**MS Windows**

**Genesi**

• **MS-DOS** (MicroSoft Disk Operating System)

– Mono-utente in modalità command line

– Non multi-programmato

– Inizialmente ispirato a CP/M

• 1981 : 1.0 (8 KB codice) . PC IBM 8088 (16 bit)

• 1986 : 3.0 (36 KB) . PC IBM/AT (i286 @ 8 MHz, . 16 MB)

**Windows 1a generazione**

– Modalità GUI solo come rivestimento di MS-DOS

– Interfaccia copia del 1o modello Macintosh di Apple

• 1990 - 1993 : 3.0, 3.1, 3.11 . i386 (32 bit)

**GUI**

• GUI (Graphical User Interface)

– Introdotto dal modello Macintosh di Apple il 24 gennaio 1984

• Vedi http://www.apple-history.com/lisa.html

– Basato sul paradigma WIMP (dispregiativo!)

• Finestre (windows), icone (icons), menu e dispositivi di puntamento (pointing)

• Realizzabile

– Sia come programma in spazio utente (GNU/Linux)

– Che come parte del S/O (Windows)

**Windows 2a generazione**

– Vero e proprio S/O multiprogrammato ma sempre mono-utente con FS su modello FAT

• 1995 : Windows 95 (MS-DOS 7.0)

• 1998 : Windows 98 (MS-DOS 7.1)

– Nucleo a procedure Incapaci di più esecuzioni simultanee

– Ogni accesso a nucleo protetto da semaforo a mutua esclusione

» Scarsissimi benefici di multiprogrammazione

– 1/4 dello spazio di indirizzamento di processo (4 GB totali) condiviso

R/W con gli altri processi; 1/4 condiviso R/W con il nucleo

» Scarsissima integrità dei dati critici

• 2000 : Windows ME (ancora MS-DOS)

**Windows 3a generazione**

– Progetto NT: abbandono della base MS-DOS

• Architettura a 16 bit

– Enfasi su sicurezza e affidabilità

– FS di nuova concezione (ntfs)

• 1993 : Windows NT 3.1 . fiasco commerciale per la mancanza di programmi di utilità

• 1996 : Windows NT 4.0 . reintroduzione di interfaccia e programmi Windows 95

– Scritto in C e C++ per massima portabilità al costo di grande complessità (16 M linee di codice!)

– Molto superiore a Windows 95/98 ma privo di supporto per plug-and-play gestione batterie e emulatore MS-DOS

– Architettura di NT 3.1 a microkernel e modello client-server

• La maggior parte dei servizi è incapsulata in processi di sistema eseguiti in modo utente e offerti ai processi applicativi tramite scambio messaggi

– Elevata portabilità

• Dipendenze hardware localizzate nel nucleo

– Bassa velocità

• Più costosa l’esecuzione in modo privilegiato

– Architettura di NT 4.0 a nucleo monolitico

• Servizi di sistema riposizionati entro il nucleo

– 1999 : **Windows 2000** (alias di NT 5.0)

• Il S/O esegue in modo nucleo

– Lo spazio di indirizzamento dei processi è interamente privato e distinto dal modo nucleo

• Periferiche rimuovibili

– Plug-and-play

• Internazionalizzazione (configurabile per lingua nazionale)

• Alcune migliorie a ntfs

• MS-DOS completamente rimpiazzato da una shell di comandi che ne riproduce e estende le funzionalità

• Enorme complessità: oltre 29 M linee di codice C[++]

– 2001: **Windows XP**

• Migliorie grafiche rispetto a Windows 98 da cui ci sivuole definitivamente staccare

– Un successo

• Non è previsto utilizzo come tipo server

– 2007: **Windows Vista**

• Windows Aero

• Flop (lento e pesante)

– Seppure più sicuro e con meno buffer overflow

• Supporto WinXP esteso fino ad aprile 2014

– 2009: **Windows** **7**

• A meno di 3 anni dall’uscita del predecessore

– Corsa ai ripari per insuccesso Vista

• Miglioramenti in efficienza e interfaccia grafica

– Supporto multitouch

• In alcune versioni include un emulatore di XP

– Per assicurare assoluta compatibilità e facilitare transizione

– 2012: **Windows 8**

• Integrazione PC, Smartphone, Tablet

• Windows Store per distribuzione app

• Interfaccia con Tiles

• Efficienza energetica (consumo batteria dispositivi mobili)

**Interfaccia di programmazione**

Basato su principio speculare a quello adottato da UNIX e GNU/Linux

– Interfaccia di sistema non pubblica

– Procedure di libreria pubblicate in Win32 API (Application Programming Interface) a uso del programmatore ma controllata da Microsoft

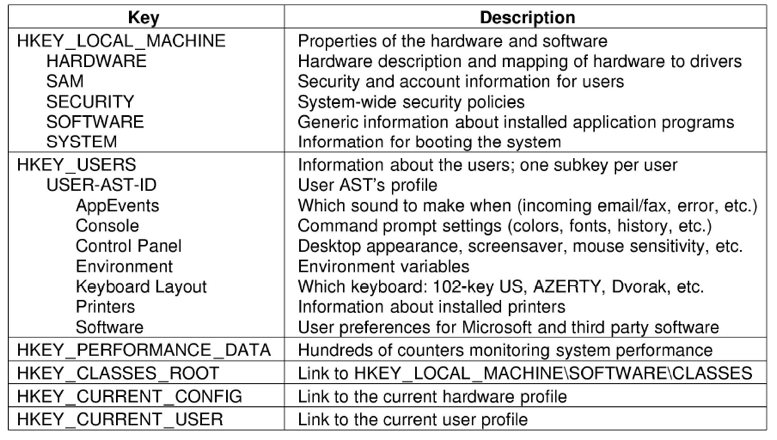
– Alcune procedure includono chiamate di sistema

– Altre svolgono servizi di utilità eseguiti interamente in

modo utente

• Nessun sforzo di evitare ridondanza o rigore gerarchico

**Informazioni di configurazione**

Tutte le informazioni vitali di configurazione del sistema sono raccolte in una specie di FS detto registry salvato su disco in file speciali detti hives

– Directory 🡪 key

– File 🡪 entry = {nome, tipo, dati}

6 directory principali con prefisso HKEY\_

– Per esempio: HKEY\_LOCAL\_MACHINE con entry descrittive dell’hardware e delle sue periferiche (HARDWARE) dei programmi installati (SOFTWARE) e con informazioni utili per l’inizializzazione (SYSTEM)

**Gestione dei processi**

**Job** = {processi gestiti come singola unità con limiti risorse}

**Processo** = possessore di risorse, con ≥ 1 thread

ID unico, 4 GB di spazio di indirizzamento (2 in modo utente e 2 in modo nucleo), inizialmente con singolo thread, simile al processo UNIX; non ha stato di avanzamento

**Thread** = flusso di controllo gestito dal nucleo

Esegue per conto e nell’ambiente del processo (che non ha stato di avanzamento), con ID localmente unico, 2 stack (1 per modo)

**Fiber** = suddivisione di thread ignota al nucleo

Esegue nell’ambiente del thread e viene gestita interamente a livello di servizi offerti dal sottosistema Win32

I thread hanno vari modi per sincronizzarsi tra loro tramite oggetti di ordinamento

– Semafori binari (mutex) o contatori

– Sezioni critiche limitate allo spazio di indirizzamento del thread che le crea

– Eventi (oggetti del kernel)

• Thread attendono che si verifichino certi eventi

– Manual-reset events (rilasci manuali)

– Auto-reset events (al verificarsi dell’evento uno e uno solo viene rilasciato)

I thread hanno vari modi per comunicare senza bisogno di sincronizzarsi

– **Pipe** : canali bidirezionali come in UNIX e GNU/Linux a sequenza di byte senza struttura oppure per messaggi (sequenze con struttura)

– **Mailslot** : canali unidirezionali anche su rete

– **Socket** : come pipe ma per comunicazioni remote

– **RPC** (chiamata di procedura remota) : per invocare procedure nello spazio di altri processi e riceverne ilrisultato localmente

– **Condivisione di memoria** : usando (porzioni di) file mappati in memoria

**Politica di ordinamento**

Ordinamento con prerilascio a priorità

– Effettuato da azioni esplicite del thread eseguite in modo nucleo🡪 nessuna entità attiva dedicata di sistema

• Thread si blocca ad un semaforo, I/O, etc.

– Già in nucleo

• Thread segnala un oggetto (es. fa up di semaforo)

– Già in nucleo

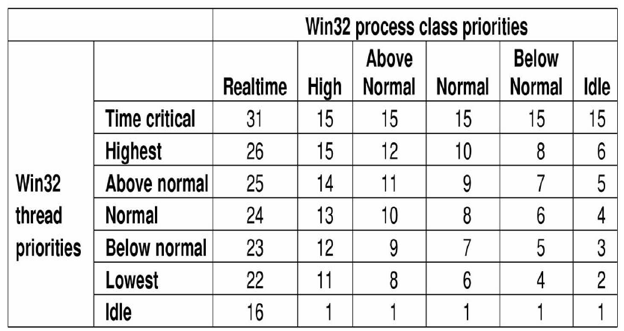
• Al completamento del proprio quanto di tempo

– passa in modo nucleo tramite DPC per concludere l’interrupt handler

– Causato da attività esterne eseguite nel contesto del thread corrente

• Azioni di ordinamento programmate come DPC (Deferred Procedure Call) associate al trattamento di eventi asincroni

– Completamento operazione I/O

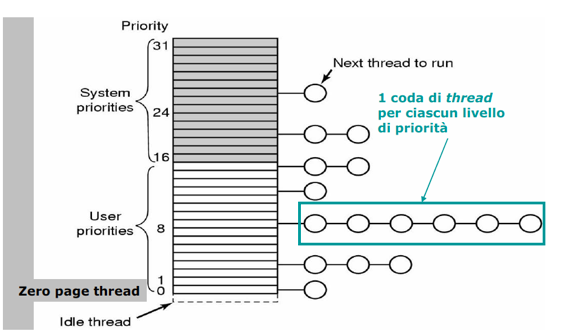
 – Scadenza timer

• 6 classi di priorità per processo

– Realtime, high, above-normal, normal, below-normal, idle

• 7 classi di priorità per thread

– Time-critical, highest, above-normal, normal, belownormal, lowest, idle

• 32 livelli di priorità (31 .. 0)

– Ciascuno associato a una coda di thread pronti

– Thread non distinti per processo di appartenenza

– 31 .. 16 priorità di sistema; 15 .. 0 priorità ordinarie

• Ricerca per priorità decrescente

• Selezione dalla testa della coda

• Priorità più elevate per processi I/O bound e interattivi

Ciascun thread ha una priorità base iniziale e una corrente che varia nel corso dell’esecuzione

– Entro la fascia della classe di priorità del processo di appartenenza

La priorità corrente si eleva quando il thread

– Completa un’operazione di I/O

• Per favorire maggior utilizzazione delle periferiche

• Insieme a un ampliamento temporaneo della durata del quanto

– Ottiene un semaforo o riceve un segnale d’evento

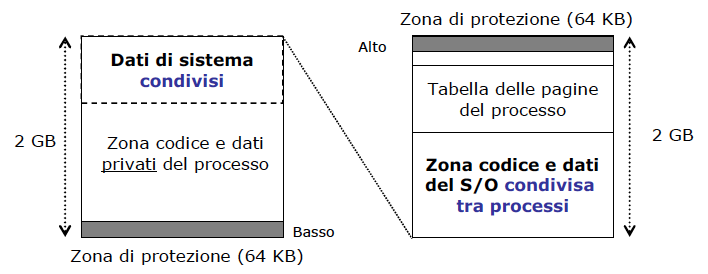
• Per ridurre il tempo di attesa dei processi interattivi

La priorità corrente decresce a ogni quanto consumato

– Il quanto può essere più ampio se è riferito alla finestra in primo piano

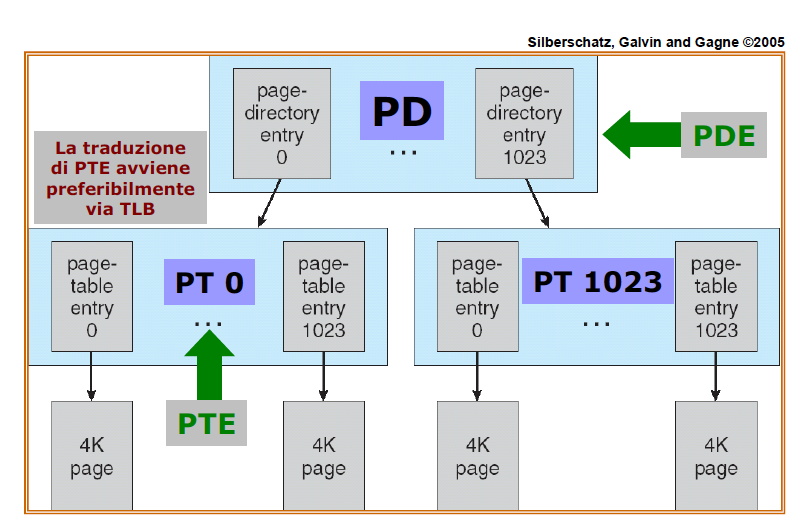
• Usa una tecnica brutale per mitigare il problema di inversione di priorità

– Un thread pronto non selezionato per un certo tempo riceve un incremento di priorità per 2 quanti

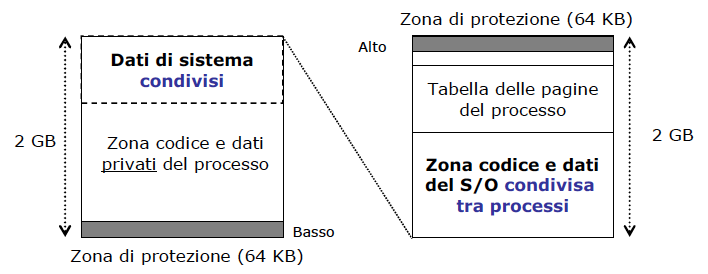
**Gestione della memoria**

Ogni processo dispone di uno spazio di indirizzamento virtuale paginato ampio 4 GB e suddiviso in 2 zone adiacenti ampie 2 GB ciascuna

– Indirizzi virtuali espressi su 32 bit



**Indirizzamento virtuale**



**Gestione della memoria**

Una pagina virtuale può essere

– **R** (lettura) / **W** (scrittura) / **E** (esecuzione)

– **Libera** (free): non riferita da alcun PTE

• Tutte le pagine di un processo sono inizialmente libere (pagingon-demand)

• Page fault

– **Assegnata** (committed): in uso per codice o dati

• Viene riferita tramite indirizzo virtuale e caricata da disco ove non fosse già presente in RAM

– **Prenotata** (reserved): non ancora in uso, ma non libera

• Per agevolare l’assegnazione di pagine contigue a processi

– Alla creazione di un nuovo processo 1MB è riservato per lo stack

Più processi possono condividere l’accesso a pagine di uno stesso file mappato in memoria

– Un libreria condivisa DLL (Dynamic Link Library) è un tipico esempio di file mappato in memoria

• Codice condiviso in sola lettura

• Dati statici R/W copiati per ciascun processo (copy-on-write)

– Ogni processo che accede a un file possiede specifici diritti di accesso che il S/O si preoccupa di far rispettare

La stessa posizione nel file può corrispondere ad indirizzi virtuali diversi per processi distinti

– Gli indirizzi riferiti nel codice condiviso di DLL devono pertanto essere espressi in modo relativo

– A cura del compilatore

Il caricamento di una nuova pagina in RAM può richiedere il rimpiazzo locale di una pagina “vecchia”

– Solo se non vi sono abbastanza pagine libere

– Il sistema mantiene una lista delle pagine libere

– A ogni processo i si associa l’insieme Ii delle sue pagine attualmente in RAM (Working Set)

• L’ampiezza del WS Ii può variare solo entro limiti prefissati

Mini ≤ #{Ii} ≤ Maxi

• Politica di rimpiazzo locale

Si ha rimpiazzo globale se e solo se un particolare processo deve scambiare proprie pagine tra RAM e disco troppo spesso

Anche il S/O stesso è visto come un processo con un proprio WS con pagine rimpiazzabili

– Min set iniziale nell’ordine di 20-50 pagine

– Max set iniziale nell’ordine di 45-345 pagine

– Solo alcune pagine del S/O sono inamovibili

Un daemon di kernel con periodo 1 s accerta che vi siano sufficienti pagine libere

– Balance set manager

Se insufficienti il daemon attiva un thread del Memory manager che esamina con una euristica i WS dei processi per rilasciarne pagine

– Working set manager

– Processi non recentemente attivi con WS ampi vengono scrutinati prima degli altri

– Le pagine necessarie si prelevano dagli WS di ampiezza vicina al massimo e con scarso uso recente

Ciascuna page frame in RAM può essere

– **In uso** e appartenere a 1 WS (≥1 se condivisa)

– **Rilasciata** e appartenere a 1 e 1 sola lista tra:

• [A] In attesa: pagina recentemente rimossa dal WS di un processo ma ancora associata a esso e non modificata

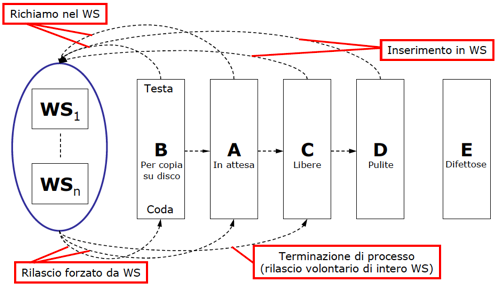
– Può essere riassegnata e sovrascritta senza problemi

• [B] Da copiare su disco: ~ A ma se rimpiazzata deve essere riportata su disco

• [C] Libera: ~ A ma non più associata ad alcun processo

• [D] Azzerata: ~ C ma con contenuto obliterato a zero per consentire riassegnazione senza travaso di info privata

• [E] Difettosa: pagina che non può più essere utilizzata a causa di difetti nella zona di memoria fisica

Lo swapper thread (daemon) del Memory manager porta in [A]o[B] le pagine dello stack dei processi i cui thread siano stati tutti recentemente inattivi.

Altri 2 daemon assicurano che vi siano abbastanza pagine in [C] salvando su disco quelle in [B] e poi accodandole in [A].

Un WS che cresce preleva pagine libere da [C] se le sovrascrive interamente (senza conservare dati precedenti) da [D] altrimenti.

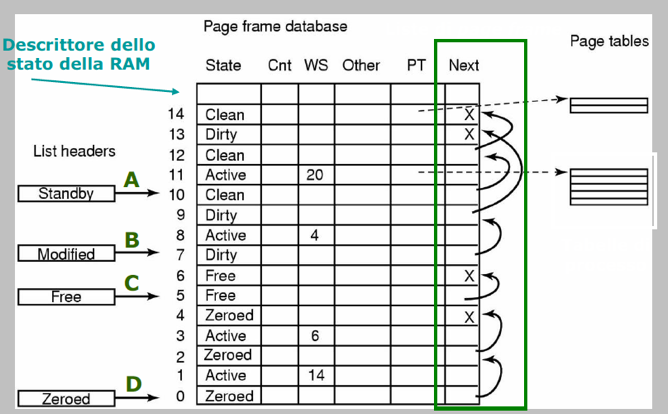
– Un daemon dedicato che opera per conto del kernel azzera periodicamente il contenuto di pagine in [C] e le pone in [D]

Euristiche complesse e non garantite governano le scelte effettuate dalle varie attività di gestione delle liste [A] – [D]

– L’amministratore di sistema può influenzare alcune euristiche mediante parametri di configurazione

Lo stato della RAM viene mantenuto in una tabella dedicata acceduta per indice di pagina fisica (page frame database)

– Pagina valida/invalida, contatore dei riferimenti, WS di appartenenza, lista di appartenenza, etc.



**File System**

**Architettura di NTFS**

NTFS file system adottato da Windows NT e poi anche da XP e Vista

supporta l’intera gamma dei FS Windows e anche ext2fs di GNU/Linux

– **FAT-16**

• Limite logico all’ampiezza di partizione

• ≤ 216 blocchi di ampiezza massima 32 KB 🡪 2 GB

– **FAT-32**

• Limite fisico all’ampiezza di partizione

• ≤ 232 settori da 512 B 🡪 2 TB

• Limite logico : 228 blocchi da 8 KB 🡪 2 TB

– **NTFS**

• Nuova concezione con indici espressi su 64 bit

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 264 B !) contenente i dati del file

– Idea copiata da Apple ☺ introdotta (anche) per compatibilità

FS ad architettura gerarchica (come ext2)

– \ invece di / come separatore nelle espressioni di cammino

– 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

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

La principale struttura dati è la MFT (Master File Table)

– Una per volume

– **Fisicamente** realizzata come un file

• Perciò può essere salvata ovunque nel volume

– Soluzione più robusta

– **Logicamente** strutturata come una sequenza lineare di ≤ 248 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)

Ciascun record contiene un numero variabile di coppie <descrittore di attributo, valore>

– Il 1o campo della coppia specifica la struttura dell’attributo (che tipo di attributo è, quanto è lungo il suo valore)

• Esistono 13 attributi di base con struttura predefinita

• Possono esistere altri attributi aggiuntivi a struttura libera

– Il 2o 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 effettivo del file

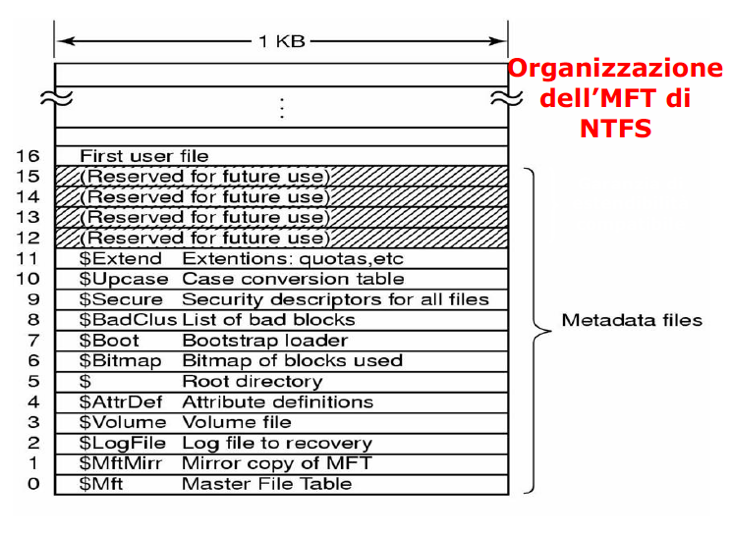
I primi 16 record dell’MFT sono riservati per “file trascendenti” di sistema (metadata)

– Questi record descrivono l’organizzazione dell’intero volume

• Il 1o record (0) descrive l’MFT stesso

• Il 2o (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 4o (3) caratterizza il volume (nome logico, versione di FS, data dicreazione, etc.)

• Il 5o (4) descrive gli attributi usati nel volume

– Gli attributi non residenti sono denotati da un puntatore di 48 bit a un record remoto e un codice di controllo di 16 bit che deve coincidere con quello del record di base in MFT (64 bit in tutto)

• Inoltre: puntatore alla radice del FS; bitmap dei blocchi liberi; copia del codice di boot di volume o suo puntatore; etc.

Il campo <descrittore di attributo> per attributi residenti ha ampiezza 24 B

– Quello per attributi non residenti è più ampio

Non tutti i 13 attributi di sistema applicano a tutti i file

– Gli attributi previsti per i file corrispondono a quelli che GNU/Linux pone negli i-node con l’aggiunta dell’identificatore dell’oggetto corrispondente

– Il contenuto dati di file di ampiezza < 1 KB viene memorizzato interamente entro un record di MFT

• Immediate file (rari)

– Per file più grandi il valore dell’attributo dati diventa la lista ordinata dei corrispondenti blocchi su disco

NTFS cerca di assegnare allo stesso file sequenze di blocchi contigui piuttosto che singoli blocchi isolati

– Strategia analoga a quella di ext2fs

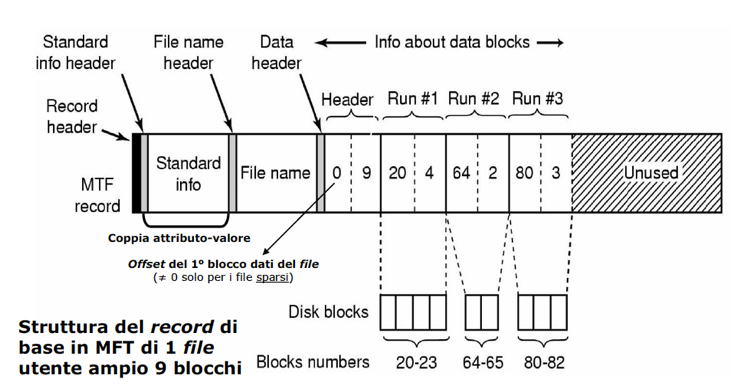
– Nel caso peggiore i dati di un file possono trovarsi su sequenze di blocchi singoli non adiacenti

Esiste 1 record base in MFT . file sequenziale presente 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 diversamente

**Record base senza estensioni**

Nella figura 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 1o 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 1o blocco coperto dal descrittore e l’offset del 1o blocco non coperto (= ampiezza)

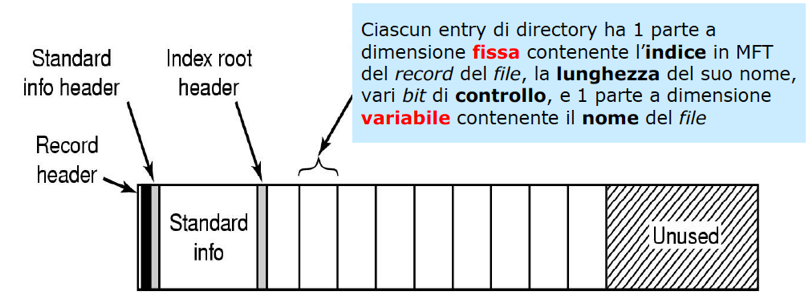
La strategia NTFS consente di rappresentare file di ampiezza virtualmente illimitata.

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 x 2 x 8 B = 336 B

– 1 file da 64 KB costituito da 64 sequenze di 1 blocco ciascuna richiede (64+1) x 2 x 8 B = 1040 B

Il record base in MFT per una directory di piccole dimensioni



**Record con estensioni**

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

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

