Architettura di NTFS – 1

- **NT 5.x** supporta l'intera gamma dei FS Windows e **anche** `ext2fs` di GNU/Linux
  - **FAT-16**
    - Limite **logico** all'ampiezza di partizione
    - ≤ 2^{16} **blocchi** di ampiezza massima 32 KB → 2 GB
  - **FAT-32**
    - Limite **fisico** all'ampiezza di partizione
    - ≤ 2^{32} **settori** da 512 B → 2 TB
    - Limite logico : 2^{28} blocchi da 8 KB → 2 TB
  - **NTFS**
    - **Nuova concezione** con indici espressi su 64 *bit*

Architettura di NTFS – 2

- Nome di *file* fino a 255 caratteri in codifica **UNICODE** (2 – 4 B/carattere)
  - Un nome espresso come **cammino** (relativo o assoluto) non può eccedere (32 K – 1) caratteri
  - Distinzione tra maiuscolo e minuscolo ma **senza effetto** per buona parte di `Win32 API` (*backward compatibility*)
- *File* come **aggregato di attributi** rappresentati come sequenza di caratteri (*byte stream*)
  - Esempio: sequenza breve contenente il nome del *file* e l'indirizzo dell'oggetto associato (+ *thumbnail preview*) + sequenza lunga (fino a 2^{64} B !) contenente i dati del *file*
  - Idea copiata da Apple ☺ introdotta (anche) per compatibilità

Architettura di NTFS – 3

- FS ad **architettura gerarchica** (come `ext2`)
  - \ invece di / come separatore nelle espressioni di cammino
  - Supporto per *directory* corrente (`wd`)
  - Supporto per entrambe le forme di `link`
- Servizi di FS resi tramite procedure di libreria `Win32 API`
  - Funzionalmente simili a GNU/Linux ma di concezione assai più bizantina

Architettura di NTFS – 4

- NTFS è una collezione di **volumi logici**
  - Un volume logico può mappare su **più** partizioni e anche su **più** dischi
  - Volume = sequenza lineare di blocchi (*cluster*) di ampiezza fissa
    - Volumi **diversi** possono avere dimensione di blocco **diverse**
      - Tra 512 B e 4 KB
      - Teoricamente fino a 64 KB
    - Blocco piccolo → ridotta frammentazione interna
    - Blocco grande → meno accessi a disco ma più frammentazione interna
- Una **MFT** (`Master File Table`) per volume
  - **Fisicamente** realizzata come un *file*
    - Perciò può essere salvata **ovunque** nel volume
      - Soluzione più robusta
  - **Logicamente** strutturata come una **sequenza lineare** di ≤ 2^{48} *record* di ampiezza da 1 a 4 KB
    - Ciascun *record* descrive 1 *file* identificato da un indice ampio 48 *bit*
    - Gli altri 16 *bit* servono come numero di sequenza (contatore di riuso)

Architettura di NTFS – 5

- Ciascun *record* contiene un numero **variabile** di coppie `<descrittore di attributo, valore>`
  - Il 1° campo della coppia specifica la **struttura** dell'attributo
    - Esistono **13 attributi di base** con struttura predefinita
    - Possono esistere altri **attributi aggiuntivi** a struttura libera
  - Il 2° campo denota il **valore** dell'attributo
    - Se possibile il valore è rappresentato interamente nel *record*
      - Attributo **residente**
    - Altrimenti rappresentato da un puntatore al suo *record* remoto
      - Attributo **non residente**
  - Il valore dell'attributo **dati** rappresenta il contenuto del *file*

Architettura di NTFS – 6

- Il campo `<descrittore di attributo>` per **attributi residenti** ha ampiezza 24 B
  - Quello per **attributi non residenti** è più ampio
- I 13 attributi di sistema non applicano a tutti i *file*
  - Gli attributi dei *file* corrispondono a quelli che GNU/Linux pone negli *i-node*
  - Con l'aggiunta dell'**identificatore dell' oggetto** corrispondente
    - 64 *bit* con valore **unico per volume**
  - Il contenuto dati di *file* di ampiezza < 1 KB viene memorizzato **interamente** entro un *record* di MFT (attributo residente)
    - ***Immediate file*** (rari)
  - Per *file* più grandi il valore dell'attributo dati diventa la lista ordinata dei corrispondenti blocchi su disco
    - Attributi non residenti

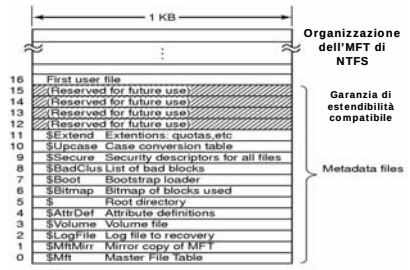
Architettura di NTFS – 7

- Il valore di **attributi non residenti** è posto su sequenze di blocchi contigui
  - NTFS cerca di assegnare allo stesso *file* sequenze di blocchi contigui piuttosto che singoli blocchi isolati
    - Strategia analoga a quella di `ext2fs`
    - Nel caso peggiore i dati di un *file* possono trovarsi su sequenze di blocchi singoli non adiacenti
  - Le sequenze non sono necessariamente contigue tra loro
- 1 *record* base in MFT ∀ *file* sequenziale nel volume
  - La struttura interna del *record* dipende dalla dimensione del *file* e dalla contiguità dei suoi blocchi
  - *File* con zone interne non utilizzate (e.g. poste a 0 e riservate per uso futuro) sono chiamati ***file* sparsi** e sono gestiti in modo distinto e particolare

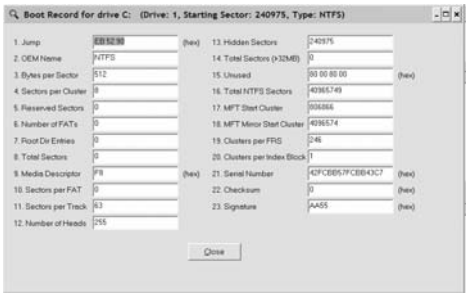
Architettura di NTFS – 8

- I primi 16 *record* dell'**MFT** sono riservati per "*file* **trascendenti**"
  - *Record* che descrivono l'organizzazione interna dell'intero volume (***metadata***)
- Il 1° *record* (#0) descrive l'MFT stesso
  - Che è trattato come un *file* i cui record compongono l'MFT (!)
- Il 2° (#1) replica i primi 16 *record* in modo **non residente** ponendone il contenuto in fondo al volume (***mirror file***)
  - Facilita il ripristino del volume in caso di corruzione dell'MFT
- Il 3° (#2) è una specie di *journal*
- Il 4° (#3) caratterizza il volume
  - Nome logico, versione di FS, data di creazione, etc.
- Il 5° (#4) descrive gli attributi usati nel volume
  - Attributi non residenti denotati da un puntatore di 48 *bit* a un *record* remoto
  - Un codice di controllo di 16 *bit* deve coincidere con il numero di sequenza del *record* di base del valore dell'attributo in MFT
  - **64 *bit* in tutto**
- Inoltre: puntatore alla radice del FS (#5); *bitmap* dei blocchi liberi; copia del codice di *boot* di volume o suo puntatore; etc.

Architettura di NTFS – 9



Architettura di NTFS – 10

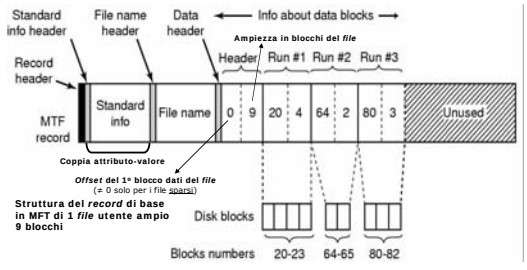


Architettura di NTFS – 11

Volume MS Windows Partition (C:)	
Volume size	= 19.53 GB
Cluster size	= 4 KB
Used space	= 15.35 GB
Free space	= 4.18 GB
Percent free space	= 21 %
Volume fragmentation	
Total fragmentation	= 0 %
File fragmentation	= 0 %
Free space fragmentation	= 1 %
File fragmentation	
Total files	= 98,161
Average file size	= 215 KB
Total fragmented files	= 0
Total excess fragments	= 0
Average fragments per file	= 1.00

Pagefile fragmentation	
Pagefile size	= 2.00 GB
Total fragments	= 1
Folder fragmentation	
Total folders	= 9,492
Fragmented folders	= 1
Excess folder fragments	= 0
Master File Table (MFT) fragmentation	
Total MFT size	= 119 MB
MFT record count	= 107,892
Percent MFT in use	= 88 %
Total MFT fragments	= 3

Record base senza estensioni – 1



Record base senza estensioni – 2

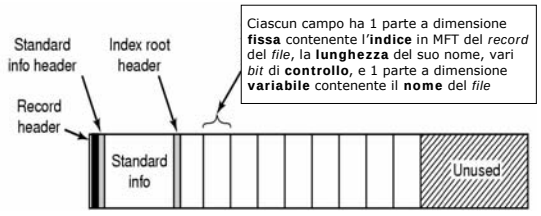
- Nella figura precedente un **singolo** descrittore basta a contenere l'intera lista di sequenze (***run***) di blocchi contigui di dati del *file*
  - 9 blocchi dati in totale suddivisi in 3 sequenze ciascuna descritta come
    - Indirizzo su disco del 1° blocco della sequenza
    - Ampiezza in blocchi della sequenza
  - L'intestazione dell'attributo dati specifica il # di sequenze presenti nel descrittore (3 in questo caso)
  - La prima coppia di attributi dati specifica l'*offset* entro il *file* del 1° blocco coperto dal descrittore e l'*offset* del 1° blocco non coperto (= ampiezza)

Record base senza estensioni – 3

- La strategia NTFS consente di rappresentare *file* di ampiezza virtualmente **illimitata**
- Il numero di *record* necessari per i dati di 1 singolo *file* dipende dalla **contiguità** dei suoi blocchi piuttosto che dalla sua ampiezza
  - 1 *file* da 20 GB costituito da 20 sequenze di 1 M blocchi da 1 KB ciascuno richiede 20+1 coppie di valori espressi su 64 *bit* ovvero $21 \times 2 \times 8$ B = 336 B
    - Ampiamente contenuto in 1 singolo *record* MFT
  - 1 *file* da 64 KB costituito da 64 sequenze di 1 blocco ciascuna richiede $(64+1) \times 2 \times 8$ B = 1040 B
    - Maggiore della capacità dell'attributo dati su 1 singolo *record*

Record base senza estensioni – 4

Il *record* base in MFT per una *directory* di piccole dimensioni

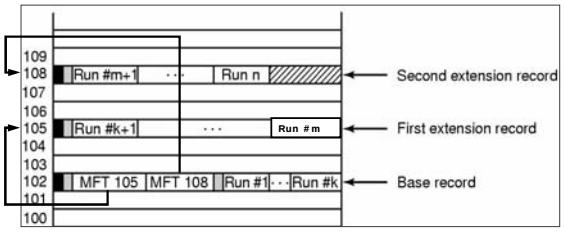


Record con estensioni – 1

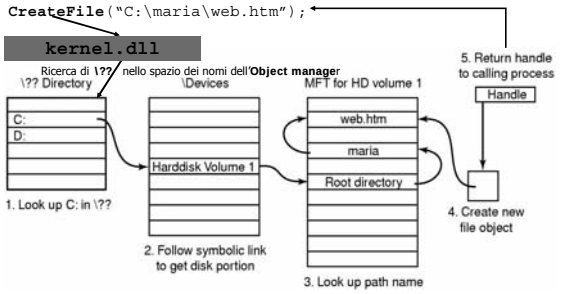
- La rappresentazione di *file* può richiedere **più** *record*
- NTFS usa per questo una tecnica a **continuazioni** analoga a quella usata dagli *i-node* di UNIX e GNU/Linux
  - Il *record* **base** in MFT contiene un puntatore a N ≥ 1 *record* **secondari** in MFT che descrivono la sequenza di blocchi del *file*
  - Lo spazio rimanente del *record* **base** può descrivere le prime sequenze di blocchi dati del *file*
- Se non vi fosse abbastanza spazio in MFT per i *record* secondari di un dato *file* la loro intera lista verrebbe trattata come un **attributo non residente** e posta in un *file* dedicato denotato da un *record* posto in MFT

Record con estensioni – 2

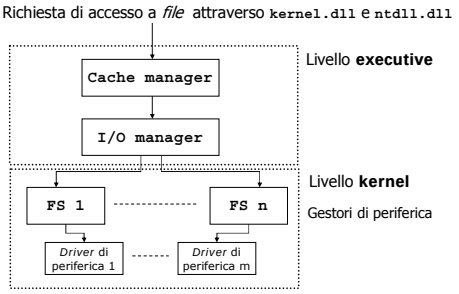
Un *file* composto da **n** sequenze di blocchi dati con descrizione specificata su 1 *record* base e 2 *record* di estensione in MFT



Creazione e localizzazione di *file*



Gestione della *cache* RAM – 1



Gestione della *cache* RAM – 2

- L'accesso a *file* avviene tramite **kernel.dll** e **ntdll.dll** e viene indirizzato al `Cache manager`
  - **Indipendentemente** dal tipo di FS
  - Il `Cache manager` tratta **blocchi virtuali** perché non conosce la struttura fisica dello specifico FS
    - Concetto analogo a quello del **VFS** di GNU/Linux
    - **Blocco virtuale** = (***stream***, ***offset***)
    - **Blocco fisico** = (**partizione, indice di blocco**)
- Ogni FS viene visto come **gestore di periferica logica** sotto il controllo dell'`I/O manager`

Gestione della *cache* RAM – 3

- Per ogni *file* in uso il `Cache manager` alloca 256 KB di spazio di indirizzamento virtuale di `kernel`
  - **Indipendentemente** dalla dimensione del *file*
  - Lo spazio complessivo a disposizione del `Cache manager` è parametro di configurazione
    - Se necessario rialloca spazio assegnato a *file* vecchi
- Le richieste di accesso a *file* vengono soddisfatte attingendo allo spazio di `kernel`
  - Per ogni dato non disponibile l'`I/O manager` tratta l'errore (*page fault*) **trasparentemente** al `Cache manager` caricando il blocco richiesto

Gestione della *cache* RAM – 4

- Ogni accesso concorrente di più *thread* a uno stesso *file* mappato in memoria riferisce la **stessa** area di `kernel` assegnata dal `Cache manager` al *file*
  - Il *file* è mappato **una sola volta** nello spazio del `kernel` indipendentemente dal numero di utenti
  - Le scritture avvengono nello spazio di `kernel` le letture copiano dati nell'area dell'utente
    - Principio del *copy-on-write*
- Questo meccanismo garantisce la coerenza della condivisione di dati su *file*