**File System**

**Considerazioni generali**:

La maggior parte dell’informazione applicativa (i dati) ha durata, ambito e dimensione più ampi della vita delle applicazioni che la usano.

Le tre esigenze principali sono:

* Persistenza dei dati
* Possibilità di condivisione dei dati tra applicazioni distinte
* Nessun limite di dimensione fissato a priori.

Il *File System* è il servizio di S/O progettato per soddisfare questi bisogni.

Il termine **file** designa un insieme di dati correlati, residenti in memoria secondaria e trattati unitariamente (file=raccoglitore).

Il termine **file system** (FS) designa la parte di S/O che si occupa di: organizzazione, gestione, realizzazione, accesso di e ai file.

La progettazione di FS affronta due problemi chiave:

* **Cosa** occorre offrire all’utente applicativo e secondo quali forme concrete (modalità di accesso a file, struttura logica e fisica di file, operazioni ammissibili su file)
* **Come** ciò possa essere realizzato in modo pratico ed economico garantendo la massima indipendenza dall’architettura fisica di supporto.

**Il *file*:**

Il file è un concetto logico realizzato tramite meccanismi di astrazione per salvare informazione su memoria secondaria e potendola ritrovare in seguito senza conoscerne ne la struttura logica ne il funzionamento. All’utente non interessa come ciò avviene. Interessa invece poter designare le proprie unità di informazione mediante nomi logici unici e distinti. L’utente vede solo nomi di file.

Le caratteristiche distintive di un file sono: attributi, struttura interna, operazioni ammesse.

**Attributi di file**:

* Nome - stringa composta da 8-255 caratteri, inclusi numeri e caratteri speciali con maggiore o uguale di una estensione che possa designare il “tipo” di file come visto dall’utente. In generale l’utente può configurare presso l’S/O l’associazione tra l’ultima estensione del file e il tipo applicativo corrispondente.
* Dimensione corrente
* Data di creazione - può non essere mostrata
* Data ultima modifica - può non essere mostrata
* Creatore e possessore: possono essere distinti
* Permessi di accesso - lettura, scrittura, esecuzione

**Strutture dati di file**:

La struttura dei dati all’interno di un file può essere considerata da tre punti di vista:

* **Livello utente** – il programma applicativo associa autonomamente significato al contenuto grezzo del file
* **Livello di struttura logica** – a monte dell’interpretazione dell’utente il S/O organizza i dati grezzi in strutture logiche per facilitarne il trattamento
* **Livello di struttura fisica** – il S/O mappa le strutture logiche sulle strutture fisiche della memoria secondaria disponibile

Le possibili strutture logiche di un file sono:

* A sequenza di byte (byte stream): la struttura logica più rudimentale e flessibile (scelta di UNIX e MS Windows). Il programma applicativo sa come dare significato al contenuto informativo del file (minimo sforzo pe il S/O). L’accesso ai dati utilizza un puntatore relativo all’inizio del file. Lettura e scrittura operano a blocchi di byte.
* A record di lunghezza e struttura interna fissa: gli spazi non utilizzati sono riempiti da caratteri speciali (es. NULL o SPACE). Il S/O deve conoscere la struttura interna del file per muoversi al suo interno. L’accesso ai dati è sequenziale e utilizza un puntatore al record corrente. Lettura e scrittura operano su record singoli. Scelta ormai obsoleta e legata a specifiche limitazioni dell’architettura di sistema.
* A record di lunghezza e struttura interna variabile: La struttura interna di ogni record viene descritta e identificata univocamente da una chiave (key) posta in posizione fissa e nota entro il record. Le chiavi vengono raccolte in una tabella a se stante, ordinata per chiave, contenente anche i puntatori all’inizio di ciascun record. L’accesso ai dati avviene per chiave. Uso abbastanza diffuso in sistemi mainframe.

**Modalità di accesso**:

* Accesso sequenziale: viene trattato un gruppo di byte (oppure un record) alla volta. Un puntatore indirizza il record (o gruppo) corrente e avanza a ogni lettura o scrittura: la lettura può avvenire in qualunque posizione del file, la quale però deve essere raggiunta sequenzialmente (come su nastro), la scrittura può avvenire solo in coda al file (append). Sul file si può operare solo sequenzialmente, ogni nuova operazione fa ripartire il puntatore dall’inizio.
* Accesso diretto: opera su record di dati posti in posizione arbitraria nel file (posizione determinata rispetto alla base, offset=0)
* Accesso indicizzato: per ogni file una tabella di chiavi ordinate contenenti gli offset dei rispettivi record nel file, informazione di navigazione non più nei record ma in una struttura a parte ad accesso veloce (hash). Ricerca binaria della chiave e poi accesso diretto. Denominatore ISAM (index sequential access method) da IBM che consente accesso sia indicizzato che sequenziale.

**Classificazione**:

Il FS può trattare diversi tipi di file (classificazione distinta da quella dell’utente)

* + File regolari (regular) – sui quali l’utente può operare normalmente, contenuto ASCII o binario
  + File catalogo (directory) – tramite i quali il FS permette di descrivere l’organizzazione di gruppi di file
  + File speciali – con i quali il FS rappresenta logicamente dispositivi orientati a carattere (es. terminale) o a blocco (es. disco)

**Operazioni ammesse**:

* + **Creazione** – file inizialmente vuoto; inizializzazione attributi
  + **Distruzione** – rilascio della memoria occupata
  + **Apertura** – deve precedere il 1° uso; predispone le informazioni utili all’accesso
  + **Chiusura** – rilascio delle strutture di controllo usate per l’accesso ed il salvataggio dei dati
  + **Cerca posizione** (seek) – solo per accesso casuale
  + **Lettura, scrittura** – read, write, append
  + **Cambia nome** – rename, può implicare lo spostamento nella struttura logica del FS
  + **Trova attributi** – per make
  + **Modifica attributi** – permetti

Azioni più complesse (es. copia) si ottengono tramite la combinazione delle operazioni di base.

Sessione d’uso di un file: Si può accedere in uso solo a un file già aperto. All’apertura del file il S/O ne predispone uno specifico strumento di accesso (handle). Dopo l’uso il file dovrà essere chiuso. UNIX mantiene una tabella dei file aperti a due livelli: livello 1 – informazioni del file comuni a “famiglie” di processi, livello 2 – dati specifici del particolare processo.

**File mappati in memoria**:

Il S/O può mappare un file in memoria virtuale. il file continua a risiedere in memoria secondaria. All’indirizzo di ogni suo dato corrisponde un indirizzo di memoria virtuale (base+offset), con segmentazione si potrebbe avere [file=segmento] potendo cosi usare lo stesso offset per entrambi. Le operazioni su file avvengono in memoria principale (chiamata di indirizzo🡪page fault🡪caricamento🡪operazione🡪rimpiazzo di pagina🡪salvataggio in memoria secondaria).

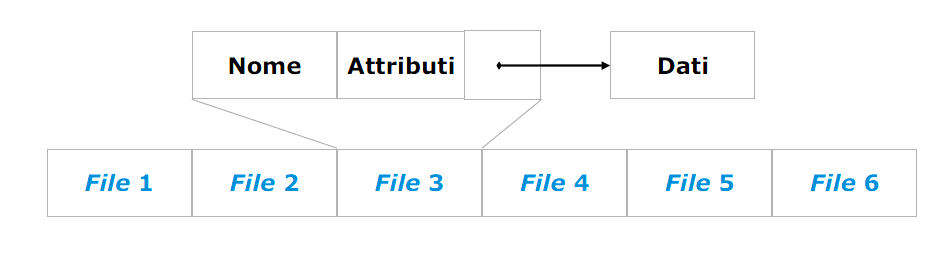
Riduce gli accessi a disco ma comporta problemi con la condivisione e con i file di enormi dimensione.

**Struttura di directory**:

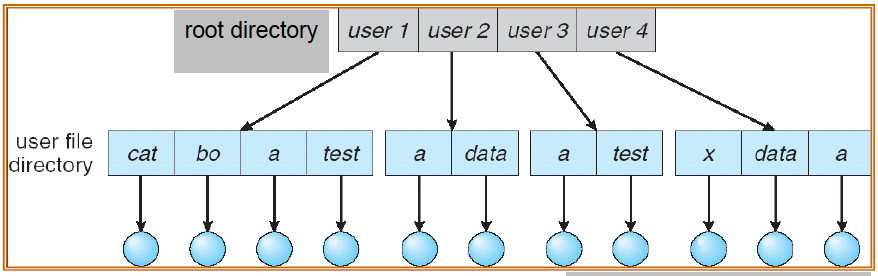
Ogni FS usa directory (catalogo) o folder (cartella) per tener traccia dei suoi file regolari.

Le directory possono essere classificate rispetto all’organizzazione di file che esse consentono:

* **Directory a livello singolo** – tutti i file sono elencati su un’unica lista lineare (“root directory”), ciascuno con il proprio nome (i nomi dei file devono pertanto essere unici). E’ semplice da capire e da realizzare. I file sono semplici da trovare ma c’è una gestione onerosa all’aumentare dei file.

****

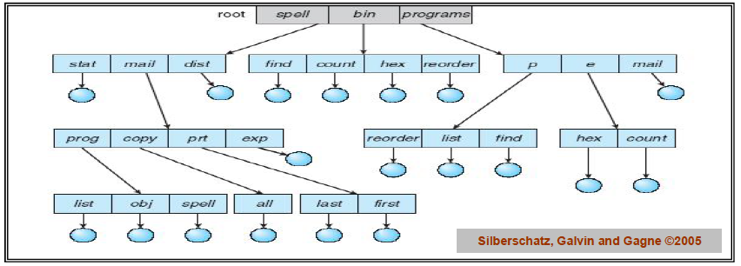
* **Directory a due livelli** – una Root Directory contiene una User File Directory (UFD) per ciascun utente di sistema. L’utente registrato può vedere solo la propria UFD (le UFD di altri solo se esplicitamente autorizzato, buona soluzione per isolare utenti in sistemi multiprogrammati). I file sono localizzati tramite percorso (path name). I programmi di sistema possono essere copiati su tutte le UFD oppure (meglio) posti in una directoru di sistema condivisa e lì localizzati mediante cammini di ricerca predefinita (search path).



Requisiti parzialmente soddisfatti: efficienza di ricerca, libertà di denominazione (ma non di riferimenti multipli allo stesso file)

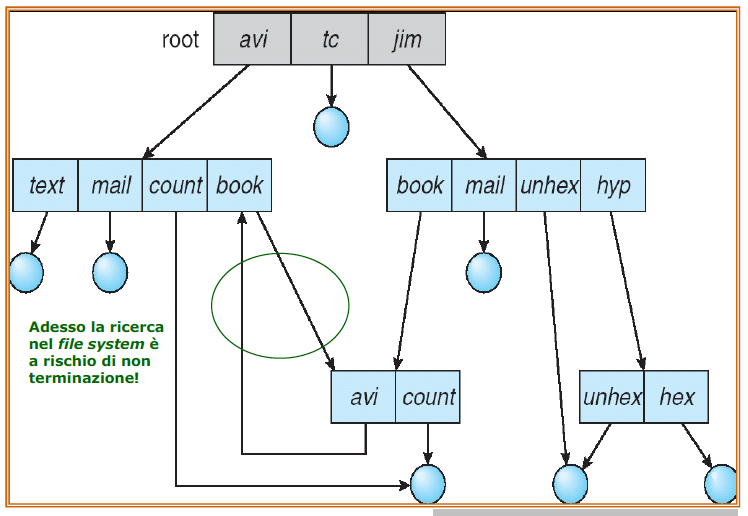
Requisiti non soddisfatti: libertà di raggruppamento.

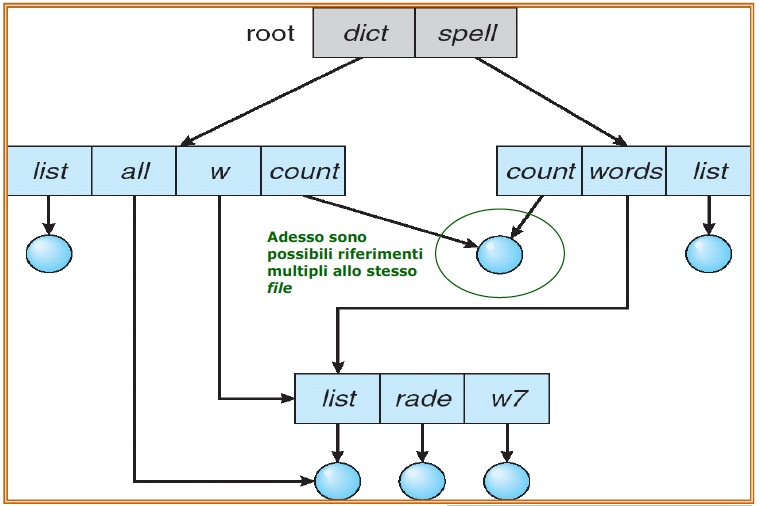
* **Directory ad albero** – numero arbitrario di livelli. Il livello superiore viene detto radice (root). Ogni directory può contenere file regolari o directory di livello inferiore. Ogni utente ha la sua directory corrente che può cambiare con comandi di sistema. Se non si specifica il cammino (path) si assume come riferimento alla directory corrente. Il cammino può essere assoluto (espresso rispetto alla radice del FS) oppure relativo (rispetto alla posizione corrente).



Requisiti parzialmente soddisfatti: ricerca efficiente, libertà di denominazione (ma non di riferimento allo stesso file), libertà di raggruppamento.

* **Directory a grafo**– L’albero diventa grafo consentendo allo stesso file di appartenere simultaneamente a più directory. INIX e GNU/Linux utilizzano collegamenti simbolici (link) tra il nome reale del file e la sua presenza virtuale, La forma generalizzata consente collegamenti ciclici e dunque riferimenti circolari. Un S/O potrebbe duplicare gli identificatori di accesso al file. (handle)🡪nomi distinti, questo però rende più difficile assicurare la coerenza del file.

Struttura a grafo aciclico: Struttura a grafo generalizzato:



I **requisiti fondamentali** a livello utente sono:

* Efficienza – realizzare un file deve essere semplice e trovarlo deve essere facile e veloce
* Libertà di denominazione – più utenti devono poter ciascuno usare lo stesso nome per un file loro proprio, lo stesso file deve poter essere “chiamato” con nomi diversi da utenti diversi
* Libertà di raggruppamento – creare gruppi logici di file sulla base di proprietà significative per l’utente

**Hard link**: un puntatore diretto a un file regolare viene inserito in una directory a esso remota (che deve risiedere nello stesso FS). Questo crea due vie d’accesso distinte dirette a uno stesso file.

**Symbolic (soft) link**: viene creato un file speciale il cui contenuto è il cammino del file originario (il file originario può avere qualunque tipo e risiedere anche in un FS remoto). Questo riferimento mantiene una sola via d’accesso al fine originario.

**Realizzazione del file system**:

I file system sono memorizzati su disco. I dischi possono essere partizionati e ogni partizione può contenere un file system distinto.

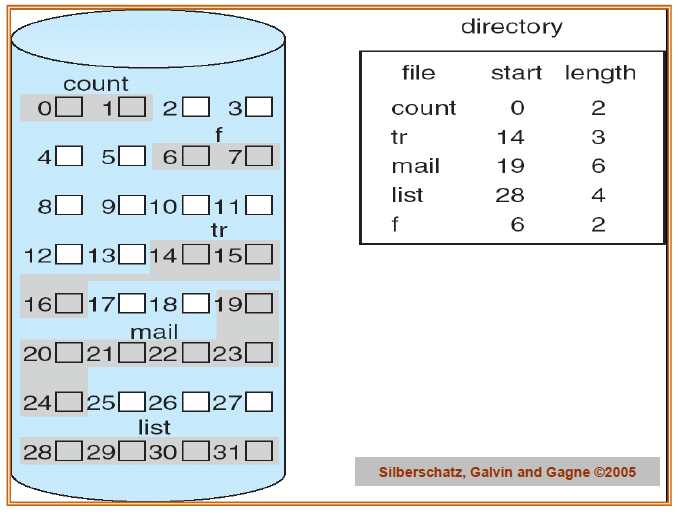
Il settore 0 del disco contiene le informazioni di inizializzazione del sistema: master Boot Record. L’inizializzazione è eseguita dal BIOS. L’MBR contiene (in 512 B) una descrizione delle partizioni che identifica quella attiva. Il primo blocco di informazioni di ogni partizione contiene le sue specifiche informazioni le sue specifiche informazioni di inizializzazione (boot block).

L’unità informativa su disco è il settore.

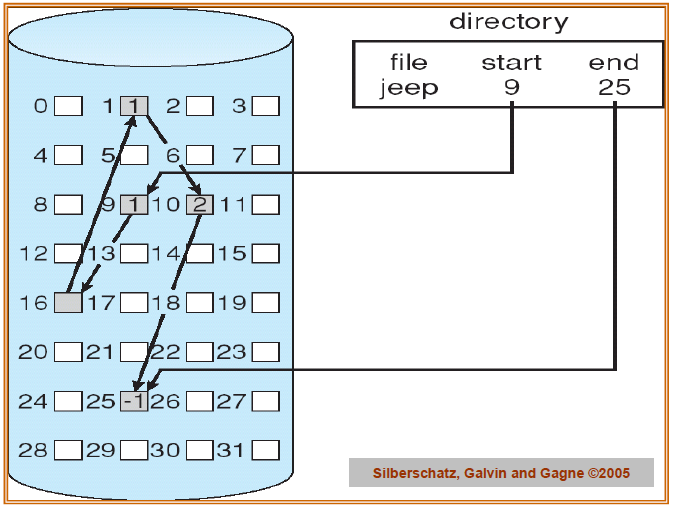
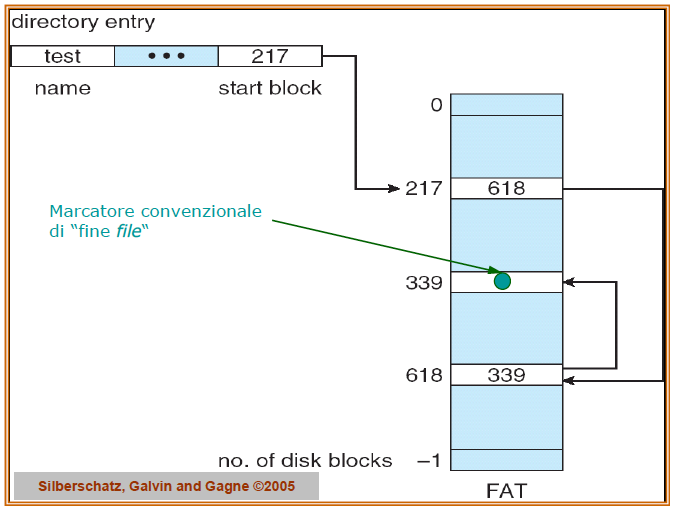
I dischi vengono però letti e scritti a blocchi (cluster per Microsoft). ! blocco = N settori (N>=1) e c’è un rischio consapevole di frammentazione interna.

La struttura interna di partizione è specifica del FS.

**Realizzazione dei file**:

A livello fisico un file è un insieme di blocchi di disco, occorre decidere quali blocchi assegnare a quale file e come tenerne traccia.

Ci sono tre strategie di allocazione di blocchi a file:

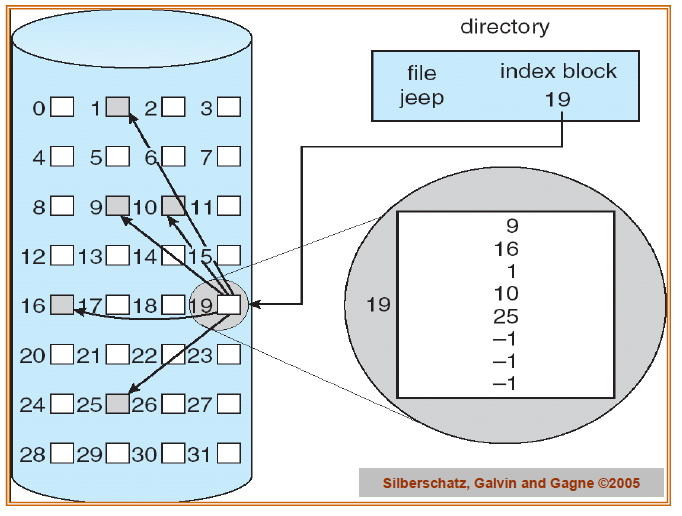
* **Allocazione contigua** - cerca di memorizzare i file su blocchi consecutivi. Ogni file è descritto dall’indirizzo del suo primo blocco e dal numero di blocchi utilizzati. Consente sia accesso sequenziale che diretto. Un file può essere letto e scritto con un solo accesso al disco (ideale per CD-ROM e DVD). Ogni modifica di file comporta il rischio di frammentazione esterna (ricompattazione periodica molto costosa. L’alternativa richiede l’utilizzo dei gruppi di blocchi liberi: mantenere la lista dei blocchi liberi e la loro dimensione che è possibile ma oneroso o conoscere in anticipo la dimensione massima dei nuovi file per farli stare in un blocco libero ma la stima è difficile e rischiosa.
* **Allocazione a lista concatenata (linked list)** – File come lista concatenata di blocchi. File identificato dal puntatore al suo primo blocco, per alcuni S/O anche dal puntatore all’ultimo blocco di file, Ciascun blocco di file deve contenere il puntatore al blocco successivo (o un marcatore di fine lista) e questo sottrae spazio ai dati. L’accesso sequenziale resta semplice ma può richiedere molte operazioni su disco (accesso diretto molto più complesso e oneroso). Un solo blocco guasto corrompe l’intero file.
* **Allocazione a lista indicizzata** – Si pongono i puntatori ai blocchi in strutture apposite, ciascun blocco contiene solo dati. Il file è descritto dall’insieme sei suoi puntatori. Ci sono due strategie di organizzazione: forma tabulare (FAT, File Allocation Table) e forma indicizzata (nodo indice, i-node). Non causa frammentazione esterna. Consente accesso sequenziale e diretto. Non richiedere di conoscere preventivamente la dimensione massima di ogni nuovo file.
  + **FAT** – File Allocation Table.

Scelta progettuale di MS-DOS, base di MS Windows.

E una tabella ordinata di puntatori. C’è un puntatore per ogni blocco (cluster) del disco. La tabella cresce con l’ampiezza della partizione.

La porzione di FAT relativa ai file in uso deve sempre risiedere internamente in RAM, consente accesso diretto ai dati seguendo sequenzialmente i collegamenti ma senza accessi a disco. Un file è una catena di indici.

* + **Nodi indice** (UNIX 🡪 GNU/Linux)

Una struttura indice (i-node) per ogni file con gli attributi del file e i puntatori ai suoi blocchi. L’i-node è contenuto in un blocco dedicato.

In RAM una tabella di i-node per i soli file in uso, la dimensione massima di tabella dipende dal massimo numero di file apribili simultaneamente (non più della capacità della partizione).

Un i-node contiene un numero limitato di puntatori a blocchi.

File di piccola dimensione: gli indirizzi dei blocchi dei dati sono ampiamente contenuti in un singolo i-node, tipicamente con un po’ di frammentazione interna.

File di media dimensione: un campo dell’i-node punta a un nuovo blocco i-node.

File di grandi dimensioni: un campo dell’i-node principale punta a un livello di blocchi i-node intermedi che a loro volta puntano ai blocchi dei dati.

Per file di dimensioni ancora maggiore basta aggiungere un ulteriore livello di indirezione.

**Gestione dei file condivisi**

Come preservarne la consistenza senza costi eccessivi:

* + Non porre blocchi di dati nella directory di residenza del file.
  + Per ogni file condiviso porre nella directory remota un symbolic link verso il file originale, esiste così un solo descrittore (i-node) del file originale, l’accesso condiviso avviene tramite cammino sul FS
  + Altrimenti si può porre nella directory remota il puntatore diretto (hard link) al descrittore (i-node) del file originale. Più possessori di descrittori dello stesso file condiviso, un solo proprietario effettivo del file condiviso, il file condiviso non può più essere distrutto fin quando esistano suoi descrittori remoti anche se il suo proprietario avesse inteso cancellarlo.

**Gestione dei blocchi liberi**

Vettore di bit (bitmap) dove ogni bit indica lo stato corrispondente del blocco: 0=libero, 1=occupato.

Lista concatenata di blocchi sfruttando i campi puntatore al successivo (questa è la scelta nell’architettura FAT).

**Realizzazione delle directory**:

La directory fornisce informazioni su nome, collocazione, attributi di file appartenenti a quel particolare catalogo. File e directory risiedono in aree logiche distinte. Conviene minimizzare la complessità gestionale della struttura interna di directory, meglio una struttura a lunghezza fissa per quanto il suo contenuto sia di ampiezza intrinsecamente variabile, [nome + attributi] oppure [nome + puntatore a nodo indice con attributi].

La ricerca di un file correla il nome (stringa ASCII) alle informazioni necessarie all’accesso, nome e directory di appartenenza del file sono determinati dal percorso indicato dalla richiesta.

La ricerca lineare in directory è di realizzazione facile ma di esecuzione onerosa.

La ricerca mediante tabelle hash è più complessa ma più veloce (F(nome)=posizione in tabella🡪puntatore al file)ùSi può anche creare in RAM una cache di supporto alla ricerca.