# CAPITOLO 11: REALIZZAZIONE DEL FILE SYSTEM

Il file system fornisce il meccanismo per la memorizzazione e l’accesso al contenuto dei file, compresi dati e programmi. Il file system risiede permanentemente nella memoria secondaria, progettata per ottenere in modo permanente grandi quantità di dati

## 11.1 STRUTTURA DEL FILE SYSTEM

I **dischi** costituiscono la maggior parte della memoria secondaria in cui si conserva il **file system**

I dischi hanno 2 caratteristiche importanti:

* si possono riscrivere localmente; si può leggere un blocco dal disco, modificarlo e quindi scriverlo nella stessa posizione;
* si può accedere direttamente a qualsiasi blocco Di informazioni del disco, quindi risulta semplice accedere a qualsiasi file, sì in modo sequenziale che modo diretto.

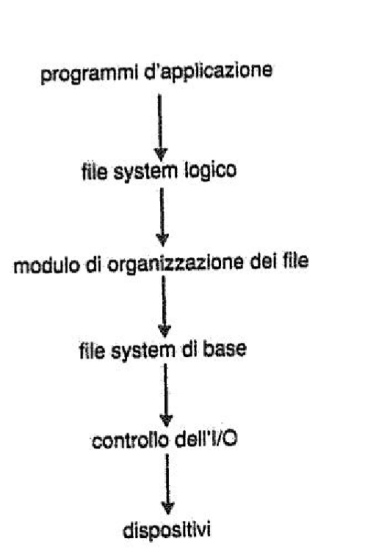
Anziché trasferire un byte alla volta, per migliorare l’efficienza dell’I/O vengono trasferiti interi blocchi, ciascun blocco è formato da uno o più settori(di dim compresa tra 32 e 4096 byte)

Per fornirne un efficiente e conveniente accesso ad disco, il SO fa uso di uno o più **file system**.

Un **file system** presenta due problemi di progettazione piuttosto diversi:

* Il primo riguarda la definizione dell’aspetto dei file system agli occhi utente (come le operaz. Permesse su quel file e gli attributi di quel file).
* Il secondo riguarda la creazione di algoritmi e strutture dati che permettano di far corrispondere il file system logico ai dispositivi fisici di memoria secondaria.

Un file system di solito è composto da molti livelli distinti.



//Esempio di file system stratificato.

Il livello più basso è il controllo dell’I/O costituito dai **driver dei dispositivi** e dai gestori di segnali d’interruzione, si occupa del trasferimento delle informazioni tra memoria centrale e memoria secondaria.

Un **driver di dispositivi** si può concepire come un traduttore che riceve comandi ad alto livello.

Il **file system di base** deve soltanto inviare dei generici comandi all'appropriato driver di dispositivo per leggere e scrivere blocchi fisici nel disco. Ogni blocco si identifica col suo indirizzo numerico nel disco(cilindro-superficie-settore). Questo stato gestisce inoltre buffer di memoria cache che conservano vari blocchi dei file system , delle directory e dei dati. Un blocco viene allocato nel buffer prima che possa verificarsi il trasferimento di un blocco nel disco. Quando il buffer è pieno, il gestore del buffer deve recuperare più spazio in memoria o liberare spazio nel buffer. Le cache servono a conservare metadati di file system usati frequentemente, in modo da migliorare le prestazioni.

Il **modulo di organizzazione dei file** è a conoscenza dei file e dei loro blocchi logici così come dei blocchi fisici dei dischi. Conoscendo il tipo di allocazione dei file usato e la locazione dei file, può tradurre gli indirizzi dei blocchi logici negli indirizzi dei blocchi fisici. I blocchi logici di ciascun file sono numerati da 0/1 a n.

Infine il **file system logico** gestisce i metadati; si tratta di tute le strutture del file system eccetto gli effettivi dati. Il **file system logico** gestisce la struttura della directory per fornire al modulo di organizzazione dei file le informazioni di cui necessita dato un nome simbolico di file. Mantiene le strutture di file tramite di **blocchi controllo dei file**(FCB) contenenti informazioni sui file,come la proprietà, i permessi,ecc. Il **file system logico** è responsabile anche della protezione e della sicurezza.

Nei **file system stratificati** la duplicazione di codice è ridotta al minimo. Il controllo dell’I/O e talvolta il codice di base del file system possono essere comuni a numerosi file system che poi gestiscono il file system logico e i moduli per l’organizzazione dei file secondo le proprie esigenze. Sfortunatamente la stratificazione può comportare un maggior overhead del SO.

Sebbene Linux possa funzionare con più di 40 file system diversi, quello standard e noto come **file system esteso** (ext2/ext3).

## 11.2 REALIZZAZIONE DEL FILE SYSTEM

## Per permettere ai processi di richiedere l'accesso al contenuto dei file, i sistemi operativi offrono le chiamate di sistema open() e close().

## 11.2.1 Introduzione

Per realizzare un file system si usano parecchie strutture dati, sia nei dischi sia in memoria. Queste strutture variano secondo il SO e il file system. Nei dischi il file system tiene informazioni su come eseguire l’avviamento di un SO memorizzato nei dischi stessi, il numero totale di blocchi, il numero e la locazione dei blocchi liberi. Molte di loro sono analizzate in modo particolare.

Fra le strutture presenti nei dischi troviamo:

* **il blocco di controllo dell’avviamento.** Il **boot control block** contiene le informazioni necessarie al sistema per l’avviamento di un SO da quel volume(di solito il primo blocco). Se un disco non contiene un SO quel blocco è vuoto.
* **Blocchi di controllo dei volumi.** Ciascuno di loro contiene i dettagli riguardanti il relativo volume(o partizione) come il numero e la dimensione dei blocchi nel disco, il contatore dei blocchi liberi ecc.
* **Le strutture delle directory.**(una per file system): usate per organizzare i file. Nel caso di Unix(UFS) comprendono i nomi dei file e i numeri di i-node associati. In Windows (file system NTFS) sono nella *tabella principale dei file* (master file table)
* **I blocchi di controllo dei file. FCB** Contenenti molti dettagli dei file compresi i permessi d’accesso ai relativi file,i proprietari, le dimensioni e le locazioni dei blocchi di dati.

Le informazioni tenute in memoria servono sia per la gestione del file system sia per migliorare le prestazioni attraverso l’uso di **cache**.

I dati si caricano al momento del montaggio e si eliminano allo smontaggio.

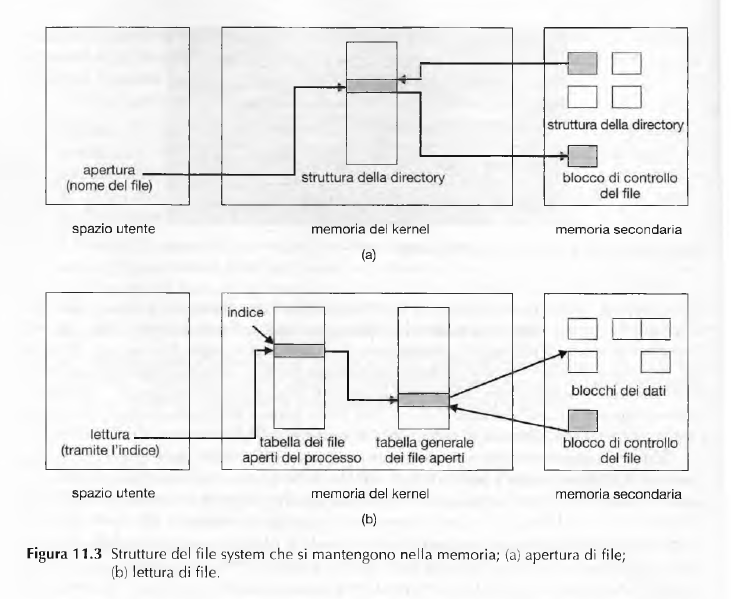
Le strutture contenenti queste informazioni comprendono:

* la **tabella di montaggio**: interna alla memoria che contiene informazioni relative a ciascun volume montato;
* la **struttura delle directory** : tenuta in memoria , contenente le informazioni relative a tutte le directory a cui i processi hanno avuto accesso di recente;
* la **tabella generale dei file aperti :** contenente una copia del blocco di controllo del file per ciascun file aperto, insieme con altre informazioni;
* la **tabella dei file aperti per ciascuno processo**: contenente un puntatore all’appropriato elemento della tabella generale dei file aperti, insieme con altre informazioni;
* i **buffer** conservano blocchi del file system durante la loro lettura o scrittura sul disco.

Le applicazioni per creare un nuovo file eseguono una chiamata al **file system logico**, il quale conosce il formato della struttura della directory. Esso crea e alloca un nuovo FCB.

Il sistema carica quindi la directory appropriata in memoria, la aggiorna con il nome del nuovo file e con il blocco di controllo associato, e la scrivo nuovamente sul disco.

-Alcuni SO, compreso Unix, trattano le directory esattamente come i file, distinguendoli con un campo per il tipo, mentre WINDOWS NT dispone di chiamate di sistema appropriate per le directory. Indipendentemente da questioni strutturali, il file system logico può basarsi sul modulo che si occupa dell'organizzazione dei file per far rispondere l’I/O su directory a numeri di blocchi di disco, che poi si passano al file system di base e il sistema per il controllo dell’I/O. una volta creato un file, deve essere aperto. La chiamata sistema open() passa un nome a file system , questa chiamata dapprima esamina la tabella dei file aperti in tutto il sistema; in caso affermativo, aggiunge un elemento alla tabella dei file aperti del processo (per ogni processo che sta usando il file ). una volta aperto il file se ne ricerca il nome all'interno della directory, e una volta trovato si copia l’FCB nella tabella generale dei file aperti . Questa tabella tiene traccia anche del numero dei processi in quel momento hanno il file aperto. successivamente, si crea un elemento nella tabella dei file aperti del processo con un puntatore alla tabella generale e con altri campi , questi altri campi possono comprendere un puntatore alla posizione corrente del file e il tipo di accesso richiesto all'apertura del file. Il nome dato all’elemento della tabella è **descrittore di file** (file descriptor) in UNIX, e **handle del file** in WINDOWS. Finché un file non viene chiuso, tutte le operazioni si compiono sulla tabella dei file aperti usando questo elemento. Quando un processo chiude il file, si cancella il relativo elemento nella tabella dei file aperti del processo e si decrementa il contatore associata al file nella tabella generale. Se tutti i processi che avevano aperto il file lo hanno richiuso, si scrive l'informazione aggiornata sul file nella struttura delle directory nei dischi e si cancella il relativo elemento nella tabella generale dei file aperti.



## 11.2.2 Partizioni e montaggio

Un disco si può configurare in vari modi, secondo il SO che lo gestisce. Si può suddividere in più partizioni, oppure un volume può comprendere più partizioni su molteplici dischi.

Qui trattiamo il caso in cui il **disco si divide in più partizioni**.

Ciascuna partizione è **priva di** **struttura logica**(*raw partition*) se non contiene alcun file system. Se nessun file system è appropriato , si usa un **disco privo di struttura logica** detto **raw disk**.

UNIX impiega una partizione priva di struttura per l’area di avvicendamento dei processi ; per questo scopo usa un formato specifico.

Un **disco privo di struttura logica** può anche contenere informazioni necessarie per sistemi RAID di gestione dei dischi.

Le informazioni relative all’avviamento del sistema si possono registrare in un’apposita partizione, che anche in questo caso ha un proprio formato, poiché nella fase d’avviamento il sistema non ha ancora caricato i driver di dispositivo del file system e quindi non può interpretarne il formato.

Nei pc si può configurare una installazione di più SO(dual booted), in questo caso l’area di avvicendamento può contenere un modulo, dentro il boot loader, capace di interpretare diversi file system e durante la fase di **boot** l’utente sceglie quale SO avviare. Nella fase di caricamento il SO esegue il **montaggio della partizione radice**(root partition) che contiene il kernel dei sistema operativo e in alcuni casi altri file di sistema.

Secondo il sistema operativo, il montaggio degli altri volumi avviene automaticamente in questa fase oppure si può compiere successivamente in modo esplicito. Durante l’operazione di montaggio, il sistema verifica la coerenza del file system verificando che la directory abbia formato corretto. Infine, annota nella struttura della **tabella di montaggio** che un file system è stato montato insieme al tipo di file system. In unix , l'operazione di montaggio di un file system si può compiere qualsiasi directory impostando un Flag nella copia dell’i-node tenuta a memoria di quella directory.

**11.2.3 File system virtuali**

Un metodo ovvio ma non ottimale per realizzare più tipi di file system è scrivere procedure di gestione di file e directory separate per ciascun tipo di file system. Al contrario la maggior parte dei SO(compreso Unix) impiega tecniche orientate agli oggetti per semplificare e organizzare in maniera modulare la soluzione. Gli utenti possono accedere a file contenuti in più file system nei dischi locali o anche in file system disponibili tramite la rete.

Per isolare le funzioni di base delle chiamate di sistema dai dettagli di realizzazione si adoperano apposite strutture dati. In questo modo la realizzazione del file system si articola in tre strati principali. Il **primo strato** è l’interfaccia del file system, basata sulle chiamate di sistema open , read, write , close e sui descrittori di file.

Il **secondo strato** si chiama strato del file system virtuale ***VFS***e svolge due funzioni importanti:

1)Separa le operazioni generiche del file system dalla loro realizzazione definendo un'interfaccia VFS uniforme, nello stesso calcolatore possono coesistere più interfacce VFS.

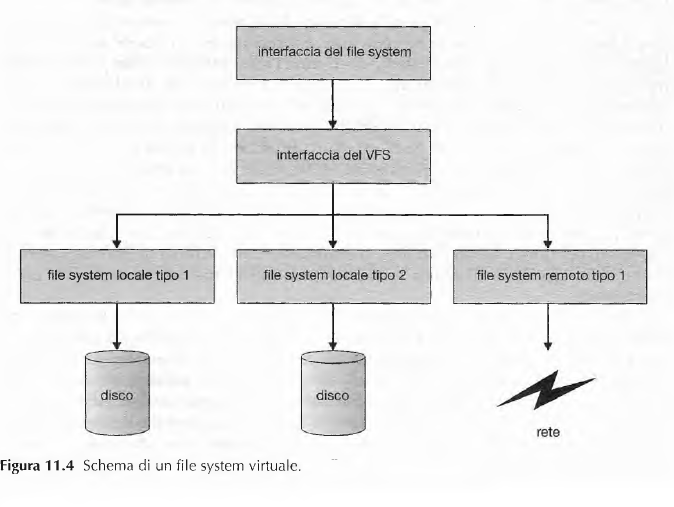
2)Permette la rappresentazione univoca di un file su tutta la rete. Il VFS è basato su una struttura di rappresentazione dei file detta ***vnode*** che contiene un indicatore numerico unico per tutta la rete per ciascun file.

Quindi, il VFS distingue i file locali da quelli remoti, e distingue i file locali secondo i relativi tipi di file system. Il VFS attiva le opzioni specifiche del file system per gestire le richieste locali secondo il tipo di file system, e invoca le procedure del controllo NFS per le richieste remote.

I quattro tipi più importanti di oggetti definiti in questo sistema sono:

* l'oggetto **inode**, che rappresenta il singolo file;
* l'oggetto **file**, che rappresenta un file aperto;
* l'oggetto **superblock**, che rappresenta un intero file system;
* l'oggetto **dentry**, che rappresenta il singolo elemento della directory.

Per ognuno di questi tipi, VFS specifica un insieme di operazioni da implementare. Ciascun oggetto di uno di questi tipi contiene un puntatore a una tabella di funzioni; questa, contiene gli indirizzi delle effettive funzioni che implementano le operazioni richieste. Ogni implementazione dell'oggetto file per uno specifico tipo di file deve implementare tutte le funzioni specificate nella definizione dell'oggetto file.



## 11.3 REALIZZAZIONE DELLE DIRECTORY

La selezione degli algoritmi di allocazione e degli algoritmi di gestione delle directory ha un grande effetto sull'efficienza, le prestazioni e l'affidabilità del file system.

## 11.3.1 Lista lineare

Il più semplice metodo di realizzazione di una directory è basato sull’uso di una **lista lineare** contenente i nomi dei file con puntatori ai blocchi di dati. Questo metodo *è* di facile programmazione, ma la sua esecuzione *è* onerosa in termini di tempo. Per creare un nuovo file occorre prima esaminare la directory per essere sicuri che non esista già un file con Io stesso nome, quindi aggiungere un nuovo elemento alla fine della directory. Per cancellare un file occorre cercare nella directory il file con quel nome, quindi rilasciare lo spazio che gli era assegnato.

Esistono vari metodi per riutilizzare un elemento della directory: si può contrassegnare l’elemento come un usato oppure può essere aggiunto a una lista di elementi di directory liberi ; una terza possibilità prevede la copiatura dell'ultimo elemento della dir in una locazione liberata e la diminuzione della lunghezza,della directory.

Il vero svantaggio dato da una lista lineare di elementi di directory è dato dalla ricerca lineare di un file. Le informazioni sulla directory vengono usate frequentemente, e gli utenti avvertirebbero una gestione lenta e accesso a tali informazioni. In effetti, molti SO impiegano una cache per memorizzare le informazioni sulla directory usata più recentemente; in questo modo si evita la rilettura delle informazioni dal disco.

Una lista ordinata permette una ricerca binaria e riduce il tempo medio di ricerca. Un vantaggio della lista ordinata *è* che consente di produrre l'elenco ordinato del contenuto della directory senza una fase d'ordinamento separata.

**11.3.2 Tabella hash**

Un'altra struttura dati che si usa per realizzare le directory *è* la **tabella hash**. In questo metodo una lista lineare contiene gli elementi di directory, ma si usa anche una struttura dati hash.

La tabella hash riceve un valore calcolato usando come operando il nome del file e riporta un puntatore al nome del file nella lista lineare.

Attraverso questa struttura dati si può diminuire nettamente del tempo di ricerca nella directory. L’inserimento e la cancellazione sono abbastanza semplici anche se occorre prender provvedimenti per evitare **collisioni** cioè situazioni in cui da due nomi di file si ottiene un riferimento alla stessa locazione.

Le maggiori difficoltà legate a una **tabella hash** sono la sua dimensione, che in genere è fissa e la dipendenza della funzione hash da tale dimensione. Alternativamente, ciascun elemento della tabella hash, anziché un singolo valore può essere una lista concatenata, ciò consente di risolvere le collisioni aggiungendovi il nuovo elemento. Le ricerche vengono alquanto rallentate poiché la ricerca per nome può richiedere l’attraversamento della lista concatenata degli elementi in **collisione** della **tabella hash**.

Probabilmente tale metodo è comunque più veloce di una ricerca lineare nell’intera directory.

## 11.4 METODI DI ALLOCAZIONE

La natura ad accesso diretto dei dischi permette una certa flessibilità nella realizzazione dei file. Molti file si memorizzano nello stesso disco. Il problema principale consiste dunque nell’allocare lo spazio per questi file in modo che lo spazio nel disco sia usato efficientemente e l’accesso ai file sia rapido. Esistono 3 metodi principali per l’allocazione dello spazio di un disco: può essere **contigua**, **concatenata** o **indicizzata**.

## 11.4.1 Allocazione contigua

Per usare il metodo di **allocazione contigua** ogni file deve occupare un insieme di blocchi contigui del disco. Gli indirizzi del disco definiscono un ordinamento lineare nel disco stesso. Con questo ordinamento l’accesso al blocco b+1 dopo il blocco b non richiede normalmente alcuno spostamento della testina.

Quindi il numero dei posizionamenti(*seek)* richiesti per accedere a file in cui lo spazio è allocato il modo contiguo è trascurabile, cosi come è trascurabile il tempo di ricerca(*seek time)*.

Accedere ad un file in cui la sua allocazione è contigua, è facile. Quando si usa un accesso sequenziale, il file system memorizza l’indirizzo dell’ultimo blocco a cui è stato fatto riferimento e se è necessario legge il blocco successivo.

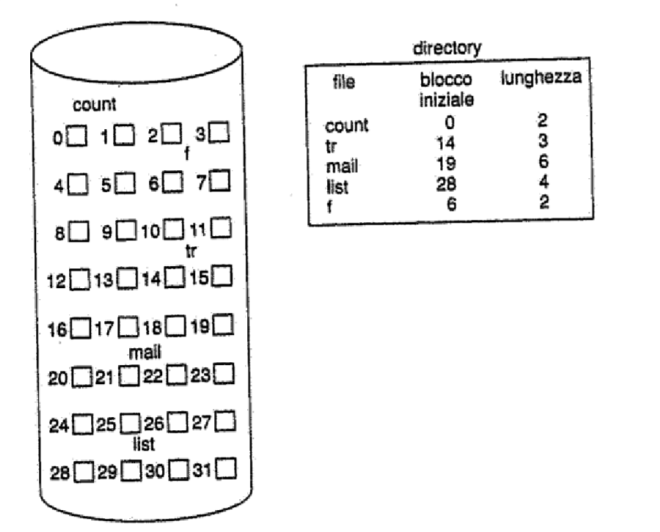
L’**allocazione contigua** presenta però alcuni problemi: una difficoltà riguarda l’individuazione dello spazio per un nuovo file. La realizzazione del **sistema di gestione dello spazio libero** determina il modo in cui tale compito viene eseguito. Si può usare ogni sistema di gestione anche se alcuni sono più lenti di altri.

*Il problema dell’****allocazione contigua*** *dello spazio dei dischi* si può considerare un’applicazione particolare del problema generale dell’**allocazione dinamica** della memoria: il problema generale infatti è quello di soddisfare una richiesta di dimensione *n* data una lista di buchi liberi. I più comuni criteri di ricerca del buco libero sono *first-fit* e *best-fit*. Questi algoritmi soffrono della **frammentazione esterna**. Una soluzione detta **deframmentazione** è quella di copiare i file system in un altro supporto e successivamente riscriverlo in modo compatto. Il costo di questa compattazione è rappresentato dal tempo necessario, ed è particolarmente pesante per i dischi di grande capacità che impiegano l’allocazione contigua.

Alcuni sistemi Richiedono l'esecuzione non in linea di questa funzionalità (file system non montato), Durante questo periodo “morto” (*down time*) il funzionamento normale del sistema non è possibile. Molti sistemi moderni invece lo fanno online (file system montato) a prezzo però di una diminuzione delle prestazioni.

*Un altro problema è la determinazione della quantità di spazio necessaria per un file quando lo si crea*. Se un file riceve poco spazio, può essere impossibile stenderlo: soprattutto nel caso in cui sia dopo il criterio di allocazione del più piccolo tre buchi abbastanza grandi, esistono allora due possibilità. La **prima** e che il programma utente si possa terminare con un idoneo messaggio di errore, l'utente in quel momento, deve allocare più spazio ed eseguire di nuovo il programma. La **seconda** consiste nel trovare un buco più grande, copiare il contenuto del file nuovo spazio e rilasciare lo spazio precedente.

Per ridurre al minimo questo problema alcuni SO fanno uso di una versione modificata: inizialmente si assegna una porzione di spazio contiguo e se questa non è abbastanza grande si aggiunge un’altra porzione di spazio.



## 11.4.2 Allocazione concatenata

**L’allocazione concatenata** risolve tutti i problemi sorti dall’allocazione contigua.

Con questo tipo di allocazione ogni file è composto da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco. La directory contiene un puntatore al primo e all’ultimo blocco del file. Questi puntatori non sono disponibili all’utente, quindi se ogni blocco è formato da 512 byte e un indirizzo ne richiede 4, l'utente vede blocchi di 508 byte.

Per creare un nuovo file si crea semplicemente un nuovo elemento della directory. Con l’allocazione concatenata, Ogni elemento della directory ha un puntatore al primo blocco del file. Un'operazione di scrittura nel file determina la ricerca di un blocco libero attraverso il sistema di gestione dello spazio libero, la scrittura in tale blocco e la concatenazione di tale blocco alla fine del file, mentre per leggere un file occorre semplicemente leggere i blocchi seguendo puntatori da un blocco all'altro. Con l’allocazione concatenata non esiste frammentazione esterna e per soddisfare una richiesta si può usare qualsiasi blocco libero , inoltre non e necessario dichiarare la dimensione del file.

L’allocazione concatenata presenta comunque alcuni svantaggi. Il problema principale riguarda il fatto che può essere usata in modo efficiente solo per i file ad accesso sequenziale.

Per trovare l'i-esimo blocco di un file occorre partire dall’inizio del file e seguire i puntatori finché non si raggiunge l’i-esimo blocco. Ogni accesso a un puntatore implica una lettura del disco, e talvolta un posizionamento della testina. Di conseguenza, per file il cui spazio è assegnato in modo concatenato, la funzione d’accesso diretto è inefficiente.

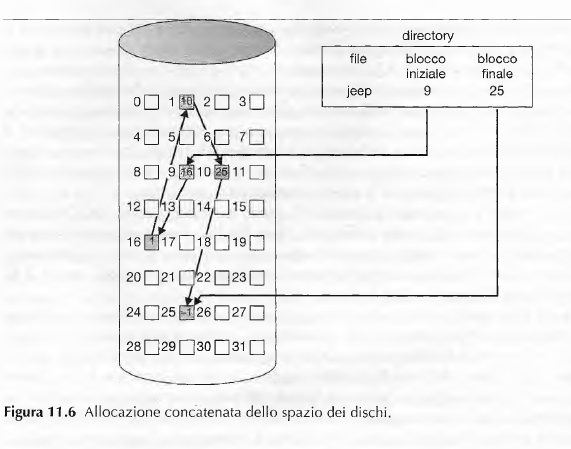
Un altro svantaggio dell’allocazione concatenata riguarda lo spazio richiesto per i puntatori.

Se un puntatore richiede 4 byte di un blocco di 512 byte, allora lo 0,78 per cento del disco è usato per i puntatori anziché per le informazioni.

La soluzione più comune a questo problema consiste nel riunire un certo numero di blocchi contigui in cluster e nell’allocare i cluster anziché i blocchi.

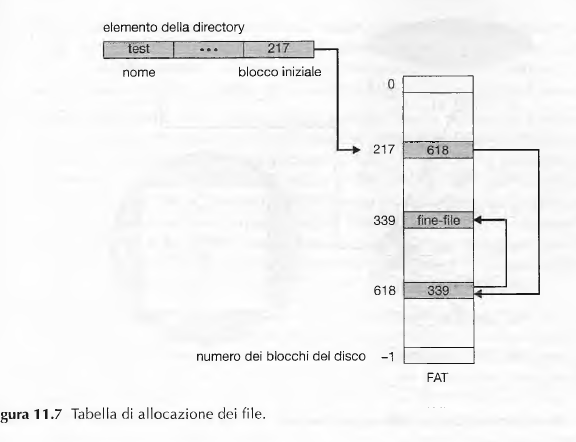
Così i puntatori usano una quantità di spazio di disco che si riduce in modo proporzionale al numero di cluster. Questo metodo migliora la produttività del disco. Il costo di questo metodo è dato da un incremento della frammentazione interna, poiché se un cluster e parzialmente pieno si spreca più spazio di quanto.se ne sprecherebbe con un solo blocco parzialmente pieno.

Un altro problema riguarda l'affidabilità, s’immagini che cosa accadrebbe se un puntatore andasse perduto o danneggiato, questo errore potrebbe causare il collegamento alla lista dei blocchi liberi oppure un altro file. Una soluzione parziale di tale problema consiste nell’usare liste doppiamente concatenate oppure nel memorizzare il nome del file e il relativo numero di blocco in ogni blocco. (schemi onerosi)



Una variante importante del metodo di allocazione concatenata consiste nell’uso della **tabella di allocazione dei file FAT**. Per contenere tale tabella si riserva una sezione del disco all’inizio di ciascun volume.

La FAT ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocco, si usa essenzialmente come una lista concatenata. L'elemento di directory contiene il numero del primo blocco del file. L'elemento della tabella indicizzato da quel numero di blocco contiene a sua volta il numero del blocco successivo del file. Questa catena continua fino all’ultimo blocco, la locazione di un nuovo blocco un file implica semplicemente la locazione del primo elemento della tabella con valore 0 e la sostituzione del valore di fine del file precedente con l'indirizzo del nuovo blocco.

Lo schema di allocazione basato sulla FAT se non si usa una cache può causare un significativo numero di posizionamenti della testina. La testina deve spostarsi all’inizio del volume per leggere la FAT e trovare la locazione del blocco in questione, quindi raggiungere la locazione del blocco stesso e poi leggere il blocco. Un vantaggio è dato dall’ottimizzazione del tempo d’accesso diretto, poiché la testina del disco può trovare la locazione di ogni blocco leggendo le informazioni contenute nella FAT.

## 11.4.3 Allocazione indicizzata

**L’allocazione concatenata** risolve il problema della **frammentazione esterna** e quello della dichiarazione della dimensione dei file. Tuttavia, in mancanza di una FAT l’allocazione concatenata non è in grado di sostenere un efficiente accesso diretto, poiché i puntatori ai blocchi sono sparsi per tutto il disco. L’**allocazione indicizzata** risolve questo problema raggruppando tutti i puntatori in una sola locazione: **il blocco indice**.

Ogni file ha il proprio **blocco indice**: si tratta di un array d’indirizzi di blocchi del disco.

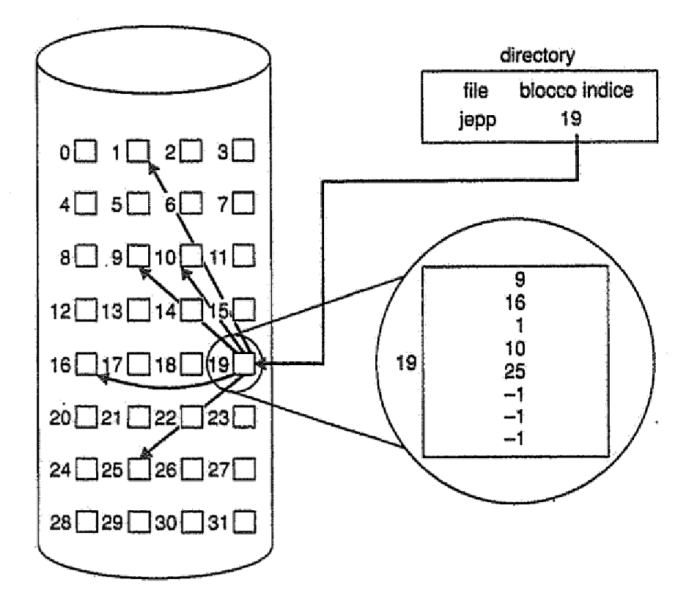
L’i-esimo elemento del **blocco indice** punta all'i-esimo blocco del file. La directory contiene l’indirizzo del **blocco indice**. Per individuare e leggere l'i-esimo blocco occorre usare il puntatore che si trova nell’i-esimo elemento del blocco indice, per poi localizzare e leggere il blocco desiderato.

Una volta creato il file, tutti i puntatori del **blocco indice** sono impostati a *nil* .

Quando si scrive l'i-esimo blocco per la prima volta, il gestore dei blocchi liberi fornisce un blocco; l’indirizzo di questo blocco viene inserito nell’i-esimo elemento del blocco indice.

Poiché ogni blocco libero del disco può soddisfare una richiesta di maggiore spazio, l’allocazione indicizzata consente l’accesso diretto senza soffrire di **frammentazione esterna**.

Lo spazio aggiuntivo richiesto dai puntatori del blocco indice e generalmente maggiore dello spazio aggiuntivo necessario per la locazione concatenata. Si consideri il comune causa di un file con uno o due blocchi, con la locazione indicizzata occorre allocare un intero blocco indice, anche se solo uno o due puntatori sono diversi da nil



Questo punto solleva la questione della dimensione del blocco indice. Ogni file deve avere un blocco indice, quindi è auspicabile che questo sia quanto più piccolo è possibile;

ma se il blocco indice è troppo piccolo non può contenere un numero di puntatori sufficiente per un file di grandi dimensioni, quindi è necessario disporre di un meccanismo per gestire questa situazione.

**♦ Schema concatenato.** Un blocco indice è formato normalmente di un solo blocco di disco; perciò ciascun blocco indice può essere letto e scritto esattamente con un’operazione. Per permettere la presenza di lunghi file è possibile collegare tra loro parecchi blocchi indice. L'indirizzo successivo, vale a dire l'ultima parola del blocco indice, è **nil** per un file piccolo oppure è un **puntatore** a un altro blocco indice per un file lungo.

**♦ Indice a più livelli.** Una variante della rappresentazione concatenata consiste nell’impiego di un blocco indice di primo livello che punta a un insieme di blocchi indice di secondo livello che loro volta puntano ai blocchi dei file. Per accedere a un blocco, il sistema operativo usa l'indice del primo livello , con il quale individua il blocco indice di secondo livello, è connesso trova il blocco dei dati richiesto. Due livelli di indici consentono 1.048.576 blocchi di dati, che permettono di avere file sino a 4GB.

**♦ Schema combinato.** Un’altra possibilità è la soluzione adottata da UNIX che consiste nell’avere i primi 15 puntatori del blocco indice nell’inode del file. I primi 12 puntano a **blocchi diretti** (cioè contengono direttamente gli indirizzi di blocchi contenenti dati del file), gli altri 3 puntano a **blocchi indiretti.** Il primo puntatore di un blocco indiretto e l'indirizzo di un blocco **indiretto singolo**; si tratta di un blocco indice che contiene indirizzi di blocchi che contengono dati. Quindi c'è un puntatore di blocco **indiretto doppio** che contiene l'indirizzo di un blocco che a sua volta contiene gli indirizzi di blocchi contenenti puntatori agli effettivi blocchi di dati. L'ultimo puntatore contiene l'indirizzo di un blocco **indiretto triplo.**

## Cattura.PNG

## 11.4.4 Prestazioni

I **metodi d’allocazione** presentati hanno diversi livelli di efficienza di memorizzazione e differenti tempi d’accesso ai blocchi di dati. Prima di sceglier un metodo di allocazione è necessario determinare il modo in cui si usano i sistemi: un sistema con una prevalenza di accessi sequenziali farà uso in un metodo differente da quello con prevalenza di accessi diretti.

Per qualsiasi tipo di accesso, l’**allocazione contigua** richiede un solo accesso per ottener un blocco. Poiché è facile tenere l’indirizzo iniziale del file in memoria, si può calcolare immediatamente l’indirizzo del disco dell’i-esimo blocco, oppure del blocco successivo, e leggerlo direttamente.

Con l’**allocazione concatenata** si può tenere in memoria anche l’indirizzo del blocco successivo e leggerlo direttamente. Questo metodo è valido per l’accesso sequenziale mentre per quel che riguarda l’accesso diretto, un accesso all’i-esimo blocco può richiedere *i* letture del disco.

Da tutto ciò segue che alcuni sistemi gestiscono i file ad **accesso diretto** usando l’**allocazione contigua** e i file ad **accesso sequenziale** tramite l’**allocazione concatenata**. Per questi sistemi, il tipo di accesso si deve dichiarare al momento della creazione del file. Un file creato per l'accesso sequenziale è un file concatenato e non si può usare per l'accesso diretto. Un file creato per l'accesso diretto è continuo e consente entrambi i tipi di accesso purché.se ne dichiari la lunghezza massima al momento della creazione.

L’**allocazione indicizzata** è più complessa. Se il blocco indice è già in memoria l’accesso può essere diretto. Tuttavia, per tenere il blocco indice in memoria occorre una quantità di spazio notevole, Se questo spazio di memoria non è disponibile, occorre leggere prima il blocco indice e quindi il blocco dati desiderato. Se un file è estremamente grande, prima di leggere il blocco dei dati occorre leggere tutti i blocchi indice per seguire la catena di puntatori.

Quindi le prestazioni di tale allocazione dipendono dalla struttura dell’indice, dalla dimensione del file e dalla posizione del blocco. Alcuni SO combinano le allocazioni in base alle proprie esigenze.

## 11.5 GESTIONE DELLO SPAZIO LIBERO

Poiché la quantità di spazio dei dischi è limitata, è necessario utilizzare lo spazio lasciato dai file cancellati per scrivere nuovi file. Per tener traccia dello spazio libero in un disco, il sistema conserva una **lista dello spazio libero**. Per creare file occorre cercare nella lista dello spazio la quantità di spazio necessaria e assegnarla al nuovo file, quindi rimuovere questo spazio dalla lista. Quando si cancella un file, si aggiungono alla lista dello spazio libero i blocchi di disco a esso assegnati.

## 11.5.1 Vettore di bit

Spesso la lista dello spazio libero si realizza come una **mappa di bit** o **vettore di bit**. Ogni blocco è rappresentato da un bit: se il blocco è libero il bit è 1, altrimenti è 0.

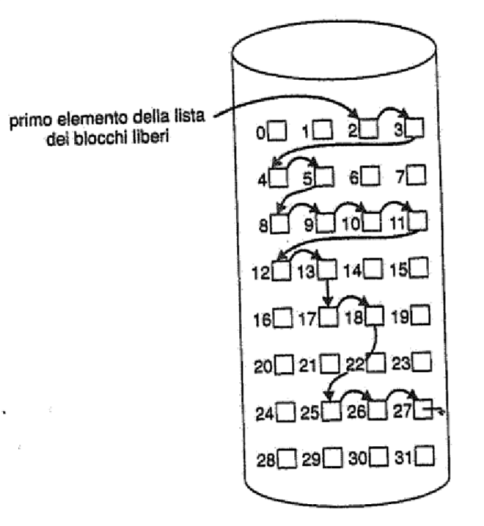
I vantaggi principali che derivano da questo metodo sono la sua relativa semplicità ed efficienza nel trovare il primo blocco libero e n blocchi liberi consecutivi nel disco.

Sfortunatamente i vettori di bit sono efficienti solo se tutto il vettore è mantenuto in memoria centrale, e viene di tanto in tanto scritto in memoria secondaria allo scopo di consentire eventuali operazioni di ripristino, tale soluzione non è applicabile ai dischi più grandi.

## 11.5.2 Lista concatenata

Un altro metodo di gestione degli spazi liberi consiste nel collegarli tutti, tenere un puntatore al primo di questi in una speciale locazione del disco e caricarlo in memoria. Questo primo blocco contiene un puntatore al successivo blocco libero e cosi via. Questo schema non è tuttavia efficiente: per attraversare la lista occorre leggere ogni blocco e l’operazione richiede un notevole tempo di I/O.

Fortunatamente l’attraversamento della lista dello spazio libero non è un’operazione frequente. Di solito il sistema operativo ha semplicemente bisogno di un blocco libero perché posso assegnarle un file , quindi si usa il primo blocco della lista. Il metodo che fa uso della FAT include il conteggio dei blocchi liberi nella struttura dati per l’allocazione.



## 11.5.3 Raggruppamento

Una possibile modifica del metodo della lista dello spazio libero prevede la memorizzazione degli indirizzi in n blocchi liberi nel primo di questi. I primi n-1 sono effettivamente liberi, l’ultimo contiene l’indirizzo ai successivi n blocchi liberi. L’importanza è data dalla possibilità di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi.

## 11.5.4 Conteggio

Generalmente più blocchi contigui si possono allocare o liberare contemporaneamente, soprattutto quando lo spazio viene allocato usando l’algoritmo di allocazione contigua. Anziché tener una lista di n indirizzi liberi, è sufficiente tenere l’indirizzo del primo blocco libero e il numero n di blocchi liberi contigui che seguono il primo blocco. Ogni elemento della lista dello spazio libero è formato da un indirizzo del disco e un contatore. Questo metodo , il tracciamento dello spazio libero, è simile al metodo generale per allocare i blocchi.

**11.6 EFFICIENZA E PRESTAZIONI**

I dischi tendono di solito a essere il principale collo di bottiglia per le prestazioni in un sistema essendo i

più lenti tra i componenti più rilevanti di un calcolatore.

## 11.6.1 Efficienza

L’uso efficiente di un disco dipende fortemente dagli algoritmi usati per l’allocazione del disco e la gestione delle directory. Ad esempio, gli *i-node* di UNIX Sono assegnati preventivamente in un volume. Anche un disco “vuoto “impiega una certa percentuale del suo spazio per gli inode. D’altra parte, l’allocazione preventiva migliorano le prestazioni del file system.

Si devono tenere in considerazione anche il tipo di dati normalmente contenuti in un elemento di una directory (o di un inode). Di solito si memorizza la *data dell’ultima scrittura* per fornire informazioni all’utente e per determinare se per il file occorre la creazione o l’aggiornamento di una copia di riserva. Per mantenere queste informazioni, ogni qualvolta si legge un file, si deve aggiornare un campo della directory. Questa modifica richiede la lettura nella memoria del blocco, la modifica della sezione e la riscrittura del blocco nel disco.

Quindi, ogni volta che si apre un file per la lettura, si deve leggere e scrivere anche l’elemento della directory ad esso associato. Ciò può essere inefficiente per file cui si accede frequentemente. In generale, è necessario considerare l’influenza sull’efficienza e sulle prestazioni di ogni informazione che si vuole associare ad un file.Una delle difficoltà nella scelta della dimensione dei puntatori, o di qualsiasi altra dimensione di allocazione fissa all’interno di un sistema operativo, è la pianificazione degli effetti provocati dal cambiamento della tecnologia.