Fondamenti dei Sistemi Operativi

Il File System

Sommario

- Necessità di organizzazione dei dati: i file
- Concetto di file e contenuto
- Strutture dei file
- Attributi comuni dei file
- Tipi di file
- Operazioni sui file
- Tabella dei file aperti
- Metodi di accesso
- File directory, Tipi di directory
- Operazioni sulle directory
- Politiche di gestione
- Protezione dei file
- Allocazione fisica dei file
- Overhead di occupazione di memoria
- L'i-node in UNIX
- Shared file
- Utilizzo spazio fisico e capacità singolo blocco
- L’architettura di un File System

Il modello stratificato del nucleo del SO suggerisce di partire dalla parte alta del kernel, quella più direttamente a contatto con l'interfaccia utente e, quindi, più visibile da parte di un utilizzatore.

Necessità di organizzazione dei dati

Alcune necessità dei processi:

- **memorizzare e trattare grandi quantità di informazioni** (maggiori della quantità di memoria principale),
- **più processi** devono avere la possibilità di **accedere alle informazioni in modo concorrente e coerente**, nello spazio e nel tempo,
- si deve garantire **integrità, indipendenza, persistenza e protezione dei dati**.

I File

La soluzione sono i **file** (archivi):

file = insieme di informazioni correlate a cui è stato assegnato un nome.

- Un file è la più piccola porzione unitaria di memoria logica secondaria allocabile dall'utente o dai processi di sistema.
- La parte del S.O. che realizza questa astrazione, nascondendo i dettagli implementativi legati ai dispositivi sottostanti, è il **file system**.
- Esternamente, il file system è spesso l'aspetto più visibile di un S.O.: come si denominano e manipolano i file, come si accede ad essi, quali sono le loro strutture, i loro attributi, ecc.
- Internamente, il file system si appoggia alla gestione dell'I/O per implementare ulteriori funzionalità.

Il concetto di file

..... spazio d'indirizzi logici contigui

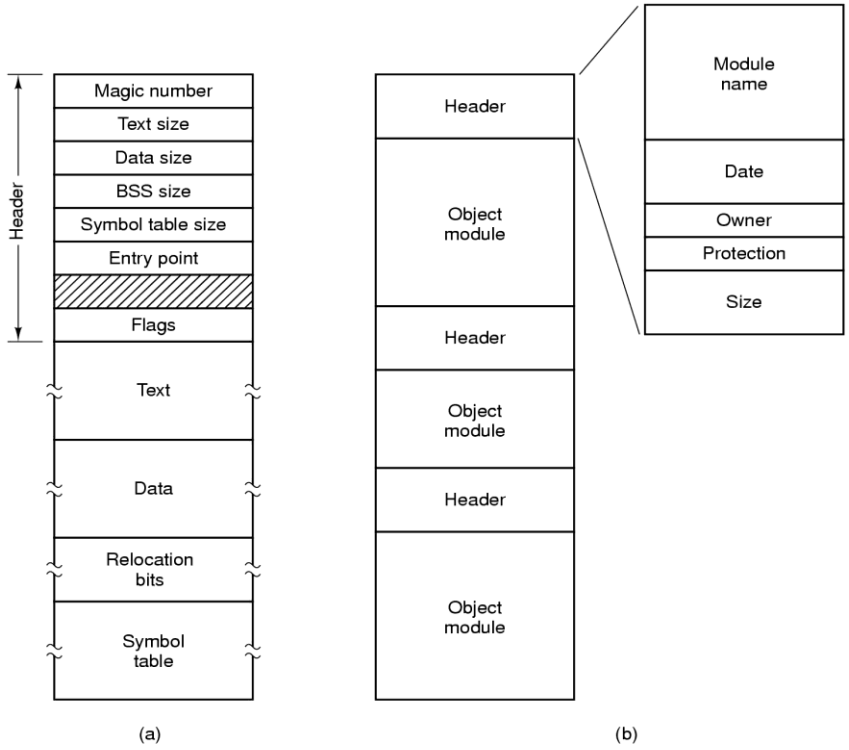
→ deve memorizzare grandi quantità di dati

→ l'informazione memorizzata deve sopravvivere al termine del processo che l'ha usata

→ più processi devono poter concorrentemente accedere all'informazione

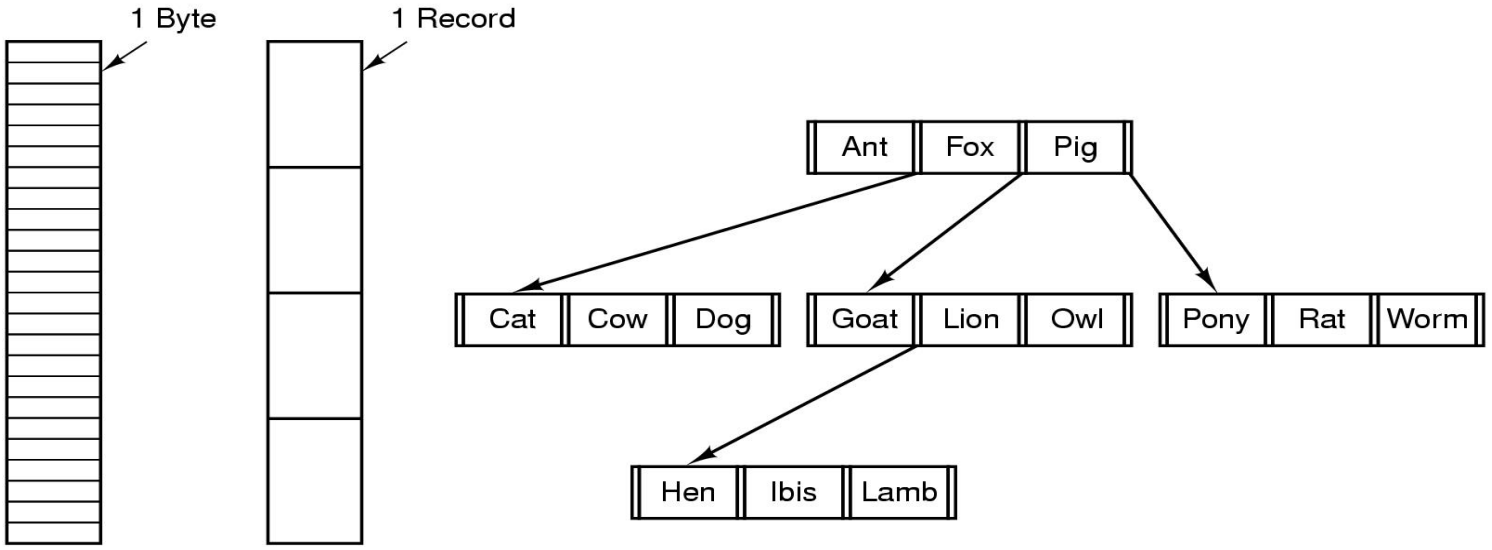
Il contenuto dei file

- Dati
  - numerici
  - caratterali
  - binari
- Programmi



Strutture dei file

- Nessuna struttura - sequenza di parole, byte
- Semplici strutture di record
  - Linee
  - di lunghezza fissa
  - di lunghezza variabile
- Strutture complesse
  - documenti formattati
  - load file rilocabili



Strutture dei file

Chi impone la struttura?

- Il sistema operativo: specificato il tipo, viene imposta la struttura e le modalità di accesso. Più astratto.
- L'utente: tipo e struttura sono delegati al programma, il sistema operativo implementa solo file non strutturati. Più flessibile.

Attributi comuni dei file

- ✓ **Nome** identificatore del file. L'unica informazione umanamente leggibile.
- ✓ **Tipo** nei sistemi che supportano più tipi di file. Può far parte del nome.
- ✓ **Locazione** puntatore alla posizione del file sui dispositivi di memorizzazione.
- ✓ **Dimensioni** attuale, ed eventualmente massima consentita.
- ✓ **Protezioni** controllano chi può leggere, modificare, creare, eseguire il file.
- ✓ **Identificatori dell'utente** che ha creato/possiede il file.
- ✓ **Varie date e timestamp** di creazione, modifica, aggiornamento info. . .

Queste informazioni (metadati: dati sui dati) sono solitamente mantenute in apposite strutture (directory) residenti in memoria secondaria.

Common file attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Tipi di file

FAT WINDOWS: name.extension

Tipo	Estensione	Funzione
Eseguibile	exe, com, bin o nessuno	programma pronto da eseguire, in liguaggio macchina
Oggetto	obj, o	compilato, in linguaggio macchina, non linkato
Codice sorgente	c, p, pas, f77, asm, java	codice sorgente in diversi linguaggi
Batch	bat, sh	script per l'interprete comandi
Testo	txt, doc	documenti, testo
Word processor	wp, tex, doc	svariati formati
Librerie	lib, a, so, dll	librerie di routine
Grafica	ps, dvi, gif	FILE ASCII o binari
Archivi	arc, zip, tar	file correlati, raggruppati in un file, a volte compressi

Tipi di file

UNIX-LINUX: nessuna assunzione

Unix non forza nessun tipo di file a livello di sistema operativo: non ci sono metadati che mantengono questa informazione.

Tipo e contenuto di un file slegati dal nome o dai permessi.

Sono le applicazioni a sapere cosa fare per ogni file.

Operazioni sui file
(FILE SYSTEM CALLS)

- **Create** — due passaggi: allocazione dello spazio sul dispositivo,e collegamento di tale spazio al file system.
- **Delete** — staccare il file dal file system e deallocare lo spazio assegnato al file.
- **Read** — dato un file e un puntatore di posizione nel file, i dati da leggere vengono trasferiti dal dispositivo in un buffer in memoria.
- **Write** — dato un file e un puntatore di posizione, i dati da scrivere vengono trasferiti sul dispositivo.
- **Get attributes** — leggere le informazioni come nome, timestamp, ecc.
- **Set attributes** — modificare informazioni come nome, timestamp, protezione, ecc.
- **Seek** — ricercare nel file tramite riposizionamento; non comporta operazioni di I/O.

Tabella dei file aperti

Queste operazioni richiedono la conoscenza dei metadati contenuti nelle directory.

Per evitare di accedere continuamente alle directory, si mantiene in memoria una tabella dei file aperti.

Sono quindi necessarie due nuove operazioni sui file:

Apertura: allocazione di una struttura in memoria (file descriptor o file control block FCB) contenente i metadati riguardanti un file.

↳ **`Open`** caricare alcuni metadati relativi al file aperto dal disco nella memoria principale, per velocizzare le chiamate seguenti.

Chiusura: quando il file viene chiuso da tutti i processi che vi accedevano, le informazioni del FCB vengono trasferite dalla memoria al dispositivo, e viene deallocato l'FCB.

↳ **`Close`** deallocare le strutture allocate nell'apertura.

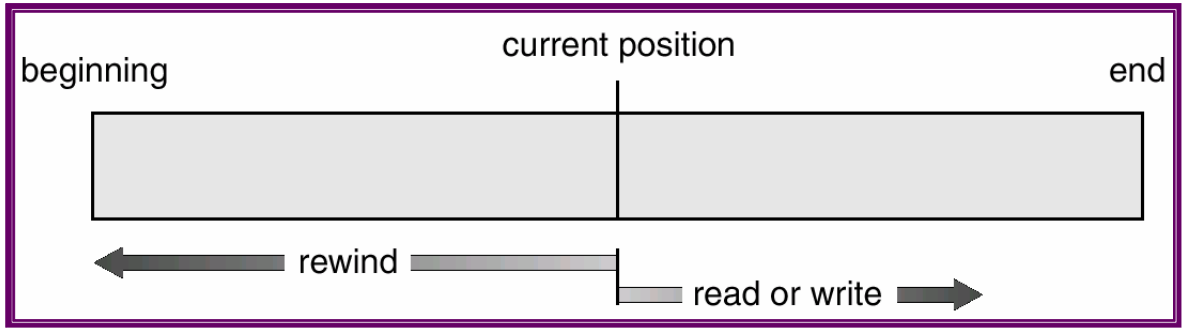
A ciascun file aperto si associa:

- Puntatore al file: posizione raggiunta durante la lettura/scrittura → record corrente.
- Contatore dei file aperti: quanti processi stanno utilizzando il file.
- Posizione sul disco.
- Diritti di accesso.

Metodi di accesso

→ Sequential Access

- Un puntatore mantiene la posizione corrente di lettura/scrittura → **record corrente**.
- Si può accedere solo progressivamente, o riportare il puntatore all'inizio del file.
- Adatto a dispositivi intrinsecamente sequenziali (p.e., nastri magnetici)



Operazioni *read next*
write next
reset
no *read after last write*
solo in alcuni file system *(rewrite)*

Metodi di accesso

→ Direct Access

- Il puntatore può essere spostato in qualunque punto del file.
- L'accesso sequenziale viene simulato con l'accesso diretto.
- Usuale per i file residenti su device a blocchi (p.e., dischi).

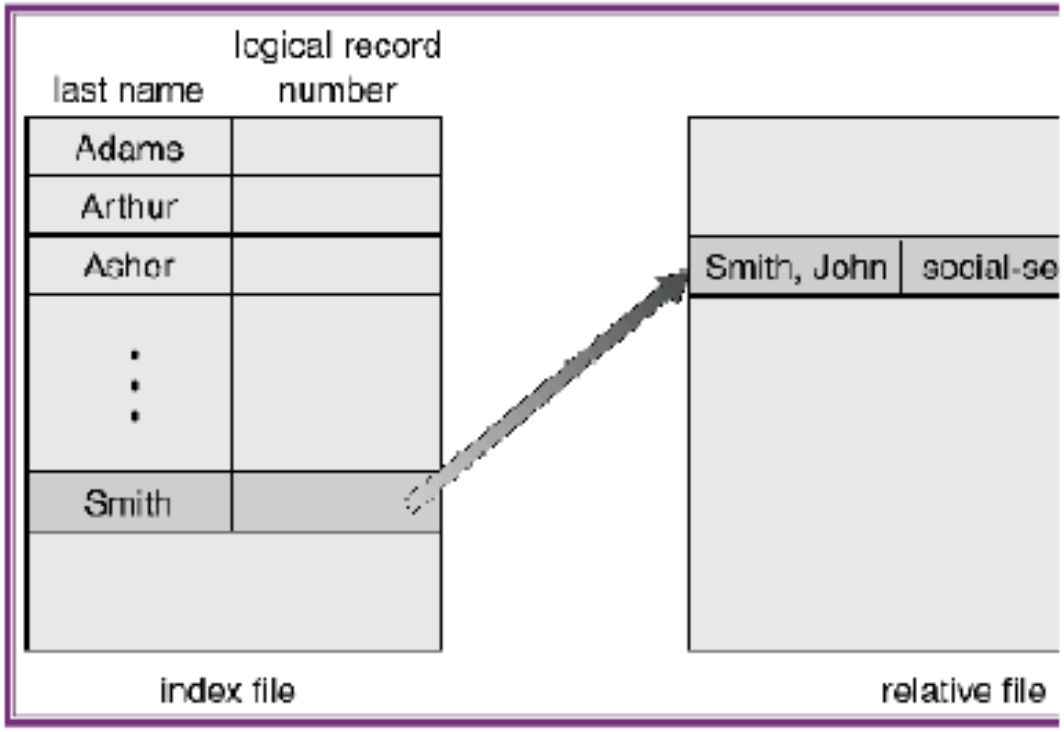
Operazioni

read n
write n
seek to n
read next
write next
rewrite n
n = relative block/record number

Metodi di accesso

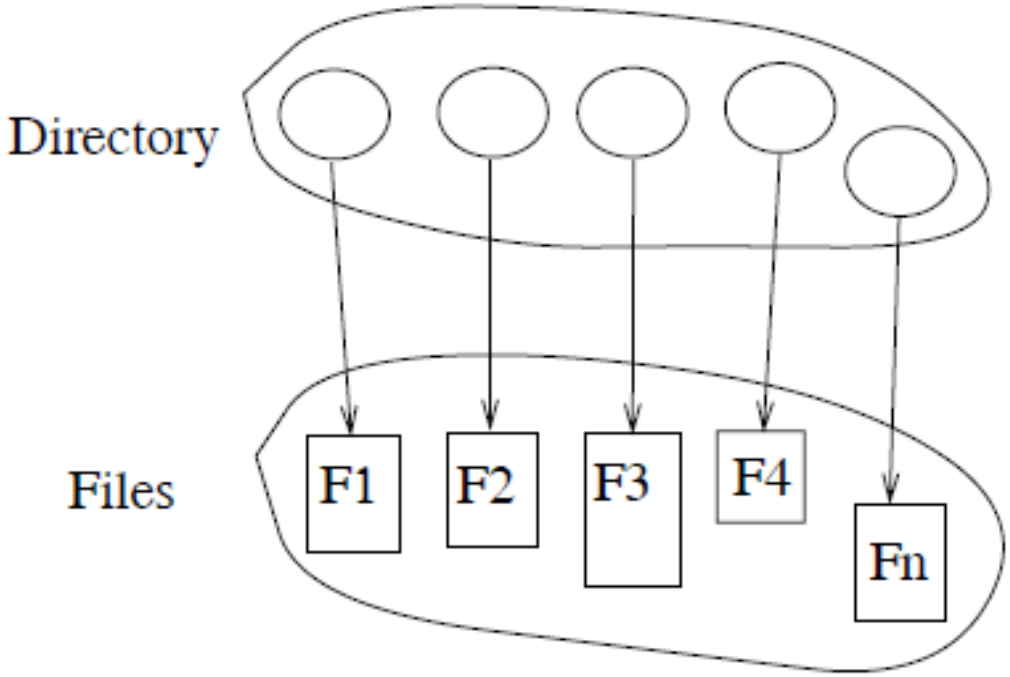
→ Indexed Access

- Un secondo file (indice) contiene solo parte dei dati, e puntatori ai blocchi (record) del vero file
- La ricerca avviene prima sull'indice, e da qui si risale al blocco
- Implementabile a livello applicazione in termini di file ad accesso diretto
- Usuale su mainframe (IBM, VMS), database. . .



File directory

- Una directory è una collezione di nodi contenente informazioni sui file (metadati).
- Sia la directory che i file risiedono su disco.
- Operazioni su una directory:
  - ricerca di un file,
  - creazione di un file,
  - cancellazione di un file,
  - lista dei file riportati nella directory,
  - rinomina di un file,
  - navigazione nel file system.



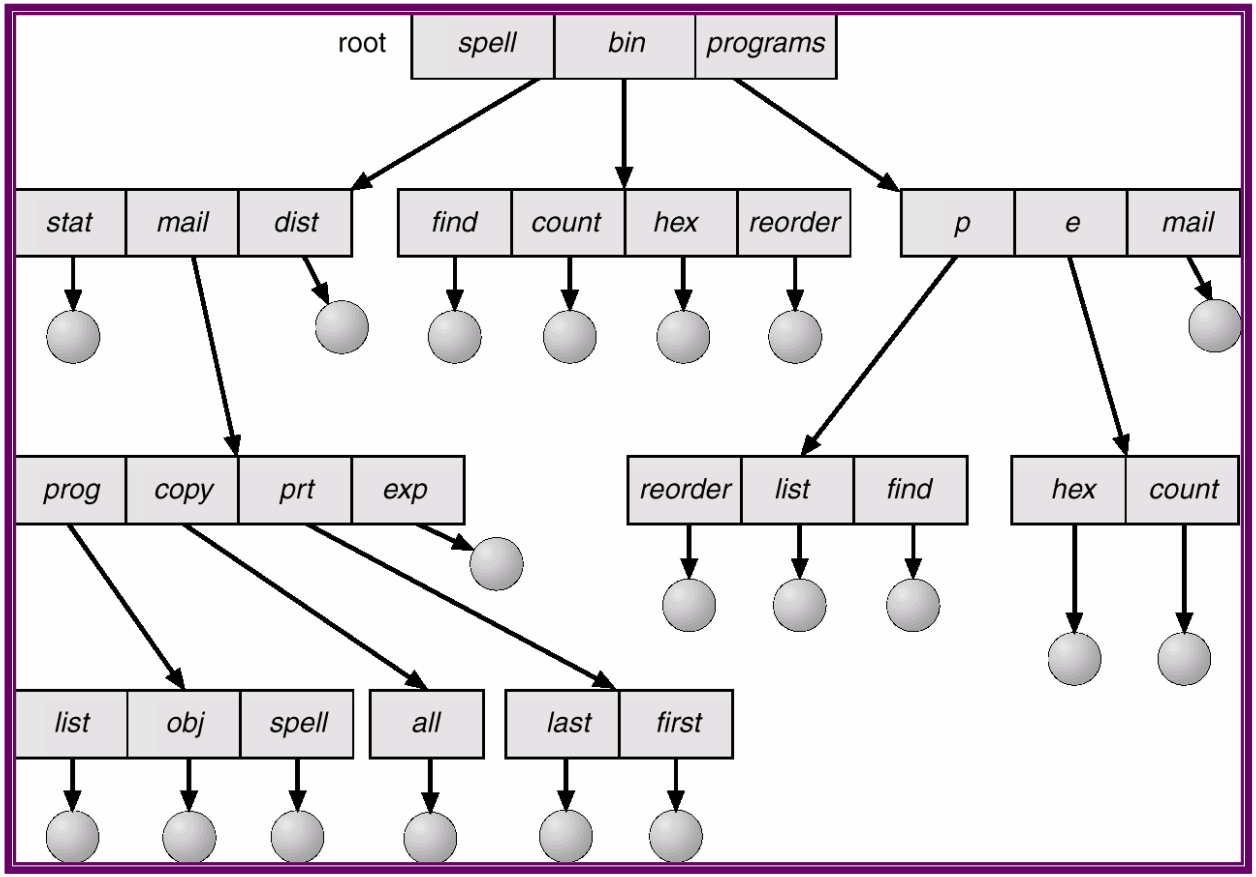
Tipi di directory

A singolo livello …… notevoli limiti

A due livelli …… una per ciascun utente

Gerarchica a più livelli (albero)

- ricerca efficiente
- capacità di raggruppamento
- directory corrente (*working directory*)
- *path name* assoluto o relativo
- creazione di un nuovo file fatta nella directory corrente
- cancellazione di un file
  - ↳ **`rm <file-name>`**
- creazione di una subdirectory fatta nella directory corrente.
  - ↳ **`mkdir <dir-name>`**

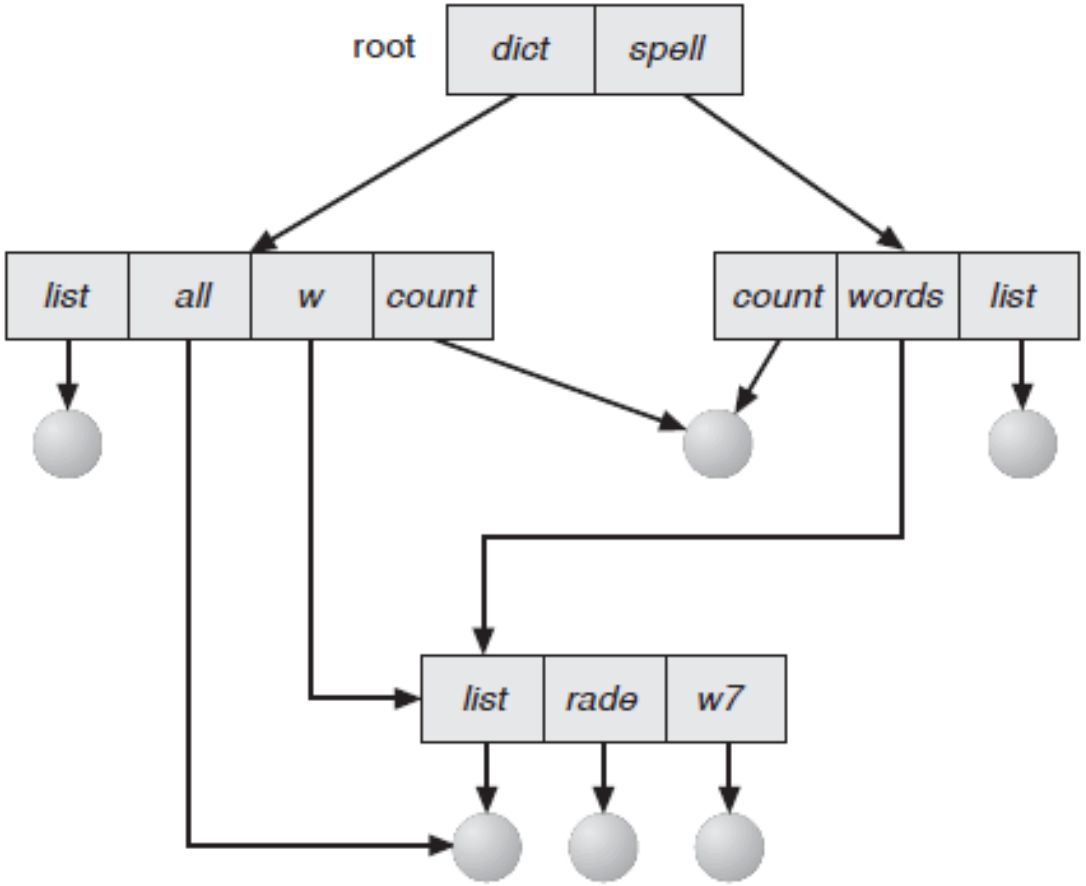


Directory a grafo aciclico (DAG)

File e sottodirectory possono essere condivise da più directory.

Due nomi differenti per lo stesso file (aliasing).

Possibilità di puntatori "dangling".



Operazioni sulle directory

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Unlink

Politiche di gestione

Le possibili politiche di gestione delle memorie di massa ad accesso diretto sono:

- preallocazione;
- allocazione dinamica.

Nel primo caso, il SO prealloca lo spazio occupato da un file.

Nel secondo caso, lo spazio viene allocato un blocco alla volta quando viene richiesta una scrittura nel file.

I sistemi UNIX-like usano una politica intermedia, preallocando dinamicamente un certo numero di blocchi.

Politiche di gestione

Preallocazione vs Allocazione dinamica

La politica di **preallocazione** impone che il programma, all'inizio della sua esecuzione, specifichi di quanto spazio di memoria di massa ha bisogno. Questo è un modo d'operare scomodo, poiché è frequente il caso in cui non si abbia idea di quanto spazio realmente serva. Esso però ha il vantaggio di una preliminare verifica della disponibilità dello spazio richiesto.

Invece, nella politica d'**allocazione dinamica**, all'utente non sono richieste informazioni sullo spazio da allocare, poiché questo viene allocato solo quando serve. Però, anche questo modo di procedere presenta degli inconvenienti, poiché si può incorrere in una imprevista indisponibilità di spazio durante l'esecuzione.

Nei sistemi UNIX si alloca un certo numero di blocchi (16). Questi vengono scritti in sequenza fino al penultimo. Nel caso in cui siano necessari altri record, il SO ne alloca altri 16, e così via. È un metodo più veloce rispetto a quello dell'allocazione dinamica, poiché la ricerca dei record liberi avviene una volta ogni 16 record allocati mentre nell'allocazione dinamica avviene ad ogni operazione di scrittura.

Protezione dei file

Importante in ambienti multiuser dove si vuole ***condividere i file.***

Il creatore/possessore (non sempre coincidono) deve essere in grado di controllare ***cosa può essere fatto e da chi*** (in un sistema multiutente).

Modi di accesso e gruppi in UNIX

Tre modi di accesso: read, write, execute

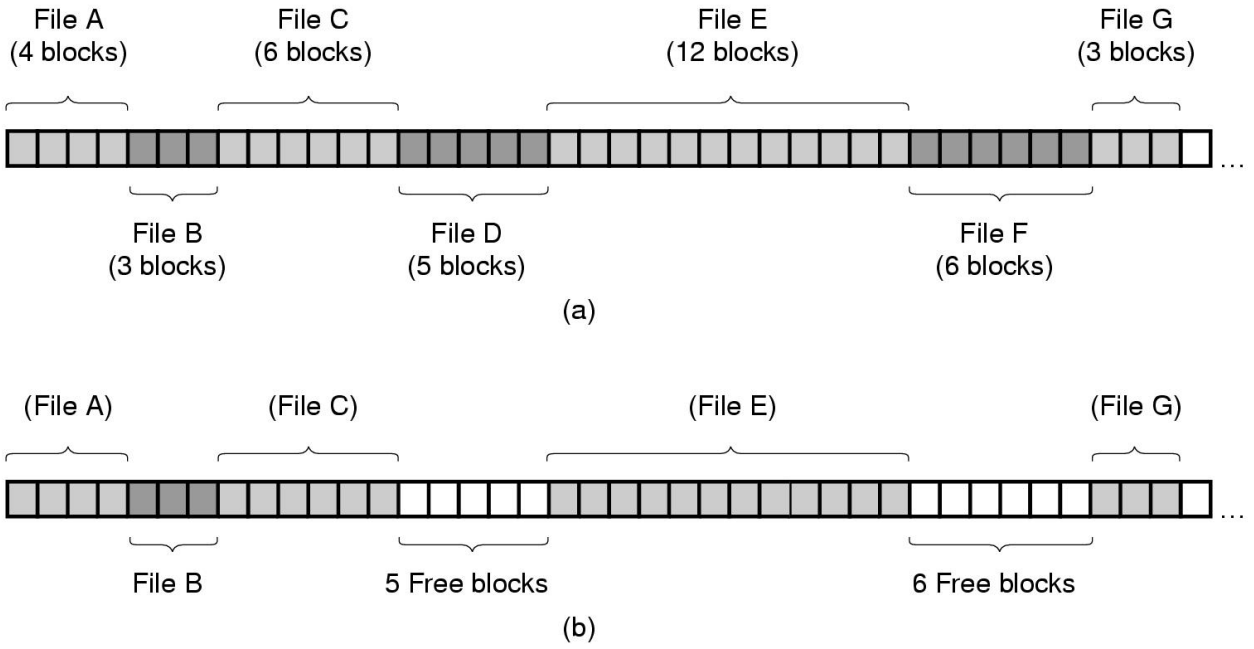
Tre classi di utenti, per ogni file

			R	W	X
a) owner access	7	→	1	1	1
b) groups access	6	→	1	1	0
c) public access	1	→	0	0	1

Ogni processo possiede User IDentification (UID) e Group IDentification (GID), con i quali si verifica l'accesso.

Allocazione fisica dei file

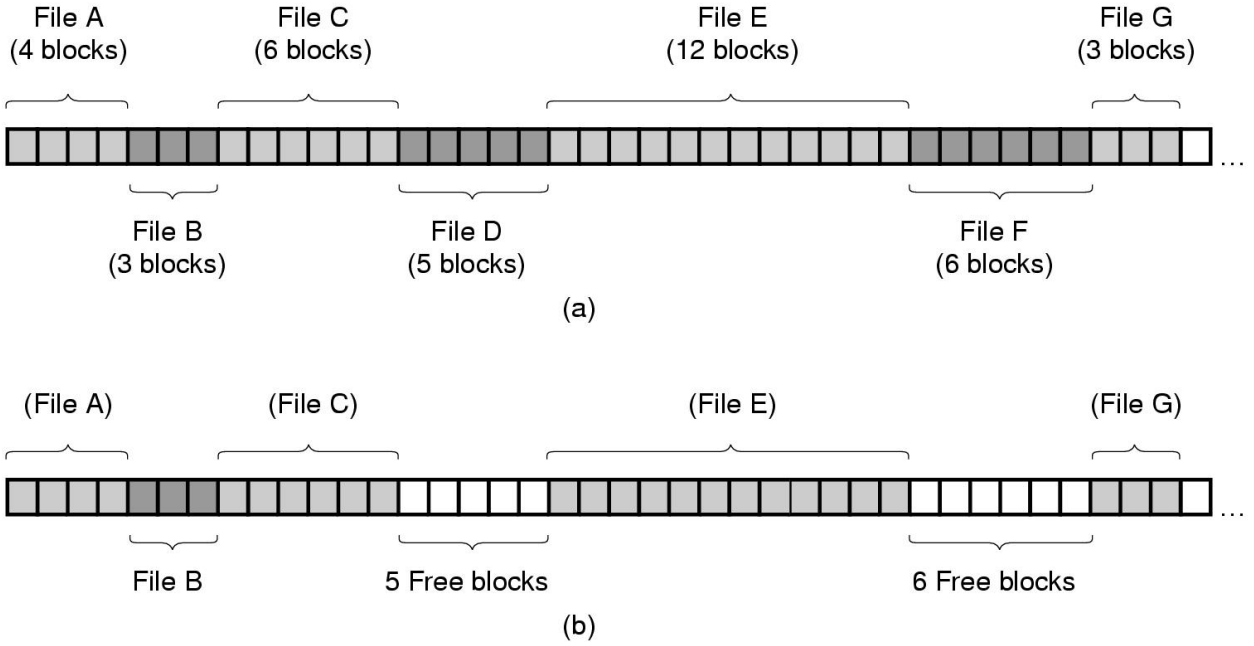
Contiguous Allocation 1/2



Contiguous allocation of disk space for 7 files
State of the disk after files D and E have been removed

Allocazione fisica dei file

Contiguous Allocation 2/2



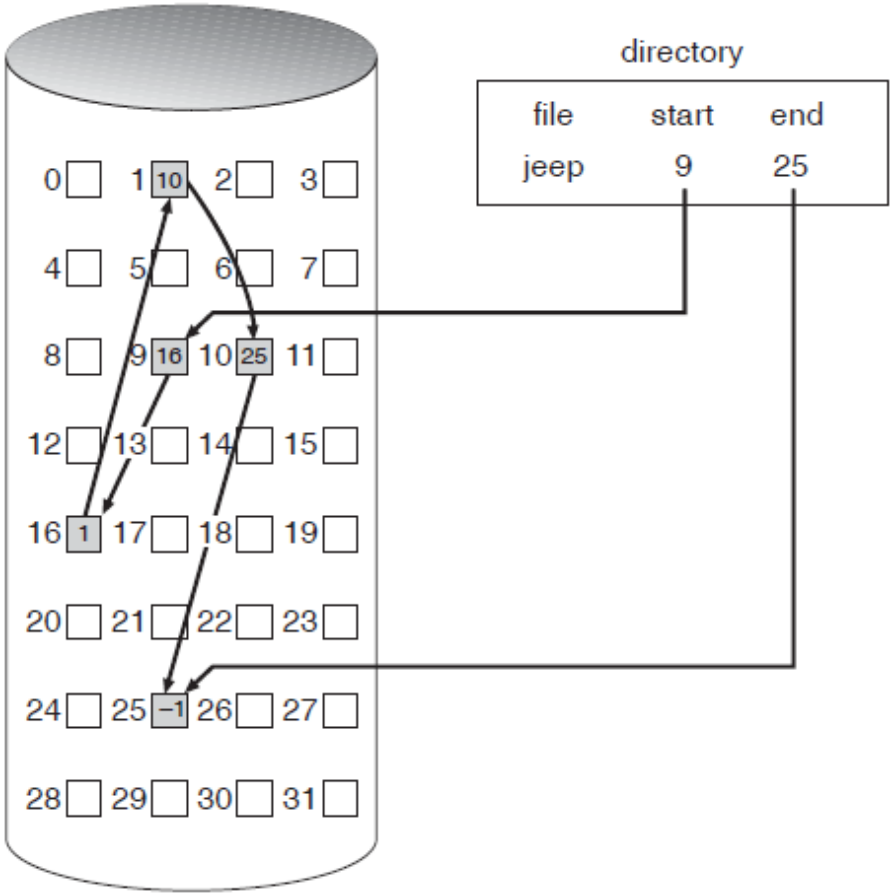
- Ogni file occupa un insieme di blocchi contigui sul disco
- Semplice: basta conoscere il blocco iniziale e la lunghezza
- L'accesso random è facile da implementare
- Frammentazione esterna. Problema di allocazione dinamica.
- I file non possono crescere (a meno di deframmentazione)
- Frammentazione interna se i file devono allocare a priori tutto lo spazio che può loro servire.

Allocazione fisica dei file

Disk blocks concatenation

Physical concatenation (pointers) 1/2

Ogni file è una linked list di blocchi, che possono essere sparpagliati ovunque sul disco

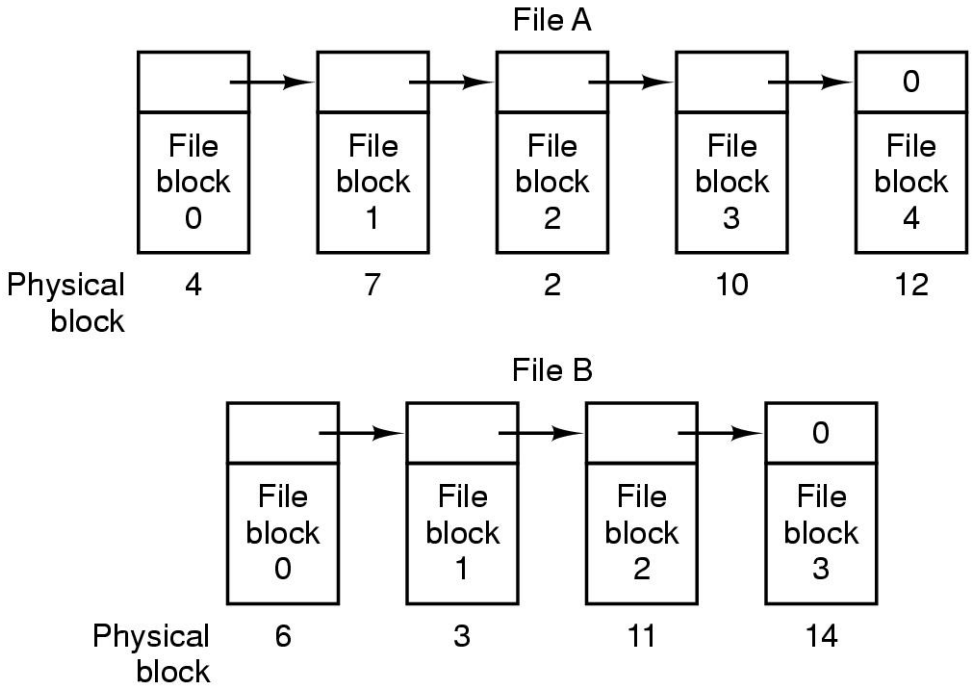


Allocazione fisica dei file

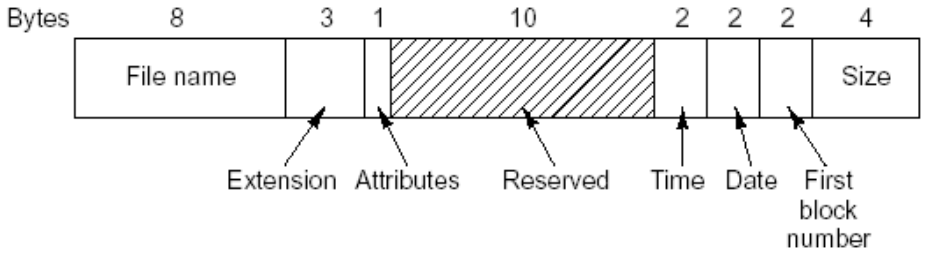
Disk blocks concatenation

Physical concatenation (pointers) 2/2

- Allocazione su richiesta; i blocchi vengono semplicemente collegati alla fine del file
- Semplice: basta conoscere l'indirizzo del primo blocco
- Non c'è frammentazione esterna
- Bisogna gestire i blocchi liberi
- Non supporta l'accesso diretto
- Spreca spazio (per i pointers) e tempo (per accedere a un singolo blocco)



File-allocation table (**FAT**) – disk-space allocation used by MS-DOS – must contain physical coordinates of the first block.



Allocazione fisica dei file

Disk blocks concatenation

Logical concatenation (linked list) 1/3

- Mantiene la linked list in una struttura dedicata, all'inizio di ogni partizione.
- I dischi sono indirizzati come dei grandi array monodimensionali di blocchi logici, dove il blocco logico è la più piccola unità di trasferimento con il controller.
- L'array monodimensionale è mappato sui settori del disco in modo sequenziale:
  - settore 0 = primo settore della prima traccia del cilindro più esterno;
  - la mappatura procede in ordine sulla traccia, poi sulle rimanenti tracce dello stesso cilindro, poi attraverso i rimanenti cilindri dal più esterno verso il più interno.

Corrispondenza bigettiva tra blocco logico nell'array e blocco fisico CTS

Si assuma che il disco abbia N_C = 200, N_T = 40, N_S = 50

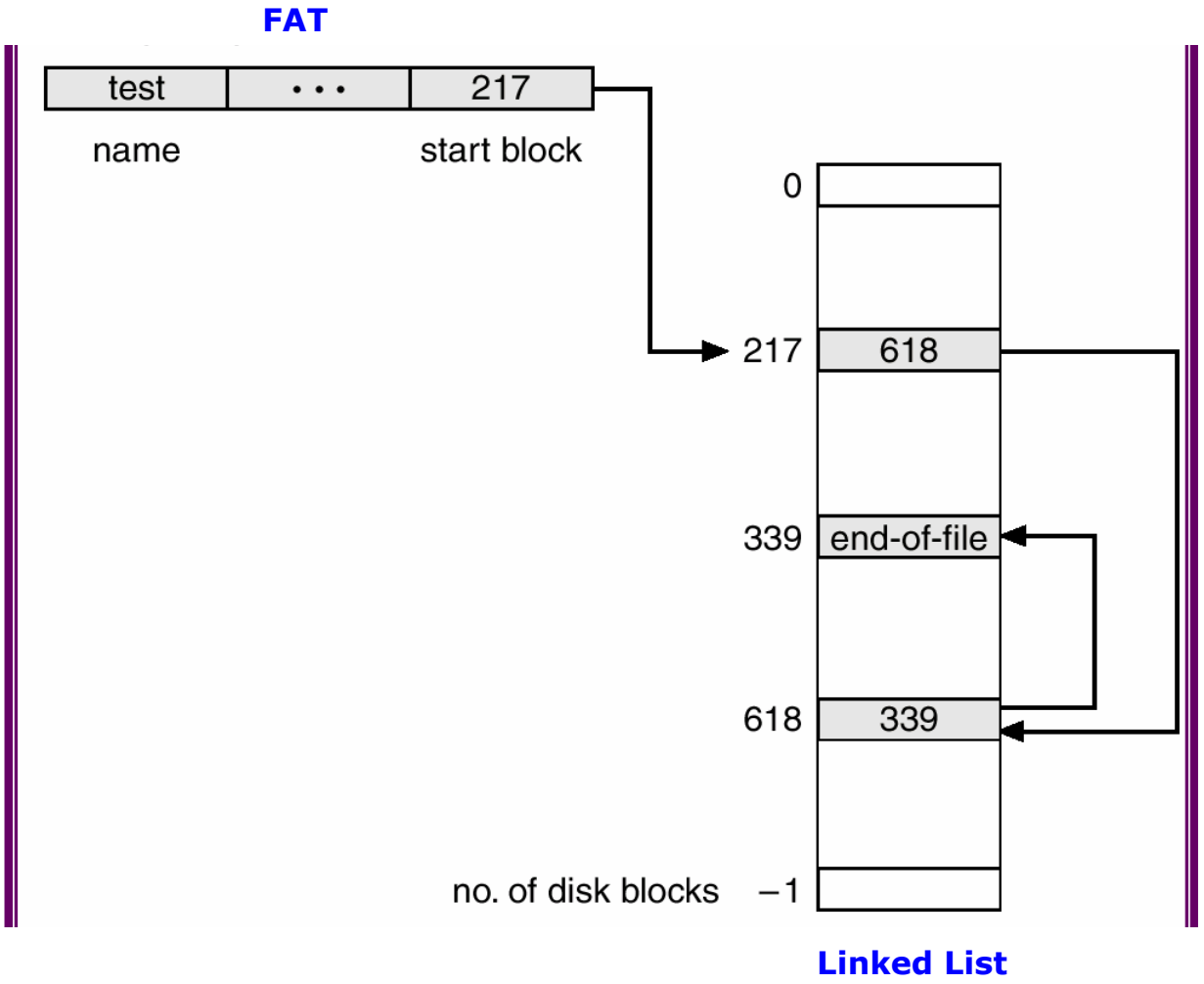
Un blocco fisico B, di coordinate C = 48, T = 27, S = 30 corrisponderà all'elemento dell'array: 48 * 40 * 50 + 27 *50 + 30 --> **97380**

Viceversa, quali saranno le coordinate fisiche dell'elemento 21350 dell'array?

Allocazione fisica dei file

Disk blocks concatenation

Logical concatenation (linked list) *2/3*

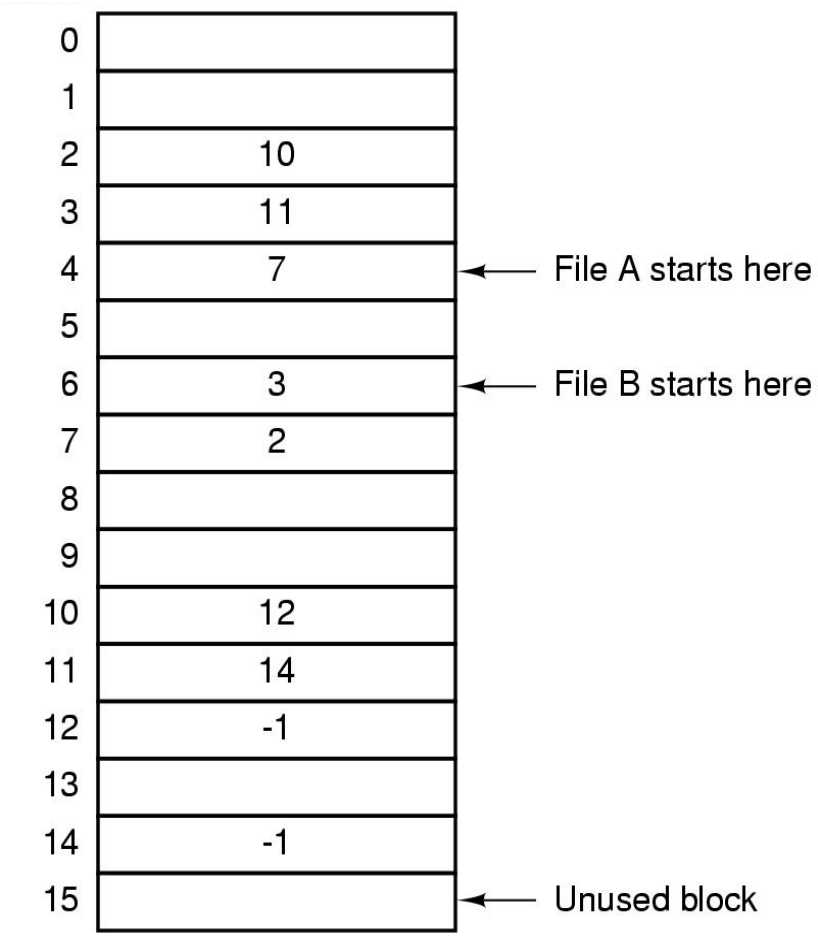


Allocazione fisica dei file

Disk blocks concatenation

Logical concatenation (linked list) 3/3

- Linked List è una tabella del contenuto di una memoria secondaria
- Ad ogni file corrisponde, nella linked list, una struttura concatenata di blocchi, che possono essere sparpagliati ovunque sul disco.
- Il numero di blocco logico iniziale di ciascun file è contenuto nella FAT.
- Si considerino gli stessi file A e B della *physical concatenation* tramite pointers.



Limiti della concatenazione logica

- ✓ Overhead di occupazione di memoria

Supponiamo di avere un disco fisso da 20 GB con 1Kb per blocco

M_{tot} = 20 Gb = 21.474.836.480 byte

M_{block} = 1 kb = 1.024 byte

Quanto spazio occupa la Linked List?

Numero di blocchi necessari = M_{tot}/M_{block} = 20.971.520

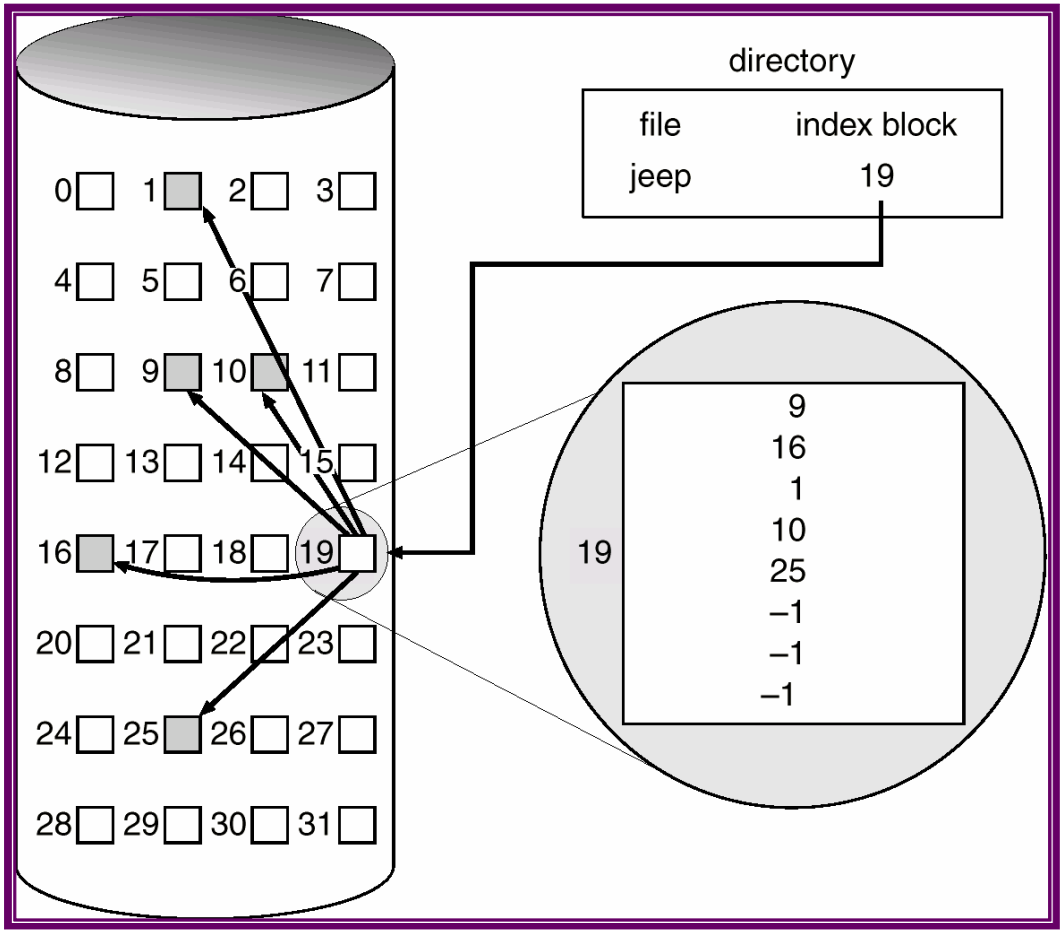
Ogni blocco è indicizzato con un intero a 32 bit = 4 byte

Spazio per la Linked List (byte) ---> 20.971.520 x 4 = 81920 Kb = 80 Mb (~ 0.4%)

- ✓ i dischi possono avere settori difettosi che vengono nascosti dal meccanismo di mappatura tra il numero dei blocchi logici e i settori fisici sul disco

Allocazione fisica dei file

Indexed allocation 1/2



Allocazione fisica dei file

Indexed allocation 2/2

Si mantengono tutti i puntatori ai blocchi di un file in una tabella indice (***index block***).

- **Supporta accesso random**
- **Consente l'allocazione dinamica senza frammentazione esterna, ma si ha l'overhead dell'index block**

Vantaggi: L'accesso diretto e l'accesso sequenziale sono efficienti
Si usa almeno un blocco indice, che è sprecato se il file è piccolo

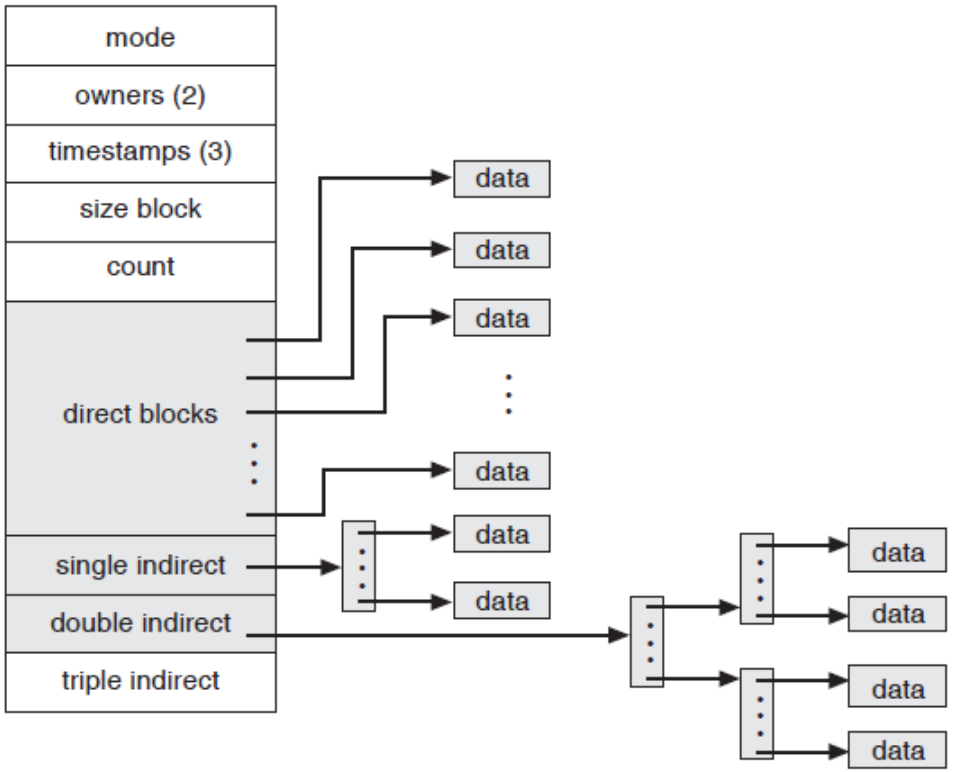
Svantaggi: Ogni blocco indice può gestire file di una dimensione fissata.
Che succede se il file eccede questa dimensione? Le soluzioni sono:
L'ultima entry del blocco indice punta ad un secondo blocco indice (**schema concatenato**)
Il blocco indice di 1° livello punta a blocchi indice di 2° livello e così via, finchè i blocchi indice di ultimo livello puntano a blocchi di dati (**schema multi-livello**)
S'impiegano in combinazione puntatori a blocchi diretti e puntatori a blocchi indiretti (**schema combinato** detto anche **schema a multiplo indirizzamento indiretto**). È questo il caso dei SO UNIX-like, di seguito illustrato.

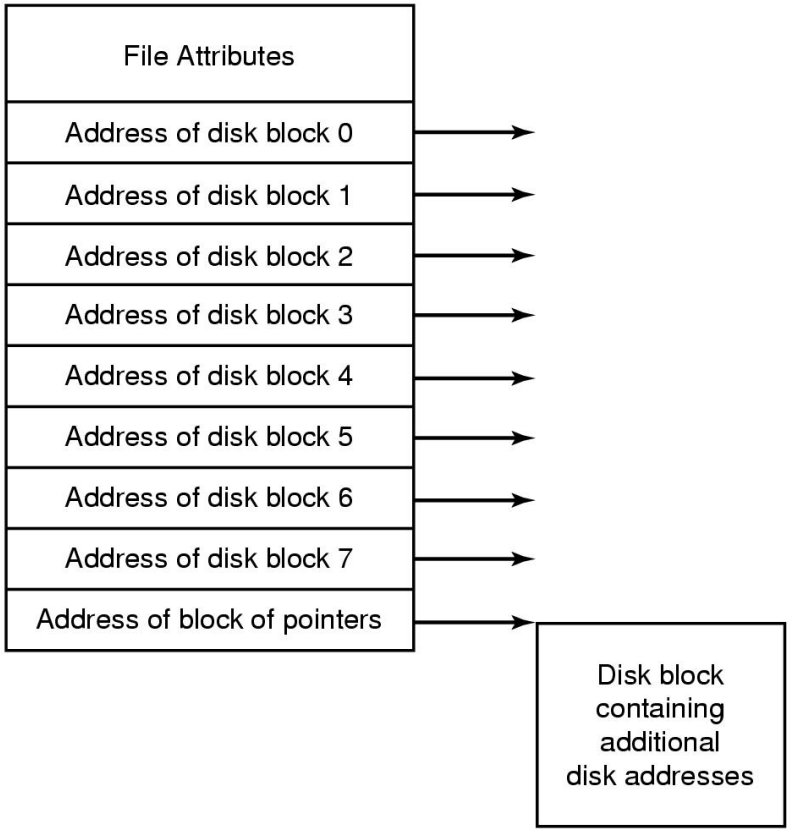
L'i-node in UNIX

Un file in Unix è rappresentato da un i-node (che contiene le informazioni fisiche circa il file).

Indexed allocation

i-node

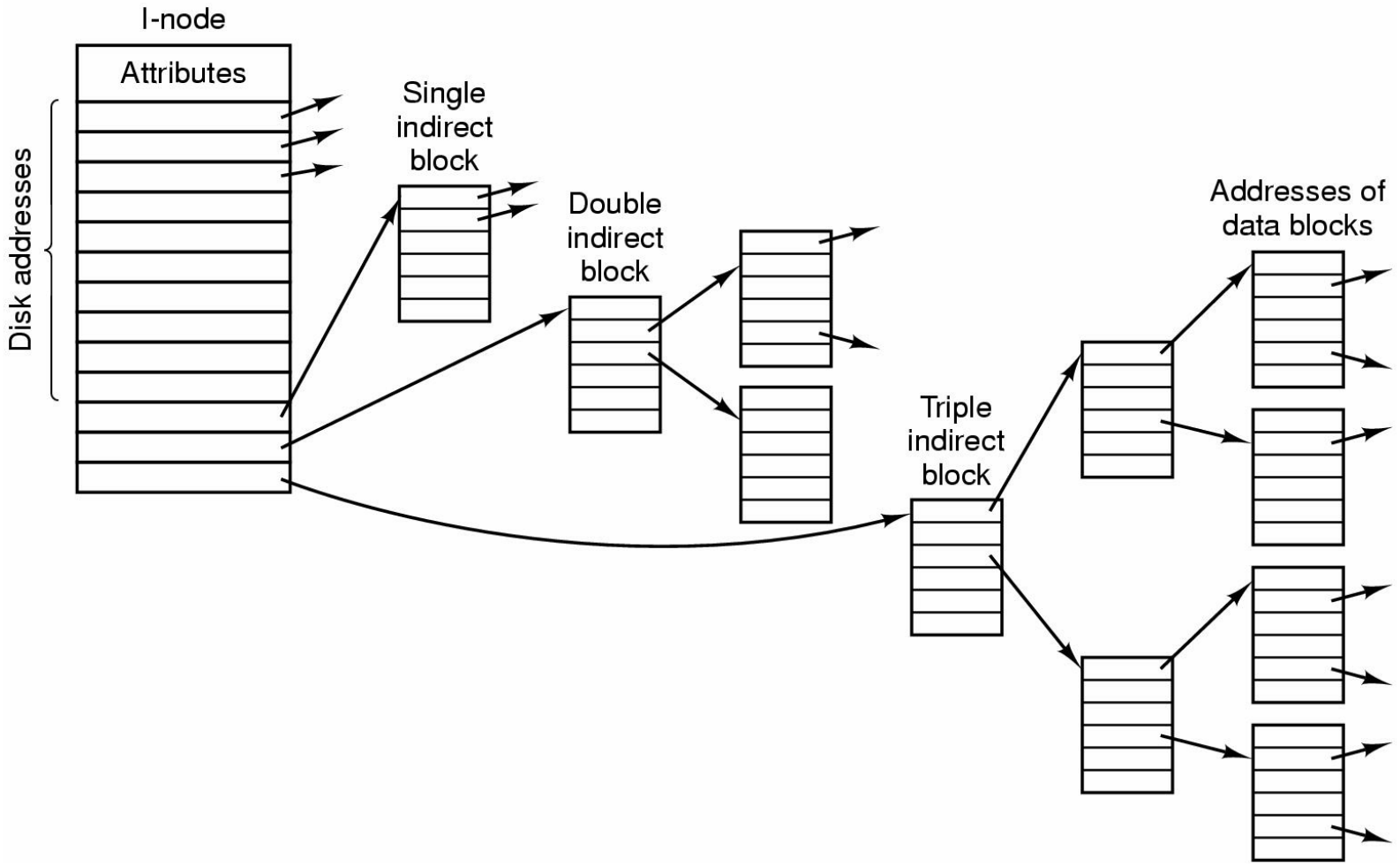




L'i-node in UNIX

Indexed allocation

Triple indexed allocation



i-node

Gli indici indiretti vengono allocati su richiesta.

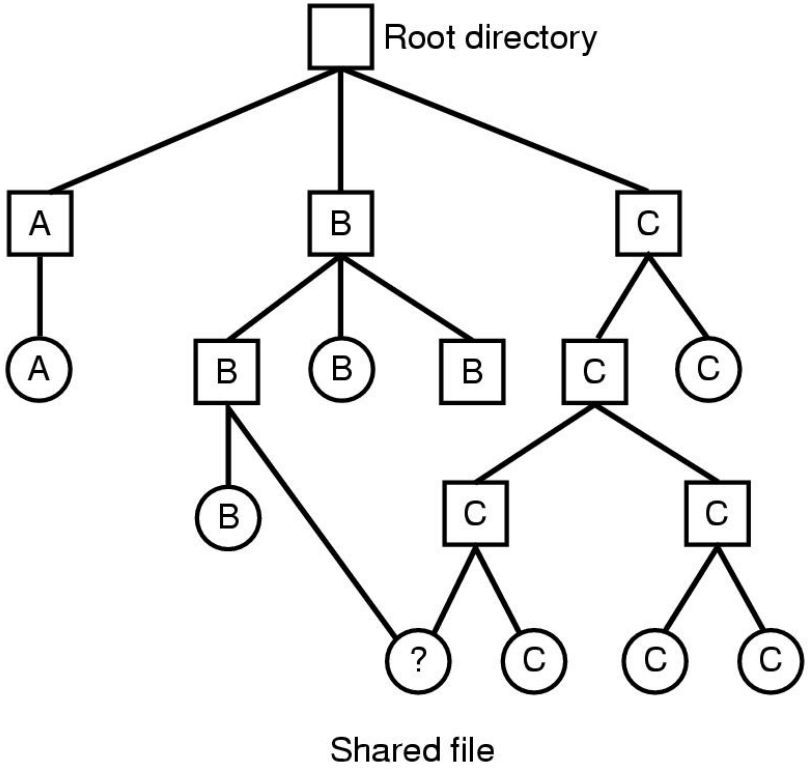
Accesso più veloce per file piccoli.

Lmax → #max di blocchi indirizzabile con blocchi da 4K e pointer da 4byte

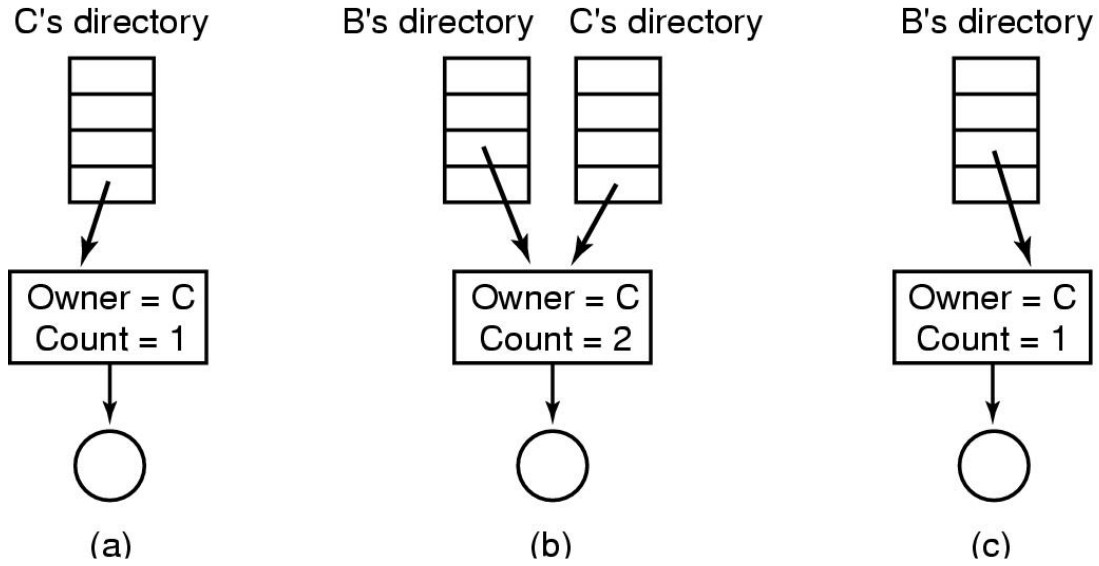
Lmax = 13 (indir. diretto) + 1024 (singolo indir. indiretto) + 1024^2 (doppio indir. indiretto) + 1024^3 (triplo indir. indiretto) > $1024^3 = 2^{30}$blk = 2^{42}byte = 4TB

MOLTO OLTRE LE CAPACITÀ DEI SISTEMI A 32 BIT.

Shared file



Shared file



(a) Situation prior to linking
(b) After the link is created
(c) After the original owner removes the file

Utilizzo spazio fisico e capacità singolo blocco

Nelle memorie secondarie soggette a *formattazione preliminare*, lo spazio viene suddiviso in blocchi di capacità fissa (dell'ordine di 2kb o multipli).

Un "blocco" o ***record fisico*** è l'unità minima soggetta a scrittura e a lettura.

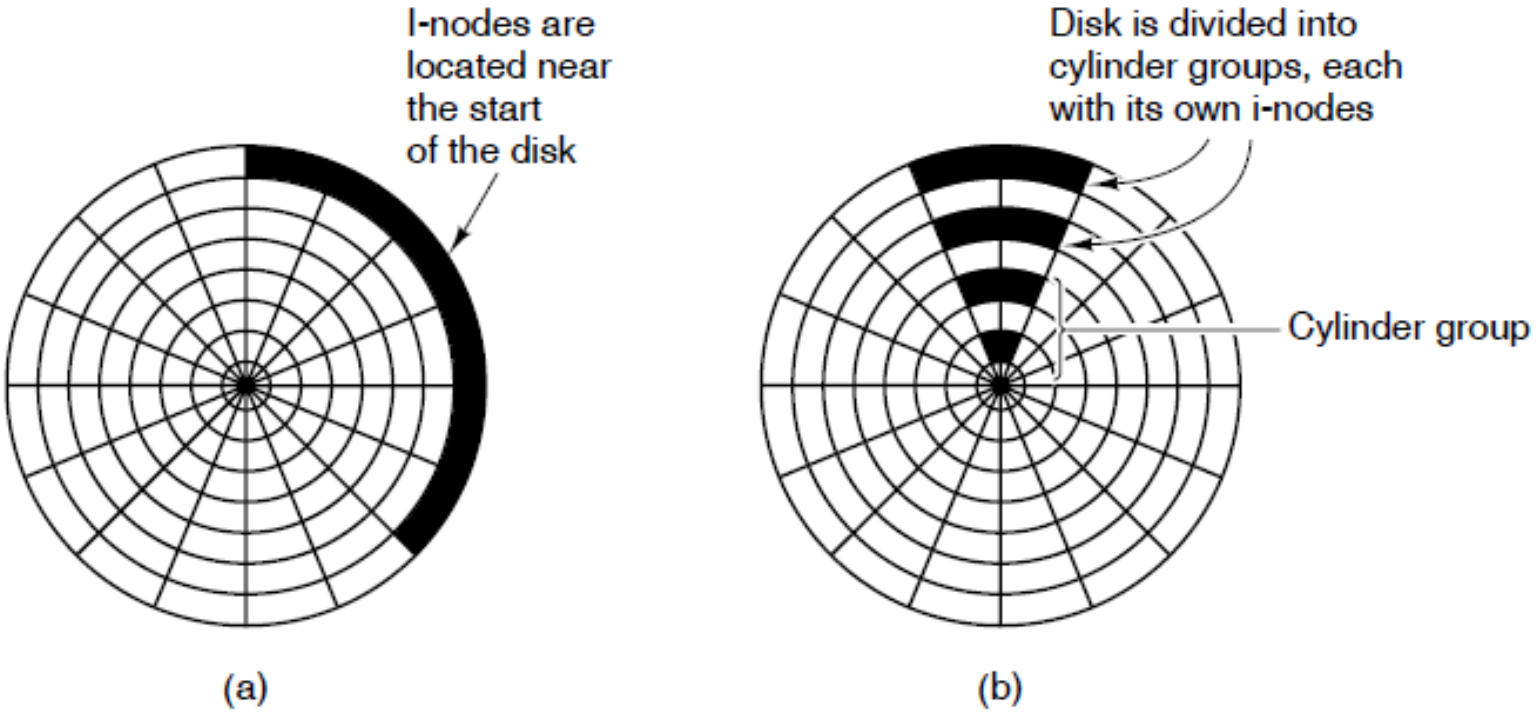
Questo può dare origine ad un cattivo utilizzo del disco, quando la dimensione del "***record logico***" (insieme dei dati che descrivono un singolo elemento dell'archivio) fosse sostanzialmente inferiore rispetto a quella del blocco (molto meno frequentemente un record logico richiede di espandere su più di un record fisico).

Per evitare tale spreco un ***record fisico*** può essere costituito al proprio interno da più "record logici". Si dice allora che *più record logici sono "bloccati" in un record fisico*. Il bloccaggio *consente anche di ridurre i tempi di lettura e scrittura*, ricorrendo all'uso di buffer di memoria: in un buffer di lettura viene alloggiato l'ultimo record fisico letto ed in un altro buffer viene alloggiato il record fisico da scrivere appena completato.

Espedienti per accelerare l'I/O

- ridurre il movimento delle testine ---> fig. (a)
- raggruppare (e leggere) i blocchi in gruppi (cluster) ---> fig. (b)

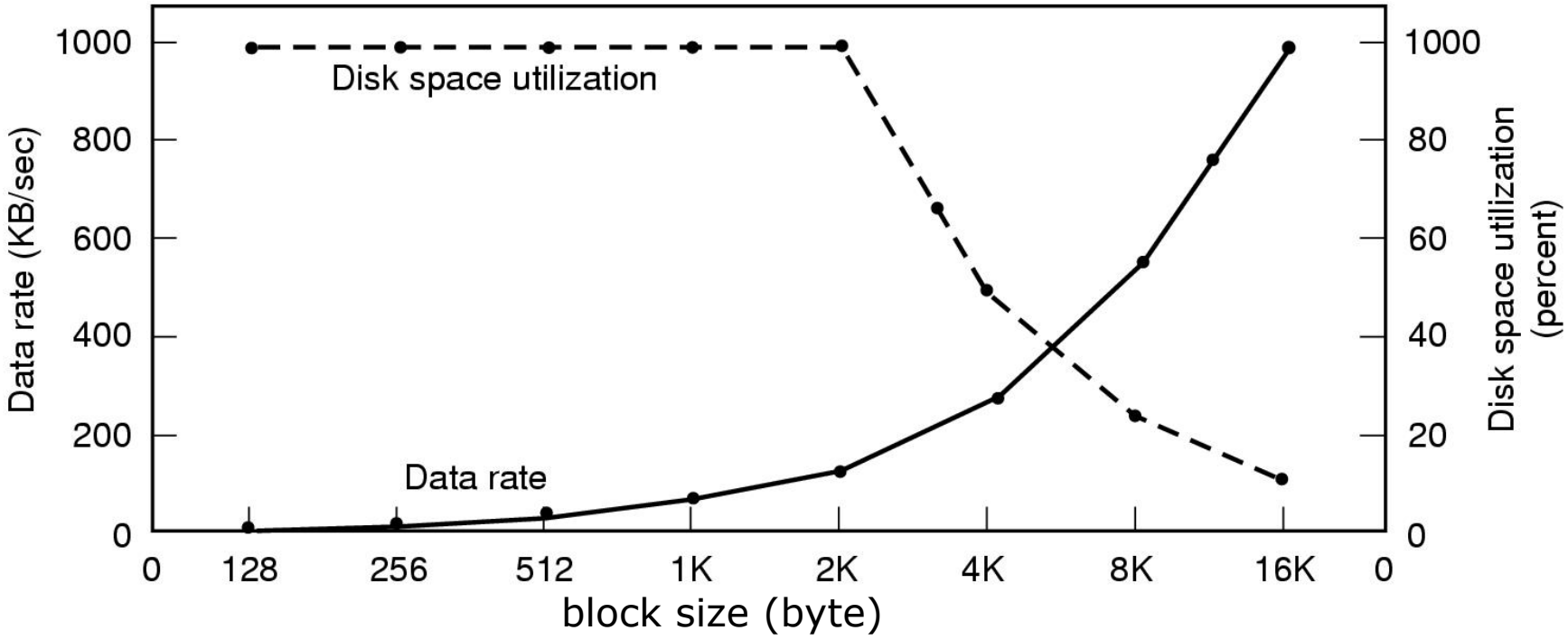
  collocare i blocchi con i metadati (inode, p.e.) presso i rispettivi dati



Utilizzo spazio fisico e capacità singolo blocco

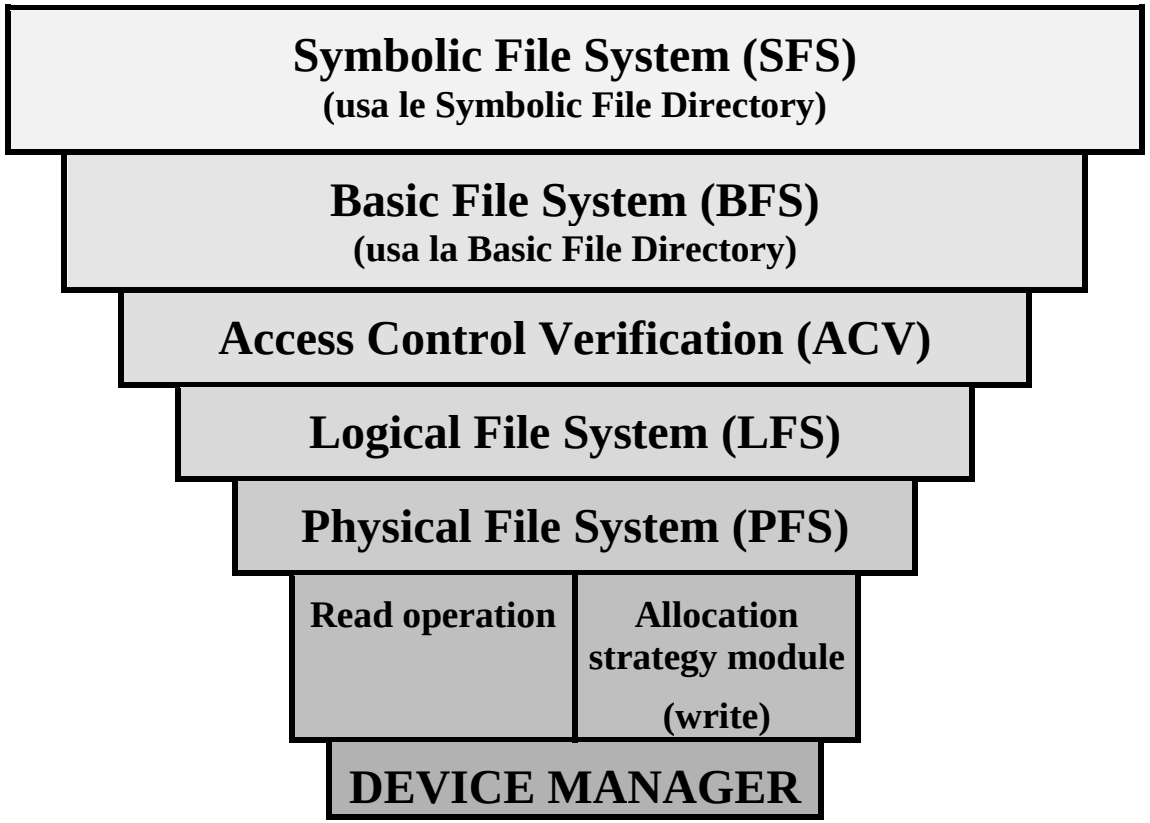
- Dark line (left hand scale) gives data rate of a disk
- Dotted line (right hand scale) gives disk space efficiency

All files 2KB



L'architettura di un File System

Il modello generale



L'architettura di un File System

Il modello generale (1/3)

Il modello prevede un file system gerarchico, formato da una directory radice e più sottodirectory.

Ogni directory o cartella (detta Symbolic File Directory o **SFD**) è un file costituito da record contenenti ciascuno:

- il nome simbolico di un file o sotto-directory;
- il record della Basic File Directory (**BFD**) in cui sono riportate le informazioni relative al file o sotto-directory.

La BFD, una per l'intero volume, è un file costituito da tanti record quanti sono i file e directory contenuti nel volume in questione.

L'architettura di un File System

Il modello generale (2/3)

La funzione svolta da ciascun livello può essere compresa analizzando le operazioni svolte per l'esecuzione, da parte di un processo-utente, del seguente comando, che qui si suppone riferito, senza perdita di generalità, ad un file sequenziale:

READ nome_volume, path, nome_file, lista_variabili

Tale comando implicitamente prevede la lettura dell'i-esimo record del file, dal quale ricavare i valori delle variabili specificate nella lista.

Il modulo **SFS**, identificato il volume (*nome_volume*) ed il percorso per raggiungere la **SFD** contenente il file richiesto (*path*), provvede al controllo dell'esistenza in essa di un file avente il *nome_file* specificato.

Il SFS provvede quindi a chiamare il BFS fornendogli il record della **BFD** associato al nome specificato.

Il **BFS** accede a tale record della BFD, copiandolo in memoria perché il suo contenuto possa essere utilizzato, a tempo debito, dai moduli sottostanti della gerarchia.

In particolare l'**ACV** provvede a verificare che il processo-utente che ha originato il comando possieda i "diritti di accesso" richiesti per eseguire l'operazione richiesta. L'ACV chiama quindi il LFS.

L'architettura di un File System

Il modello generale (3/3)

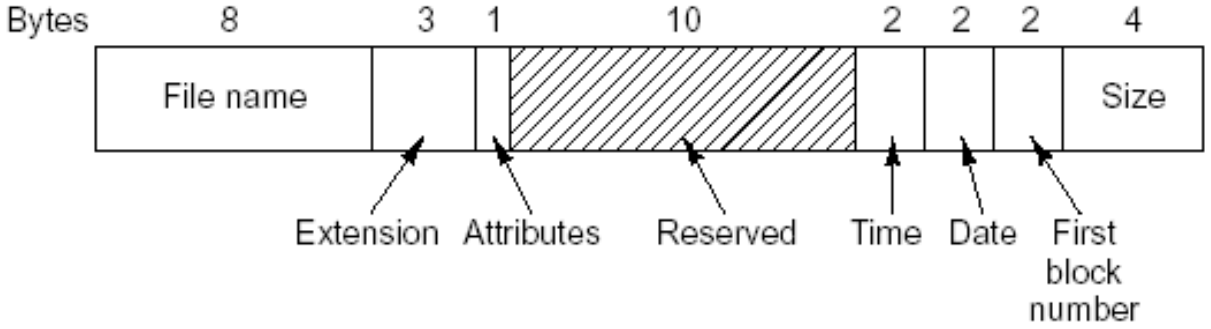
Il **LFS** determina il record logico al quale si sta facendo riferimento. Dal numero del record logico e dalla struttura del file (lunghezza del record logico e numero di record logici costituenti il singolo record fisico), si ricava il record fisico al quale si deve accedere. Quest'ultima informazione, viene passata al modulo PFS.

Il **PFS** può quindi procedere alla individuazione delle coordinate del record fisico sul disco ed avviare la successiva operazione di lettura.

Se il comando richiedesse un'operazione di scrittura, invece che una di lettura, l'Allocation Strategy Module provvederebbe a ricavare il blocco libero su cui effettuare l'operazione di scrittura.

Modello generale e sua realizzazione

Il DOS non distingue tra symbolic e basic file directory; infatti adotta un unico tipo di directory: la File Allocation Table (FAT).
I singoli record della FAT indicano se si riferiscono a file o directory. La FAT contiene l'indirizzo del primo record del file.



The MS-DOS directory entry.

Invece, **UNIX adotta il modello distinguendo tra directory** (equivalente alla symbolic file directory) **e i-list** (equivalente alla basic file directory del modello). I suoi record, uno per ciascun file, si chiamano i-node. **I file possono essere di tipo ordinario, di tipo directory o di tipo speciale** (drivers). Tali files speciali sono contenuti nella directory /DEV.
Il proprietario del file definisce i diritti di lettura, di scrittura e d'esecuzione. Tali diritti vengono definiti per sé stesso (cioè il proprietario), per il suo gruppo e per gli "altri". In totale, le informazioni sui permessi occupano nove bit.