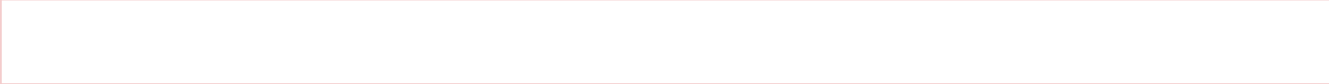


# UNIT-5

## FILE MANAGEMENT



# Syllabus:

**File system Interface-** The concept of a file, Access Methods, Directory structure, File system mounting.

**File System implementation-** File system structure, allocation methods, free-space management, Mass-storage structure, Disk scheduling.

**System Protection:** Goals of protection, Principles and domain of protection, Access Matrix, Access Control, Revocation of access rights.

## What is the file?

- ★ The file can be explained as the smallest unit of storage on a computer system.
- ★ The user can perform file operations like open, close, read, write, and modify.


## File concept

- ★ The operating system can provide a logical view of the information stored in the disks, this logical unit is known as a file.
- ★ The information stored in files is not lost during power failures.
- ★ A file helps to write data on the computer. It is a sequence of bits, bytes, or records, the structure of which is defined by the owner and depends on the type of the file.

**A file's attributes** vary from one operating system to another but typically consist of these:

- **Name:** Name is the symbolic file name and is the only information kept in human readable form.
- **Identifier:** This unique tag is a number that identifies the file within the file system; it is in non-human-readable form of the file.
- **Type:** This information is needed for systems which support different types of files or its format.
- **Location:** This information is a pointer to a device which points to the location of the file on the device where it is stored.
- **Size:** The current size of the file (which is in bytes, words, etc.) which possibly the maximum allowed size gets included in this attribute.
- **Protection:** Access-control information establishes who can do the reading, writing, executing, etc.
- **Date, Time & user identification:** This information might be kept for the creation of the file, its last modification and last used. These data might be useful for in the field of protection, security, and monitoring its usage.

# File Access Methods in Operating System

A diagram consisting of a red L-shaped line that originates from the bottom-left corner of the title bar and points towards the list of file access methods.

- ★ **Sequential-Access**

- ★ **Direct Access**

- ★ **Index sequential Method**

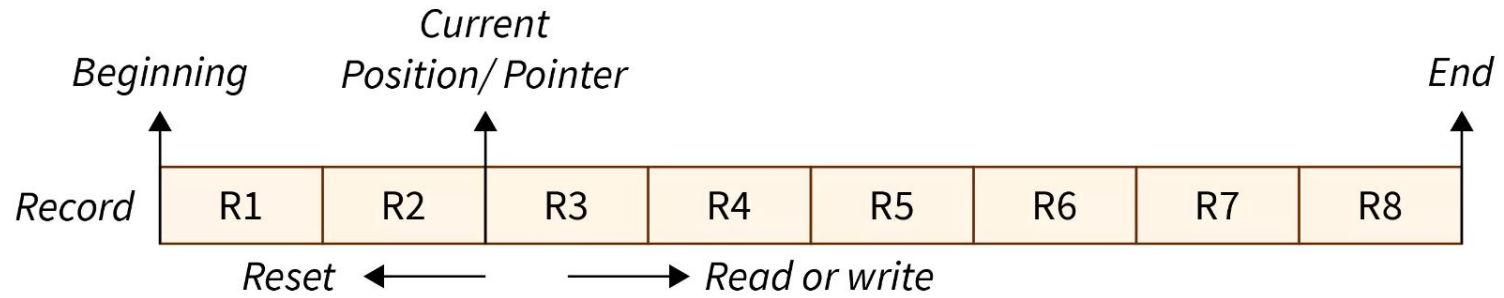
- A file is a collection of bits/bytes or lines which is stored on secondary storage devices like a hard drive (magnetic disks).
- File access methods in OS are nothing but techniques to read data from the system's memory.

There are various ways in which we can access the files from the memory like:

- Sequential Access
- Direct/Relative Access, and
- Indexed Sequential Access.

## Sequential Access

- The operating system reads the file word by word in sequential access method of file accessing.
- A pointer is made, which first links to the file's base address.
- If the user wishes to read the first word of the file, the pointer gives it to them and raises its value to the next word. This procedure continues till the file is finished.
- It is the most basic way of file access. The data in the file is evaluated in the order that it appears in the file and that is why it is easy and simple to access a file's data using sequential access mechanism.
- For example, editors and compilers frequently use this method to check the validity of the code.

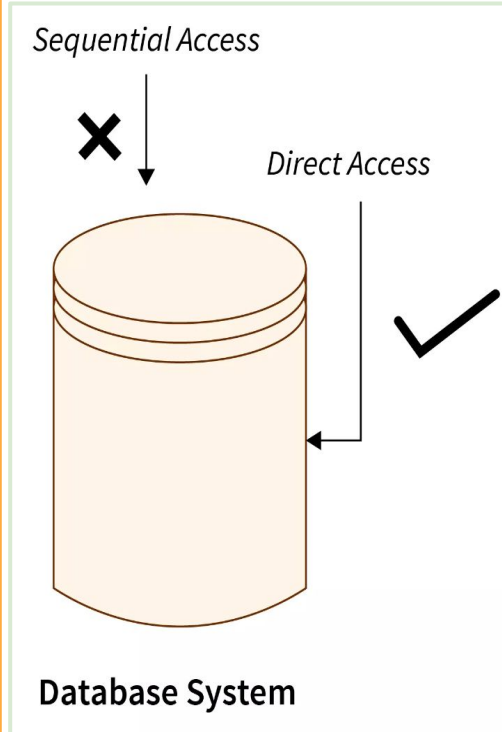


**Sequential Access Mechanism**



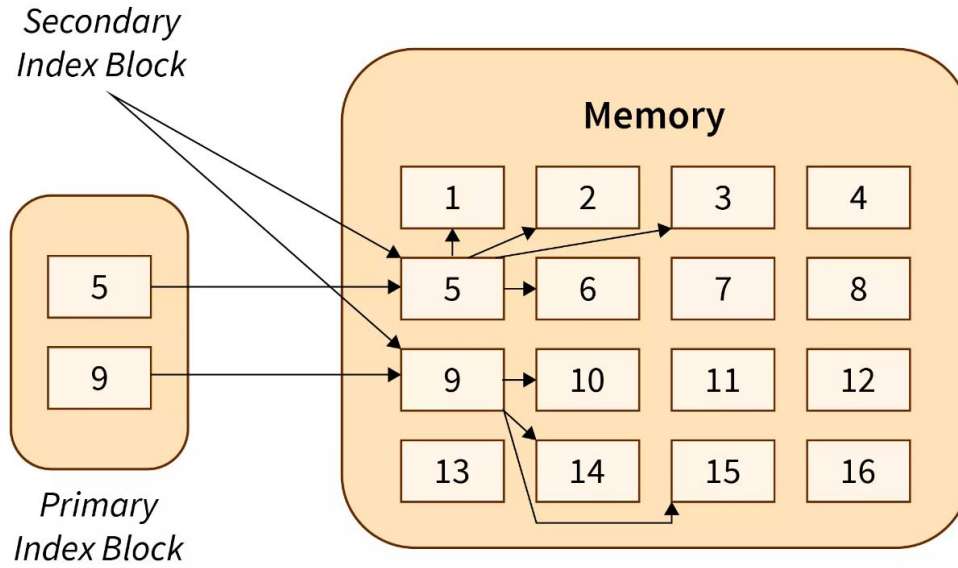
## Direct (or Relative) Access

- A Direct/Relative file access mechanism is mostly required with the database systems.
- In the majority of the circumstances, **we require filtered/specific data from the database**, and in such circumstances, sequential access might be highly inefficient.
- Assume that each block of storage holds four records and that the record we want to access is stored in the **tenth block**.
- In such a situation, sequential access will not be used since it will have to traverse all of the blocks to get to the required record, while direct access will allow us to access the required record instantly.
- The direct access mechanism requires the OS to perform some additional tasks but eventually leads to much faster retrieval of records as compared to the sequential access.



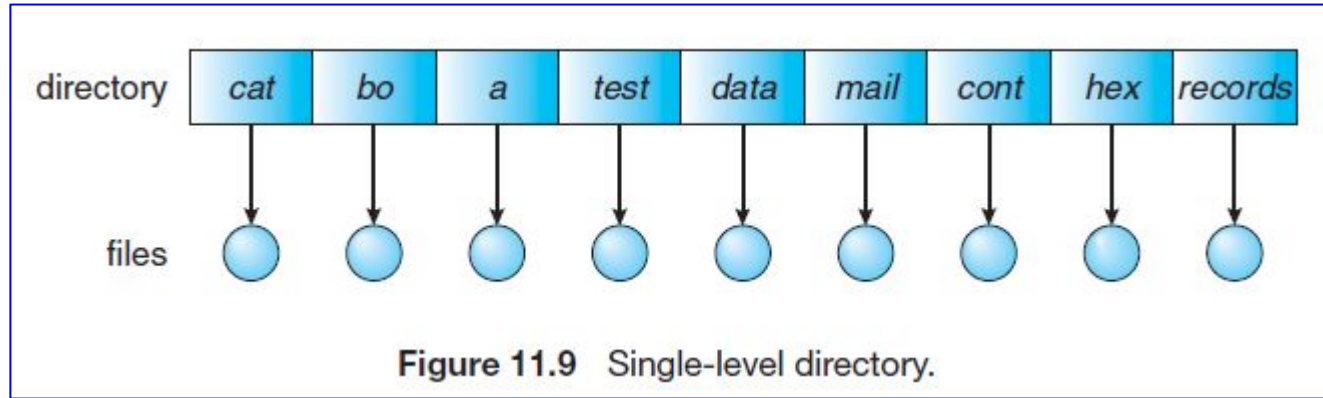
## Indexed Sequential Access

- It's the other approach to access a file that's constructed on top of the sequential access mechanism.
- This method is practically similar to the pointer to pointer concept in which we store an address of a pointer variable containing address of some other variable/record in another pointer variable.
- The indexes, similar to a book's index (pointers), contain a link to various blocks present in the memory. To locate a record in the file, we first search the indexes and then use the pointer to pointer concept to navigate to the required file.
- Primary index blocks contain the links of the secondary inner blocks which contains links to the data in the memory.



**Indexed Sequential Access Of File**

## Single-Level Directory:-



- The single-level directory is the simplest directory structure. In it, all files are contained in the same directory which makes it easy to support and understand.
- A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user.
- Since all the files are in the same directory, they must have a unique name. if two users call their dataset test, then the unique name rule violated.

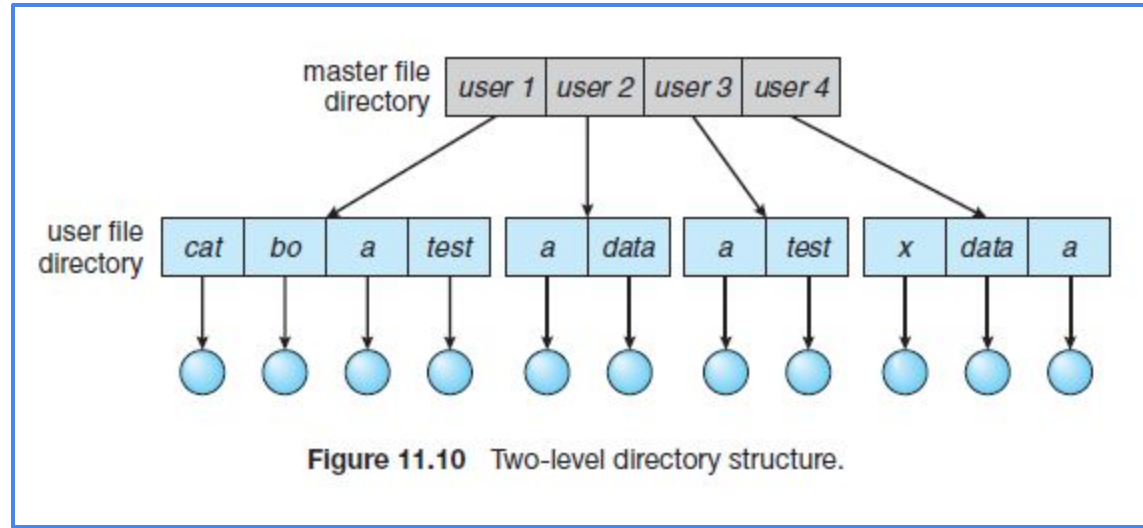
## **Advantages**

- Implementation is very simple.
- If the sizes of the files are very small then the searching becomes faster.
- File creation, searching, deletion is very simple since we have only one directory.

## **Disadvantages:**

- There may chance of name collision because two files can have the same name.
- Searching will become time taking if the directory is large.
- This can not group the same type of files together

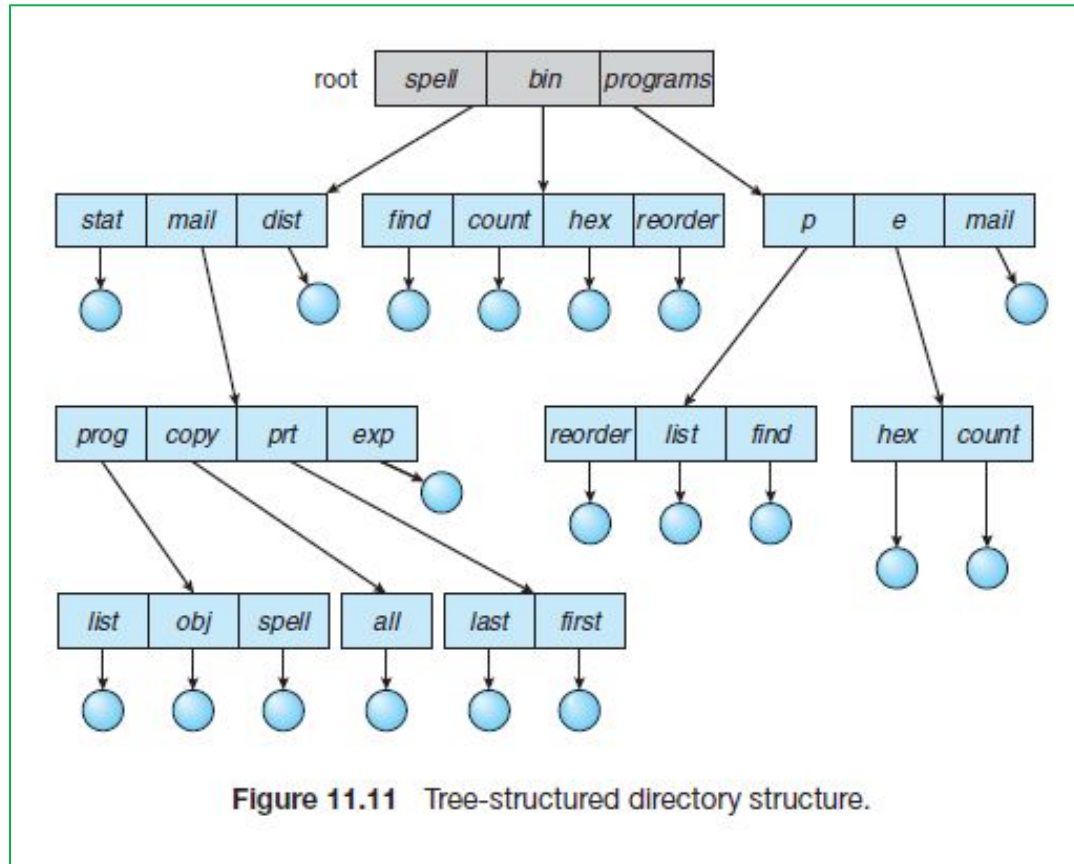
## Two-Level Directory:-



- A single-level directory often leads to confusion of file names among different users.
- **The standard solution is to create a separate directory for each user.**
- In the two-level directory structure, each user has his own user file directory (UFD).
- The UFDs have similar structures, but each lists only the files of a single user.
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched.
- The MFD is indexed by user name or account number, and each entry points to the UFD for that user.

- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
- To create a file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists.
- To delete a file, the operating system confines its search to the local UFD; Thus, it cannot accidentally delete another user's file that has the same name.

## Tree-Structured Directories:-



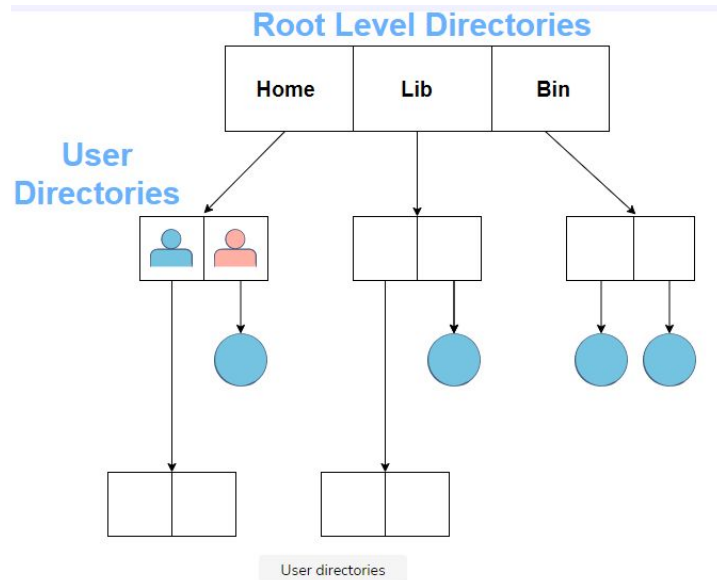


- Once we have seen how to view a two-level directory as a two-level tree, the natural generalization is to extend the directory structure to a tree of arbitrary height (Figure 11.11).
- This generalization allows users to create their own subdirectories and to organize their files accordingly.
- A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories.
- A directory is simply another file, but it is treated in a special way.
- All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.

- ➔ In normal use, each process has a current directory. The current directory should contain most of the files that are of current interest to the process.
- ➔ When reference is made to a file, the current directory is searched.
- ➔ If a file is needed that is not in the current directory, then the user usually must either specify a path name or change the current directory to be the directory holding that file.
- ➔ To change directories, a system call is provided that takes a directory name as a parameter and uses it to redefine the current directory.

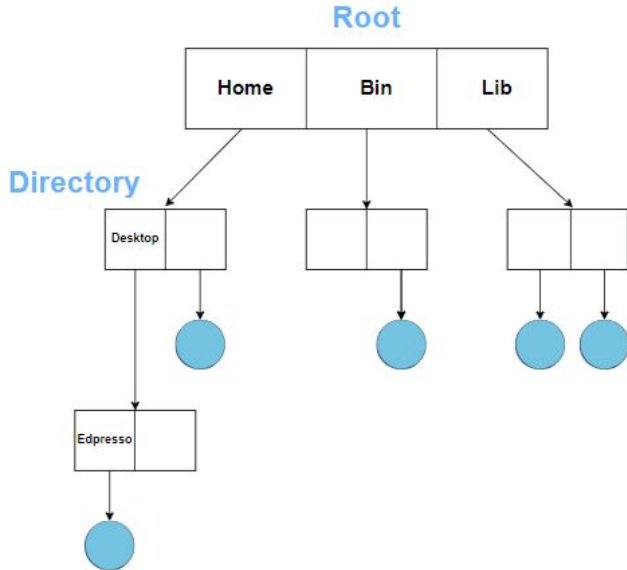
## Users

- Each user has their own directory and cannot access the directory of another user.
- All users can read data from the root directory.
- However, they cannot write to or modify the root directory.
- This privilege is reserved for the system administrator who has complete access to the root directory.



## Paths

- We can access a file using two types of paths. These are:
  - Absolute path
  - Relative path
- The absolute path is the path to the file with respect to the root directory.
- The relative path is the path to the file with respect to the current working directory.



Consider the above directory tree. Let's assume that the current working directory is: `root/home/Desktop`.

To access the file under the directory labeled *Edpresso*, we can use the following paths:

- Relative path: `Edpresso`
- Absolute path: `root/home/Desktop/Edpresso`

# File System Mounting

- In operating systems, file system mounting refers to the process of attaching a file system to a specific directory in the file hierarchy.
- When a file system is mounted, it becomes available for reading from and writing to, allowing users to access the files and directories contained within it.
- The specific method of mounting a file system depends on the operating system being used.

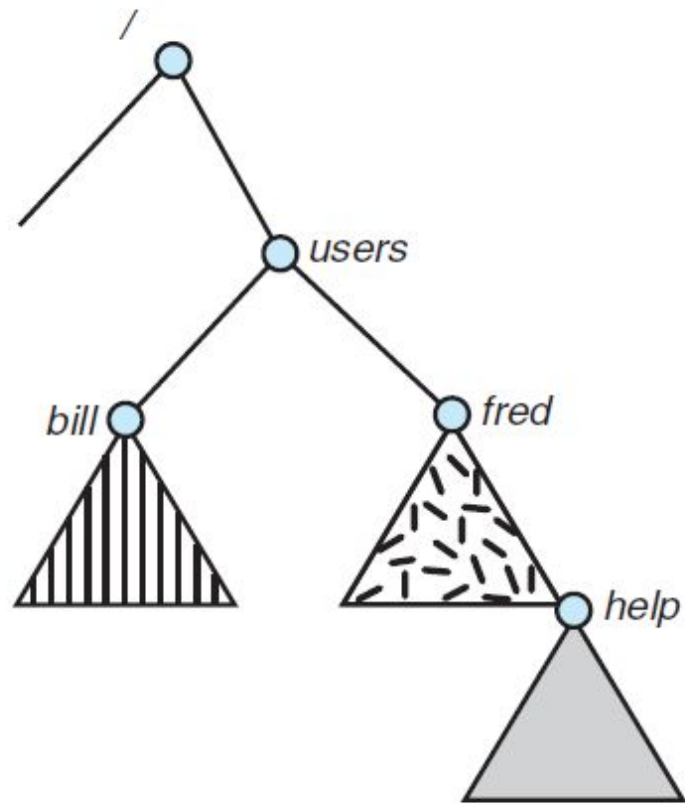
**The mount procedure is straightforward.**

- The operating system is given the name of the device and the mount point (Mount Point : The location within the file structure where the file system is to be attached) .
- Some operating systems require that a file system type be provided, while others inspect the structures of the device and determine the type of file system.
- Typically, a mount point is an empty directory.
- For instance, on a UNIX system, a file system containing a user's home directories might be mounted as /home;
- Then, to access the directory structure within that file system, we could precede the directory names with /home, as in /home/jane.
- Mounting that file system under /users would result in the path name /users/jane, which we could use to reach the same directory.

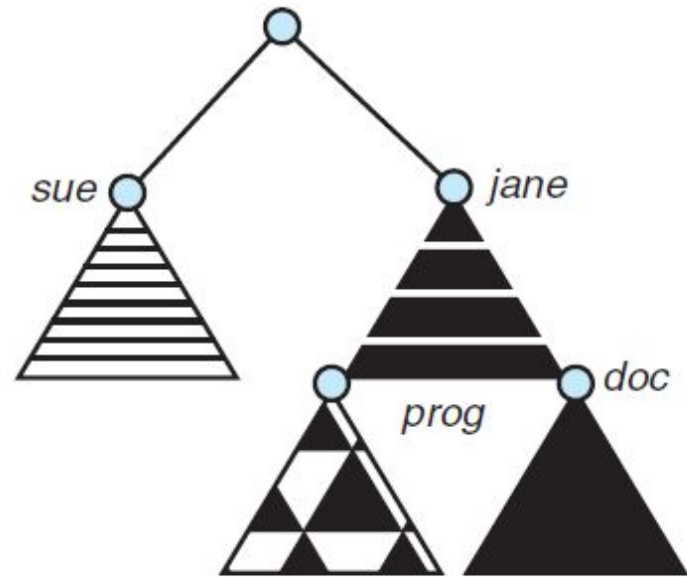
- Next, the operating system verifies that the device contains a valid file system.
- It does so by asking the device driver to read the device directory and verifying that the directory has the expected format.
- Finally, the operating system notes in its directory structure that a file system is mounted at the specified mount point.
- This scheme enables the operating system to traverse its directory structure, switching among file systems, and even file systems of varying types, as appropriate.

#### Example:

- To illustrate file mounting, consider the file system depicted in Figure 11.14, where the triangles represent subtrees of directories that are of interest.
- Figure 11.14(a) shows an existing file system, while Figure 11.14(b) shows an unmounted volume residing on /device/dsk.
- At this point, only the files on the existing file system can be accessed. Figure 11.15 shows the effects of mounting the volume residing on /device/dsk over /users.
- If the volume is unmounted, the file system is restored to the situation depicted in Figure 11.14.



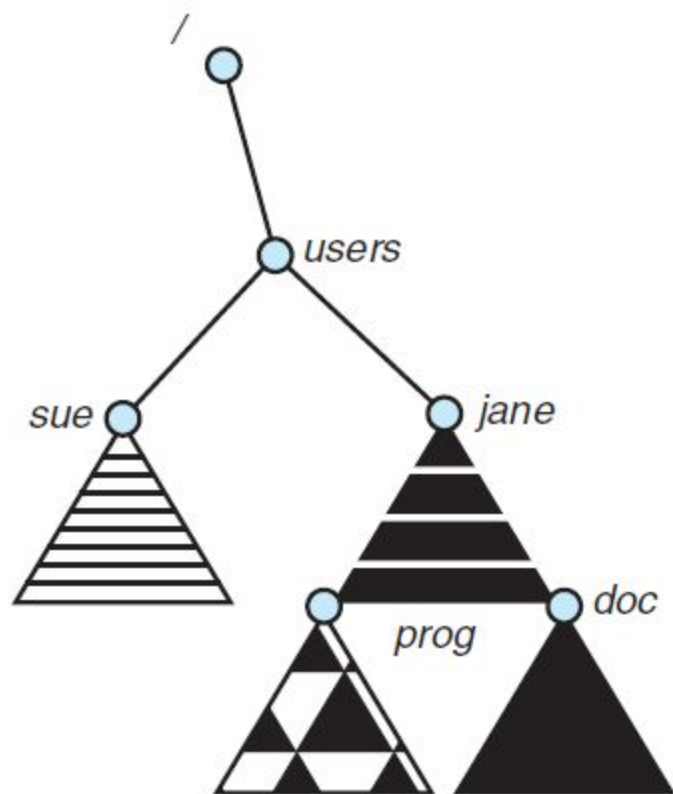
(a)



(b)

**Figure 11.14** File system. (a) Existing system. (b) Unmounted volume.





**Figure 11.15** Mount point.

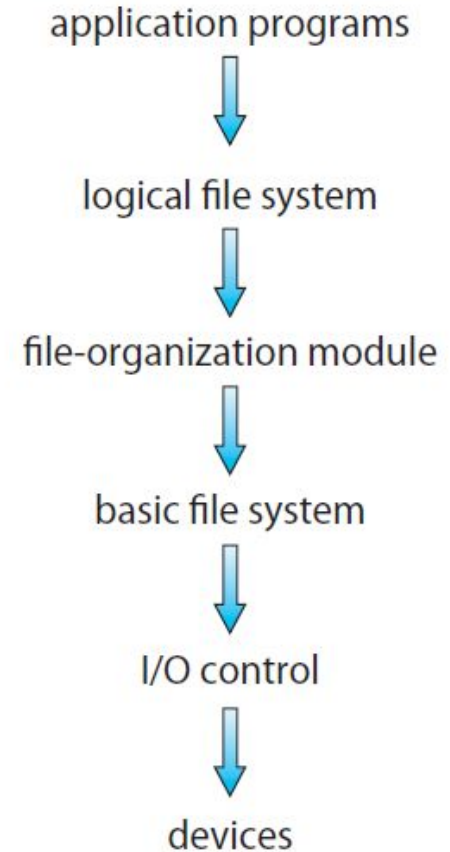
## **File system Implementation**

### **File system structure**

- ➔ Disks provide most of the secondary storage on which file systems are maintained. Two characteristics make them convenient for this purpose:
  1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same place.
  2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires only moving the read–write heads and waiting for the disk to rotate.
- ➔ **File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.**
- ➔ The file system itself is generally composed of many different levels.
- ➔ The structure shown in Figure 12.1(Next Slide) is an example of **a layered design.**
- ➔ **Each level in the design uses the features of lower levels to create new features for use by higher levels.**

## I/O Control layer:

- The I/O control level consists of **device drivers and interrupt handlers** to transfer information between the main memory and the disk system.
- A **device driver** can be thought of as a **translator**.
- Its input consists of high level commands such as “retrieve block 123.”, Its output consists of low-level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O device to the rest of the system.



**Figure 12.1** Layered file system.

### **Basic file system:**

- It Issues general commands to device driver to read and write physical blocks on disk.
- It manages the memory buffers and caches.
- A block in buffer can hold the contents of the disk block and cache stores frequently used file system metadata.

### **File organization Module:**

- It has information about files, location of files and their logical and physical blocks.
- Physical blocks do not match with logical numbers of logical block numbered from 0 to N.
- It also has a free space which tracks unallocated blocks.

### **Logical file system :**

- It manages metadata information about a file i.e includes all details about a file except the actual contents of file.
- It maintains file structure via file-control blocks
- File control block (FCB) has information about a file – owner, size, permissions, location of file contents.

# File allocation methods

- ★ The allocation methods define **how the files are stored in the disk blocks**.
- ★ There are three main disk space or file allocation methods.

- ★ Contiguous Allocation
- ★ Linked Allocation
- ★ Indexed Allocation

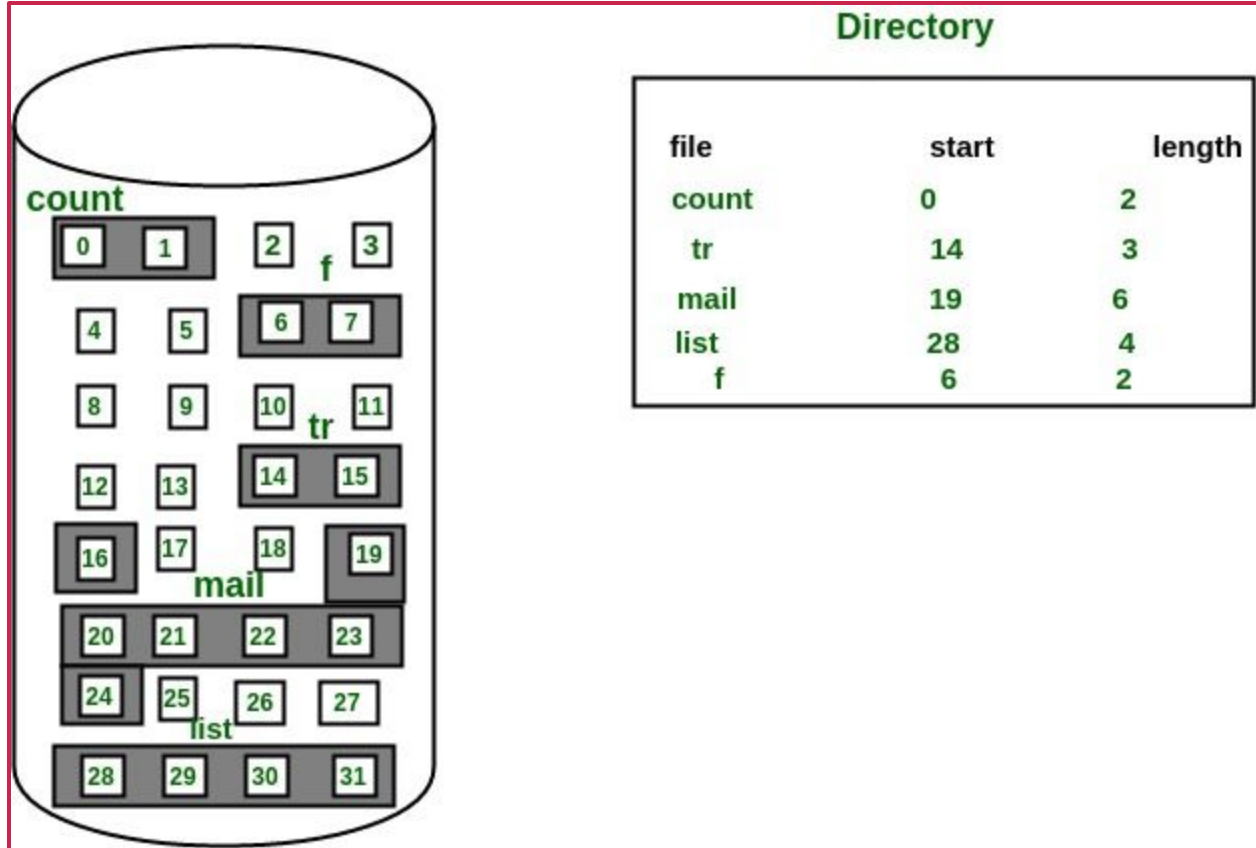
The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

## Contiguous Allocation

- In this scheme, each file occupies a contiguous set of blocks on the disk.
- For example, if a file requires  $n$  blocks and is given a block  $b$  as the starting location, then the blocks assigned to the file will be:  $b, b+1, b+2, \dots, b+n-1$ .
- This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
- The directory entry for a file with contiguous allocation contains
  - Address of starting block
  - Length of the allocated portion.
- The file 'mail' in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.

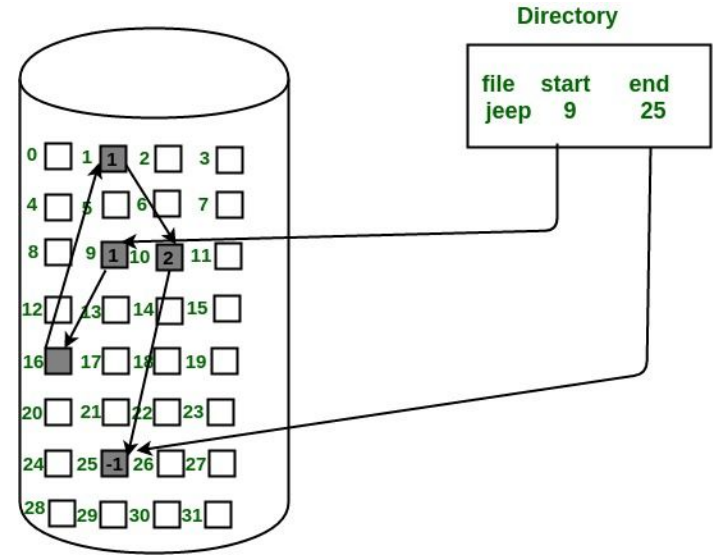
# Contiguous Allocation





## Linked Allocation:

- In this scheme, each file is a linked list of disk blocks which need not be contiguous.
- The disk blocks can be scattered anywhere on the disk.
- The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.
- The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.

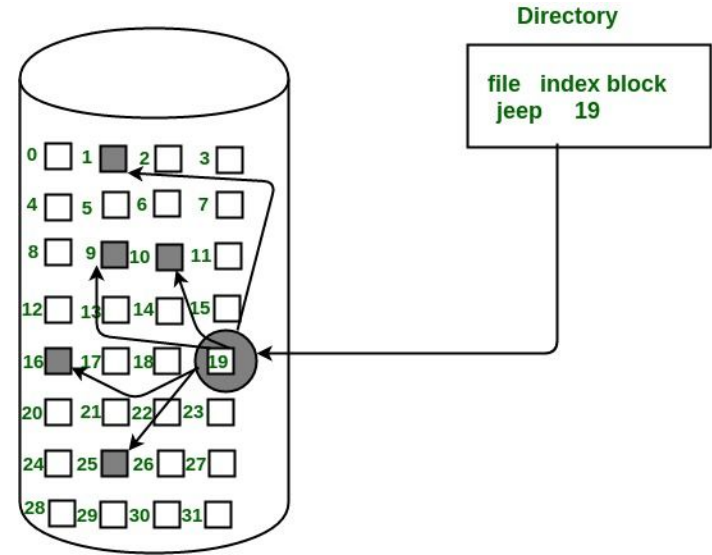


## Indexed Allocation

- In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Each file has its own index block.
- The  $i$ th entry in the index block contains the disk address of the  $i$ th file block. The directory entry contains the address of the index block as shown in the image.

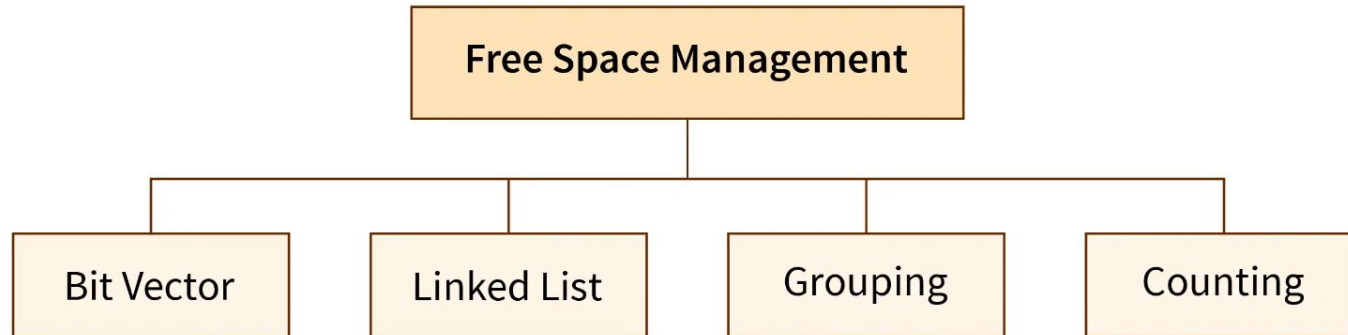
### Advantages:

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.



# Free-space management

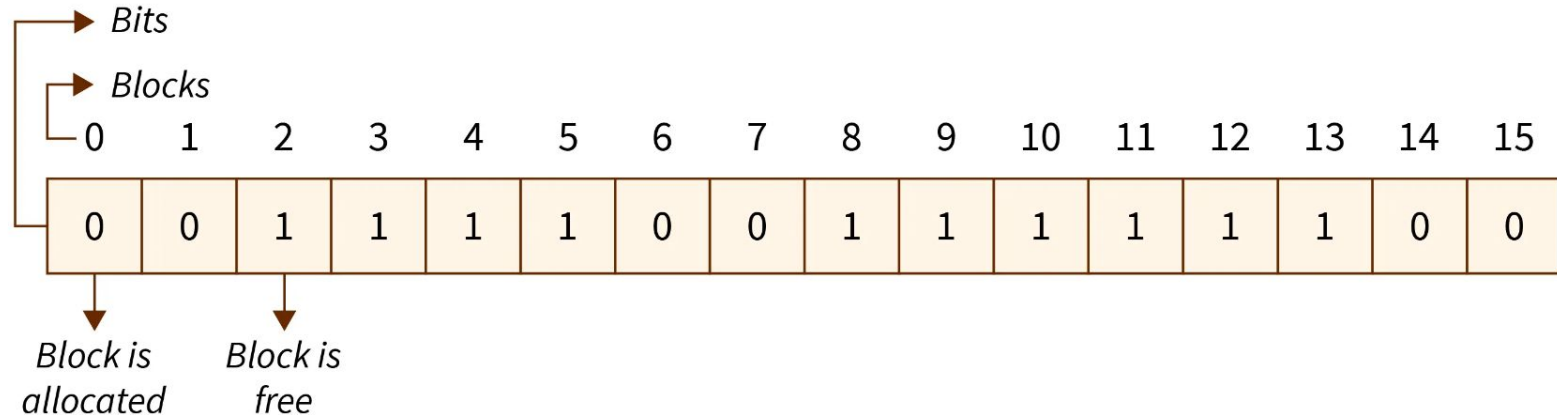
- ★ Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- ★ To keep track of free disk space, the system maintains a **free-space list**.
- ★ **The free-space list** records all free disk blocks ,those not allocated to some file or directory.
- ★ **To create a file**, we search the free-space list for the required amount of space and allocate that space to the new file.This space is then removed from the free-space list.
- ★ **When a file is deleted**, its disk space is added to the free-space list.



## Bit Vector or Bit Map:

- ★ Frequently, the free-space list is implemented as a bit map or bit vector.
- ★ Each block is represented by 1 bit.
  - If the block is **free**, the bit is **1**.
  - If the block is **allocated**, the bit is **0**.

For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, and 13 are free and the rest of the blocks are allocated. The free-space bit map would be



- We can find the free block number from the bit vector using the following method-

$$\text{Block number} = (\text{Number of bits per word}) * (\text{Number of 0-value words}) + (\text{Offset of first bit})$$

- We will now find the first free block number in the above example.
- The first group of **8 bits (00111100)** constitutes a non-zero word since all bits are not 0.
- After finding the non-zero word, we will look for the first 1 bit. This is the **third character** of the non-zero word. Hence, **offset = 3**.

Therefore, the first free block number =  $8 * 0 + 3 = 3$ .

## Advantages

- It is simple to understand.
- It is an efficient method.
- It occupies less memory.

## Linked List:

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- Blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 were free and the rest of the blocks were allocated.
- In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.

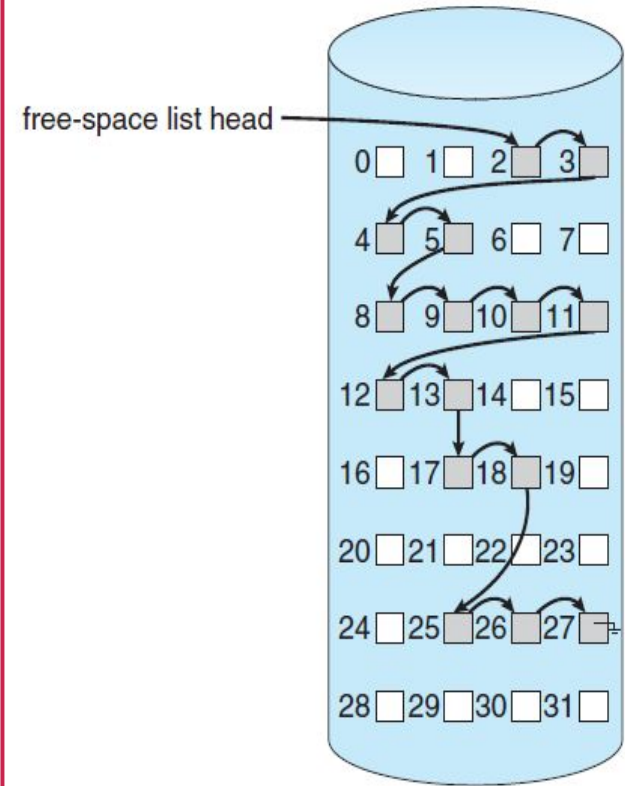


Figure 12.10 Linked free-space list on disk.

## Grouping:

- This method is the modification of the linked list method.
- In this method, **the first free block stores** the addresses of the **n** free blocks.
- The first n-1 of these blocks is free. The last block in these n free blocks contains the addresses of the next n free blocks, and so on.
- For example, consider a disk having 16 blocks where block numbers **3, 4, 5, 6, 9, 10, 11, 12, 13, and 14 are free**, and the rest of the blocks **1, 2, 7, 8, 15 and 16 are allocated to some files**.
- If we apply the Grouping method considering **n to be 3**, **Block 3 will store the addresses of Block 4, Block 5, and Block 6**.
- **Block 6 will store the addresses of Block 9, Block 10, and Block 11**.
- **Block 11 will store the addresses of Block 12, Block 13, and Block 14**.

Block 3 -> 4, 5, 6  
Block 6 -> 9, 10, 11  
Block 11 -> 12, 13, 14



## Counting:

- This method is also a modification of the linked list method.
- This method takes advantage of the fact that several contiguous blocks may be allocated or freed simultaneously.
- In this method, a linked list is maintained but in addition to the pointer to the next free block, a count of free contiguous blocks that follow the first block is also maintained.
- Thus each free block in the disk will contain two things-
  - A pointer to the next free block.
  - The number of free contiguous blocks following it.

## Counting:

**For example:**

Consider a disk having 16 blocks where block numbers 3, 4, 5, 6, 9, 10, 11, 12, 13, and 14 are **free**, and the rest of the blocks, i.e., block numbers 1, 2, 7, 8, 15 and 16 are **allocated** to some files.

- If we apply the counting method, Block 3 will point to Block 4 and store the count 4 (since Block 3, 4, 5, and 6 are contiguous).
- Similarly, Block 9 will point to Block 10 and keep the count of 6 (since Block 9, 10, 11, 12, 13, and 14 are contiguous).

Block 3 -> 4  
Block 9 -> 6

# Mass Storage Structure

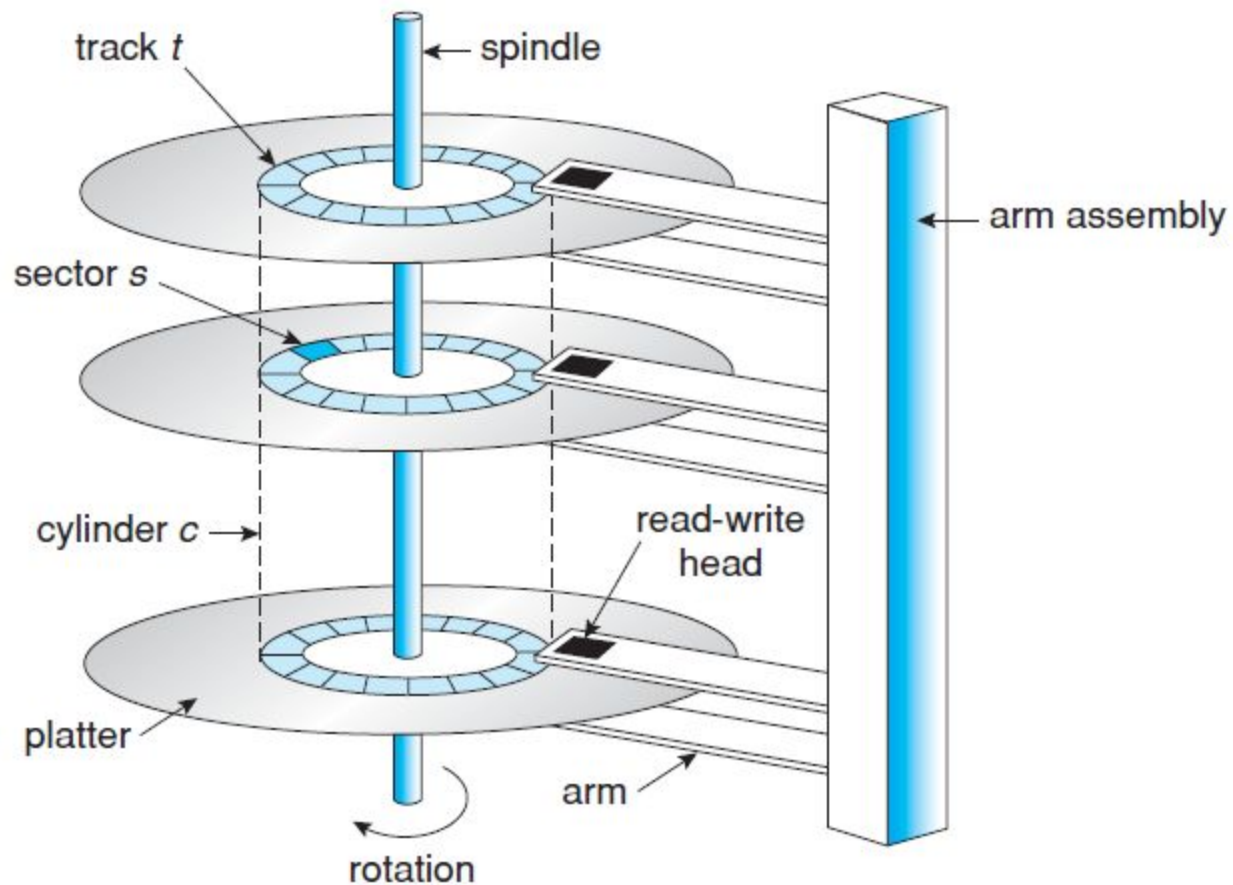


Figure 10.1 Moving-head disk mechanism.

- Magnetic disks provide the bulk of secondary storage for modern computer systems. Conceptually, disks are relatively simple (Figure 10.1-Next slide).
- Each disk platter has a flat circular shape, like a CD.
- Common platter diameters range from 1.8 to 3.5 inches.
- The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.
- A read–write head “flies” just above each surface of every platter.
- The heads are attached to a disk arm that moves all the heads as a unit.
- The surface of a platter is logically divided into circular tracks, which are subdivided into sectors.
- The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.
- The storage capacity of common disk drives is measured in gigabytes.

- When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute(RPM). Common drives spin at 5,400, 7,200, 10,000, and 15,000 RPM.
- Disk speed has two parts.
  - The transfer rate is the rate at which data flow between the drive and the computer.
  - The positioning time, or random-access time, consists of two parts:
    - The time necessary to move the disk arm to the desired cylinder, called the seek time.
    - The time necessary for the desired sector to rotate to the disk head, called the rotational latency.
- Typical disks can transfer several megabytes of data per second, and they have seek times and rotational latencies of several milliseconds.

# Disk Scheduling Algorithms



- Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk.
- Disk scheduling is also known as I/O scheduling.

- ★ FCFS Scheduling Algorithm
- ★ SSTF Scheduling Algorithm
- ★ SCAN Scheduling Algorithm
- ★ C-SCAN Scheduling Algorithm
- ★ LOOK Scheduling Algorithm

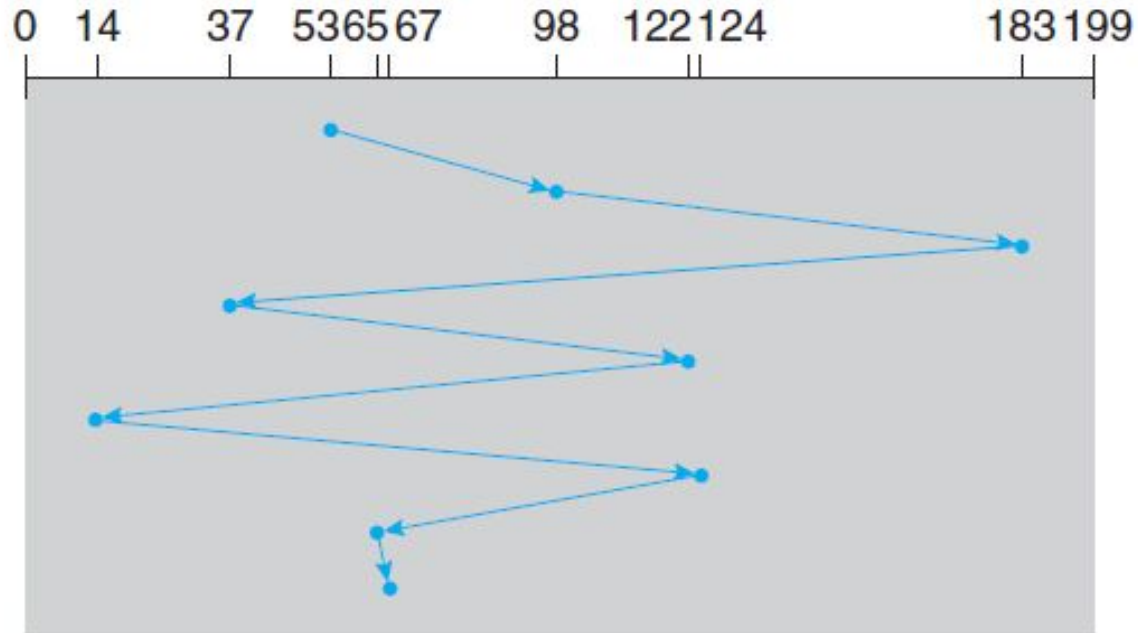
# FCFS Scheduling Algorithms

- The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm.
- This algorithm is intrinsically fair, but it generally **does not provide the fastest service.**



**Example:** A disk queue with requests for I/O to blocks on cylinders

**98, 183, 37, 122, 14, 124, 65, 67 , head starts at 53.**



**Figure 10.4** FCFS disk scheduling.

★ If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67.

★ **Total head movements** incurred while servicing these requests

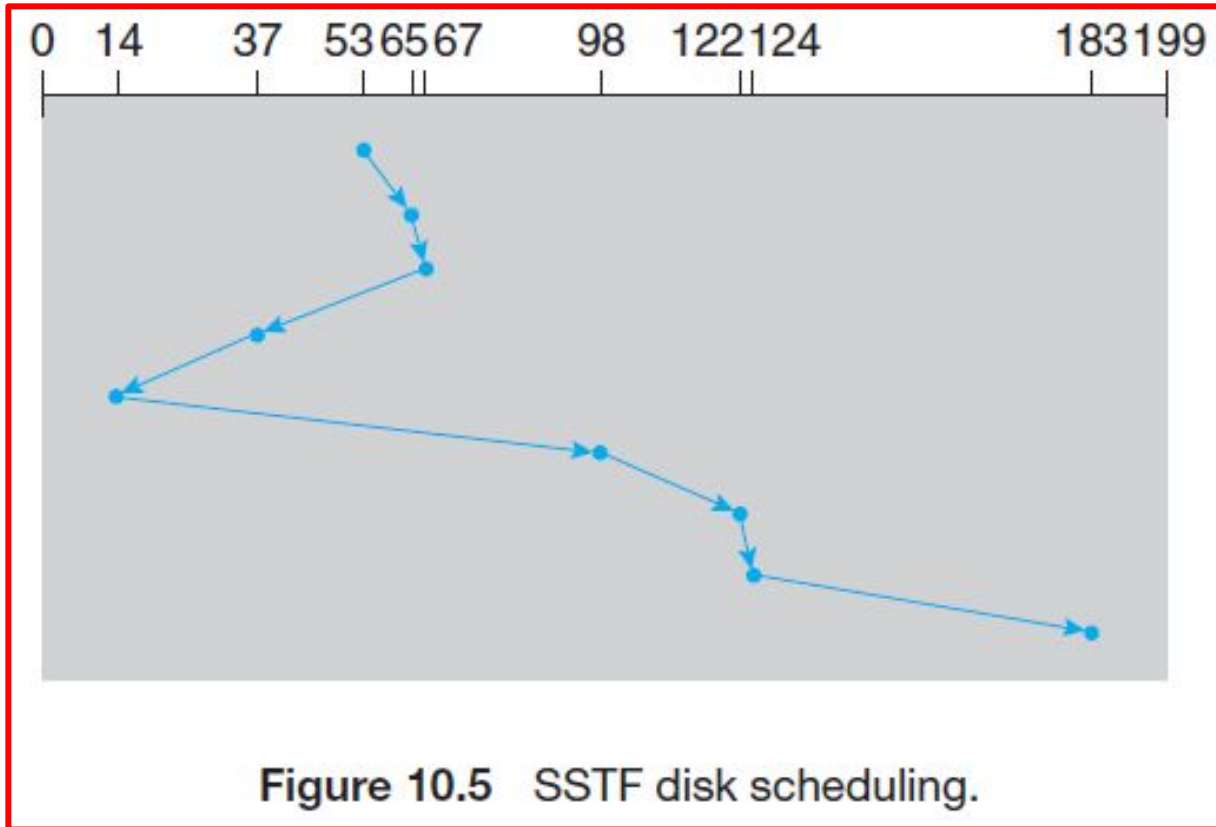
$$\begin{aligned} &= (98 - 53) + (183 - 98) + (183 - 37) + (122 - 37) + (122 - 14) + (124 - 14) + (124 - 65) + (67 - 65) \\ &= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2 \\ &= \mathbf{640} \end{aligned}$$

# SSTF Scheduling Algorithms

- The **SSTF(Shortest-Seek-Time-First)** algorithm selects the request with the least seek time from the current head position.
- In other words, SSTF chooses the pending request closest to the current head position.

**Example:** A disk queue with requests for I/O to blocks on cylinders

**98, 183, 37, 122, 14, 124, 65, 67 , head starts at 53.**



★ For our example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124, and finally 183 (Figure 10.5-Previous Slide).

★ **Total head movements** incurred while servicing these requests

$$\begin{aligned} &= (65 - 53) + (67 - 65) + (67 - 37) + (37 - 14) + (98 - 14) + (122 - 98) + \\ &\quad (124 - 122) + (183 - 124) \\ &= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59 \\ &= \mathbf{236} \end{aligned}$$

# **The Elevator Scheduling Algorithm (or)**

## **SCAN Scheduling Algorithms**

- In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
- At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.
- The SCAN algorithm is sometimes called **the elevator algorithm**.

**Example:** A disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67 , cylinders are numbered from 0 to 199 , head starts at 53.

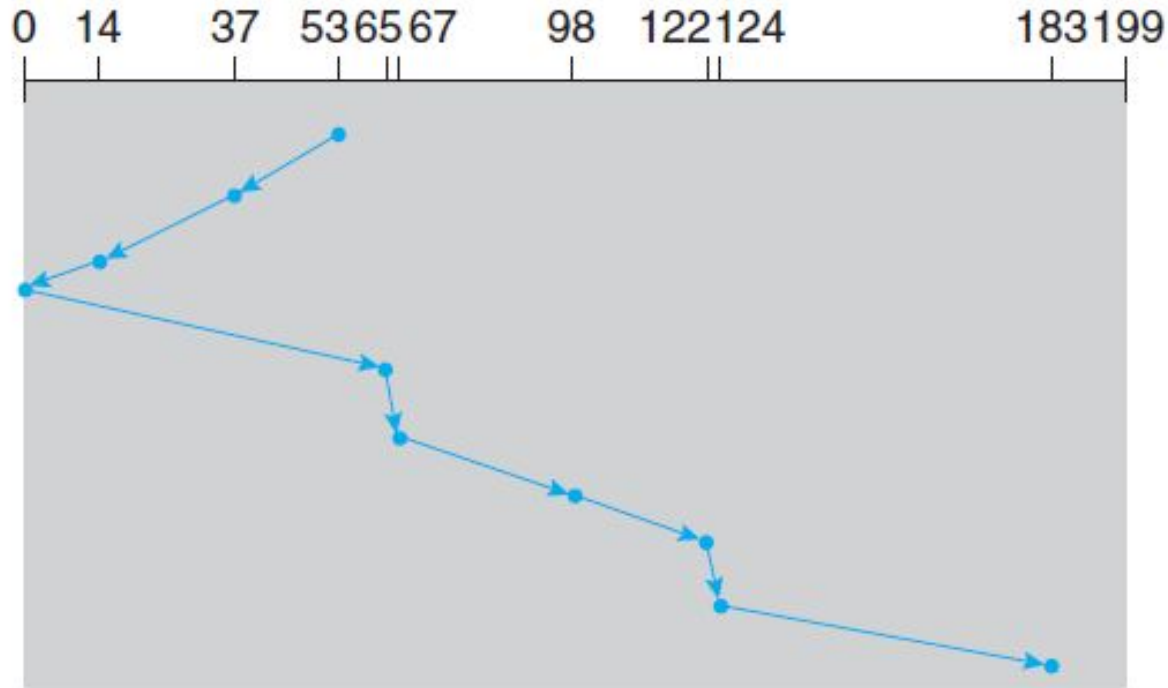


Figure 10.6 SCAN disk scheduling.

- ★ Before applying SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement in addition to the head's current position.
- ★ Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14.
- ★ At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183 (Figure 10.6).

★ **Total head movements incurred while servicing these requests**

$$= (53 - 37) + (37 - 14) + (14 - 0) + (65 - 0) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124)$$

$$= 16 + 23 + 14 + 65 + 2 + 31 + 24 + 2 + 59$$

$$= \mathbf{236}$$

**or**

$$= (53 - 0) + (183 - 0)$$

$$= 53 + 183$$

$$= \mathbf{236}$$

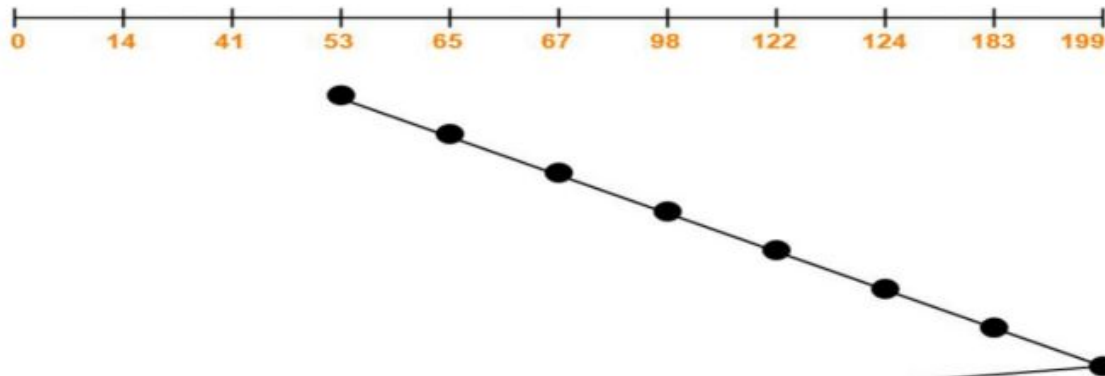


**Eg: 2**

Consider a disk queue with requests for I/O to blocks on cylinders

98, 183, 41, 122, 14, 124, 65, 67

The SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is \_\_\_\_\_.



$$= (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) + (199 - 183) + (199 - 41) + (41 - 14)$$

$$= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27 = 331$$

**(OR)**

$$\begin{aligned} &= (199 - 53) + (199 - 14) \\ &= 146 + 185 = 331 \end{aligned}$$

# C-SCAN Scheduling Algorithms

- **Circular SCAN (C-SCAN)** scheduling is a variant of SCAN designed to provide a more uniform wait time.
- Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.
- When the head reaches the other end, however, it immediately returns to the beginning of the disk **without servicing any requests on the return trip.**

**Example:** A disk queue with requests for I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67 , cylinders are numbered from 0 to 199 , head starts at 53.

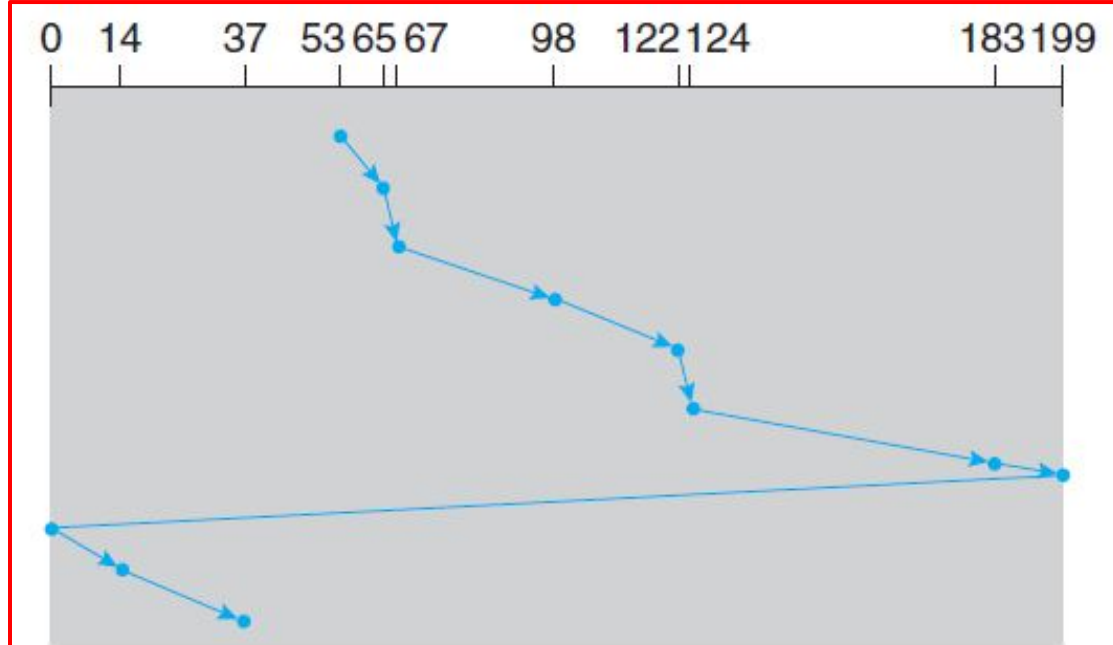


Figure 10.7 C-SCAN disk scheduling.

- ★ Before applying SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement in addition to the head's current position.
- ★ Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14.
- ★ At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183 (Figure 10.6).
- ★ **Total head movements incurred while servicing these requests**

$$\begin{aligned}
 & (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) + (199 - 183) + (199 - 0) + (14 - 0) + (37 - 14) \\
 &= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 23 \\
 &= 382
 \end{aligned}$$

Alternatively,

$$\begin{aligned}
 & \text{Total head movements incurred while servicing these requests} \\
 &= (199 - 53) + (199 - 0) + (41 - 0) \\
 &= 146 + 199 + 37 = 382
 \end{aligned}$$