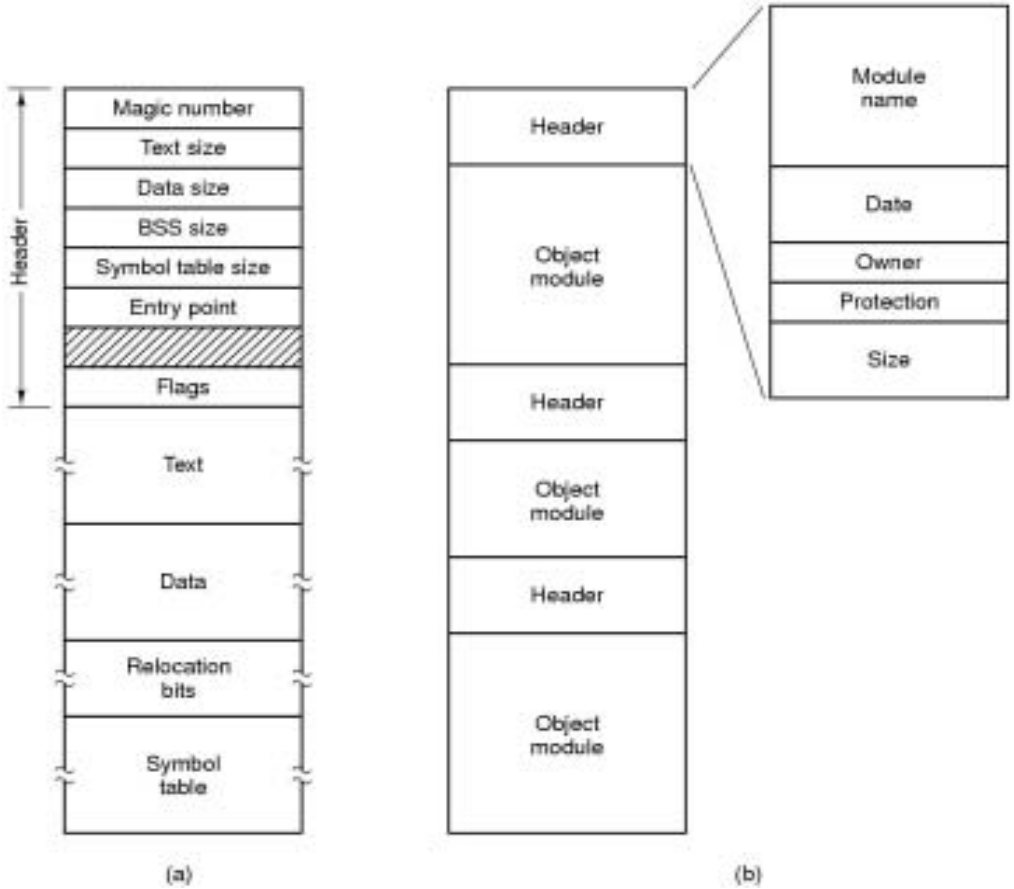
THE FILE CONCEPT

….. contiguous logical address space

- Must store large amounts of data
- Information stored must survive the termination of the process using it
- Multiple processes must be able to access the information concurrently

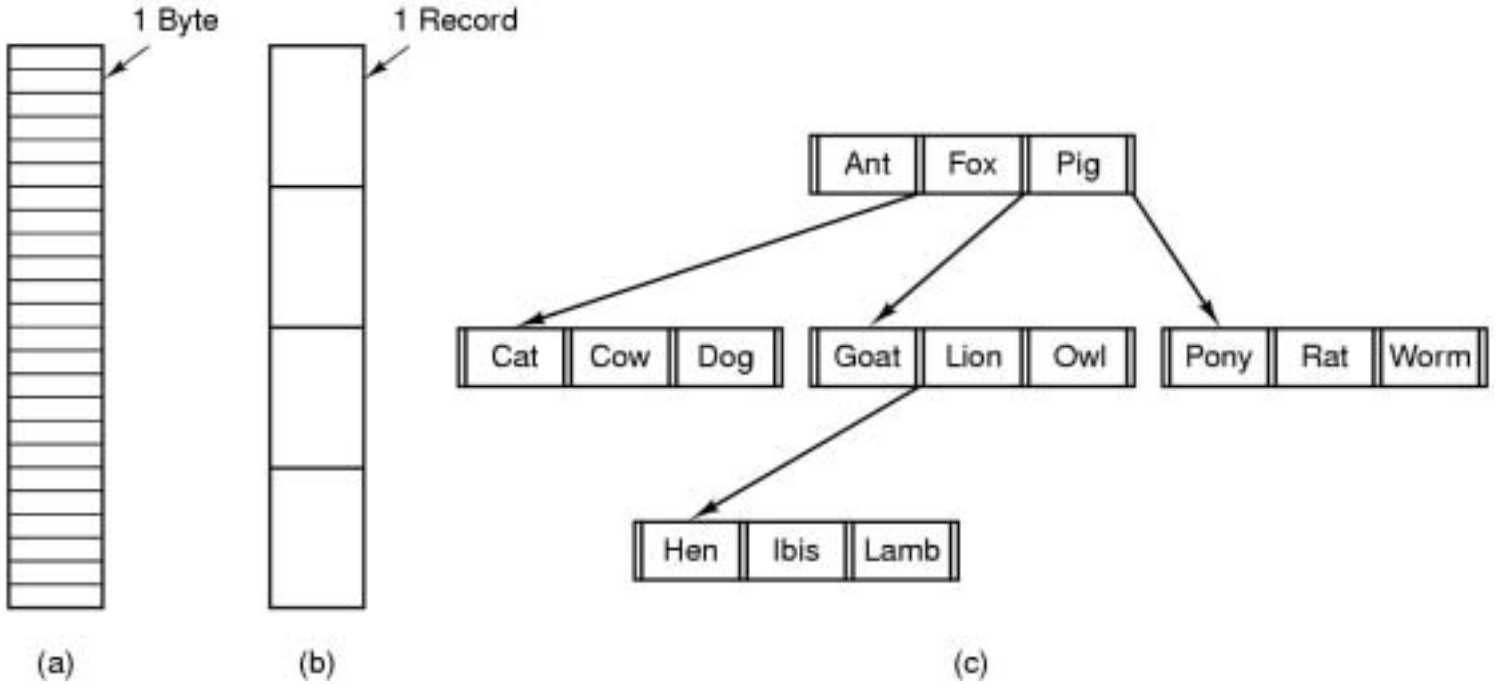
FILE CONTENT

- Data
  - numeric
  - character
  - binary
- Program



FILE STRUCTURES

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file



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

FILE OPERATIONS (FILE SYSTEM CALLS)

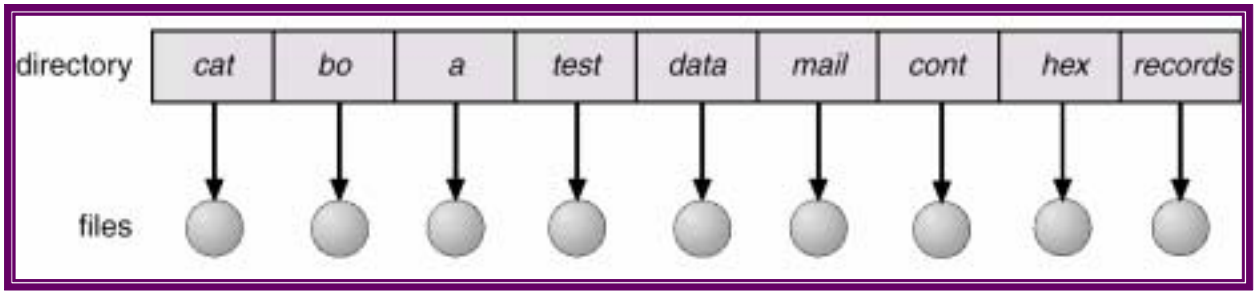
- Create
- Write
- Read
- Seek
- Delete
- Rename
- Get attributes
- Set attributes
- Open(*Fi*) – search the directory structure on disk for entry *Fi*, and move the content of entry to memory.
- Close (*Fi*) – move the content of entry *Fi* in memory to directory structure on disk.

DIRECTORY FILE

One level Directory

A single directory for all users.

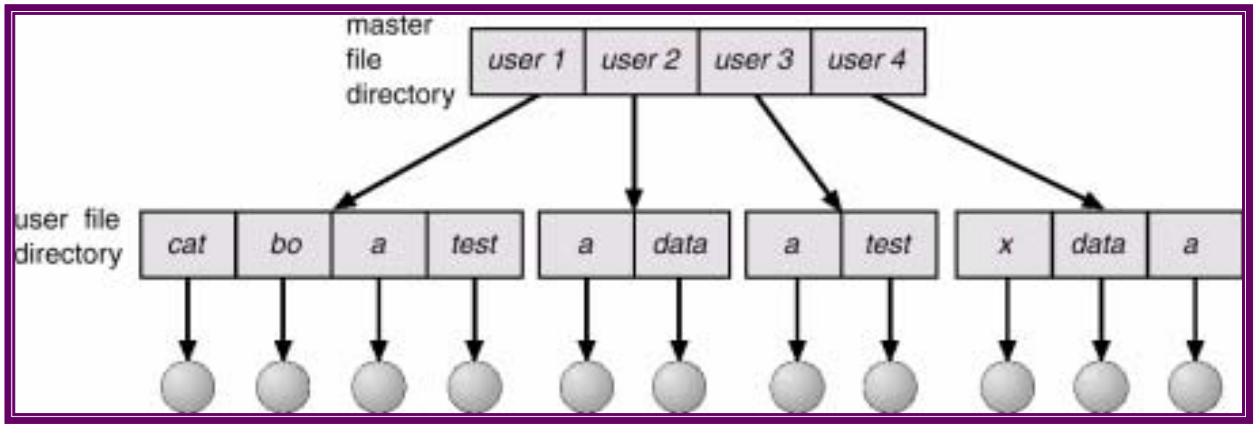
- Naming problem
- Grouping problem



Two levels Directory

Separate directory for each user.

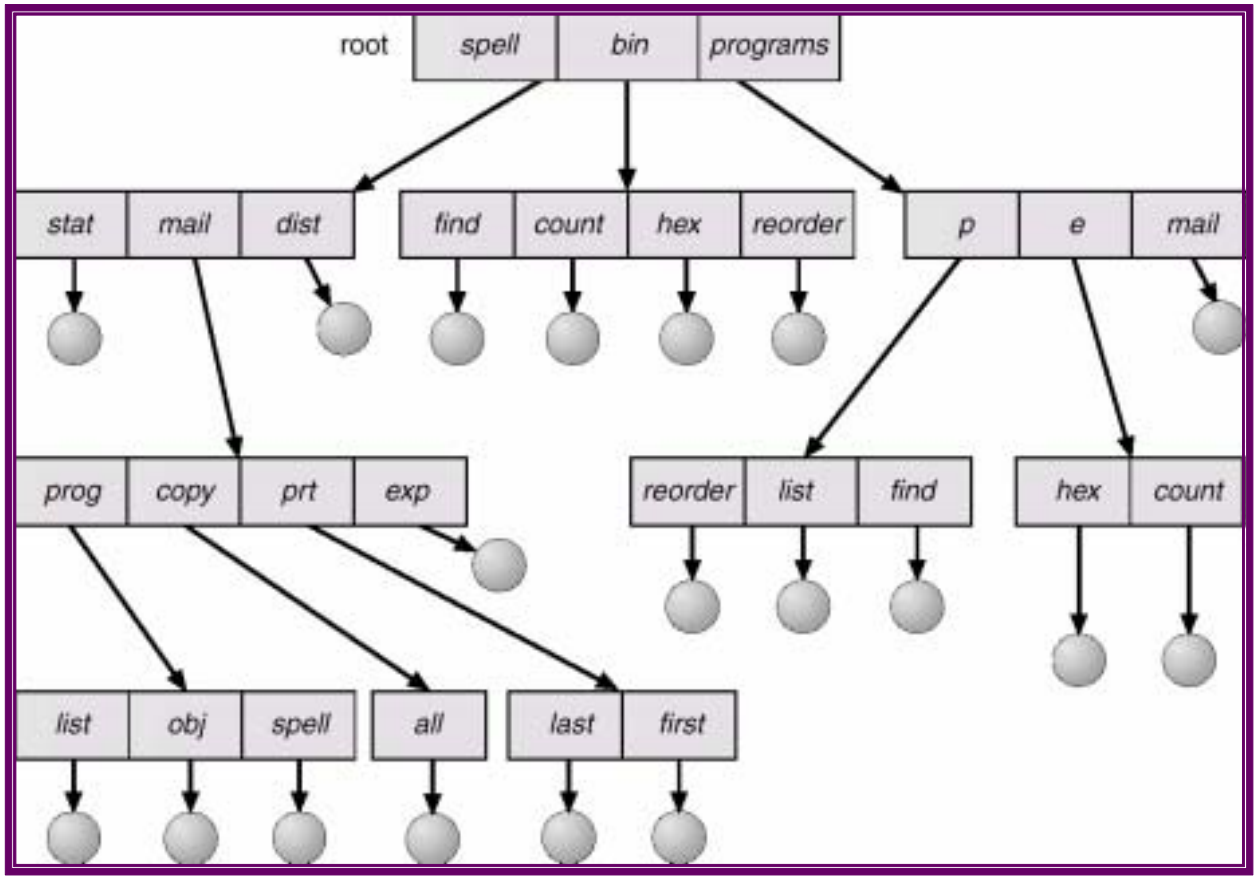
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability



DIRECTORY FILE

Directory Hierarchy

- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - cd /spell/mail/prog
  - type list
- Absolute or relative path name
- Creating a new file is done in current directory.
- Delete a file
  - rm <file-name>
- Creating a new subdirectory is done in current directory.
  mkdir <dir-name>

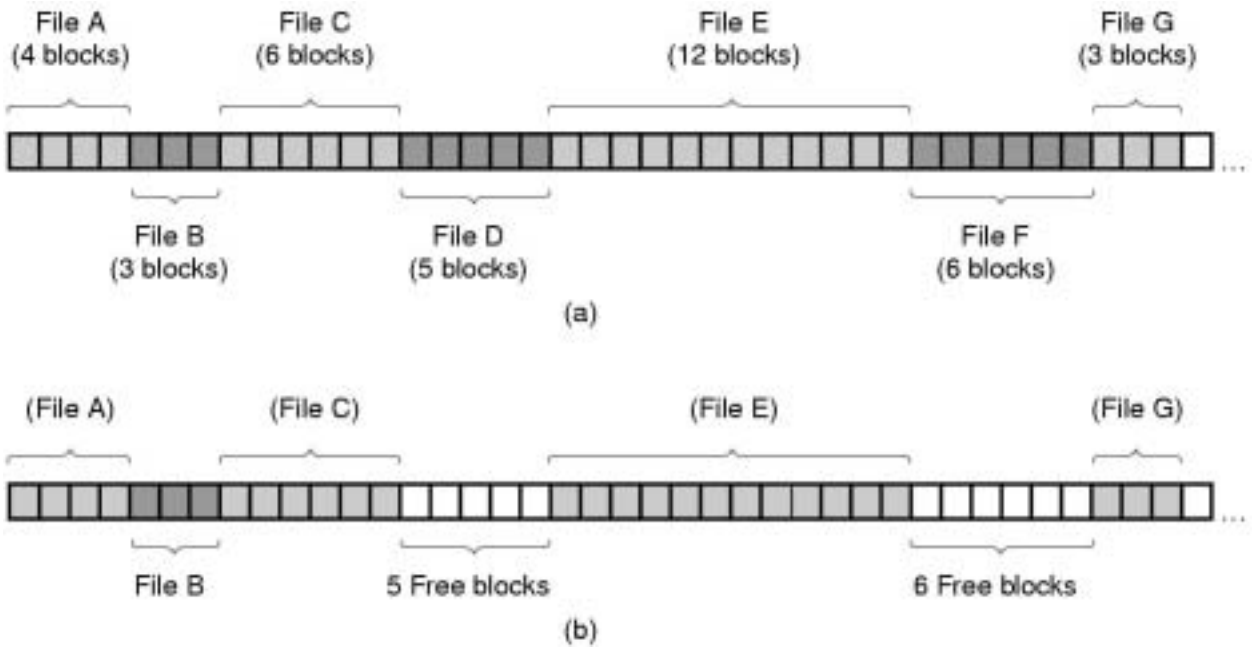


DIRECTORY OPERATIONS

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

PHYSICAL ALLOCATION OF FILES

Contiguous Allocation



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

- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.

PHYSICAL ALLOCATION OF FILES

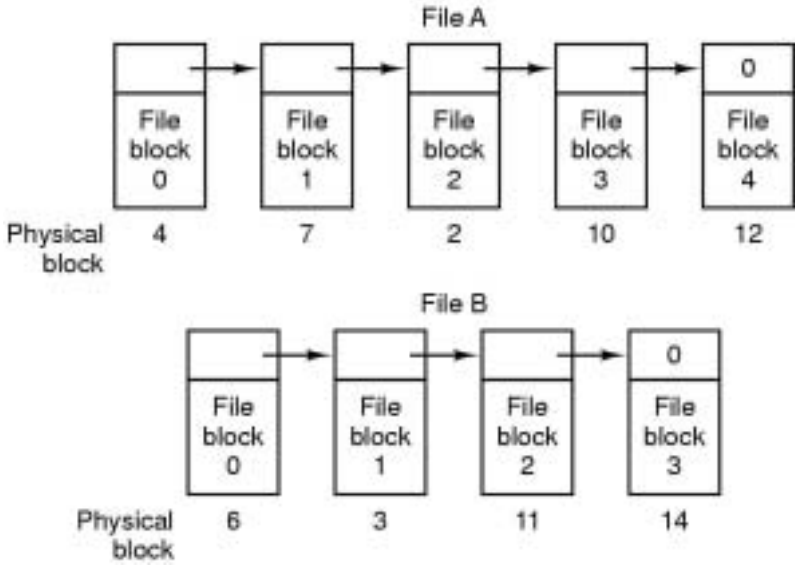
Disk blocks chained list

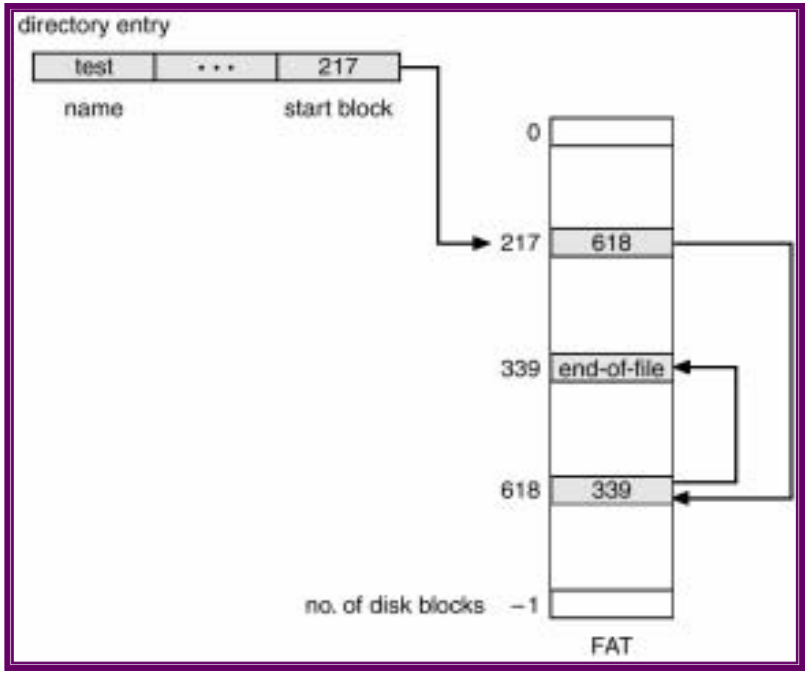
Each file is a chained list of disk blocks: blocks may be scattered anywhere on the disk.

- ✦ Simple – need only starting address
- ✦ Free-space management system – no waste of space
- ✦ Mapping

File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2 can have as a pointer to the start block of the file:

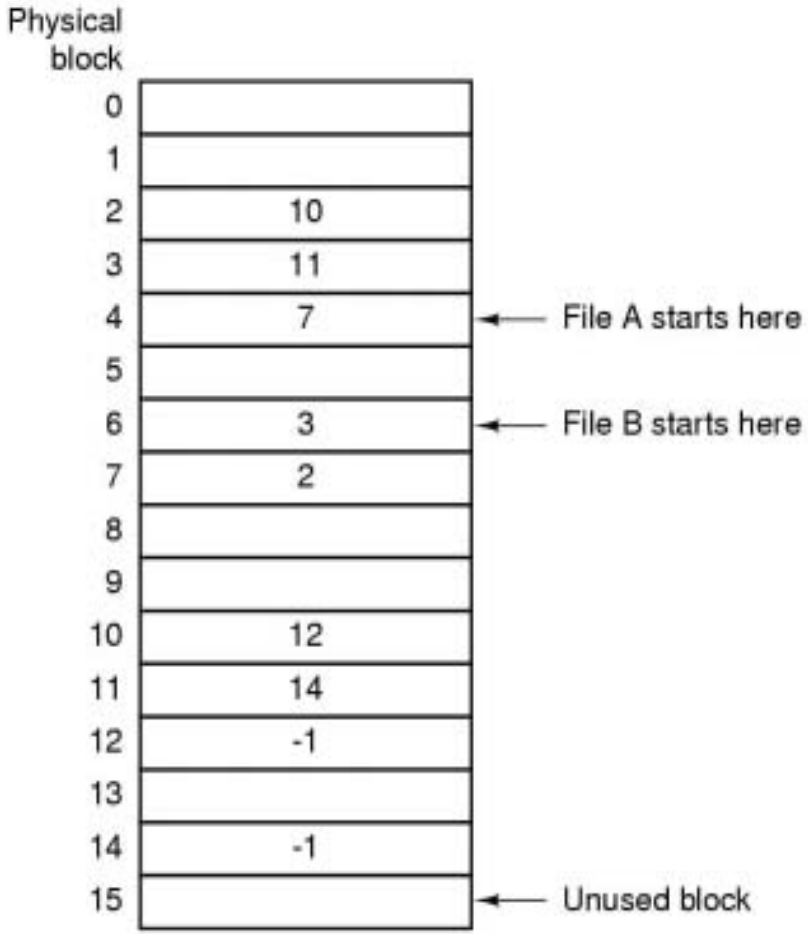
- the ***physical address*** (cyl, trk, sector) of the first physical block
- the first element of the file in the ***linked-list***, an array implementing an easy bi-univocal correspondence among logical and physical address.





PHYSICAL ALLOCATION OF FILES

Linked List through a memory table



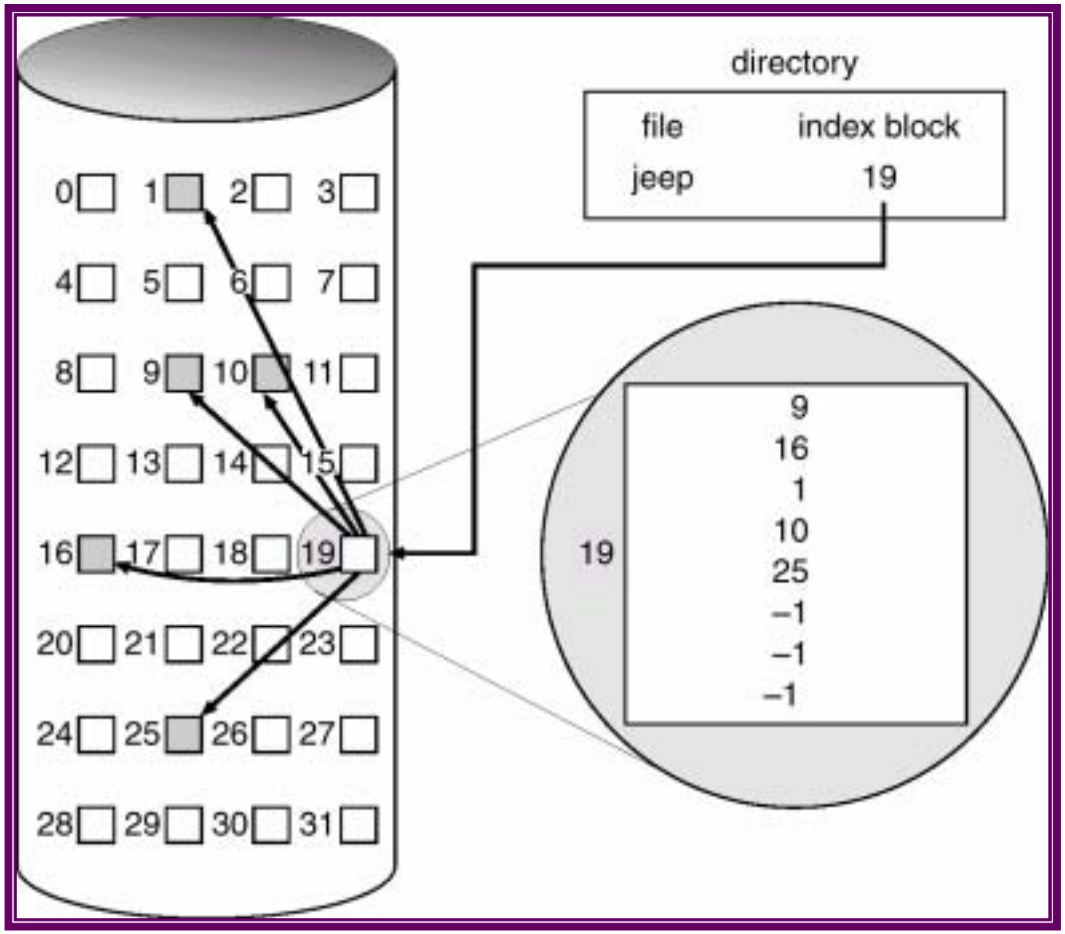
PHYSICAL ALLOCATION OF FILES

Indexed allocation

Brings all pointers together into the *index block.*

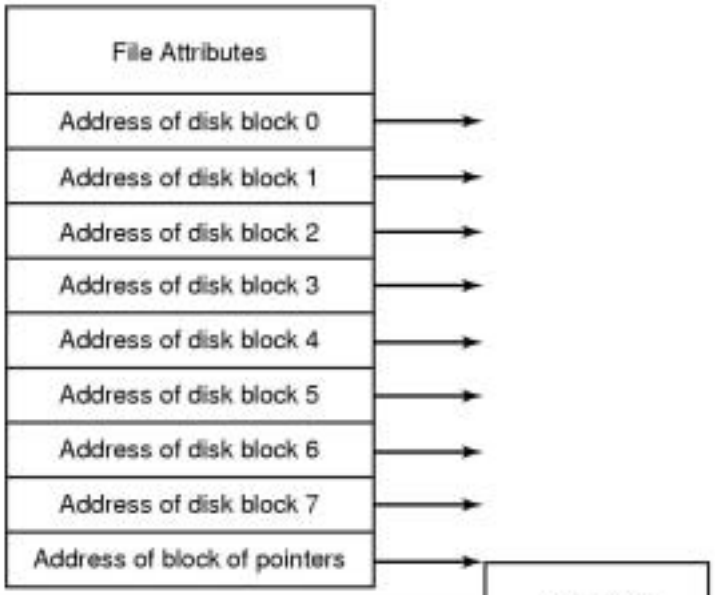
Logical view.

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.

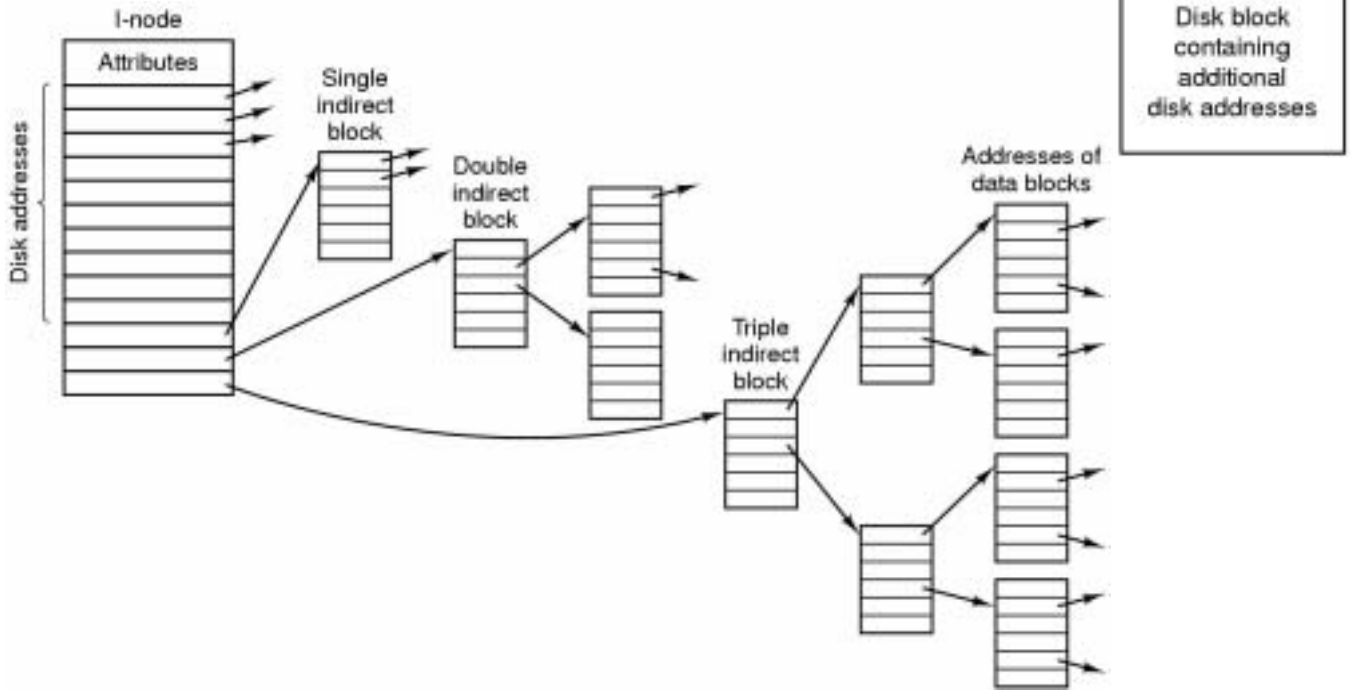


PHYSICAL ALLOCATION OF FILES

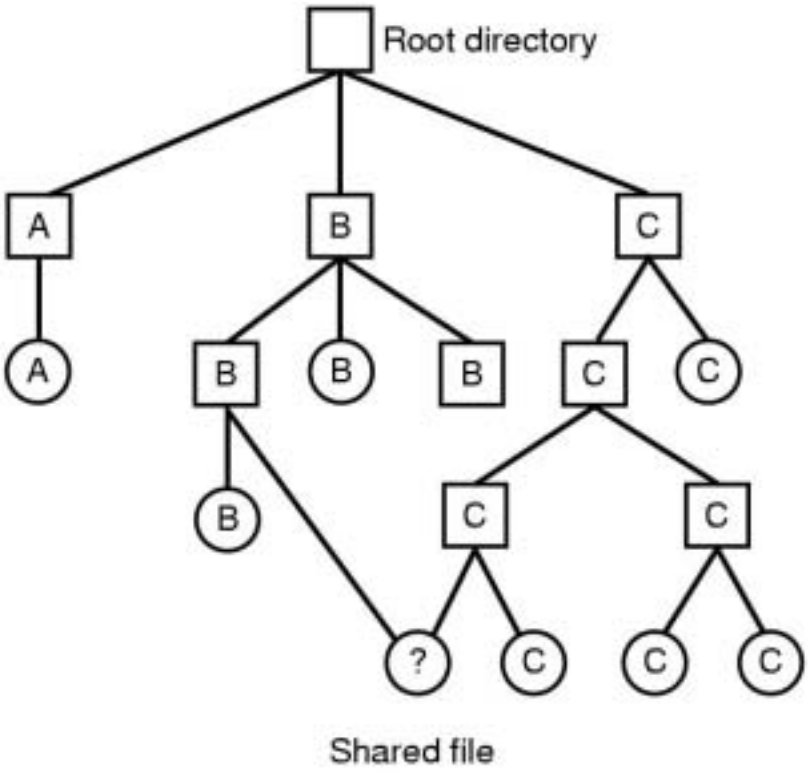
Indexed allocation (i-node)

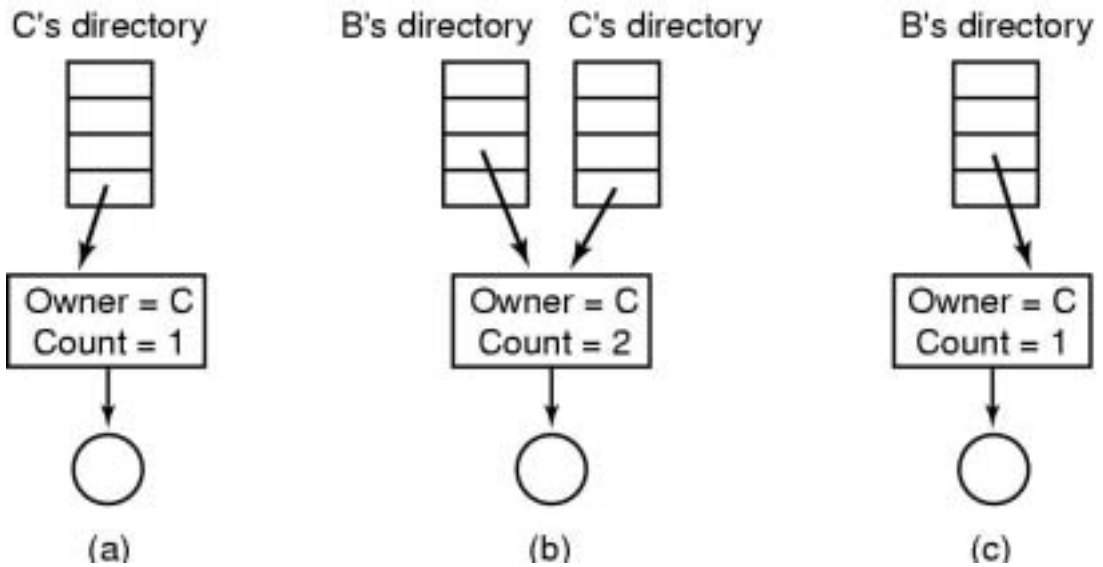


Triple indexed allocation



SHARED FILES





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

IL FILE SYSTEM

Il file system è il livello più esterno del nucleo del SO, cioè quello più vicino all'utente.

La sua funzione è quella di gestire le memorie secondarie fornendo una politica d'utilizzo per le stesse.

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.

IL FILE SYSTEM

Preallocazione vs Allocazione dinamica

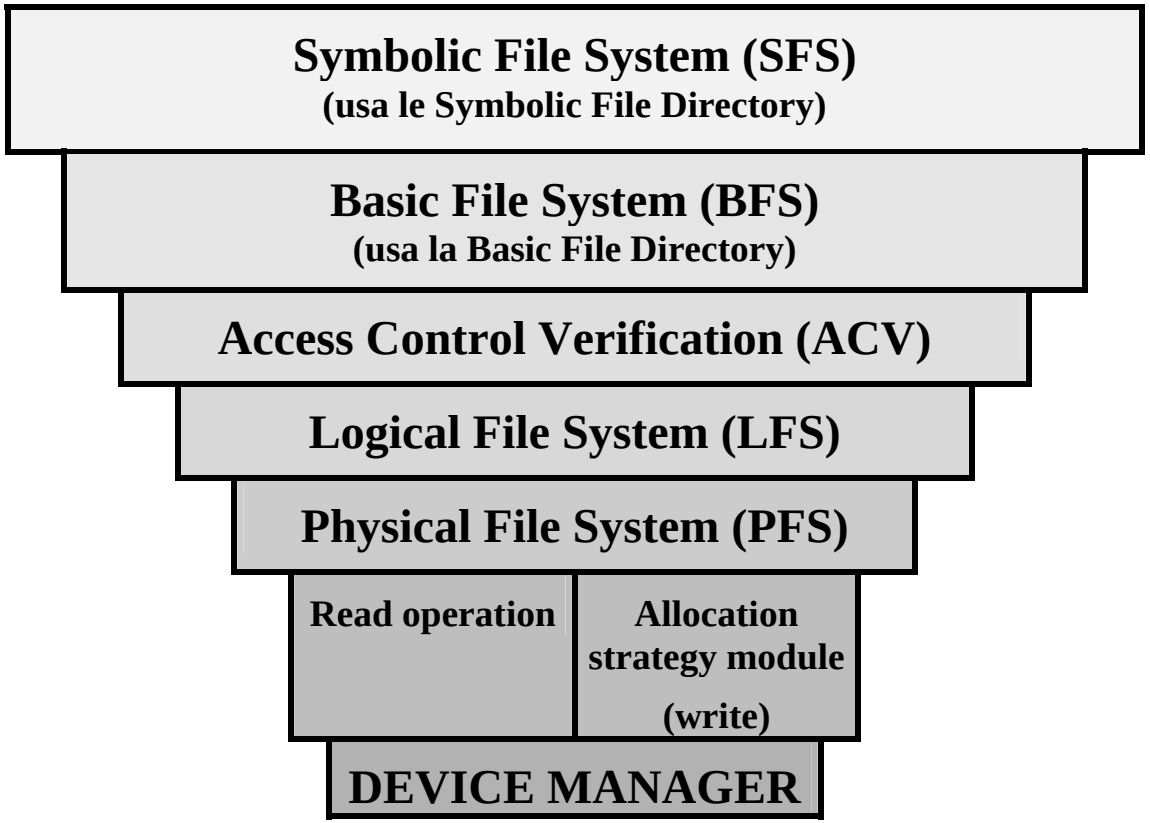
La politica di **preallocazione** impone che il programma, all'inizio della sua esecuzione, specifichi di quanto spazio di memoria di massa ha bisogno. Questo è un modo d'operare alquanto scomodo, poiché è frequente il caso in cui non si abbia la minima idea di quanto spazio realmente serva. Esso però 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 su disco, 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, quando un programma chiede di scrivere su disco, 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. Questo è un metodo più veloce rispetto a quello dell'allocazione dinamica, poiché la ricerca dei record liberi (nella tabella) avviene una volta ogni 16 record allocati. Invece, nell'allocazione dinamica, la ricerca dei record liberi avviene ad ogni operazione di scrittura.

L'ARCHITETTURA DI UN FILE SYSTEM

Il modello generale



L'ARCHITETTURA DI UN FILE SYSTEM

Il modello generale

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

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.

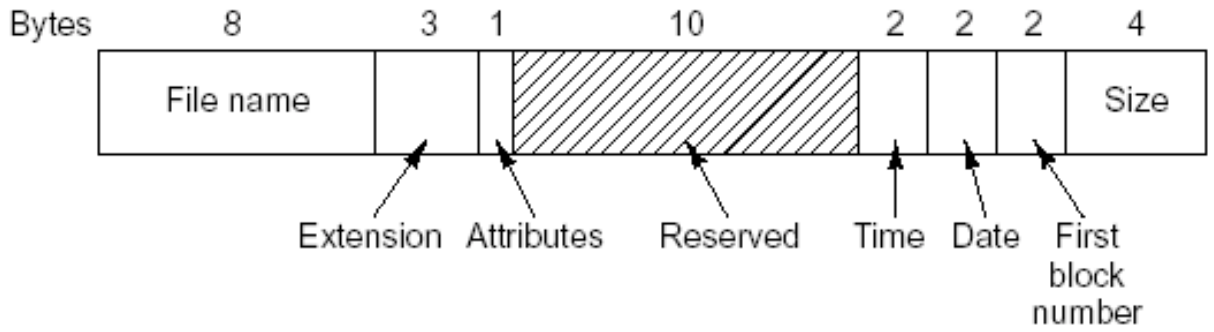
Il **LFS** determina il record logico al quale si sta facendo riferimento. Dal numero del record logico e dalla struttura del file (lunghezza del record logico e numero di record logici costituenti il singolo record fisico), si ricava il record fisico al quale si deve accedere. Quest'ultima informazione,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