UNIT IV
Disk Structure

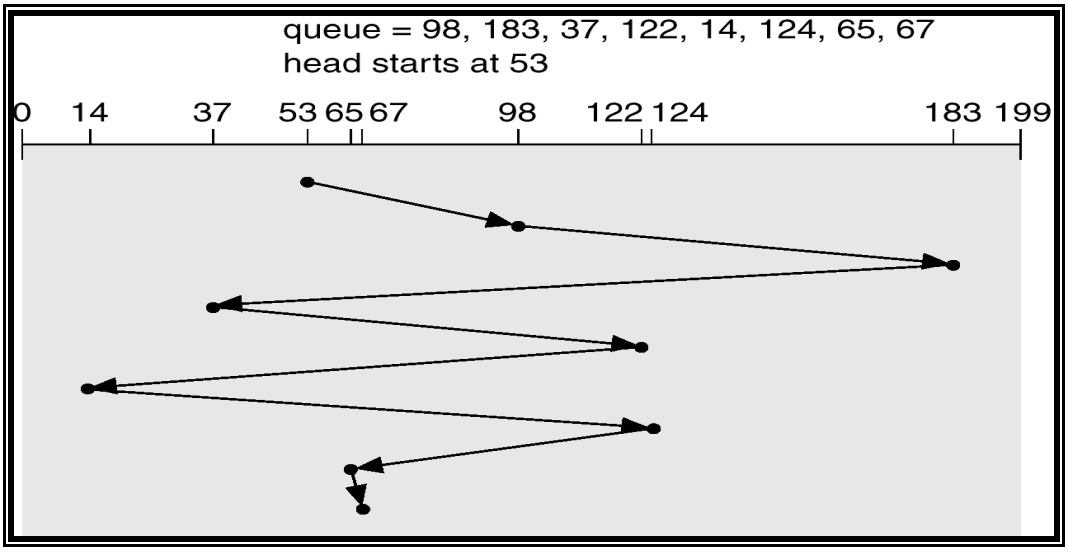
- Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
  - Sector 0 is the first sector of the first track on the outermost cylinder.
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
  - *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
  - *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time ≈ seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
- Several algorithms exist to schedule the servicing of disk I/O requests.

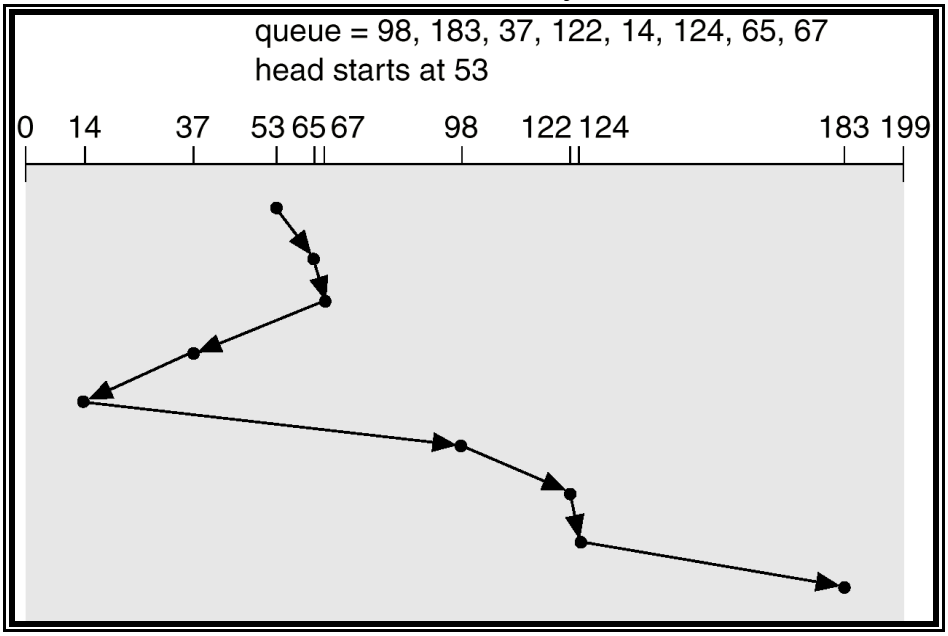
- We illustrate them with a request queue (0-199).
  98, 183, 37, 122, 14, 124, 65, 67
  Head pointer 53

FCFS



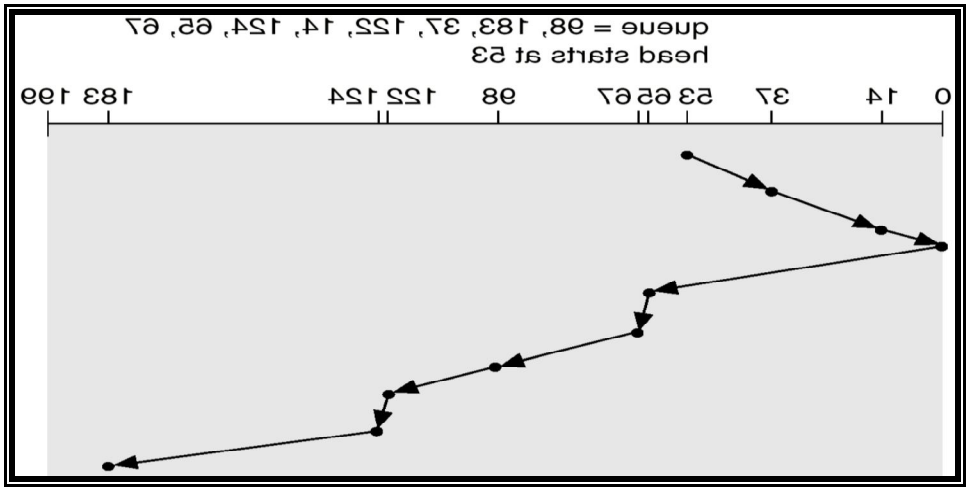
SSTF

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.



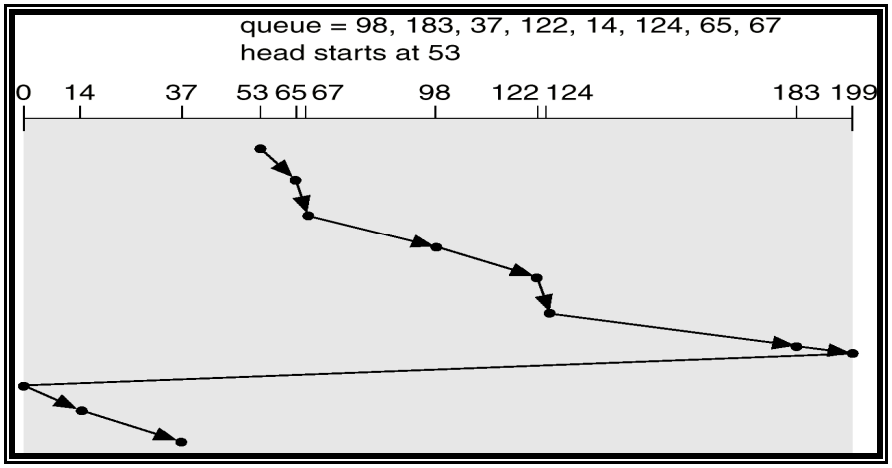
SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
- Illustration shows total head movement of 208 cylinders.



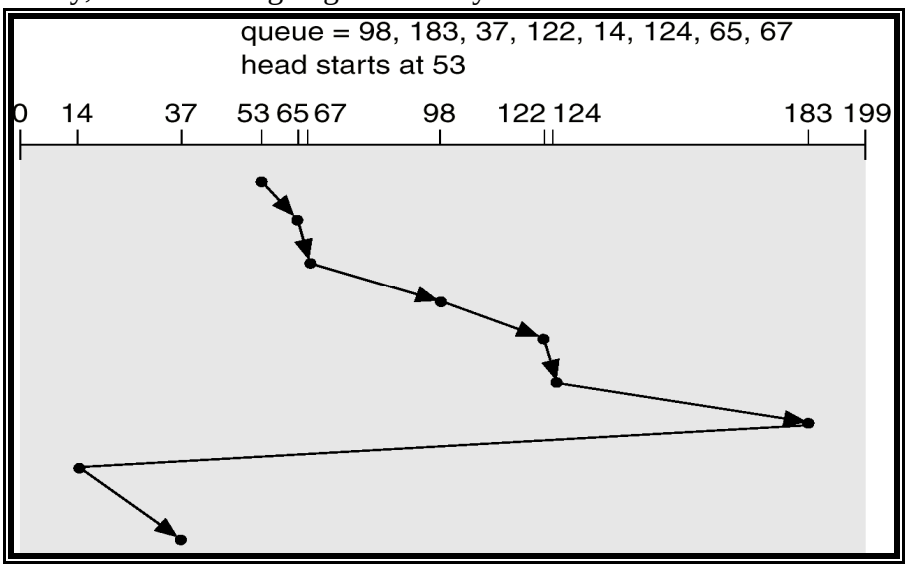
C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.



C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.



Selecting a Disk-Scheduling Algorithm

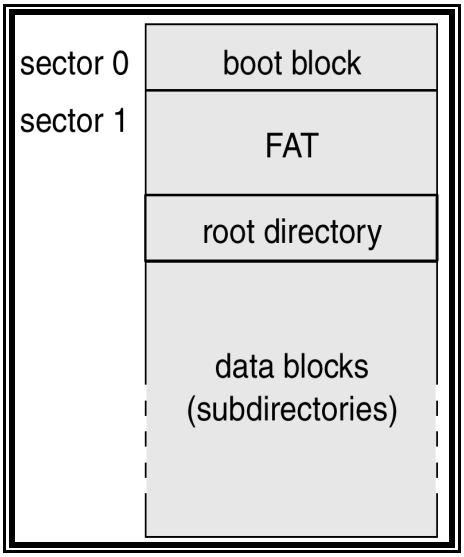
- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.

- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

Disk Management

- *Low-level formatting,* or *physical formatting* — Dividing a disk into sectors that the disk controller can read and write.
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - *Partition* the disk into one or more groups of cylinders.
  - *Logical formatting* or "making a file system".
- Boot block initializes system.
  - The bootstrap is stored in ROM.
  - *Bootstrap loader* program.
- Methods such as *sector sparing* used to handle bad blocks.

MS-DOS Disk Layout



Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory.
- Swap-space can be carved out of the normal file system,or, more commonly, it can be in a separate disk partition.
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds *text segment* (the program) and *data segment.*
  - Kernel uses *swap maps* to track swap-space use.
  - Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

File-System Interface

File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters.
- Who decides:
  - Operating system
  - Program

File Attributes

- **Name** – only information kept in human-readable form.
- **Type** – needed for systems that support different types.
- **Location** – pointer to file location on device.
- **Size** – current file size.
- **Protection** – controls who can do reading, writing, executing.
- **Time**, **date**, **and user identification** – data for protection, security, and usage monitoring.

Information about files are kept in the directory structure, which is maintained on the disk

File Operations

- Create
- Write
- Read
- Reposition within file – file seek
- Delete
- Truncate
- Open(F_i) – search the directory structure on disk for entry F_i, and move the content of entry to memory.
- Close (F_i) – move the content of entry F_i in memory to directory structure on disk.

File Types – Name, Extension

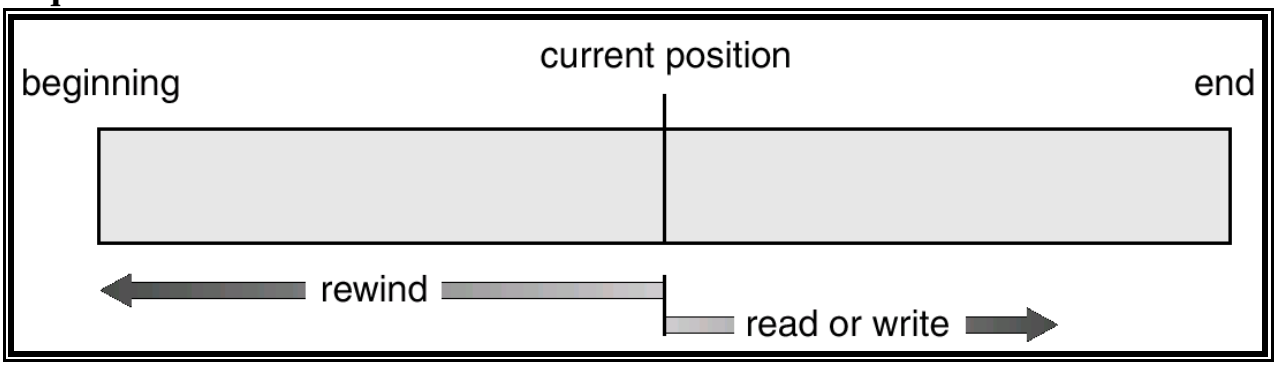
file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Access Methods

- **Sequential Access**
  - *read next*
  - *write next*
  - *reset*
  - no *read after last write*
    - *(rewrite)*
- **Direct Access**
  - *read n*
  - *write n*
  - *position to n*
    - *read next*
    - *write next*
  - *rewrite n*

n = relative block number

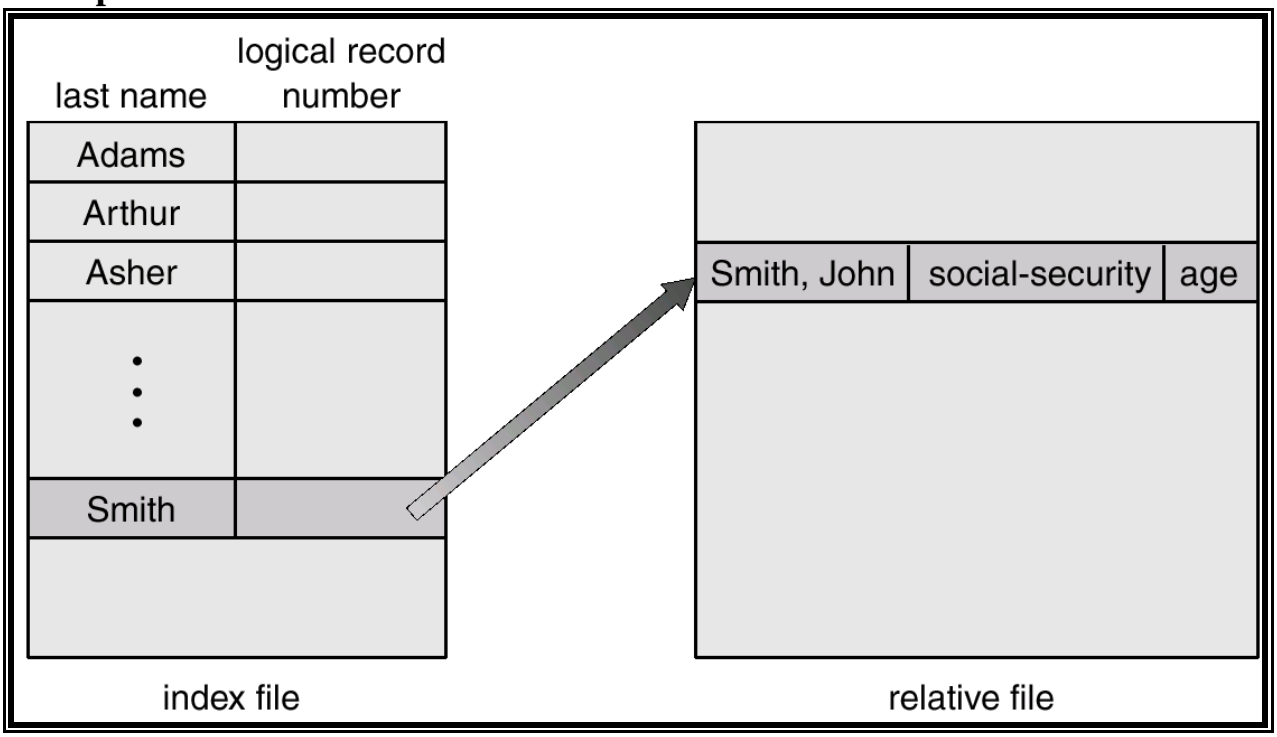
Sequential-access File



Simulation of Sequential Access on a Direct-access File

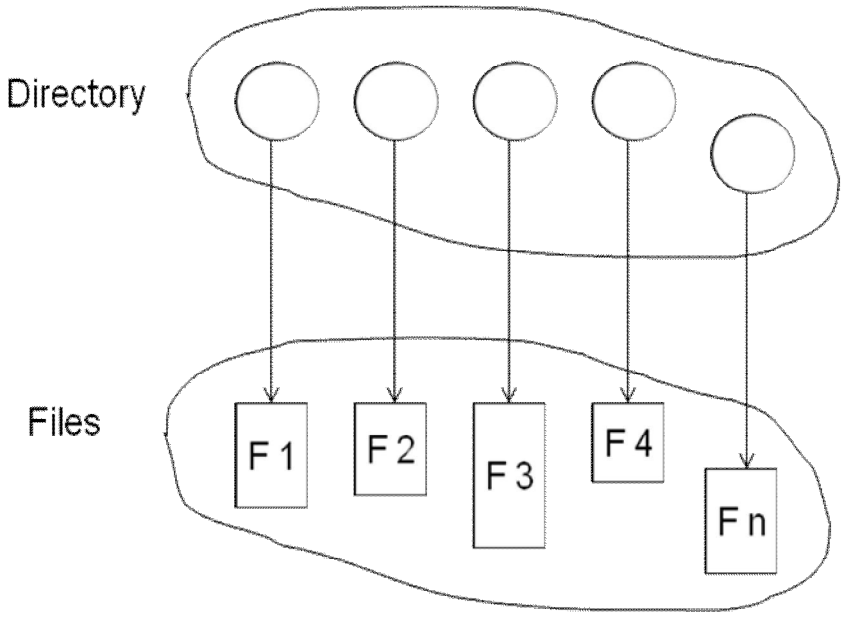
sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp*+1;
write next	*write cp*; *cp* = *cp*+1;

Example of Index and Relative Files

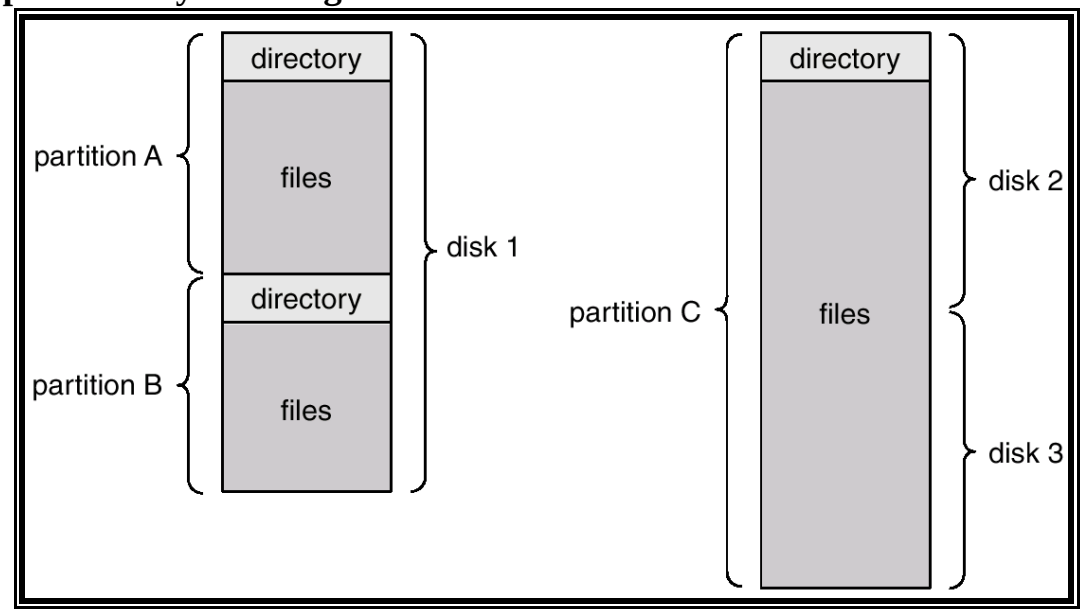


Directory Structure

- A collection of nodes containing information about all files.
- Both the directory structure and the files reside on disk.
- Backups of these two structures are kept on tapes.



A Typical File-system Organization



Information in a Device Directory

- Name
- Type

- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID (who pays)
- Protection information (discuss later)

Operations Performed on Directory

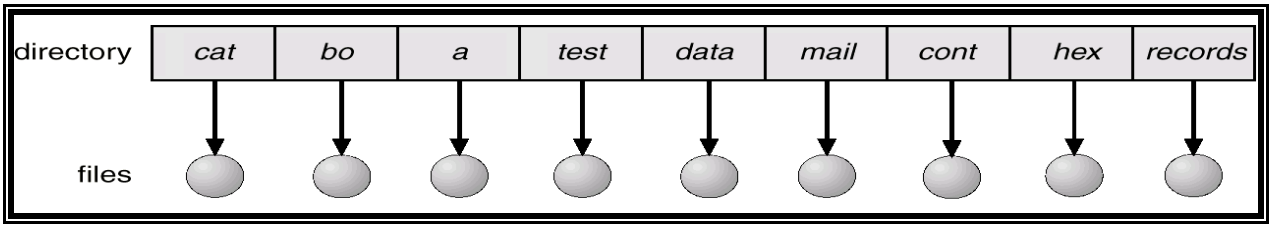
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (Logically) to Obtain

- **Efficiency** – locating a file quickly.
- **Naming** – convenient to users.
  - Two users can have same name for different files.
  - The same file can have several different names.
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

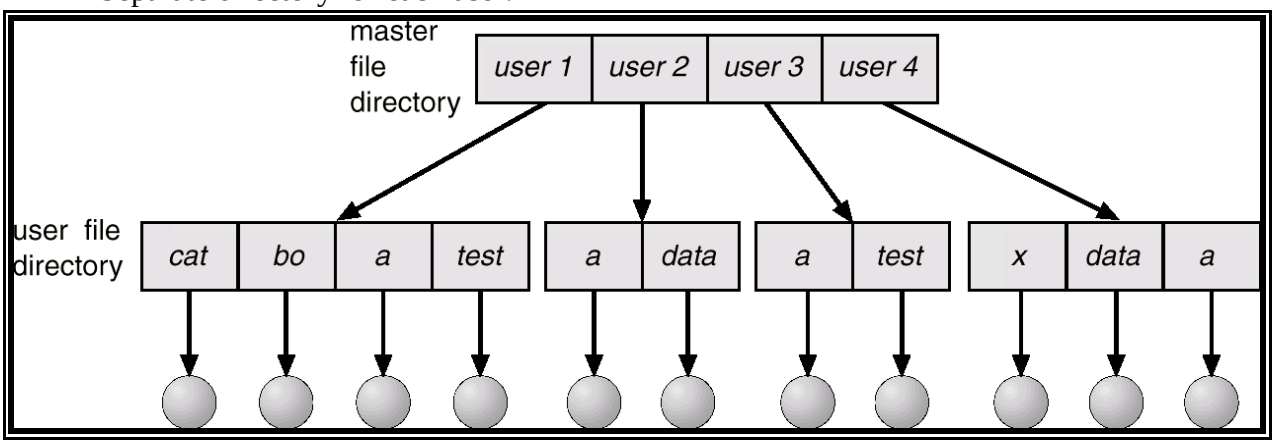
- A single directory for all users.



Naming problem
Grouping problem

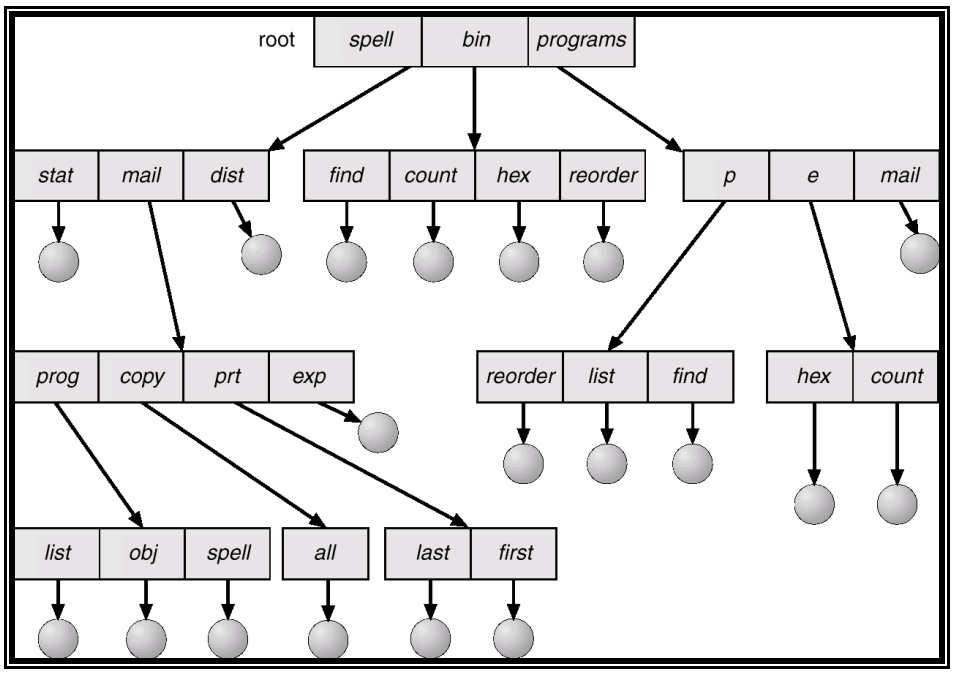
Two-Level Directory

- Separate directory for each user.



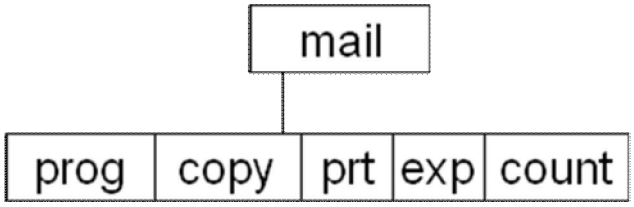
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



- Efficient searching
- Grouping Capability
- Current directory (working directory)
  - **cd** /spell/mail/prog
  - **type** list
- **Absolute** or **relative** path name

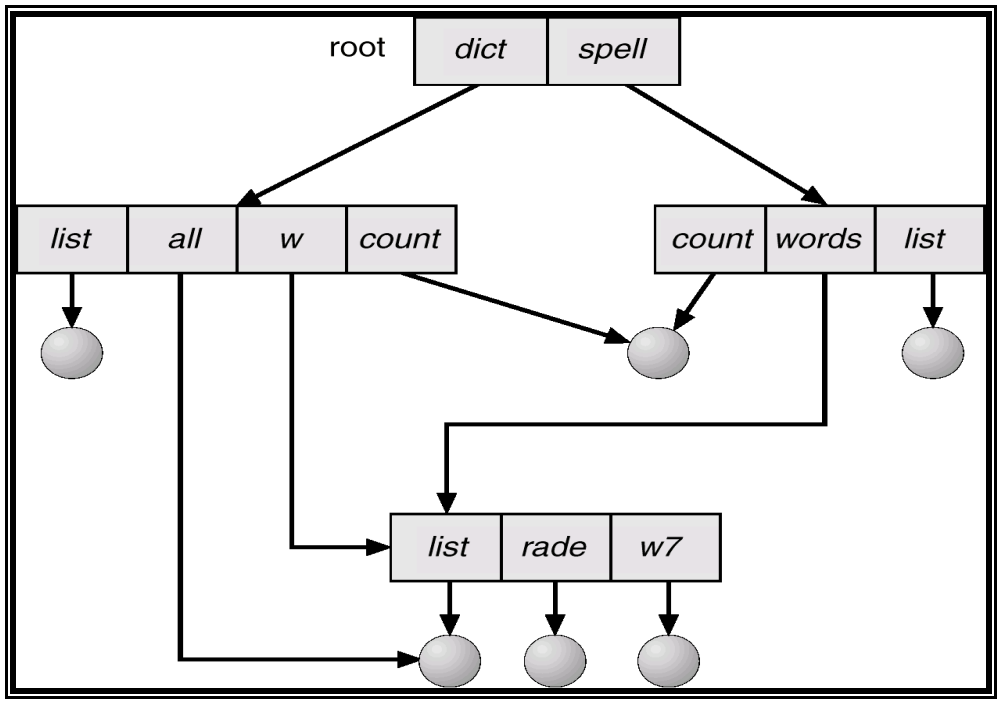
- Creating a new file is done in current directory.
- Delete a file
  rm <file-name>
- Creating a new subdirectory is done in current directory.
  mkdir <dir-name>
  Example: if in current directory **/mail**
  mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail".

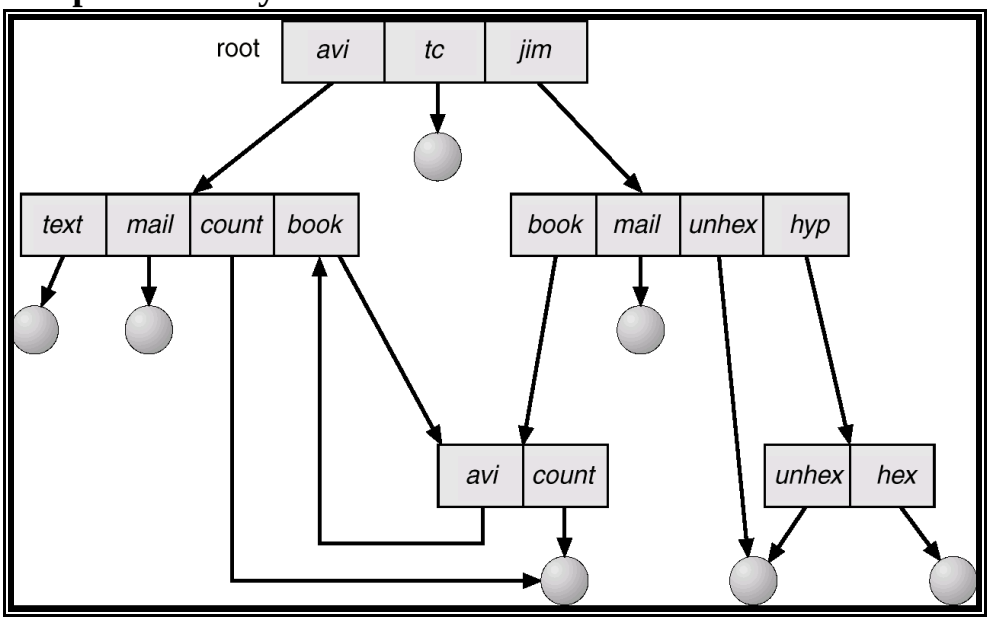
Acyclic-Graph Directories

- Have shared subdirectories and files.



- Two different names (aliasing)
- If *dict* deletes *list* ⇒ dangling pointer.
  Solutions:
  - Backpointers, so we can delete all pointers.
    Variable size records a problem.
  - Backpointers using a daisy chain organization.
  - Entry-hold-count solution.

General Graph Directory

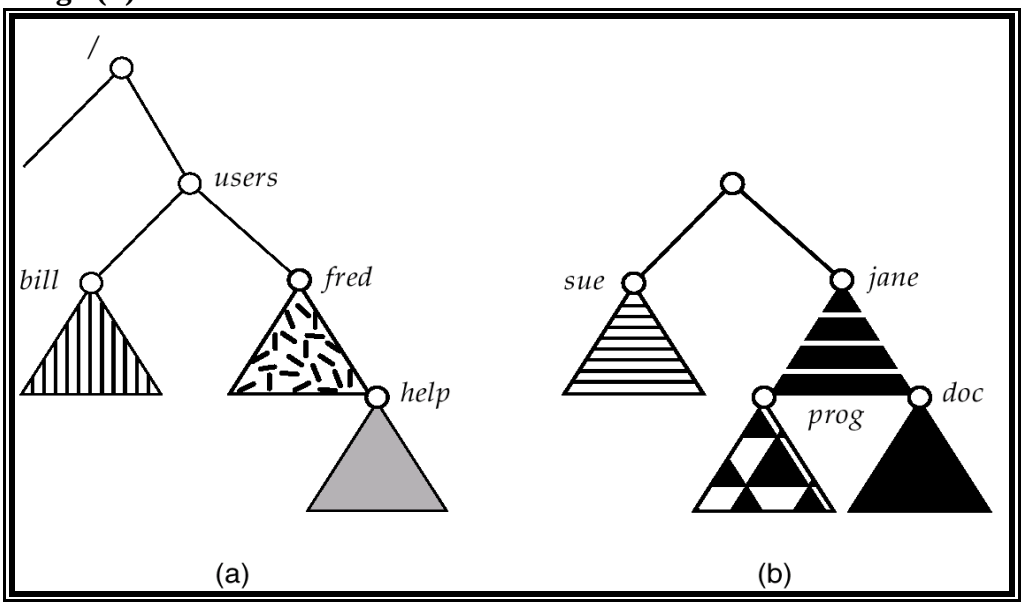


- How do we guarantee no cycles?
  - Allow only links to file not subdirectories.
  - Garbage collection.
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK.

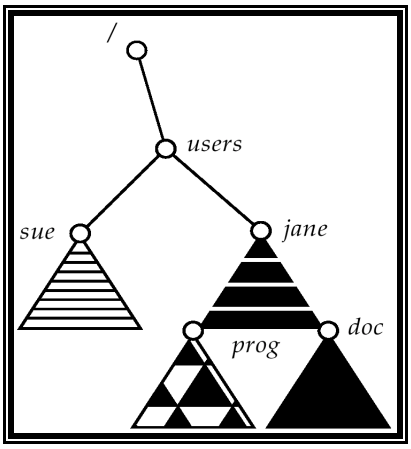
File System Mounting

- A file system must be **mounted** before it can be accessed.
- A unmounted file system (I.e (b)) is mounted at a **mount point**.

(a) Existing. (b) Unmounted Partition



Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a *protection* scheme.
- On distributed systems, files may be shared across a network.
- Network File System (NFS) is a common distributed file-sharing method.

Protection

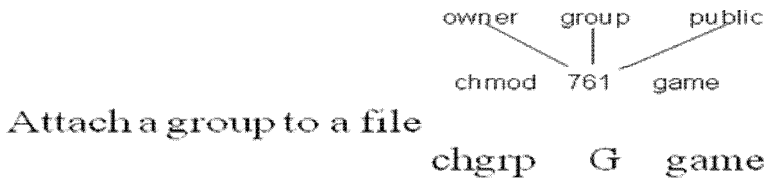
- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

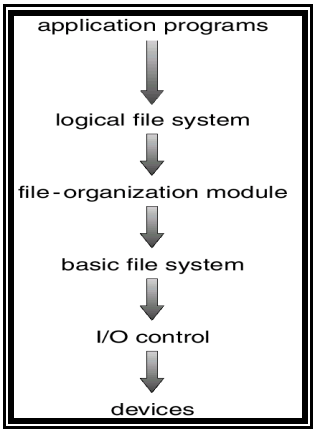


File System Implementation

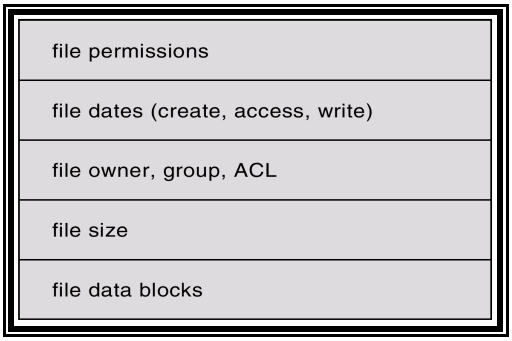
File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- *File control block* – storage structure consisting of information about a file.

Layered File System

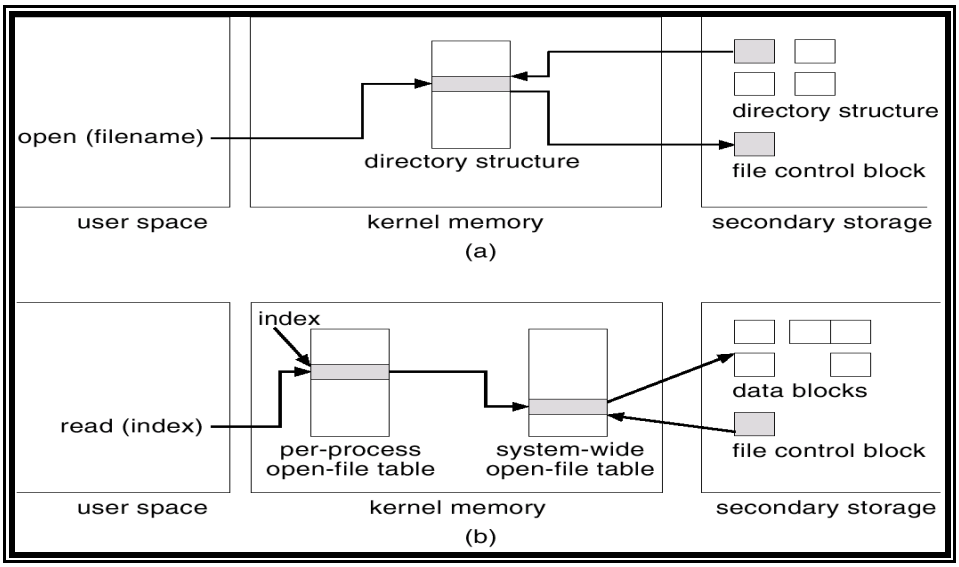


A Typical File Control Block



In-Memory File System Structures

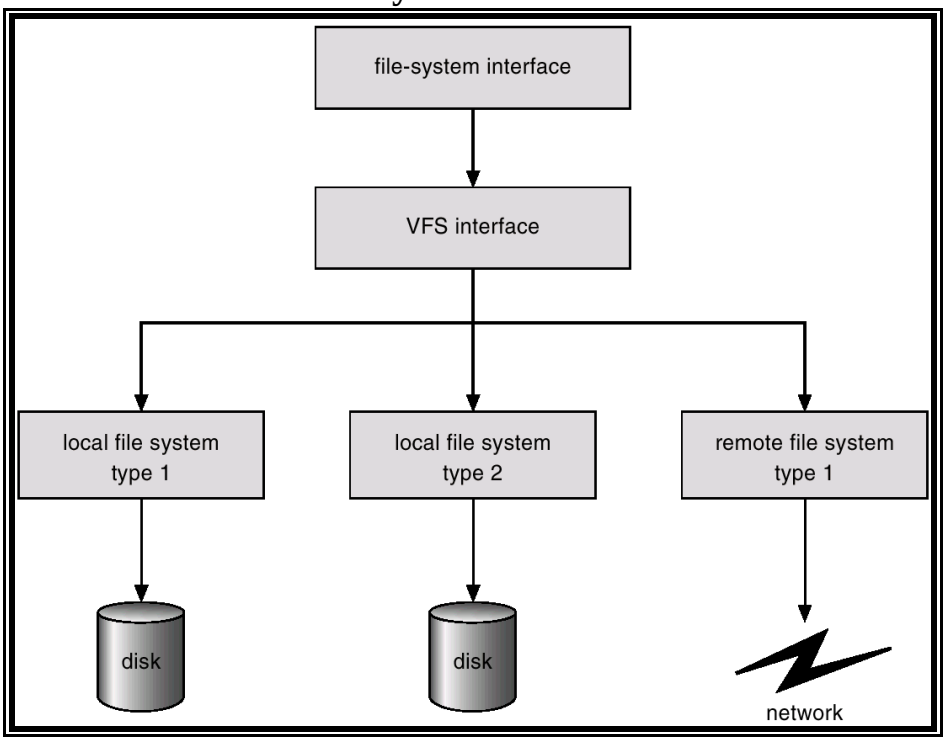
- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.



Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Directory Implementation

- Linear list of file names with pointer to the data blocks.

- simple to program
- time-consuming to execute

- Hash Table – linear list with hash data structure.
  - decreases directory search time
  - *collisions* – situations where two file names hash to the same location
  - fixed size

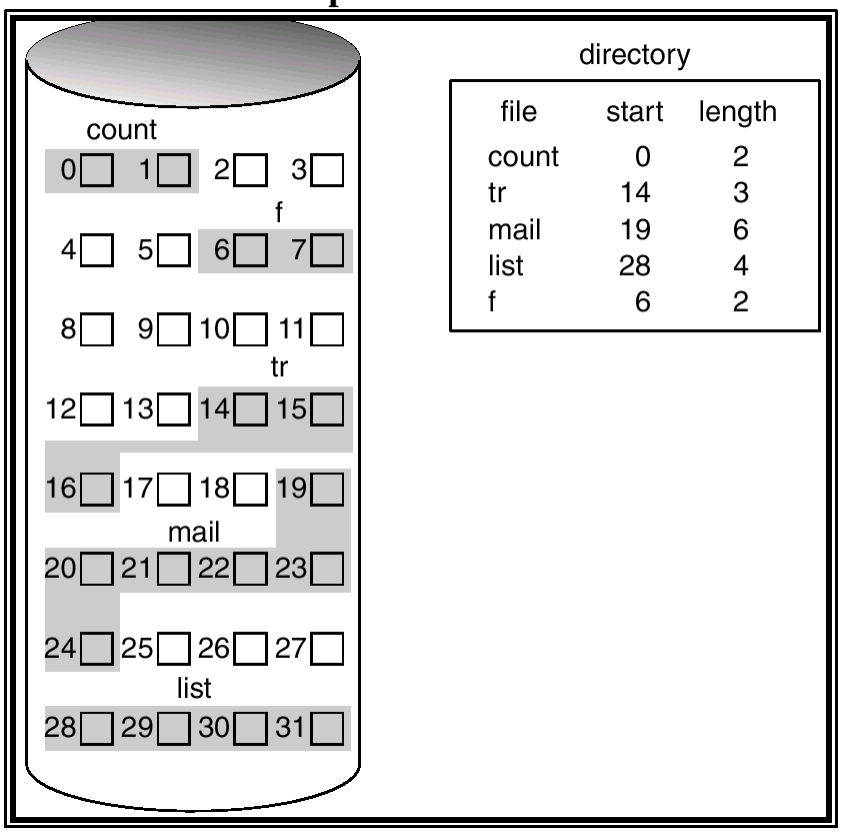
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.

Contiguous Allocation of Disk Space

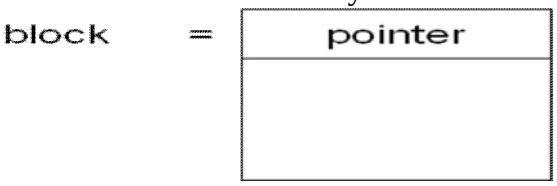


Extent-Based Systems

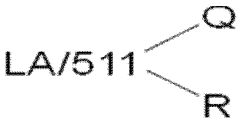
- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme.
- Extent-based file systems allocate disk blocks in **extents**.
- An **extent** is a contiguous block of disks. Extents are allocated for file allocation. A file consists of one or more extents.

Linked Allocation

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

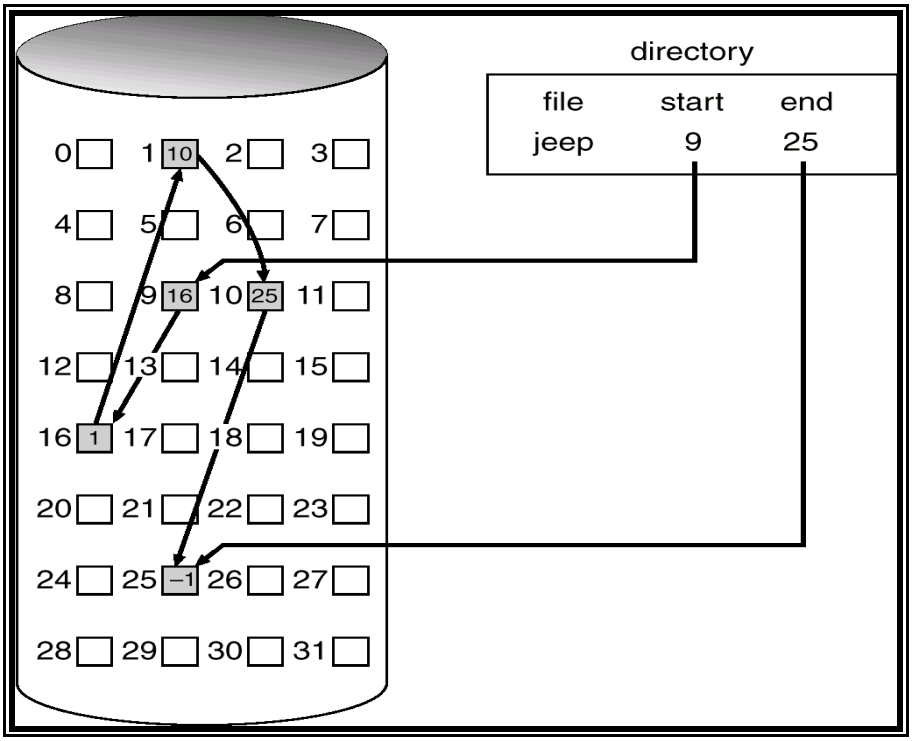


- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

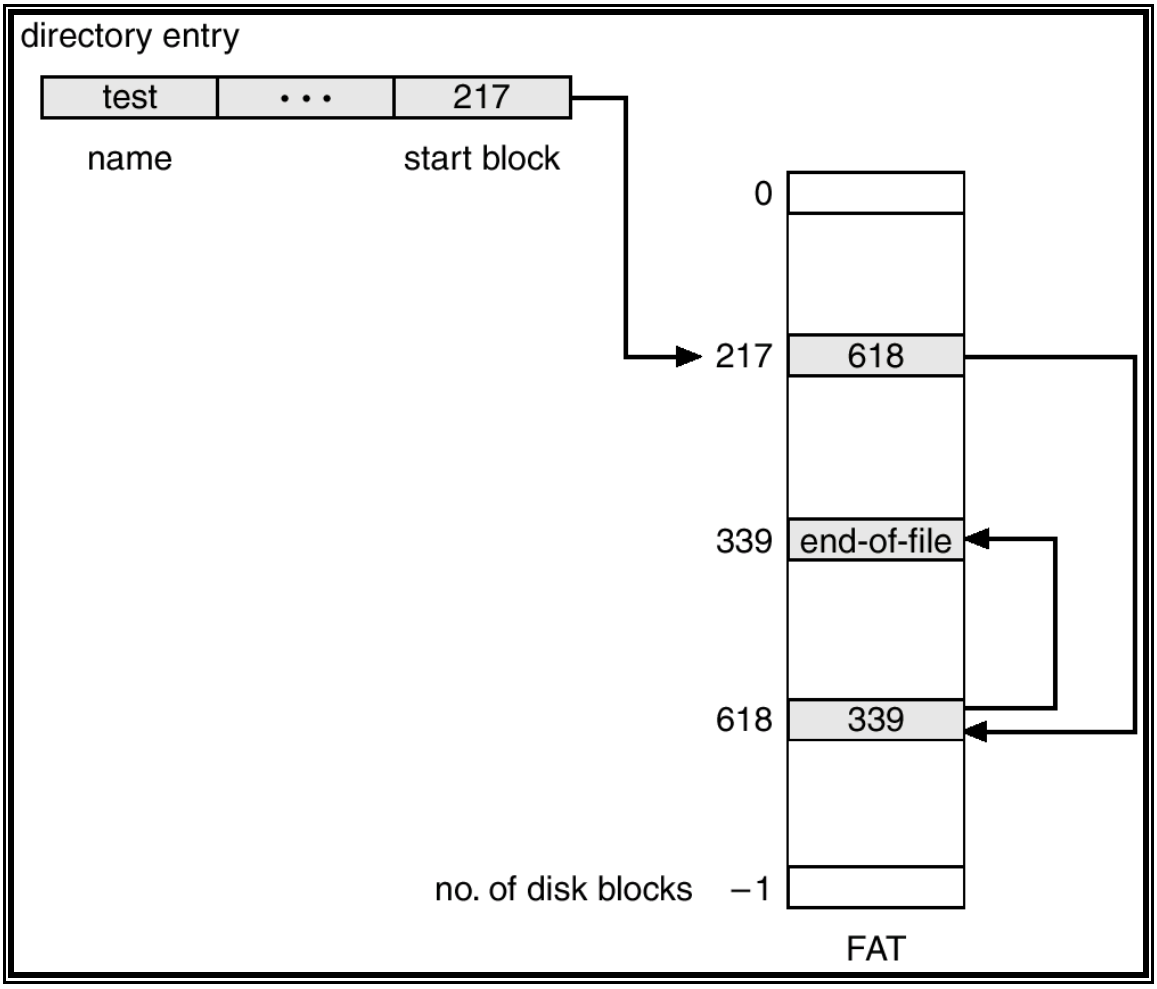


Block to be accessed is the Qth block in the linked chain of blocks representing the file.
Displacement into block = R + 1
File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation

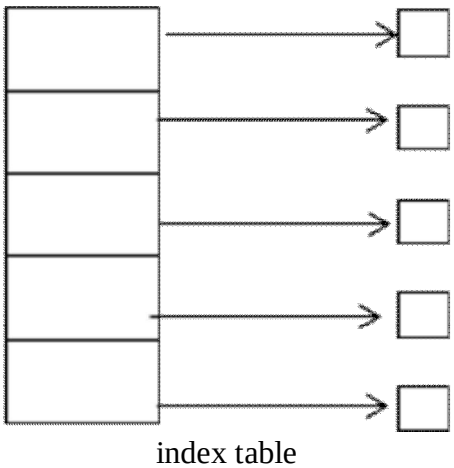


File-Allocation Table



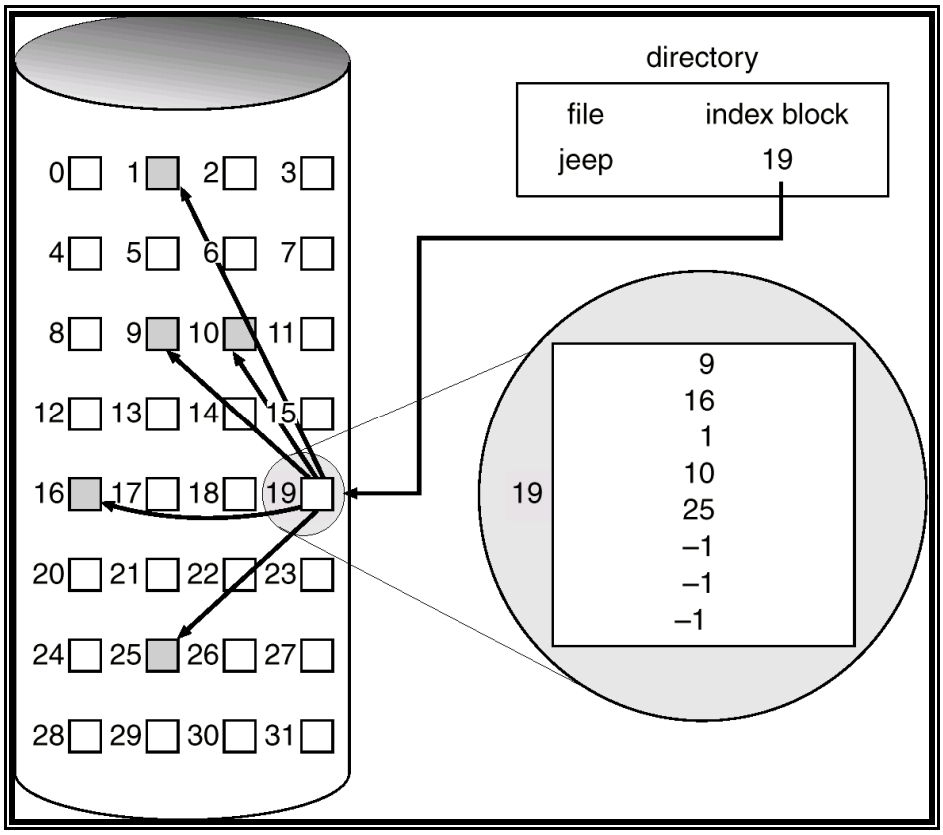
Indexed Allocation

- Brings all pointers together into the *index block.*
- Logical view.

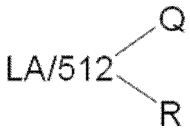


index table

Example of Indexed Allocation

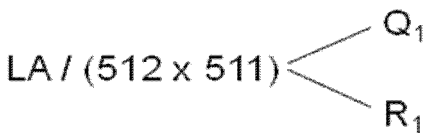


- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.



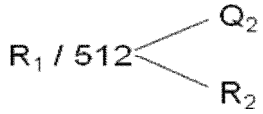
Q = displacement into index table
R = displacement into block

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).



Q_1 = block of index table

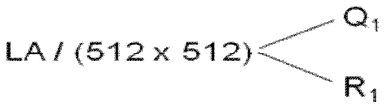
R_1 is used as follows:



Q_2 = displacement into block of index table

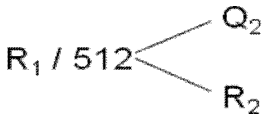
R_2 displacement into block of file:

- Two-level index (maximum file size is 512^3)

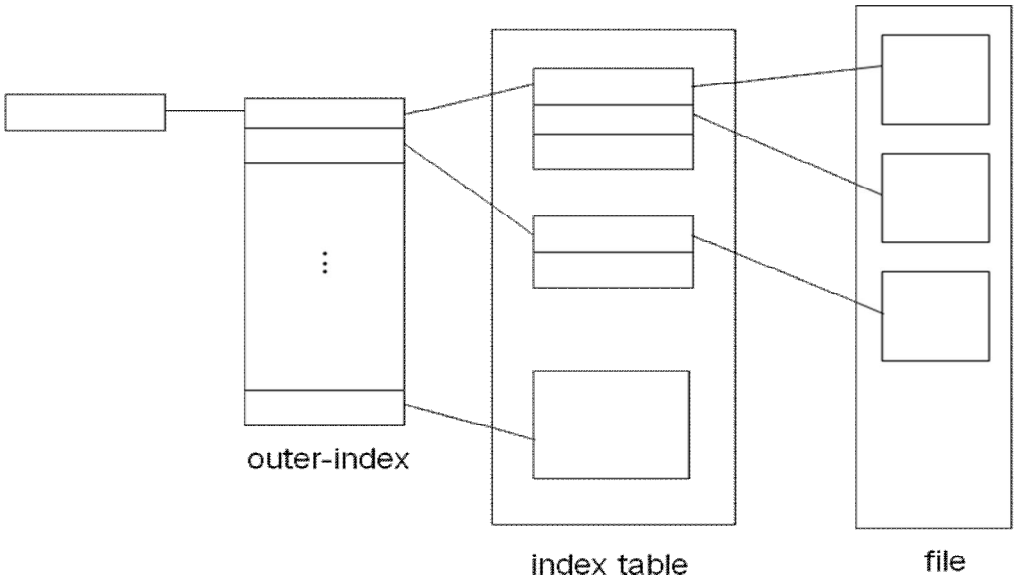


Q_1 = displacement into outer-index

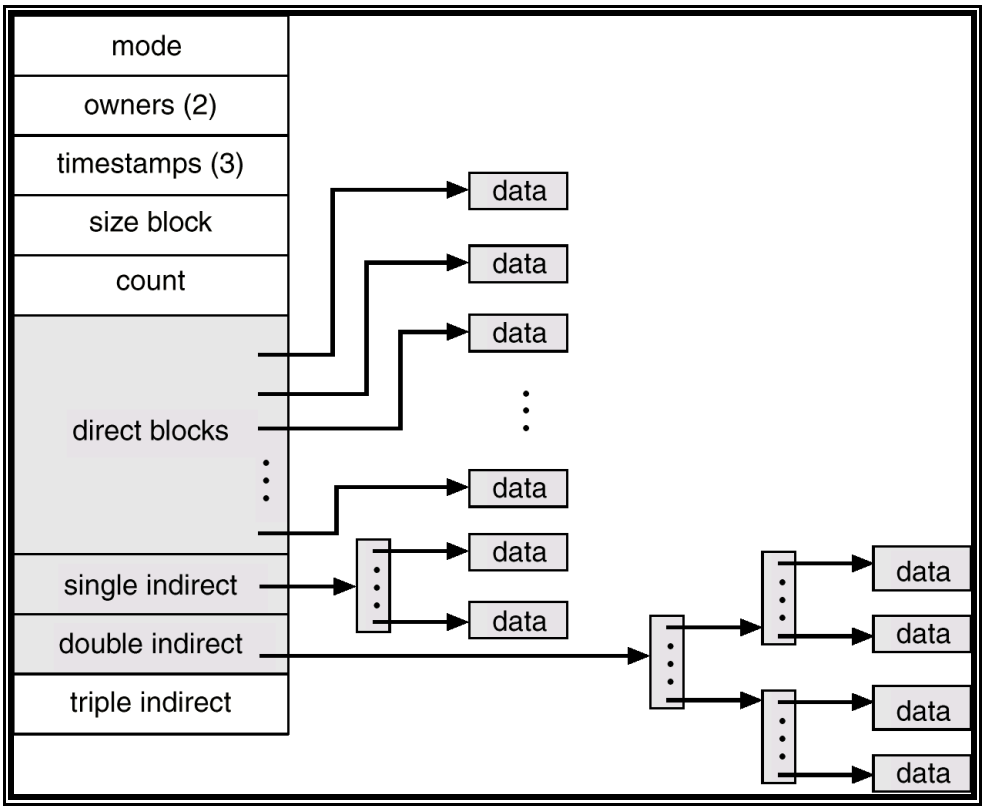
R_1 is used as follows:



Q_2 = displacement into block of index table
R_2 displacement into block of file:



Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- Bit vector (*n* blocks)

$$bit[i] = \begin{cases} 0 \Rightarrow block[i] \text{ free} \\ 1 \Rightarrow block[i] \text{ occupied} \end{cases}$$

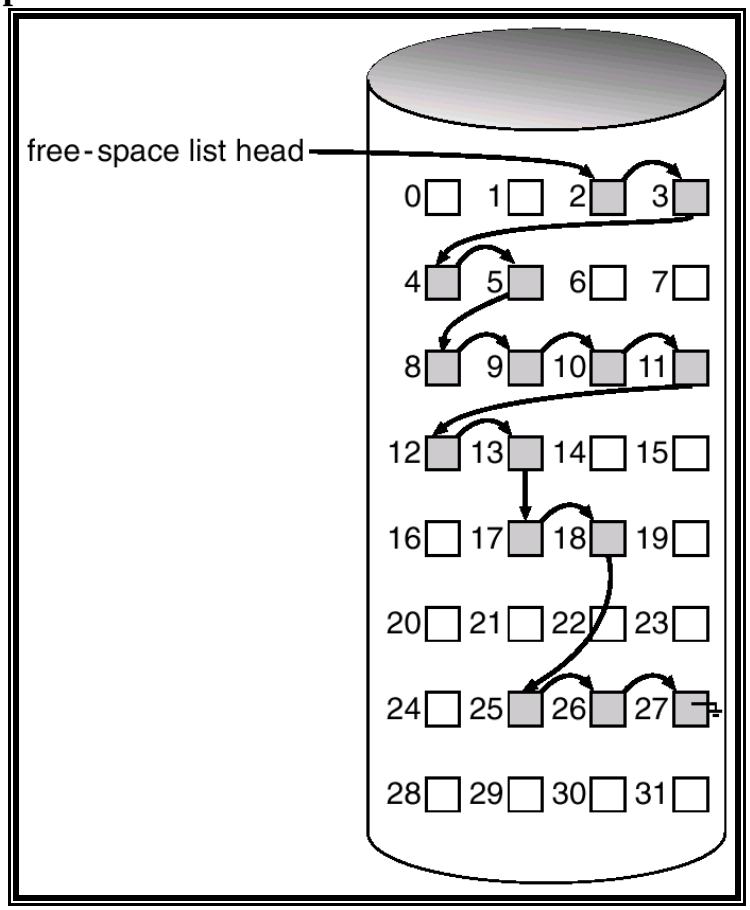
Block number calculation
(number of bits per word) *(number of 0-value words) +offset of first 1 bit

- Bit map requires extra space. Example:
  block size = 2^{12} bytes
  disk size = 2^{30} bytes (1 gigabyte)
  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
- Counting

- Need to protect:
  - Pointer to free list
  - Bit map
    - Must be kept on disk
    - Copy in memory and disk may differ.
    - Cannot allow for block[*i*] to have a situation where bit[*i*] = 1 in memory and bit[*i*] = 0 on disk.
  - Solution:
    - Set bit[*i*] = 1 in disk.
    - Allocate block[*i*]
    - Set bit[*i*] = 1 in memory

Linked Free Space List on Disk



Efficiency and Performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.