in pratica un file a se stante (con nome, dimensioni, puntatori ecc.).
- L'indirizzamento è a 64 bit.
- Tipicamente, ci sono brevi stream per i metadati (nome, attributi, Object ID) e un lungo stream per i veri dati, ma nulla vieta avere più stream di dati.
- I nomi sono lunghi fino a 255 caratteri Unicode.

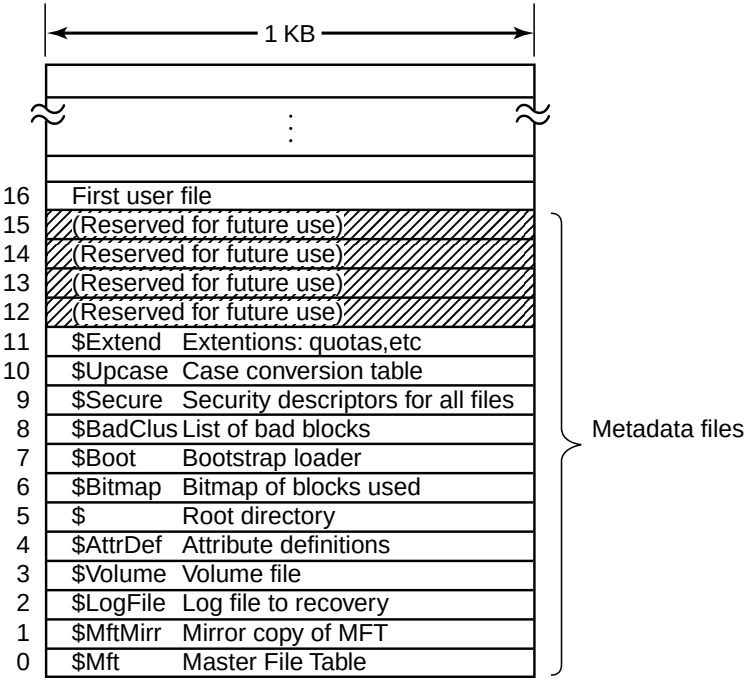
Struttura di NTFS

- Creato da zero, incompatibile all'indietro.
- Diverse partizioni possono essere unite a formare un *volume logico*
- Ha un meccanismo transazionale per i metadati (logging)
- Lo spazio viene allocato a *cluster:* potenze di 2 dipendenti dalla dimensione del disco (tra 512byte e 4K). All'interno di un volume, ogni cluster ha un *logical cluster number*.

Master File Table (MFT)

- È un file di record di 1K, ognuno dei quali descrive un file o una directory
- Può essere collocato ovunque, sul disco, e crescere secondo necessità
- Il primo blocco è indicato nel boot block.
- Le prime 16 entry descrivono l'MFT stesso e il volume (analogo al superblock di Unix). Indicano la posizione della root dir, il bootstrap loader, il logfile, spazio libero (gestito con una bitmap)...
- Ogni record successivo è uno header seguito da una sequenza di coppie (attributo header,valore). Ogni header contiene il tipo dell'attributo, dove trovare il valore, e vari flag.

  I valori possono seguire il proprio header (*resident attribute*) o essere memorizzati in un blocco separato (*nonresident attribute*)



Attributi dei file NTFS

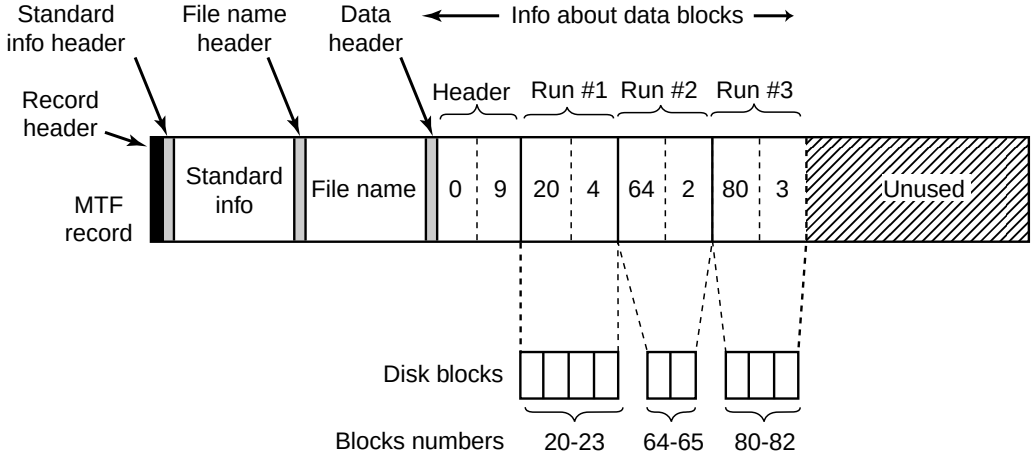
NTFS definisce 13 attributi standard

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

I Data contenono i veri dati; se sono residenti, il file si dice "immediate".

File NTFS non residenti

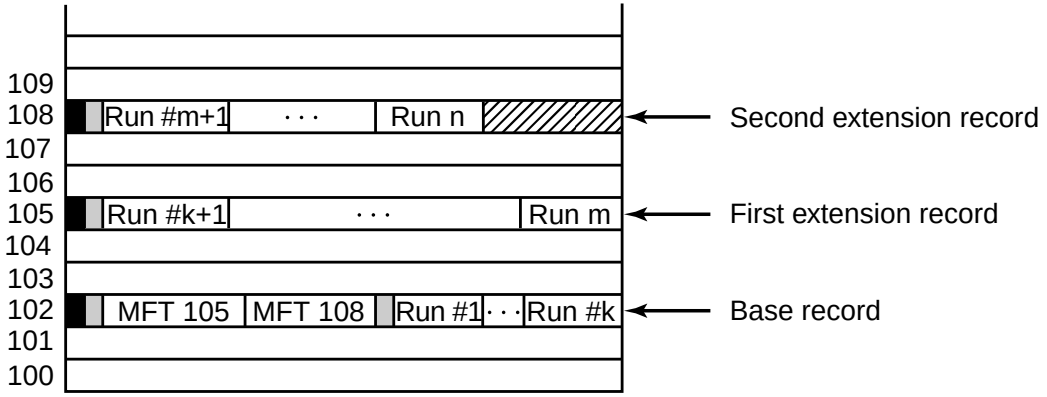
I file non immediati si memorizzano a "run": sequenze di blocchi consecutivi. Nel record MFT corrispondente ci sono i puntatori ai primi blocchi di ogni run.



Un file descritto da un solo MFT record si dice *short* (ma potrebbe non essere corto per niente!)

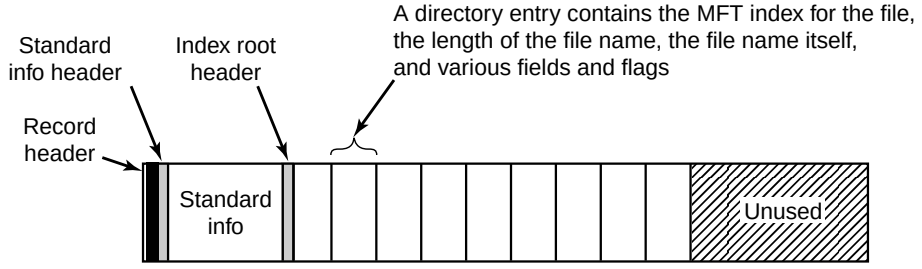
File "long"

Se il file è lungo o molto frammentato (es. disco frammentato), possono servire più di un record nell'MFT. Prima si elencano tutti i record aggiuntivi, e poi seguono i puntatori ai run.



Directory in NTFS

Le directory corte vengono implementate come semplici liste direttamente nel record MFT.



Directory più lunghe sono implementate come file nonresident strutturati a B+tree.

I passi per usare un disco sotto Unix

La prima volta:	1. Collegarlo alla macchina :–) 2. Partizionamento: *fdisk*, *cfdisk*, *druid*, . . . 3. Creazione dei file system: *mkfs*, *newfs*
A regime:	4. Mounting dei file system: *mount* 5. Buon Lavoro! 6. Unmounting: *umount*
Manutenzione:	7. Verifica e correzione dei problemi: *fsck* 8. Ottimizzazione (molto rara): *tunefs*. La deframmentazione non è necessaria!

Perché partizionare un disco

Gli HD vengono partizionati per diversi motivi:

Usi differenti: diversi file system, area di swap, altri S.O....

Robustezza: se una partizione si danneggia, può essere riformattata senza modificare le altre

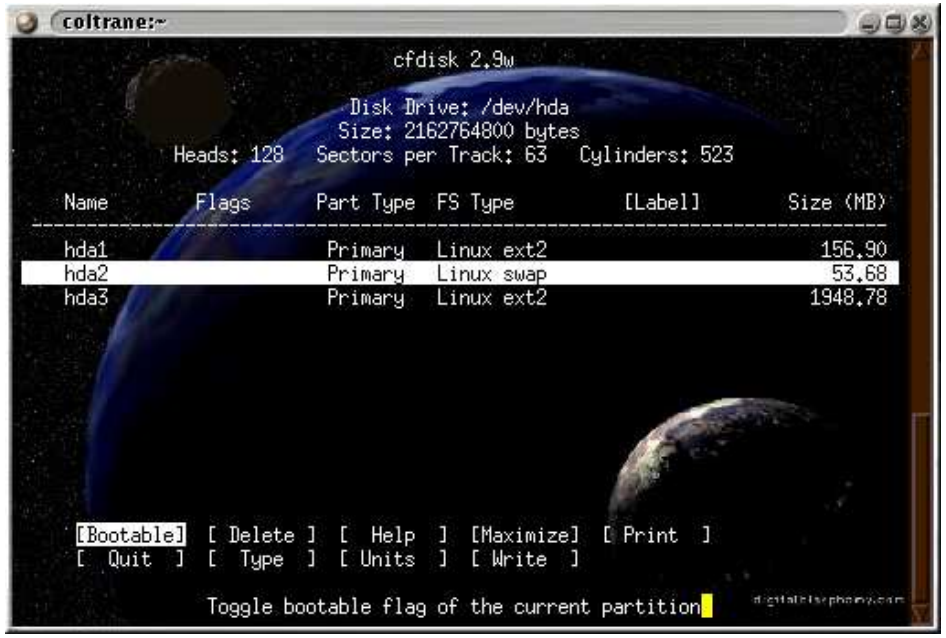
Confinamento: i file non possono superare i confini della partizione

Maggiore velocità al reboot e al backup

Partizionare un disco: fdisk

```
[root@coltrane miculan]# fdisk /dev/hda

Command (m for help): p

Disk /dev/hda: 128 heads, 63 sectors, 523 cylinders
Units = cylinders of 8064 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        38    153184+  83  Linux
/dev/hda2            39        51     52416   82  Linux swap
/dev/hda3            52       523   1903104   83  Linux

Command (m for help):
```


Partizionare un disco: cfdisk



Creazione di un file system

```
mkfs [-t fstype] [fs-options] device [size]
```

Diversi file system supportati: UFS, EXT2, SYSV, VFAT, ...

Esempi di opzioni (per EXT2 e UFS)

-b block-size dimensione del bocco

-f fragment-size dimensione del fragment (solo UFS)

-i bytes-per-inode densità degli inode (default: 4K/inode)

-m reserved-percentage riserva per il sistemista (default: 5%)

-R raid-opts supporto per RAID

Sotto Solaris, si può usare anche *newfs* (più semplice).

```
miculan@coltrane:miculan$ mkfs /dev/fd0
mke2fs 1.12, 9-Jul-98 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```


Montare il file system: mount(1m)

```
mount [-t fstype] [-o opzione] device mountpoint
```

device è un dispositivo a blocchi, *mountpoint* è il path assoluto di una directory (già esistente).
Senza argomenti: mostra i file system montati correntemente:

```
miculan@coltrane:Slides$ mount
/dev/hda1 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/hda5 on /usr type ext2 (rw)
ten:/var/spool/mail on /var/spool/mail
       type nfs (rw,bg,actimeo=0,soft,addr=158.110.144.132)
ten:/user/ospiti on /user/ospiti
       type nfs (rw,addr=158.110.144.132)
```


Il VFS: la tabella /etc/fstab (o /etc/vfstab)

```
#device      mount         FS      mount       dump? fsck
#to mount    point         type    options           pass
#
/dev/hda1    /             ext2    defaults        1 1
/dev/hda3    /usr          ext2    defaults        1 2
/dev/hda2    swap          swap    defaults        0 0
/dev/hdc     /home         ext2    defaults        1 2
/dev/fd0     /mnt/floppy   vfat    noauto,user     0 0
/dev/sda4    /mnt/zip      vfat    noauto,user     0 0
/dev/cdrom   /mnt/cdrom    iso9660 noauto,ro,user  0 0
none         /proc         proc    defaults        0 0
none         /dev/pts      devpts  gid=5,mode=620  0 0
ten:/var/mail /var/spool/mail nfs  defaults        0 0

(defaults = rw,suid,dev,exec,auto,nouser,async)
```

Al boot, viene eseguito *mount -a*: tutti i fs "auto" sono montati.

Mounting (2)

In generale, solo root può montare file system.

Il file system da montare deve essere in uno stato consistente (ossia, deve essere stato smontato correttamente—altrimenti, si veda *fsck*)

La directory coperta diventa inaccessibile (*overlay*).

```
[root@coltrane /mnt]# mount
/dev/hda1 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/hda5 on /usr type ext2 (rw)
[root@coltrane /mnt]# ls -al /mnt/floppy
drwxrwxr-x   2 root     root     1024 Feb  6  1996 .
drwxr-xr-x   6 root     root     1024 Oct  9  1998 ..
-rw-r--r--   1 root     root       15 Apr 13 14:50 pippo
[root@coltrane /mnt]#
```

Mounting (3)

```
[root@coltrane /mnt]# mount /dev/fd0 /mnt/floppy
[root@coltrane /mnt]# mount
/dev/hda1 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/hda5 on /usr type ext2 (rw)
/dev/fd0 on /mnt/floppy type ext2 (rw)
[root@coltrane /mnt]# ls -al floppy
total 14
drwxr-xr-x   3 miculan  ospiti   1024 Apr 13 12:49 .
drwxr-xr-x   6 root     root     1024 Oct  9  1998 ..
drwxr-xr-x   2 root     root    12288 Apr 13 12:49 lost+found
[root@coltrane /mnt]#
```

Ora il file system del disco è integrato nel file system logico.

Smontare un file system

Prima di spegnere la macchina, o di togliere il dispositivo (floppy, zip,...), il suo fs deve essere staccato dal file system logico (pena la perdita di dati!):

`umount <device>|<mountpoint>`

Il file system viene sincronizzato e marcato "consistente", prima di essere staccato. Allo shutdown, viene automaticamente eseguito `umount -a`.

Attenzione: un file system non può essere smontato se è "busy" (attivo), ossia se nel kernel sono presenti inode di tale file system sono. Ciò succede quando

- sono ancora aperti dei file di tale fs, oppure
- uno o più processi hanno la CWD in tale fs.

Controllare il file system: df(1m)

df dà informazioni sull'uso dei file system montati:

```
miculan@coltrane:hfs$ df -k
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/hda1              46815   36664     7734     83%   /
/dev/hda5            1930659  834155   996714     46%   /usr
ten:/var/spool/mail   567583  153597   357236     30%   /var/spool/mail
ten:/user/ospiti      962983  699454   215384     76%   /user/ospiti
/dev/fd0                1390      13     1305      1%   /mnt/floppy
miculan@coltrane:hfs$ df -i
Filesystem           Inodes   IUsed   IFree  %IUsed Mounted on
/dev/hda1             12096    3039    9057    25%  /
/dev/hda5            499712   52532  447180    11%  /usr
ten:/var/spool/mail       0       0       0     0%  /var/spool/mail
ten:/user/ospiti          0       0       0     0%  /user/ospiti
/dev/fd0                360      11     349     3%  /mnt/floppy
miculan@coltrane:hfs$
```


Controllare e riparare un file system: fsck

fsck permette di controllare e riparare i file system *non* montati

- a mano, con il comando `fsck [opzioni] <device>`
- automaticamente al boot, se indicato da /etc/fstab o se il fs non è stato smontato correttamente (e.g., crash della macchina).

I principali controlli che fsck esegue sono:

- blocchi allocati da più inodes
- blocchi allocati ma oltre i limiti del file system
- blocchi non allocati né presenti sulla lista libera
- blocchi allocati ma presenti sulla lista libera
- numero errato di link negli inode
- mancanza di "." e ".." nelle directory
- checksum del superblock
- numero di blocchi per gli inode errato
- numero di blocchi/inode liberi errato

Esempi di fsck

```
miculan@coltrane:miculan$ fsck /dev/fd0                ( Linux )
e2fsck 1.12, 9-Jul-98 for EXT2 FS 0.5b, 95/08/09
/dev/fd0: clean, 11/360 files, 63/1440 blocks


root@bodoni$ fsck /dev/rdsk/c0t0d0s3                   ( Solaris )
** /dev/rdsk/c0t0d0s3
** Last Mounted on /export
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
9447 files, 2760313 used, 665249 free (1313 frags, 82992 blocks,
                                       0.0% fragmentation)
```


Riparare file system: fsck

Quando *fsck* trova dei problemi, chiede se deve ripararli. Esempi:

I=1892371: Incorrect link count (fixed) l'inode era puntato da un numero di directory diverso da quello segnato

I=128731: unreferenced inode l'inode non era puntato da nessuna directory, e quindi è stato liberato

Succedono quando il file system è stato smontato malamente (e.g., crash della macchina): lo stato degli inode in memoria non è stato `sync`-ronizzato con quello su disco, e quindi il file viene perduto.

Se il superblock è danneggiato, fsck può sostituirlo con una sua copia:

```
$ fsck -o b=32  /dev/hda
```