Lezione 24 [09/12/22]

Capitolo 12: Realizzazione del File System

- Struttura del file system
- Realizzazione del file system
- Realizzazione delle directory
- Metodi di allocazione
- Gestione dello spazio libero
- Efficienza e prestazioni
- Ripristino
- NFS
- Esempio: il file system WAFL

File system

Tutte le applicazioni informatiche hanno bisogno di memorizzare e recuperare informazioni. Abbiamo tre requisiti essenziali per la memorizzazione delle informazioni a lungo termine:

- Si deve poter memorizzare un'enorme quantità di informazioni
- Le informazioni devono sopravvivere alla terminazione del processo che le usa
- Più processi devono poter accedere alle informazioni in modo concorrente

Il **file system** è l'insieme della strutture dati e dei metodi che ci permettono la registrazione e l'accesso a dati e programmi presenti in un sistema di calcolo

Il file system risiede permanentemente nella memoria secondaria, progettata per mantenere in maniera non volatile grandi quantità di dati

Struttura del file system

In generale i dischi sono il supporto di memoria secondaria su cui viene conservato i file system

I dischi hanno due caratteristiche importanti:

- Possono essere riscritti localmente:
  - E' possibile leggere un blocco dal disco, modificarlo e quindi riscriverlo nella stessa posizione
- E' possibile accedere direttamente a qualsiasi blocco del disco
  - Quindi risulta semplice accedere a qualsiasi file, sia in modo sequenziale, casuale, che in modo diretto
  - Si può passare da un file all'altro solamente spostando le testine di lettura-scrittura e attendendo la rotazione del disco

Per migliorare l'efficienza dell'I/O, i trasferimenti tra memoria centrale e disco avvengono per **blocchi**

Ogni blocco è composto da uno o più settori; la dimensione di essi può variare tra 32 byte a 4098 byte a seconda del tipo di unità disco (di solito è 512 byte)

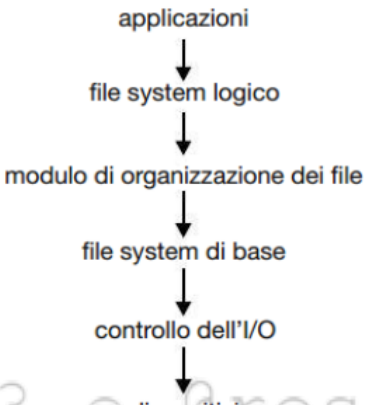
Per fornire un efficiente accesso al disco, il S.O. fa uso di uno o più **file system**

Il file system presenta due problemi di progettazione:

- Interfaccia utente: implica la definizione di un file dei suoi attributi, delle operazioni permesse e della struttura delle directory per l'organizzazione dei file
- Algoritmi e strutture dati: in modo da permettere la corrispondenza del file system logico ai dispositivi di memoria secondaria

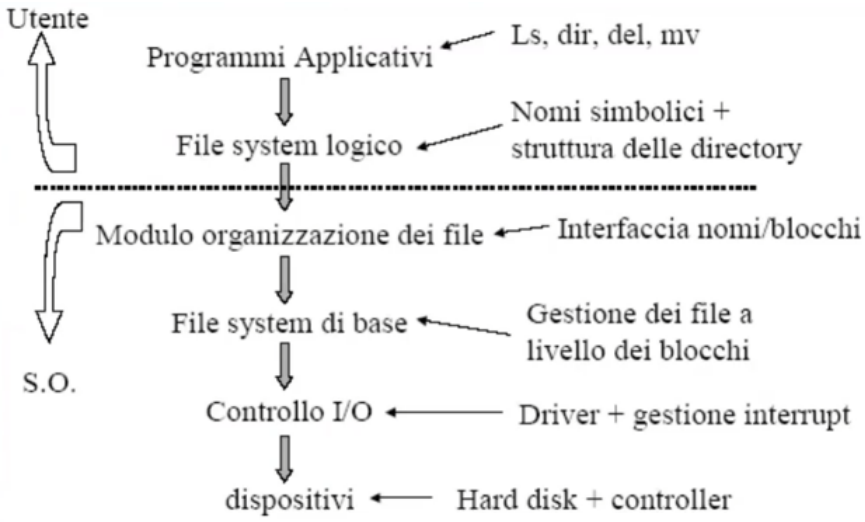
Il file system è composto da molti livelli distinti; è **stratificato**

Ogni livello si serve delle funzioni dei livelli inferiori per crearne di nuove impiegate dai livelli superiori



- **File system logico** → Gestione dei metadati
- **Modulo di organizzazione dei file** → Gestione dei blocchi logici e fisici
- **File system di base** → Gestione dell'invio dei dati
- **Controllo dell'I/O** → Driver dei dispositivi

File system stratificato



- Il livello più basso è il **controllo dell'I/O**, è costituito dai **driver dei dispostivi** e dai gestori di interrupt; si occupa di trasferire le informazioni tra memoria centrale e memoria secondaria
  - Un driver di dispositivi può essere pensato come un traduttore
  - Il suo input consiste di comandi ad alto livello; il suo output consiste di istruzioni a basso livello (usate dal controllore che fa da interfaccia tra i dispostivi di I/O e il resto del sistema). Esso scrive in locazioni di memoria del controllore per indicare quali azioni deve compiere il dispositivo e su quali locazioni
- Il **file system di base** deve soltanto inviare dei generici comandi all'appropriato driver di dipositivo per leggere e scrivere blocchi fisici sul disco
  - Ogni blocco fisico è identificato dal suo indirizzo numerico nel disco
  - Questo strato gestisce anche il buffer di I/O (dove vengono allocati i blocchi prima del trasferimento) e la cache (dove si conservano i metadati del file system che sono usati più frequentemente, per 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 utilizzati e la locazione dei file, il modulo di organizzazione dei file può tradurre gli indirizzi dei blocchi logici (numerati da 0 a N) che il file system di base deve trasferire, negli indirizzi dei blocchi fisici. Il numero di blocchi logici non ha lo stesso numero dei blocchi fisici, per questo c'è bisogno di una traduzione
  - Il modulo di organizzazione dei file comprende anche il **gestore dello spazio libero** che i blocchi non allocati e li mette a disposizione del modulo di organizzazione dei file quando sono richiesti
- Il **file system logico** gestisce i **metadati**: si tratta di tutte le strutture del file system, eccetto gli effettivi dati (il contenuto dei file)
  - Gestisce la struttura della directory per fornire al modulo di organizzazione dei file le informazioni di cui questo necessita, dato un nome simbolico di file
  - Mantiene le strutture di file tramite i **blocchi di controllo dei file** (**file control block**, **FCB**) (detti **inode** in UNIX)
  - Il file system logico è responsabile anche della protezione e della sicurezza:
    - Alcuni S.O., tra cui UNIX, trattano una directory esattamente con un file, con un campo che indica che si tratta di una directory
    - Altri S.O., ad esempio Windows NT, usano system call diverse per file e directory

Nei file system stratificati la duplicazione è ridotta al minimi: il controllo dell'I/O e il codice base del file system possono essere utilizzati da più file system. Ogni file system, poi, ha i propri moduli che gestiscono il file system logico e l'organizzazione dei file

I file system stratificati aumentando l'overhead del S.O che genera un decadimento delle prestazioni

In UNIX si usa il **file system UNIX (UFS),** mentre Windows adotta i formati FAT, FAT32 E NTFS

Nonostante LINUX possa funzionare con più di quaranta file system diversi, quello standard è noto come **file system esteso** (versioni ext2 e ext3)

Esistono anche file system distribuiti, in cui un file system su server è montato da uno o più client in una rete

Realizzazione del file system

Strutture su disco utilizzate dal file system (Introduzione)

Per realizzare un file system sono necessarie numerose strutture, sia nei dischi che in memoria

Fra le strutture presenti nei dischi ci sono le seguenti, diverse in base all'implementazione del file system:

- **Blocco di controllo d'avviamento (boot control block)**: contiene le informazioni necessarie al sistema per l'avviamento di un sistema operativo da quel volume (o partizione); se il disco non contiene un sistema operativo, tale blocco può essere vuoto. Di solito è il primo blocco di un volume. In UFS si chiama **blocco di avviamentob (boot block)**; in NTFS **settore d'avviamento della partizione (partition boot sector)**
- Il **blocco di controllo del volume o blocco di controllo delle partizione (volume control block o partition control block)**: contiene i dettagli riguardanti il relativo volume (o partizione), come il numero e la dimensione dei blocchi nella partizione, il contatore dei blocchi liberi e i relativi puntatori , il contatore degli FCB liberi e i relativi puntatori. In UFS si chiama **superblocco**; in NTFS si chiama **tabella principale dei file** (master file table, **MFT**)
- **Struttura di directory** (una per file system)**:** che si usa per organizzare i file. Per UFS comprende i nomi dei file e i numeri di **inode** associati; per NTFS è memorizzata nella tabella principale dei file
- **Blocco di controllo del file (FCB):** contiene molti dettagli relativi al file. Ha un identificatore unico per poterlo associare a una voce della directory. In NTFS , queste informazioni sono memorizzate all'interno della tabella principale dei file
- **Descrittore di file (ad es inode)**: che contengono permessi di accesso ai file, proprietari dimensioni e locazioni dei blocchi di dati, etc

Queste strutture variano a seconda della particolare implementazione del file system (UNIX, WINDOWDS NT ed XP, etc)

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 montaggio, si aggiorna in file system e si eliminano allo smontaggio (?)

Fra le strutture in memoria ci sono le seguenti:

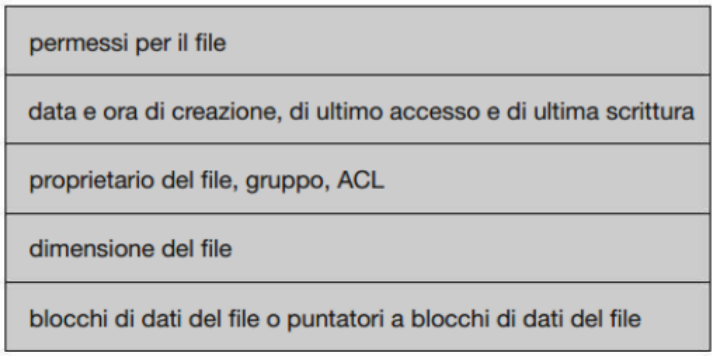
- **Tabella di montaggio o tabella delle partizioni**: contiene le informazioni relative a ciascun volume (partizione) montato/a
- **Cache della struttura di directory:** contenente le informazioni relative a tutte le directory a cui i processi hanno fatto recentemente accesso (per le directory che costituiscono dei punti di montaggio, può esserci un puntatore alla tabella dei volumi)
- **Tabella generale (di sistema) dei file aperti:** contenente una copia dell'FCB per ciascuno file aperto, insieme ad altre informazioni
- **Tabella dei file aperti per ciascun processo:** contenente un puntatore all'appropriato elemento della tabella generale dei file aperti, insieme ad altre informazioni
- **Buffer** che conservano blocchi del file system durante la loro lettura o scrittura sul disco

Apertura di file

Per creare un file, le applicazioni eseguono una chiamata al file system logico, che conosce il formato della struttura della directory. Si crea quindi un nuovo FCB (o, nel caso dei file system che creano tutti gli FCB al momento della loro istallazione, se ne alloca uno libero

Il sistema carica in memoria la directory appropriata, la aggiorna con il nome del nuovo file e il FCB associato e la scrive di nuovo sul disco

FCB:



Indipendentemente da questioni strutturali, il file system logico può basarsi sul modulo che si occupa dell'organizzazione dei file per far corrispondere l'I/O su directory ai numeri di blocchi di disco, che poi si passano al file system di base e al sistema per il controllo dell'I/O

Prima che possa essere impiegato per procedure di I/O, un file deve essere **aperto**

La chiamata di sistema `open()` passa un nome di file al file system logico. Per controllare se il file è già in usod a parte di qualche altro processo, la chiamata di `open()` prima esamina la tabella dei file aperti di sistema. Nel caso venga trovato, aggiunge un elemento alla tabella dei file aperti del processo che punta alla tabella dei file aperti di sistema (facendo così si elimina l'overhead)

Se il file non è aperto, se ne cerca il nome nella directory. Alcune porzioni della struttura della directory sono tenute in memoria (cache) per accelerare le operazioni sulle directory

Una volta trovato il file, si copia l'FCB nella tabella (dimensione, proprietario, permessi d'accesso, allocazione dei blocchi di dati etc) dei file aperti di sistema tenuta in memoria. Questa tabella tiene anche traccia del numero di processi che hanno il file aperto

Oltre alla creazione di un elemento nella tabella dei file aperti del processo con un puntatore alla tabella di sistema, si creano anche altri campi: un puntatore alla posizione corrente nel file (per successive `read()` e `write()` e il tipo di accesso specificato all'apertura del file. La `open()` riporta una puntatore all'elemento appropriato nella tabella dei file aperti del processo, in modo che tutte le operazioni sul file si svolgeranno tramite quel puntatore

L'indice della tabella dei file aperti di sistema viene riportato al programma utente e tutti i successivi riferimenti vengono effettuati attraverso tale indice (il libro parla di "nome del file")

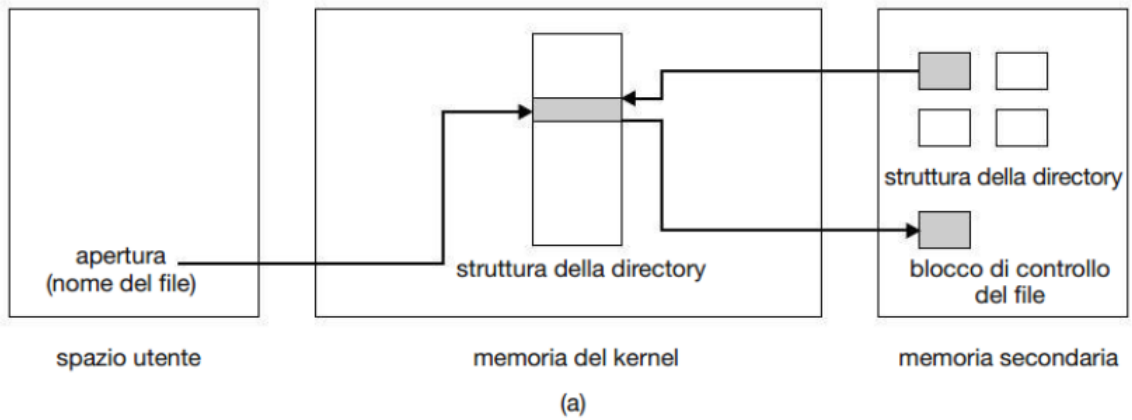
- L'indice prende nomi diversi: nel sistema UNIX è detto **descrittore di file;** in Windows NT **file handle** (maniglia), in altri sistemi, **file control block**

Quando un file viene chiuso da un processo, si cancella il relativo elemento nella tabella dei file aperti del processo e si decrementa il contatore associato al file nella

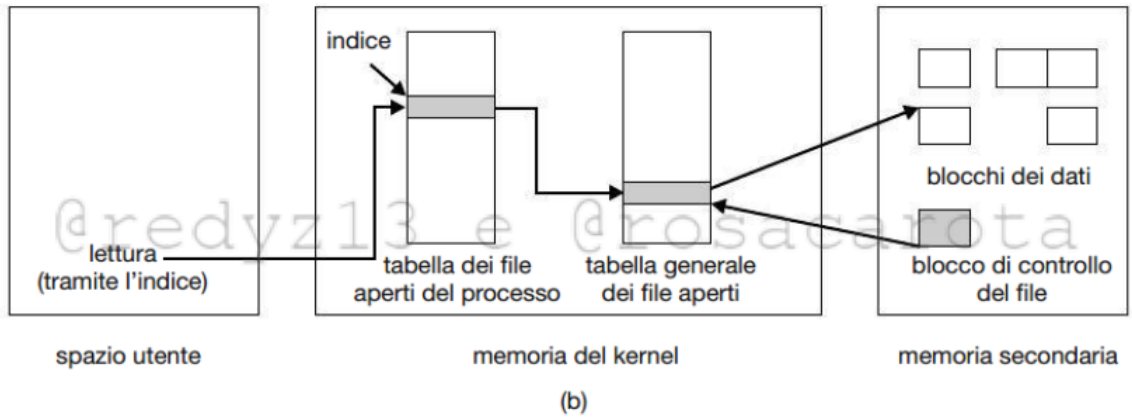
tabella di sistema

Se il file viene chiuso da tutti i processi, si riscrivono i metadati aggiornati nella struttura della directory nei dischi e si cancella il relativo elemento nella tabella di sistema dei file aperti

Apertura e lettura di un file:



(a)



(b)

a) si riferisce all'apertura di un file

b) si riferisce alla lettura di un file

Partizioni e montaggio

Un disco può essere strutturato in vari modi a seconda del S.O. che lo gestisce:

- Può essere suddiviso in più partizioni
- Un volume può comprendere più partizioni su molteplici dischi (RAID)

Trattiamo il primo caso (perché i RAID già cianna abbuffat a ualler abbastanza)

Raw partition: partizione priva di struttura logica, altrimenti contiene un file system

Raw disck: disco privo di struttura logica, si usa se nessun file system risulta appropriato all'utilizzo che se ne vuole fare. Per esempio, in UNIX si utilizza questo tipo di disco come area di avvicendamento dei processi

Le informazioni riguardante l'avvio del sistema possono essere mantenute su un'apposita partizione, che anche in questo caso ha il proprio formato. Questa partizione è più una serie sequenziale di blocchi, che si carica in memoria come un'immaginee. L'esecuzione di questa avviene a una locazione prefissata

// sproloquio sul boot loader che boh

In alcuni casi, l'area di avviamento può anche contenere un **boot loader** capace di caricare il kernel e avviarne l'esecuzione. Viene spesso utilizzato su sistemi dove sono installati più S.O. Esso, infatti, è capace di interpretare diversi file system e diversi S.O.
Il disco può avere diverse partizioni contenenti diversi SO e diversi file system

Nella fase di caricamento del S.O. si esegue il montaggio della **partizione radice** (**root partition**) che contiene il kernel del sistema operativo e in alcuni casi altri file di sistema. Il montaggio degli altri volumi può avvenire sia automaticamente che manualmente.
durante il montaggio, il sistema verifica che il dispositivo contenga un file system valido chiedendo al dispositivo di leggere la directory di dispositivo e verificando che la directory abbia formato corretto. Se non fosse così, c'è bisogno di una correzione (che potrebbe anche essere da parte dell'utente)

Il sistema, quindi, annota nella **tabella di montaggio** in memoria che il file system è stato montato e il tipo di file system

File system virtuali

I sistemi operativi moderni si ritrovano a gestire diversi tipi di file system contemporaneamente

Bisogna quindi considerare il modo in cui un SO può consentire l'integrazione di diversi tipi di file system in un'unica struttura di directory in modo da permettere agli utenti di spostarsi da un tipi di file system ad un altro

Un metodo potrebbe essere quello di scrivere procedure di gestione separate di file e directory per ciascun file system

- Poco efficiente e dispendioso in termini di dimensioni del S.O.
- La maggior parte dei S.O. impiega tecniche orientate agli oggetti per semplificare ed organizzare in maniera modulare la soluzione.
  L'uso di queste tecniche rende possibile l'implementazione, nella stessa struttura, di tipi di file system diversi , compresi i file system di rete come l'NFS

Si possono quindi isolare le funzioni di base delle chiamate di sistema dai dettagli implementativi del singolo file system adoperando opportune 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 del sistema `open()`, `write()`, `read()`, `close()` e sui descrittori di file
- Il secondo strato si chiama **file system virtuale (virtual file system - VFS)** e svolge le seguenti funzioni:
  - Separa le operazioni generiche del file system dalla loro implementazione, definendo un'interfaccia uniforme
    Nello stesso calcolatore coesistono più interfacce VFS che permettono l'accesso ai diversi file system
  - 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 ha un indicatore unico per ciascun file per tutta la rete (gli inode di UNIX sono unici solo all'interno dei un singolo file system). Questo risulta necessario per gestire i file system di rete. (il kernel ha una struttura vnode per ogni nodo attivo, che

sia un file o una directory). Identifica univocamente ciascun file presente su una partizione montata

- Il VFS distingue i file locali da quelli remoti, e i file locali stessi a seconda del file system (attiva procedure specifiche di un file system per gestire le richieste locali e invoca le procedure del protocollo NFS per le richieste remote)

Gli handle dei file si costruiscono a partire dai vnode e si inviano alle procedure attivate dal VFS come argomenti

- Il terzo strato, lo strato che realizza il codice di uno specifico file system o il protocollo NFS, è il più basso dell'architettura

VFS di Linux

I 4 tipi più importanti di oggetti definiti nel VFS sono:

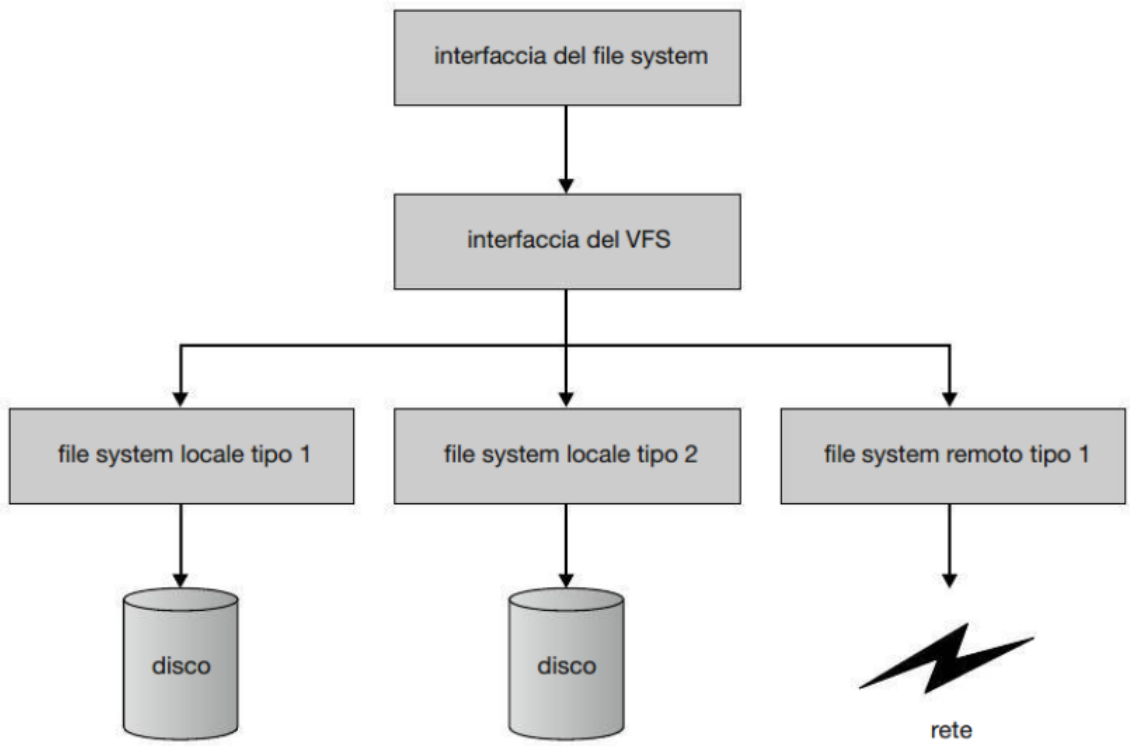
- L'**oggetto inode**, che rappresenta il singolo file
- L'**oggetto file**, che rappresenta un file aperto
- L'**oggetto superblock**, che rappresenta un intero file system
- L'**oggetto dentry**, che rappresenta il singolo elemento della directory

Per ognuno, VFS specifica un insieme di operazioni da implementare

Le funzioni specificate sono nella definizione degli oggetti e, una volta implementate, ogni oggetto conterrà un puntatore a una tabella delle funzioni (che contiene gli indirizzi delle vere funzioni)

VFS non conosce l'implementazione delle funzioni, fa solo riferimento alla tabella delle funzioni per le sua operazioni

Schema di un file system virtuale:



Directory

Quando un file viene aperto il sistema operativo usa il nome di percorso fornito dall'utente per individuare l'elemento corrispondente all'interno della directory

La voce nella directory fornisce le informazioni necessarie a trovare i blocchi sul disco

A seconda del metodo di rappresentazione scelto, questa informazione può essere:

- l'indirizzo sul disco dell'intero file (allocazione contigua)
- l'indice del primo blocco
- l'indice di i-node

La funzione principale della struttura delle directory è di tradurre il nome ASCII del file nelle informazioni necessarie a individuare i dati

Realizzazione delle directory

Come realizzare le directory:

Lista lineare che contiene i nomi dei file con puntatori ai blocchi di dati

- Facile da realizzare
- Dispendioso in termini di tempo da eseguire

Creazione di un file:
Si controlla prima la directory per essere sicuri che non esista già un file con lo stesso nome quindi si aggiunge un nuovo elemento alla fine della directory

Cancellare un file:
Si ricerca nella directory il file con quel nome, quindi rilasciare lo spazio che gli era stato assegnato

Riutilizzo di un elemento della directory:

- Si contrassegna come non usate (tramite nome speciale o bit d'uso)
- Può essere aggiunto ad una lista di elementi di directory liberi
- Si copia l'ultimo elemento della durectiry un una locazione liberata e si diminuisce la lunghezza della directory

Lo svantaggio dato dalla lista lineare è quello di ricerca lineare di un file. Per ovviare a questo problema si puà usare una cache software per memorizzare le informazioni di directory usate più recentemente

Una lista ordinata permette la ricerca binaria, ma complica la creazione e la cancellazione

Tabella hash - lista lineare con strutture dati hash

Riceve un valore calcolato a partire dal nome del file e riporta un puntatore al nome del file nella lista lineare

- Diminuisce il tempo di ricerca nella directory
- Bisogna gestire le **collisioni** (situazione dove due file diversi anno lo stesso indirizzo hash)
- Ha dimensione fissata
  Soluzione: ciascun elemento della tabella hash può essere una lista concatenata anziché un singolo valore, ma vengono rallentate le ricerche

Un problema strettamente collegato all'allocazione dei dati è dove debbano essere memorizzati gli attributi

Una possibilità ovvia è di memorizzarli direttamente all'interno della directory

Metodi di allocazione

La natura ad accesso diretto dei dischi permette una certa flessibilità nell'implementazione dei file

In quasi tutti i casi, molti file vengono memorizzati sullo stesso disco

Il problema principale consiste dunque nell'allocare lo spazio per i file in modo che lo spazio sul disco venga utilizzato efficientemente e l'accesso ai file sia rapido

Il metodo di allocazione dello spazio su disco descrive come i blocchi fisici del disco vengono allocati ai file

Esistono tre metodi principali per l'assegnazione dello spazio di un disco:

- Allocazione contigua
- Allocazione concatenata
- Allocazione indicizzata

Allocazione contigua

Lo schema più facile consiste nel memorizzare ogni file come un blocco contiguo di dati sul disco

Ciascun file occupa un gruppo di blocchi contigui sul disco, cioè **run**

Questo metodo ha due vantaggi significativi:

- E' semplice da implementare, dato che tutti i blocchi del file sono univocamente identificati da un numero
- Le prestazioni sono eccellenti dal momento che l'intero file può essere letto dal disco con singola operazione

  Supponendo che un unico job stia accedendo al disco, l'accesso al blocco b+1 dopo il blocco b non richiede spostamento di testina. Se la testina deve esser spostata (dall'ultimo settore di un cilindro al primo settore del cilindro successivo, lo spostamento è di una sola traccia. Di conseguenza , questo metodo diminuisce il numero di posizionamenti richiesti (**seek**) per accedere ai file, e risulta inoltre trascurabile il tempo di ricerca (**seek time**) quando è necessario

Per reperire il file occorrono solo la **locazione iniziale** (#blocco iniziale) e la **lunghezza** (**numero di blocchi**). L'elemento nella directory per ciascun file indica l'indirizzo del blocco iniziale e la lunghezza dell'aria assegnata

L'accesso a file può essere sia diretto che sequenziale:

- **Accesso sequenziale:**

  Il file system memorizza l'indirizzo dell'ultimo blocco a cui è stato fatto riferimento e, se necessario, legge il blocco successivo

- **Accesso diretto (casuale):**

  In caso di accesso diretto al blocco i di un file che inizia al blocco b, si può accedere immediatamente al blocco $b+i$

Problema principale: allocazione dello spazio per nuovi file

Questo si rifà al problema dell'allocazione dinamica della memoria (soddisfare una richiesta di dimensione n data una lista di buchi liberi). Come già sappiamo, gli algoritmi prediletti per questo sono first-fit e best-fit, ma hanno problemi di frammentazione esterna

Una soluzione per questo problema è quella di copiare un intero file system su una altro disco, liberando così lo spazio nel primo disco. Una volta fatto questo, si ricopia man mano tutti i file nel primo disco, che ora è diventato un enorme blocco di memoria libero

Questo schema **compatta** tutto lo spazio in uno spazio contiguo, ma il tempo necessario per questa compattazione è alto (molti ci impiegherebbero ore). Alcuni sistemi, inoltre, richiedono che questa operazione venga svolta **offline (non in linea)**, cioè con il file system non montato. In questo modo, il sistema non potrebbe essere utilizzato

La maggiorparte dei sistemi moderni sono capaci di eseguire la **deframmentazione online (in linea)**, cioè durante il loro normale funzionamento

Altro problema: determinazione dello spazio necessario per allocare un file

Se un file riceve poco spazio, risulta impossibile estenderlo in modo contiguo

Esistono due soluzioni:

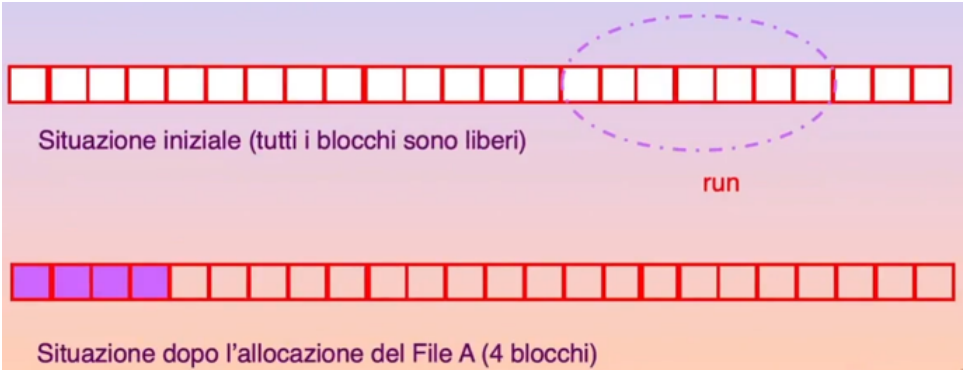
- La prima è terminare il programma utente con un messaggio di errore. L'utente dovrà quindi allocare più spazio ed eseguire di nuovo il programma
- La seconda è trovare il buco più grande, copiare il contenuto del file nel nuovo spazio e rilasciare lo spazio precedente (porta a overhead)

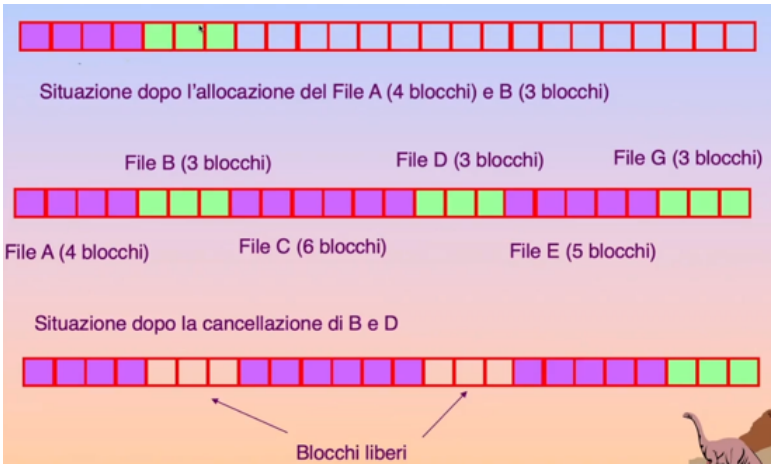
Invece, file che crescono nel tempo potrebbero soffrire di una grande frammentazione interna

Per evitare questo, alcuni S.O. fanno uso di uno schema di allocazione contigua modificato: si assegna una porzione, se questa non risulta sufficiente se ne assegna un'altra chiamata **estensione**
Quindi, si salva la locazione dei blocchi di file come una locazione e un numero di blocchi, insieme con l'indirizzo del primo blocco della prossima estensione

La frammentazione interna potrebbe ancora essere un problema se si assegnano estensioni troppo grandi, e potrebbero insorgere ulteriori problemi di frammentazione esterna in caso di assegnamento e rilascio di estensioni di dimensione variabile

Esempi:

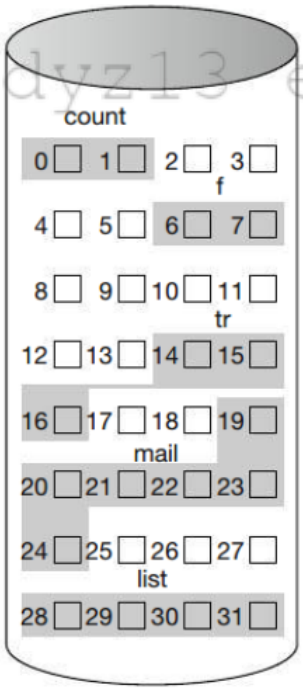




Fenomeno della **frammentazione interna:**

L'allocazione contigua viene spesso utilizzata nei file system dei CD-ROM e dei DVD

Allocazione contigua dello spazio dei dischi:



directory

file	blocco iniziale	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocazione concatenata

L'allocazione concatenata risolve i problemi dell'allocazione contigua:

- Ogni file è costituito da una lista concatenata di blocchi del disco i quali possono essere sparsi in qualsiasi punto del disco stesso
- Ogni blocco contiene un puntatore al blocco successivo

- La directory contiene un puntatore al primo e all’ultimo blocco del file

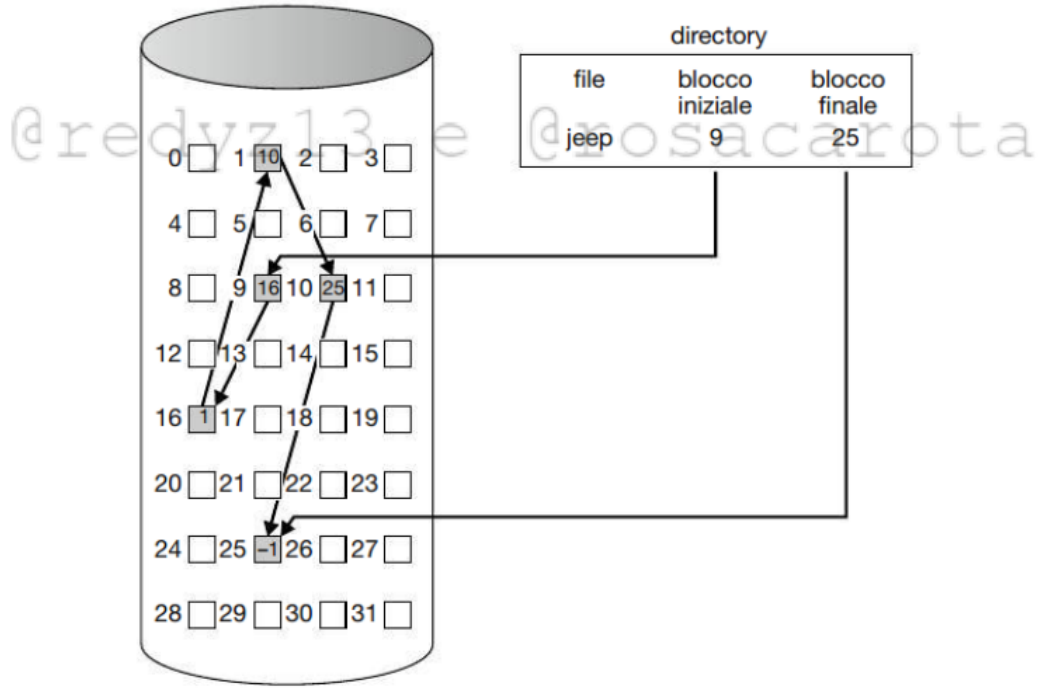
Per creare un nuovo file, si crea un nuovo elemento nella directory.
Ogni elemento della directory ha un puntatore al primo blocco del file.
Se il file è vuoto, questo puntatore è inizializzato a `null` se il file è vuoto e anche la dimensione si imposta a 0

Se si esegue un’operazione di scrittura su un file, si ricerca un blocco libero tramite il sistema di gestione dello spazio libero, si scrive nel blocco e si concatena il blocco alla fine del file

Per leggere, basta leggere seguendo i puntatori da un blocco all’altro

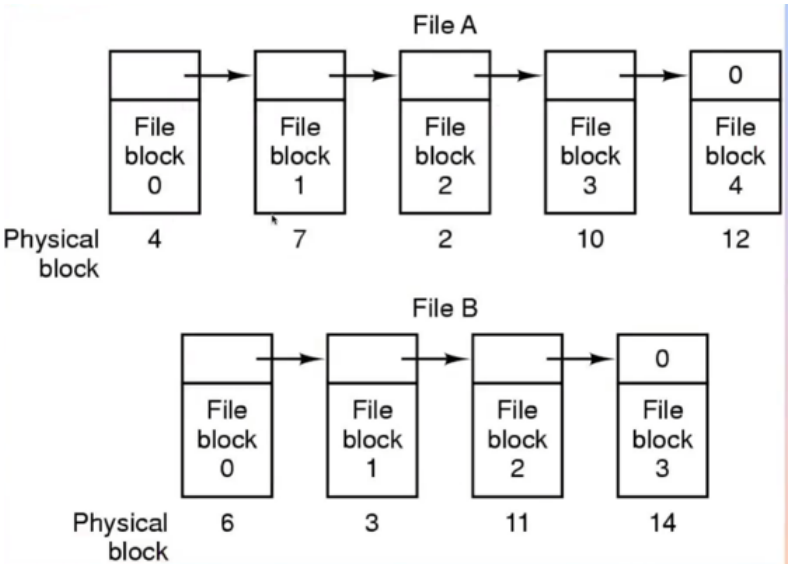
Con l’allocazione concatenata, si elimina il problema della frammentazione esterna

Allocazione concatenata dello spazio nei dischi:



File: lista concatenata di blocchi:

Memorizzazione come lista concatenata di blocchi



Problema principale: può essere usata in modo efficiente solo per i file ad accesso sequenziale

Per trovare l'i-esimo blocco di un file, si parte dall'inizio del file seguendo i puntatori fino all'i-esimo
Ogni accesso ad un puntatore implica una lettura del disco e quindi un probabile spostamento della testina. Quindi per il file a cui è stato assegnato uno spazi contiguo risulta inefficiente questa metodologi

Altro problema: spazio richiesto per i puntatori

La memorizzazione dei riferimenti riduce lo spazio disponibile per i dati: i puntatori al blocco successivo non sono disponibili all'utente

Quindi se ogni blocco è formato da 512 byte e un indirizzo del disco richiede 4 byte, allora l'utente vede blocchi di 508 byte

Quindi con blocchi di 512 byte e riferimenti (puntatori) di 4 byte si ha uno spreco di spazio pari allo 0.78% del disco

La soluzione più comune a questo problema consiste nel raccogliere i blocchi in gruppi , detti **cluster**, e nell'allocare i cluster anziché i blocchi

Ad esempio, il file system può definire un cluster di 4 blocchi e operare nel disco soltanto per unità di cluster, permettendo così di occupare meno spazio su disco

Facendo così si ha/hanno:

- Meno riposizionamenti della testina (migliora il throughput)
- Una riduzione dello spazio utilizzato per i riferimenti
- Maggiore frammentazione interna :(. Se un cluster è parzialmente pieno si spreca più spazio di quanto se ne sprecherebbe con un solo blocco parzialmente pieno

Tabella di allocazione dei file

Altro problema dell'allocazione concatenata: affidabilità

Un puntatore potrebbe essere perso o si potrebbe danneggiare

Le liste di blocchi sono fragili:

- la perdita di un solo riferimento può render inaccessibile grandi quantità di dati

Una soluzione parziale a questo problema consiste nell'utilizzare liste doppiamente concatenate oppure nel memorizzare il nome del file e il relativo numero di blocco in ogni blocco

- Questi schemi richiedono però un maggiore overhead

Una variante del metodo di assegnazione concatenata consiste nell'utilizzo di un indice detto **tabella di allocazione dei file (file allocation table - FAT)** per ogni partizione

La FAT:

- Ha un elemento per ogni blocco del disco ed è indicizzata dal numero di blocchi
- Viene utilizzata essenzialmente come una lista concatenata
- Per contenerla si riserva una sezione del disco all'inizio di ciascun volume e per utilizzarla la si porta in memoria centrale

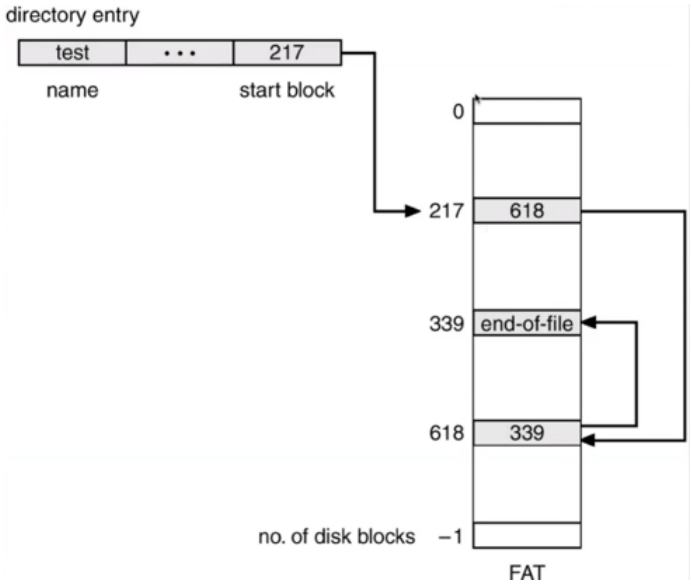
L'elemento di directory contiene il numero di blocco del primo blocco del file
L'elemento della tabella indicizzato da quel numero di blocco contiene a sua volta il numero di blocco del blocco successivo del file

Questa catena continua fino all'ultimo blocco, che ha come elemento della tabella un carattere speciale di fine file

I blocchi inutilizzati sono indicati da un valore 0 nella tabella

L'allocazione di un nuovo blocco consiste nel localizzare il primo elemento della tabella con valore 0, sostituire il valore di fine del file precedente con l'indirizzo del nuovo blocco e sostituire l'elemento trovato in precedenza con il valore di fine del file

L'utilizzo della FAT aumenta lo spostamento della testina: deve spostarsi all'inizio del volume per leggere la FAT e trovare la locazione del blocco; diminuisce però il tempo per l'accesso diretto



Allocazione indicizzata

Raggruppa tutti i puntatori in una sola locazione: il **blocco indice**

Si trova quindi, l'elenco dei blocchi che compongono un file

Ogni file ha il proprio blocco indice: è un array di indirizzi di blocchi del disco

La directory contiene l'indirizzo del blocco indice

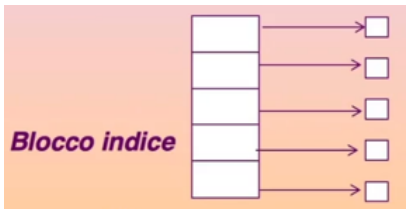
L'i—esimo blocco viene letto utilizzando il puntatore alla i-esima posizione

Per accedere ad un file, si carica in memoria il suo blocco indice e si utilizzano i puntatori in esso contenuto
Usando questa struttura l'intero blocco diventa disponibile per contenere dati

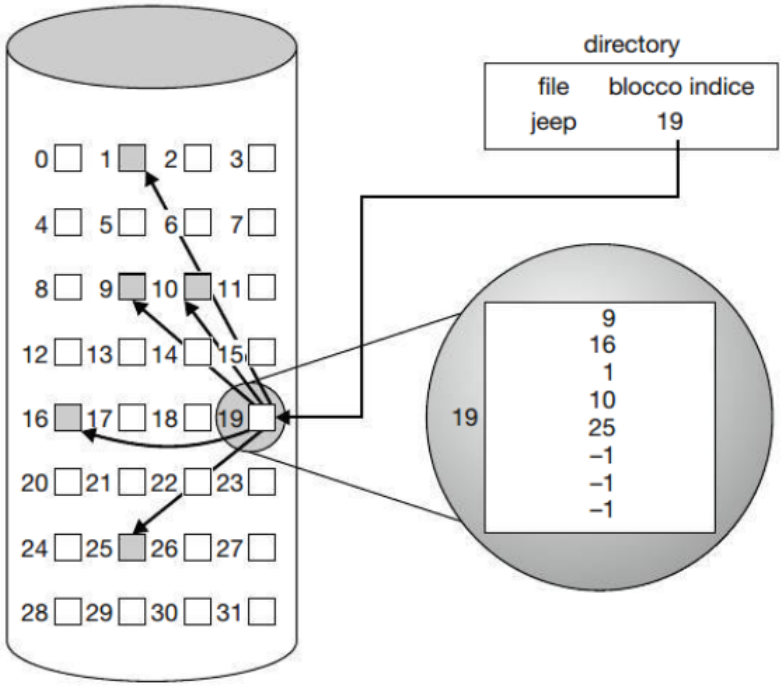
quando viene creato un file, i puntatori del blocco indice puntano tutti a null. Quando si va a scrive l'i-esimo blocco, il gestore dei blocchi liberi fronisce un blocco; a questo punto l'indirizzo del nuovo blocco è inserito nell'i-esimo elemento del blocco indice

L'accesso casuale è molto più semplice, in quanto la tabella è contenuta interamente in memoria.

E' sufficiente, come nei casi precedenti, gestire un solo intero (l'indice del primo blocco) per ogni elemento della directory per essere in grado di individuare tutti i blocchi, qualsiasi sia la grandezza del file



Allocazione indicizzata dello spazio dei dischi:



Vantaggi:

- Risolve il problema della frammentazione esterna
- Permette una gestione efficiente dell'accesso diretto
- Il blocco indice deve essere caricato in memoria solo quando il file è aperto

Svantaggi:

- La dimensione del blocco indice determina l'ampiezza massima del file
- Utilizzare blocchi indici troppo grandi comporta un notevole spreco di spazio

Ogni file deve avere un blocco indice, quindi è necessario cercare di mantenere le dimensioni dei blocchi indice più piccole possibile

Naturalmente se il blocco è troppo piccolo non può contenere un numero di puntatori sufficiente un file di grandi dimensione

Dimensione del blocco indice

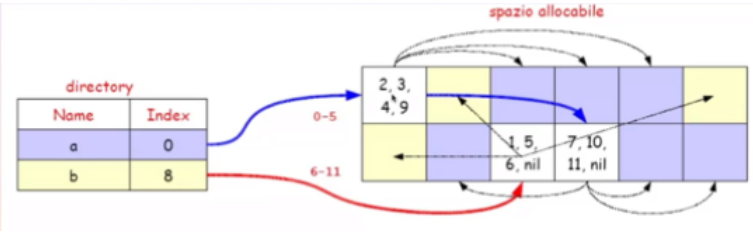
Possibili meccanismi per memorizzare il blocco indice:

- **Schema concatenato:** il blocco indice è formato da un singolo blocco di disco; per permettere la presenza di file lunghi è possibile collegare tra loro più blocchi indice. L'ultime parola del blocco indice sarà **null** oppure un puntatore ad un altro blocco indice
- **Indice a più livelli:** un blocco indice di primo livello punta ad un insieme di blocchi indice di secondo livello che, a loro volta, puntano ai blocchi dei file
- **Schema combinato:** (in UNIX) si tengono i primi 15 puntatori del blocco indice nell'**inode** associato al file
  - I primi 12 vengono usati come puntatori a **blocchi diretti**, ciiè contengono direttamente gli indirizzi di blocchi contenenti dati del file
  - Gli altri 3 puntano a **blocchi indiretti**:
    - Il primo ad un **blocco indiretto singolo**, cioè ad un blocco indice che non contiene dati, ma indirizzi di blocchi che contengono dati
    - Il secondo ad un **blocco indiretto doppio**: contiene l'indirizzo di un blocco che a sua volta contiene gli indirizzi di blocchi contenenti puntatori agli effettivi blocchi di dati
    - Il terzo ad un **blocco indiretto triplo**

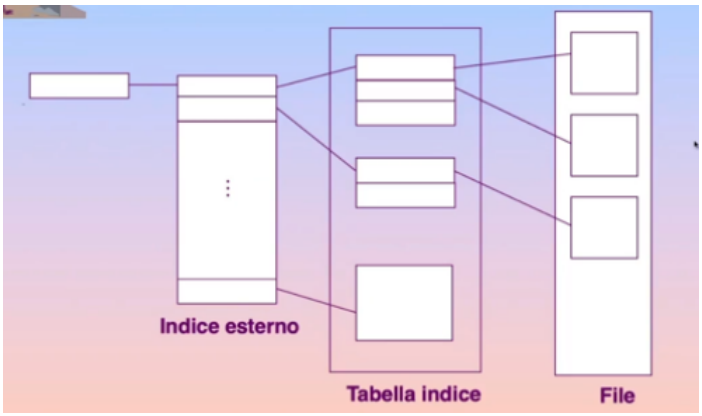
Schema concatenato

L'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo

Si ripropone il problema dell'accesso diretto a file di grandi dimensioni



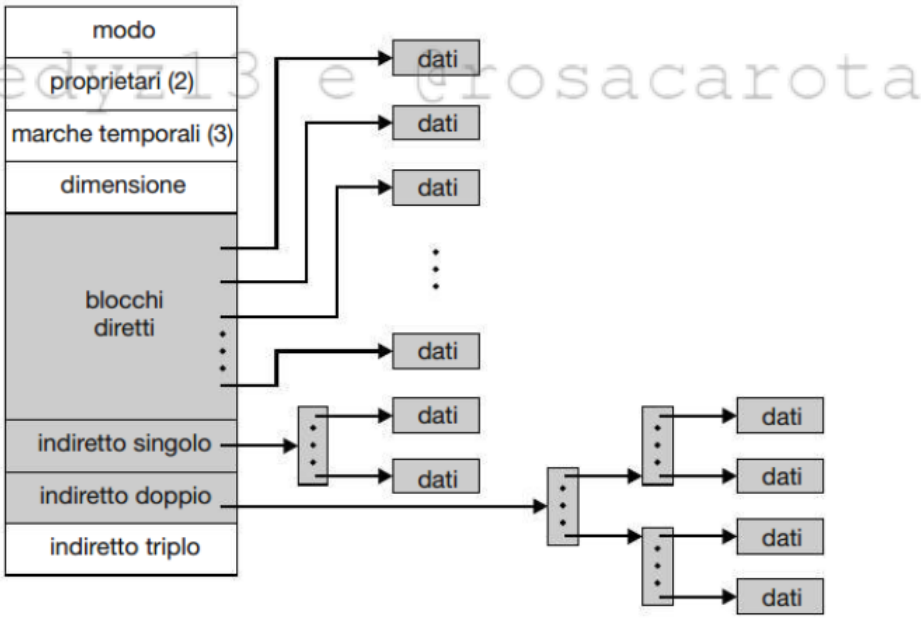
Indice a più livelli



Si utilizza un blocco indice dei blocchi indice

Degradano le prestazioni, in quanto è richiesto un maggior numero di accessi al disco

Schema combinato: Inode di UNIX (4K byte per bocco, 4 byte per puntatore: 4GB indirizzabili)



Prestazioni

I metodi presentati hanno:

- Diversi livelli di efficienza di memorizzazione
- Diversi tempi d’accesso ai blocchi di dati

La scelta di quale tipo di allocazione effettuare dipende dal sistema: se il sistema ha prevalenza di accesso sequenziale, si sceglierà un tipo di allocazione rispetto ad

un’altra

Allocazione contigua:

- Richiede un solo accesso per qualsiasi tipo di accesso (sia sequenziale che diretto)

Allocazione concatenata:

- Accesso sequenziale: Valido perché si tiene in memoria anche l’indirizzo al blocco successivo
- Accesso diretto: un accesso all’i-esimo blocco può richiedere i letture del disco

Alcuni sistemi usano entrambi i tipi di allocazione nel caso di accesso sequenziale e diretto

Allocazione indicizzata:

- Si può avere accesso diretto solo per i blocchi che si trovano già in memoria, altrimenti si è costretti ad effettuare un accesso sequenziale
  Le prestazioni dell’allocazione indicizzata dipendono dalla struttura dell’indice, dalla dimensione del file e dalla posizione del blocco desiderato
- Essa può essere combinata con l’allocazione contigua: contigua per file piccoli e indicizzata per file grandi

Gestione dello spazio libero

Lo spazio su disco è limitato ed è quindi necessario riutilizzare lo spazio lasciato dai file cancellati per scrivervi, se possibile, nuovi file

Per tenere traccia dello spazio libero, il sistema conserva un **elenco dei blocchi liberi** dove sono registrati tutti i blocchi non assegnati ad alcun file o directory

Per creare un file occorre cercare nell’elenco dei blocchi liberi la quantità di spazio necessaria e assegnarla al nuovo file, quindi rimuovere questo spazio dall’elenco dei blocchi liberi

Quando si cancella un i file si aggiungono all’elenco dei blocchi liberi i blocchi di disco ad esso assegnati

Vettore di bit

Spesso la lista dei blocchi liberi viene implementata come una **mappa, o vettore di bit**

Ogni blocco è rappresentato da un bit:

- Libero → bit 1
- Allocato → bit 0

Il vantaggio di questo metodo è la sua relativa semplicità ed efficienze nel trovare il primo blocco libero o n blocchi liberi consecutivi nel disco

Una tecnica per trovare il primo blocco libero è quello di controllare in modo sequenziale ogni parola nella mappa di bit per verificare che il valore non sia 0, poiché una parola con valore 0 ha tutti i bit a 0 e rappresenta un insieme di blocchi assegnati. La prima parola non 0, viene scandita per ricercare il primo bit 1, ciè la locazione del primo blocco libero

Il numero del blocco è dato da:

- $(numero\ di\ bit\ per\ parola) \times (numero\ di\ parole\ di\ valore\ 0) + offset\ del\ primo\ bit\ 1$

I vettori di bit sono efficienti solo se tutto il vettore è mantenuto nella memoria centrale, soluzione non applicabile ai dischi più grandi

Lista concatenata

I blocchi liberi vengono mantenuti in una lista concatenata

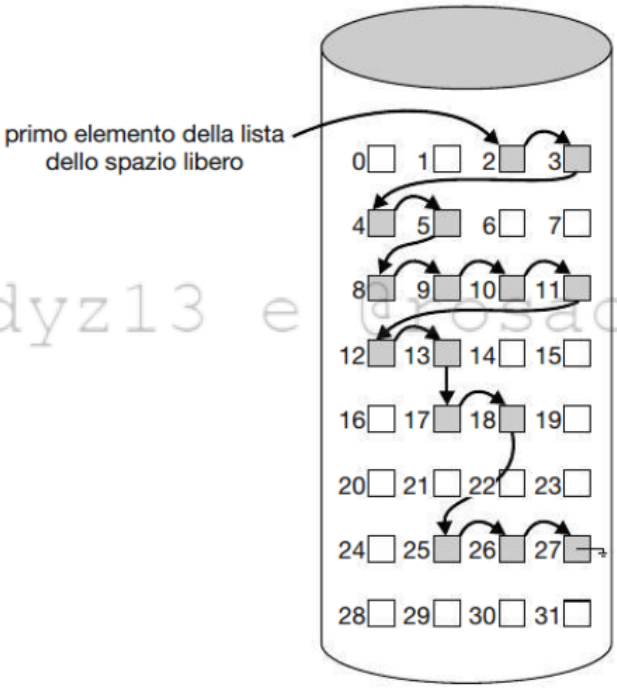
Un puntatore al primo blocco della lista (che a sua volta contiene un puntatore al secondo, e così via) viene mantenuto in una speciale locazione del disco e caricato in memoria

Vantaggi:

- Richiede poco spazio in memoria centrale

Svantaggi:

- L'allocazione di un'area di ampie dimensioni è costosa:
  - Per attraversare la lista occorre leggere ogni blocco e l'operazione richiede un notevole tempo di I/O (non è frequente)
- L'allocazione di aree libere contigue è molto difficoltosa



Raggruppamento e Conteggio

Raggruppamento: memorizzazione degli indirizzi di n blocchi liberi nel primo di questi.

I primi $n-1$ di questi blocchi sono effettivamente liberi; l'ultimo blocco contiene gli indirizzi di altri n blocchi liberi, e così via

L'importanza di questa implementazione, è data dalla possibilità di trovare rapidamente gli indirizzi di un gran numero di blocchi liberi

Conteggio: generalmente, più blocchi contigui possono essere allocati o liberati contemporaneamente

Quindi, anziché tenere 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 dei blocchi liberi è formato da un indirizzo del disco e un contatore

Anche se ogni elemento richiede più spazio di quanto ne richieda un semplice indirizzo del disco, se il contatore è generalmente maggiore di 1 la lista globale risulta più corta

Chiedere ad Anto Mappe di spazio (?)

Efficienza

I dischi tendono ad essere il principale collo di bottiglia per le prestazioni di un sistema essendo i più lenti tra i componenti di un calcolatore

L'uso efficiente di un disco dipende dalle tecniche di allocazione dello spazio su disco e dagli algoritmi di gestione delle directory

- Ad esempio, gli inode di UNIX sono preallocati in una partizione
- Anche un disco "vuoto" impiega una certa percentuale del suo spazio per gli inode
- D'altra parte, preallocando gli inode e distribuendoli lungo la partizione si migliorano le prestazioni del file system

Queste migliori prestazioni sono il risultato degli algoritmi di allocazione e di gestione dei blocchi liberi adottati da UNIX

Questi algoritmi mantengono i blocchi di dati di un file vicini al blocco che ne contiene l'inode allo scopo di ridurre il tempo di posizionamento

Anche il tipo di dati normalmente contenuti in un elemento di una directory (o di un inode) deve essere tenuto in considerazione

Solitamente viene memorizzata la **data di ultima scrittura** per fornire informazioni all'utente e per determinare se il file necessita o meno della creazione o aggiornamento di una copia di backup

Alcuni sistemi mantengono anche la **data di ultimo accesso** per consentire all'utente di risalire all'ultima volta che un file è stato letto

Il risultato del mantenere queste informazioni è che ogni volta che un file viene letto si dovrà aggiornare un campo della directory

- Questa modifica richiede la lettura in memoria del blocco, la modifica della sezione e la riscrittura del blocco su disco, poiché sui dischi è possibile operare solamente per blocchi (o gruppi di blocchi)

Quindi, ogni volta che un file viene aperto in lettura, anche l'elemento della directory a esso associato deve essere letto e scritto

Questo requisito può risultare inefficiente per file a cui si accede frequentemente, quindi, al momento della progettazione del file system, è necessario confrontare i benefici con i costi in termini di prestazioni

In generale, è necessario considerare l'influenza sull'efficienza e sulle prestazioni di ogni informazione che si vuole associare a un file