**Unit 5: Device Management**

**5.1 Device Management Function**
**5.2 Device Characteristics**
**5.3 Disk Space Management**
**5.4 Allocation and Disk Scheduling Methods**

## 5.1 Device Management Function

- Device management in an operating system means controlling the Input/output devices like disk, microphone, keyboard, printer, magnetic tape, USB ports, camcorder, scanner, other accessories, and supporting units like supporting units control channels.
- A process may require various resources, including main memory, file access, and access to disk drives, and others.
- If resources are available, they could be allocated, and control returned to the CPU.
- Otherwise, the procedure would have to be postponed until adequate resources become available.
- The system has multiple devices, and in order to handle these physical or virtual devices, the operating system requires a separate program known as an ad device controller.
- It also determines whether the requested device is available.

Device Management is needed for offering a uniform and consistent approach to all I/O operations. There are considerable differences between all the system's devices; their speed, how data are transferred and represented, how to prevent and detect errors and how they are handled.

**The basic functions of device management are as mentioned below:**

1. Keep tracks of all devices and the program with the help of I/O controller.
2. Monitoring the status of each device such as storage drivers, printers and other peripheral devices.
3. Deciding on policy to determine who gets a device, for how long and when. A wide range of techniques is available for implementing these policies. There are 3 basic techniques for implementing the policies of device management:

    a. **Dedicated**: Assigned to only one job at a time until that job releases them.
    b. **Shared**: A technique whereby a device is being shared by many processes.
    c. **Virtual**: A technique where one physical device is simulated on another physical device.

4. Allocation- physical assigning device to a process. Likewise, the corresponding control units and channels must be assigned.
5. De-allocation policy and techniques. De-allocation may be done on either a process or a job level. On a job level, a device is assigned for as long as the job exists in the system. On process level, a device is assigned for as long as the process needs it.
6. Optimizes the performance of individual devices.

## 5.2 Device Characteristics

In an **operating system (OS)**, **device management** involves handling the characteristics and operations of hardware devices to ensure smooth interaction with software and users.

Key elements of device characteristics include:

1. **Device Types:** The OS manages different categories of devices such as:
    - **Input devices** (keyboard, mouse, touchscreen) for user interaction.
    - **Output devices** (monitor, printer, speakers) for displaying or outputting data.
    - **Storage devices** (HDDs, SSDs, USB drives) for data storage and retrieval.
    - **Network devices** (Ethernet cards, Wi-Fi adapters) for connectivity.
    - **Peripheral devices** (cameras, scanners, game controllers).
2. **Device Drivers:** Drivers are software components that enable the OS to communicate with hardware. They define device capabilities and manage operations, ensuring the device performs as intended.
3. **Hardware Resources:** Devices rely on system resources for operation, such as:
    - **Interrupts (IRQ):** Notify the CPU to handle a device request.
    - **I/O Ports:** Facilitate communication between the CPU and the device.
    - **Memory:** Devices may require access to system memory (e.g., via Direct Memory Access).
4. **Plug-and-Play (PnP):** The OS automatically detects and configures devices, eliminating manual setup. PnP assigns drivers, allocates resources, and ensures compatibility.
5. **Device States:** The OS tracks devices in various states:
    - **Active:** In use and processing requests.
    - **Idle:** Ready for use but not currently active.
    - **Suspended:** In low-power mode to save energy.
    - **Disabled:** Temporarily unavailable or turned off.
6. **Access Modes:** Devices operate in:
    - **Character mode:** Processing data one byte/character at a time (e.g., keyboards).
    - **Block mode:** Processing data in blocks, commonly used in storage devices.
7. **Data Transfer Methods:**
    - **Buffering:** Temporary data storage to manage speed mismatches between device and application.
    - **Interrupt-driven I/O:** Devices signal the CPU when ready for interaction.
    - **Direct Memory Access (DMA):** Allows devices to access memory directly, bypassing the CPU.
8. **Security and Permissions:** The OS ensures only authorized users and applications can access devices, using measures like access controls and encryption.
9. **Error Handling:** The OS monitors devices for issues, logs errors, and attempts recovery methods like resetting or notifying the user.
10. **Device Monitoring:** The OS provides tools to monitor device performance, health, and resource usage, ensuring smooth operation and diagnosing problems when they arise.

These characteristics help the OS manage hardware efficiently, providing a seamless user experience while maintaining security, compatibility, and performance.

### 5.3 Disk Space Management

Disk management is one of the critical operations carried out by the operating system. Disk space management in an operating system (OS) refers to how the OS organizes, allocates, and monitors storage space on a disk drive (e.g., HDD-Hard disk drive, SSD-solid state drive). Effective disk space management ensures that files and data are stored efficiently, retrieved quickly, and the disk's space is utilized optimally. It also carries out the function of optimizing the data and making sure that the data is safe by implementing various disk management techniques.

The four main operating system management functions (each of which are dealt with in more detail in different places) are:

- Process Management
- Memory Management
- File and Disk Management
- I/O System Management

Key aspects include:

1. Disk Format
2. Booting from disk
3. Bad block recovery

**1. Disk Format**

- A new magnetic disk is a blank platter of a magnetic recording material.
- Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting, or physical formatting.
- Low-level formatting fills the disk with a special data structure for each sector.
- The data structure for a sector typically consists of a header, a data area, and a trailer.
- The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC).
- When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
- When the sector is read, the ECC-is recalculated and is compared with the stored value.
- If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad.
- The ECC is an error-correcting code because it contains enough information that, if only a few bits of data have been corrupted, the controller can identify which bits have changed and can calculate what their correct values should be.

  To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It performs this with following two steps:

i. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk.
ii. After partitioning, the second step is logical formatting (creation of a file system). The OS stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters. Disk I/O is done via blocks, but file system I/O is done via clusters.

**2. Booting from disk**

- When a computer starts running or reboots to get an instance, it needs an initial program to run. This initial program is known as the bootstrap program, and it must initialize all aspects of the system, such as:
  - First, initializes the CPU registers, device controllers, main memory, and then starts the operating system.
  - The bootstrap program finds the operating system kernel on disk to do its job and then loads that kernel into memory.
  - And last jumps to the initial address to begin the operating-system execution.
- The bootstrap is stored in read-only memory (ROM), which is convenient as it doesn't require initialization and can be executed when powered up or reset.
- ROM stands for read-only memory, and it is not susceptible to computer viruses. However, ROM and hardware chips must be modified in order to change the bootstrap code. Systems have a little bootstrap loader software stored.
- The full bootstrap program is easily modified and stored in "boot blocks" on a fixed disk, also known as a boot disk or system disk.
- The boot ROM code instructs the disk controller to read boot blocks and execute code, while the full bootstrap program is more sophisticated, loading the entire operating system from a non-fixed location.

**How Boot Block Works?**

- Windows 2000 allows hard disk partitioning into one or more partitions, with the boot partition containing the operating system and device drivers.
- Booting starts with code in ROM memory, read from the master boot record (MBR), and reads the first sector from memory.
- The boot process continues with loading system services.

**3. Bad Block Recovery**

- A bad block is an area of storage media that is no longer reliable for storing and retrieving data because it has been completely damaged or corrupted.
- Bad blocks are also referred to as bad sectors.
- We know disks have moving parts and have small tolerances. They are level to failure.
- When the failure is complete, the disk needs to be replaced and its contents restored from backup media to the new disk.

- More frequently, one or more sectors become defective.
- These blocks are handled in many ways, but it depends upon the disk and controller.
- Bad blocks are handled manually for some disks with IDE controllers or simple disks.
    - The first strategy is to scan the disk for bad blocks during formatting, flagging them as unusable. If bad blocks occur during normal operation, a manual program is run.
    - More sophisticated (stylist) disks are smarter about bad-block recovery. The work of the controller is to maintain the list of bad blocks. The list formed by the controller is initialized during the low-level formatting at the factory and updated over the disk's life.

Low-level formatting holds the spare sectors which are not visible to the operating system. In the last, a controller replaces each bad sector logically with the spare sectors. This process is also known as sector sparing and forwarding.

**Some common disk management techniques used in operating systems include:**

1. **Partitioning:** This involves dividing a single physical disk into multiple logical partitions. Each partition can be treated as a separate storage device, allowing for better organization and management of data.

2. **Formatting:** This involves preparing a disk for use by creating a file system on it. This process typically erases all existing data on the disk.

3. **File system management:** This involves managing the file systems used by the operating system to store and access data on the disk. Different file systems have different features and performance characteristics.

4. **Disk space allocation:** This involves allocating space on the disk for storing files and directories. Some common methods of allocation include contiguous allocation, linked allocation, and indexed allocation.

5. **Disk defragmentation:** Over time, as files are created and deleted, the data on a disk can become fragmented, meaning that it is scattered across the disk. Disk defragmentation involves rearranging the data on the disk to improve performance.

## 5.4 Allocation and Disk Scheduling Methods

### Disk Allocation

- When file is created, storage space is allocated to it.
- Also, when new data is added in an existing file, file size grows and it needs extra storage space.
- In a similar way, storage space is released when a file is deleted or data from a file is deleted.
- An important function of the file system is to manage space on the secondary storage, which includes keeping track of both disk blocks allocated to files and the free block available for allocation.

There are two main goals, which should be fulfilled while allocating space on disk to files.
1. Disk space should be utilized effectively.
2. Files should be accessed quickly.

- At the time of space allocation, system must keep track of which disk blocks go with which files.
- Here the disk is considered as a collection of fixed size of blocks, where size varies from system to system.
- Most commonly it is of 512 bytes of 1kb in size.

These blocks are being numbered starting from 0 or 1 to some maximum.

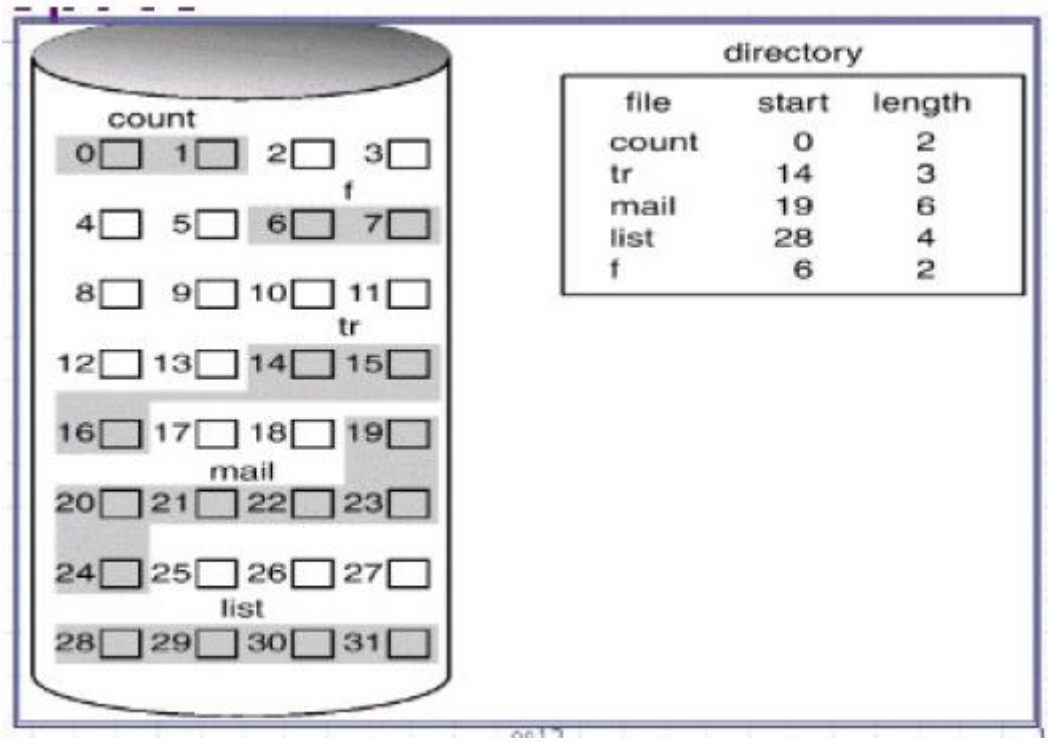
But secondary storage introduces two additional problems:
1. slows disk access time and
2. larger number of blocks to deal with

- In spite of that, many considerations are similar to both environments, particularly, contiguous and non-contiguous allocation of files.
- Whenever there is need to allocate storage space to a file one or more disk blocks are allocated to the file. In a similar way, whenever a file is being deleted, allocated blocks will become free for reallocation.

There are three main methods for disk allocation:
1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

**1. Contiguous Allocation**

- In contiguous allocation, files are assigned to contiguous areas of secondary storage.
- A user specifies in advance the size of the area needed to hold a file to be created.
- If the desired amount of contiguous space is not available, the file cannot be created.
- Each file occupies asset of contiguous blocks on the disk.
- Thus, on a disk block of 1kb, a file of 50 kb would contain 50 consecutive disk blocks. Whereas, if block size is 2kb; it will occupy 25 consecutive blocks.
- When a file is created, a disk is searched to find out a chunk of free memory having enough size to store a file. If such chunk is found, required memory is allocated.
- The directory entry contains file name, starting block number and length of a file.




This method is widely used on CD-ROMs. Here, all the files' sizes are known in advance. Also, they will never change during subsequent uses of CD-ROM. Figure shown besides is depicting such allocation for 4 different files.

**Advantages:**

1. Simple to implement. Information here required is only two things; one starting block #; and second, length of file as a total number of blocks.
2. File access is quick. All the data blocks will be on the same or neighbour tracks of disk, requiring less seek time.

**Disadvantages:**

1. Finding free space for a new file is time consuming. This requires searching an entire disk until required free memory is found.
2. If size of an existing file increases, it may not be possible to accommodate such extension.
3. **External fragmentation** is possible. When file is deleted, its blocks are free leaving hole on the disk. With time, disk will consist of files and holes. Such hole may be too small to accommodate new files and will waste space on the disk. It is known as external fragmentation. Solution for this problem is defragmentation or compaction the file.

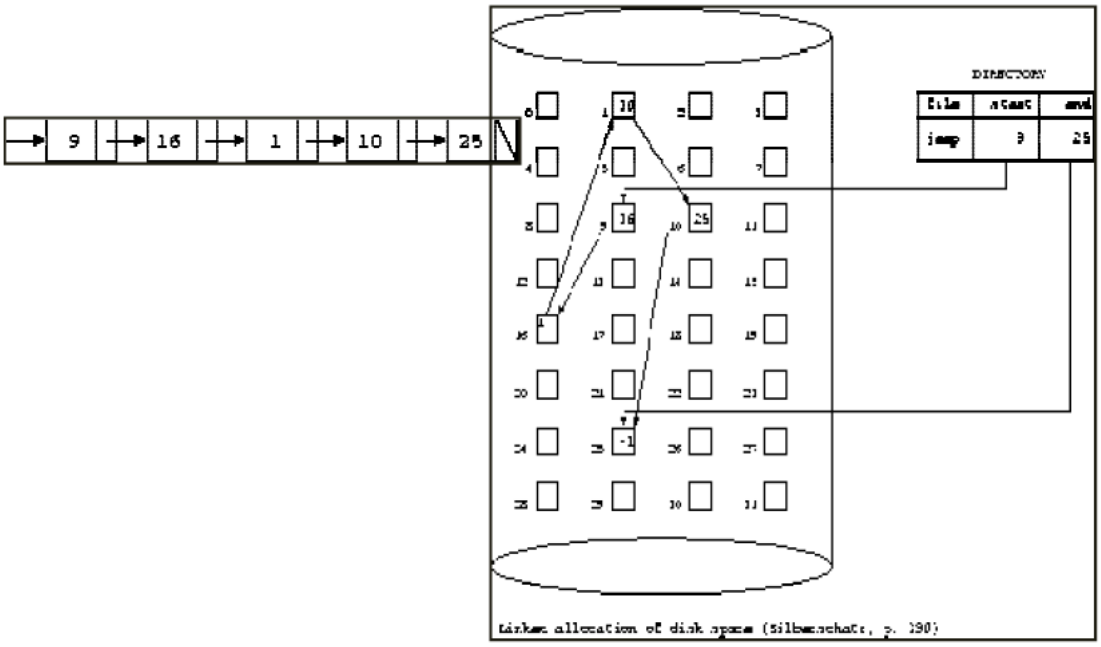
**2. Linked Allocation**

- Each file is a linked list of disk blocks.
- Each linked block contains pointer to the next block in the list.
- These disk blocks may be scattered anywhere on the disk.
- Directory entry contains the start and last block numbers in a linked list.

The directory entry structure is shown below:

| File Name | Starting Block # | Last Block # |
|---|---|---|

- When a new file is created; a new directory entry is created. Initially it contains 'null' as both the block number.
- A write to the file causes free data blocks to be added to the file; such blocks are added to the end of linked list.
- Directory entry is updated on each such occasion. To read a file, all blocks are read by following the pointers from block to block.
- Following figure is depicting the linked allocation.




In the figure beside; to reach block # 10, it is required to traverse through block # 9, 16 and 1. An important variation on the linked allocation method, called File Allocation Table (FAT), is used by the MS-DOS and older versions of Windows Operating Systems.

**Advantages:**

1. It does not suffer from external fragmentation.
2. Any free disk block can be allocated to a file. Such block does not need to be a consecutive block as in previous method. So, disk space can be utilized effectively.

**Disadvantages:**

1. File access is time consuming. It is required to access all the data blocks in a linked list to reach some particular block.
2. Random access is not possible directly.
3. Extra space is required for pointers in each data block.

### 3. Index Allocation

- In linked allocation, pointers to various disk blocks are scattered on disk among various disk blocks.
- Due to this reason, linked allocation cannot support efficient direct access.
- Indexed allocation solves this problem.
- It brings all the pointers together into one location; the Index Block.
- Each file contains its own index block.
- An index block is an array of ‘disk block addresses.
- The ‘$i^{th}$’ entry in the index block points to the ‘$i^{th}$’ block of the file.
- Directory entry contains file name and the index block #.

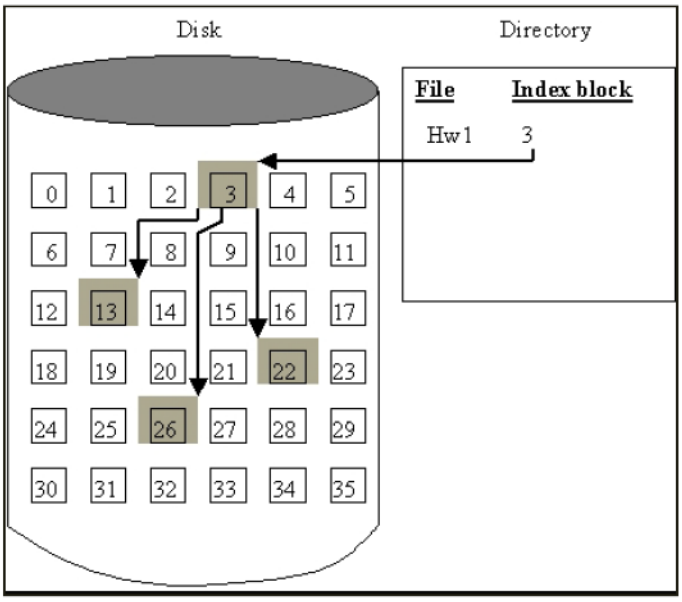
This looks as shown below:

| File Name | Index Block# |
|---|---|

- When new file is created, a new directory entry is created. Initially all pointers in index block are set null.
- When the ‘$i^{th}$’ block is first written, a free disk block is allocated and its address is put in the ‘$i^{th}$’ entry in the index block.

The following figure depicts the indexed allocation for the file ‘Hw1’. The directory entry for the index block is done as block #3 for this file. This index block entry is composed of all other block # entries which are having the data contents of the file Hw1. The index entry consists of all the block entries which consists of the data Contents.

| 13 | 26 | 22 |
|---|---|---|



**Advantages:**

1. It does not suffer from external fragmentation.
2. Direct access is efficient.

**Disadvantages:**

1. It suffers from wasted space. Index block may be partially filled. These was remaining memory space of an index block. For example, if file contains two data blocks, the index block will have only two entries to point to these two data blocks, remaining entire index block will be wasted.
2. Maximum allowable file size depends on the size of an index block.

**Solutions:**

In this allocation method the main problem is the size of index block. If it is too large, it may waste space. If it is too small it cannot accommodate enough pointers for a large file.

Following are the solutions:

1. **Linked Scheme**

- An index block is normally of one disk block size.
- For larger files more than one index block can be used by linking them together by a linked list.

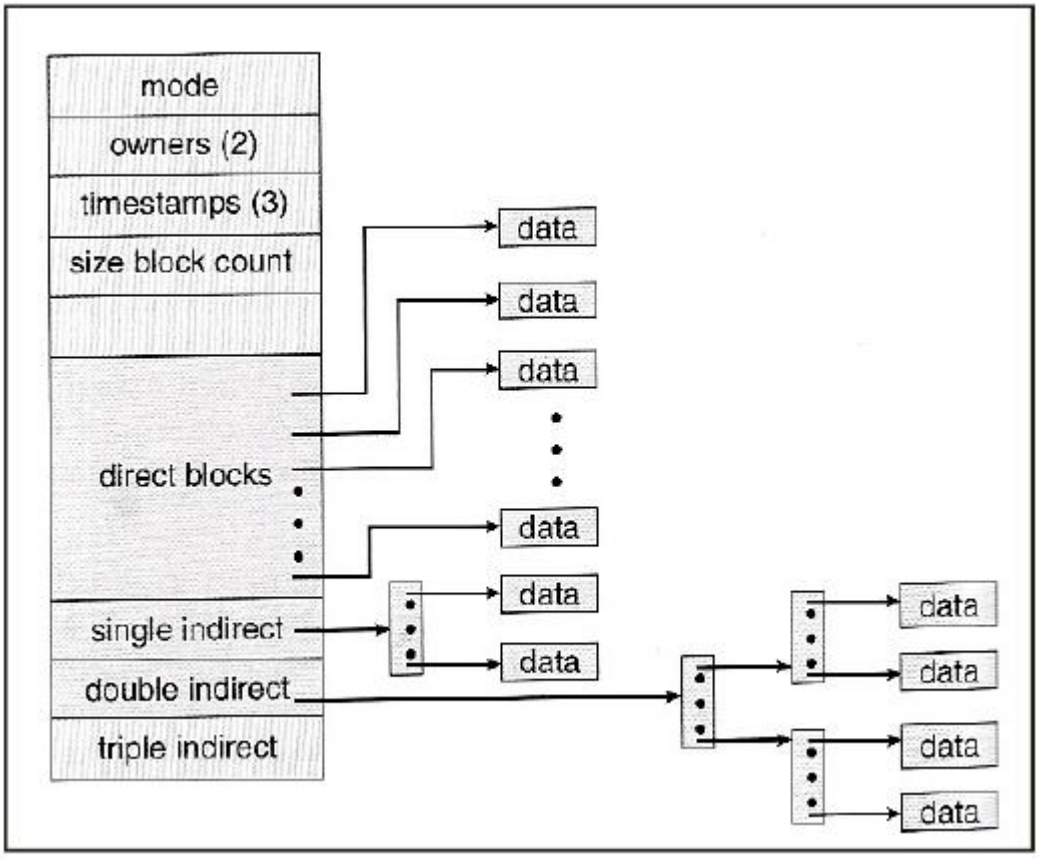
2. **Multilevel Index / Hierarchical Indexing**

- Two or more level of index blocks is used here.
- First level index blocks point to a set of second level index blocks. Second level index blocks point to disk blocks containing the data. For larger and larger file, levels can be increased.

3. **Combined Scheme**

- In this solution, partial entries from indirect block point to direct block containing file data.
- Partial entries point to indirect blocks, which are index blocks.
- They do not contain file data, but they contain address of disk blocks (pointers) that do contain file data.
- UNIX OS is using such type of combined scheme.

  Figure shown besides is an example of such combined scheme which is implemented in the UNIX operating system for storing the node.




**Disk Access Time**

**Seek Time**

The time taken by the read/ write head to reach the desire track from its current position
is called seek time.

**Latency Time**

Time taken by the sector to come under the read/write is called Rotational latency/latency time.

**Transfer Time**

It is the time to transfer data. It depends on the rotating speed of the disk and number of bytes to be transferred.

**Access Time**

Disk Access Time= Seek Time + Rotational Latency + Transfer Time.

**Bandwidth**

Total number of bytes transferred divided by the total time between first request for service and completion of last transfer.

**Disk Response Time**

Response Time is the average of time spent by a request waiting to perform its I/O operation. Average Response time is the response time of the all requests. Variance Response Time is measure of how individual request are serviced with respect to average response time. So, the disk scheduling algorithm that gives minimum variance
response time is better.

**Disk Scheduling**

- One of the responsibilities of the operating system is to use the hardware efficiently.
- For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.
- The access time has two major components. The SEEK TIME is the time for the disk arm to move the heads to the cylinder containing the desired sector. The LATENCY is the additional time for the disk to rotate the desired sector to the disk head. The 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. We can improve both the access time and the bandwidth by managing then order in which disk I/O requests are serviced.
- Whenever a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information:
    - Whether this operation is input or output
    - What the disk address for the transfer is
    - What the memory address for the transfer is
    - What the number of sectors to be transferred is

- If the desired disk drive and controller are available, the request can be serviced immediately.
- If the drive or controller is busy, any new requests for service will be placed in the queue of pending requests for that drive. For a multiprogramming system with many processes, the disk queue may often have several pending requests.
- Thus, when one request is completed, the operating system chooses which pending request to service next.
- How does the operating system make this choice? Any one of several disk scheduling algorithms can be used, and we discuss them next.

**Types of Disk Scheduling Algorithms**

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms:

1. First Come-First Serve (FCFS)
2. Shortest Seek Time First (SSTF)
3. Elevator (SCAN)
4. Circular SCAN (C-SCAN)
5. LOOK

6. C-LOOK

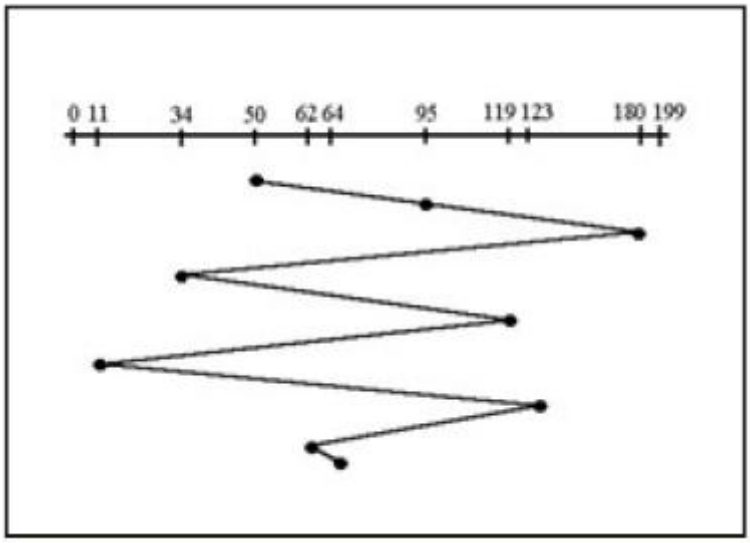
**Types of Disk Scheduling algorithms**

What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why CLOOK is the best algorithm to use in trying to establish less seek time.

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. **First Come First Serve (FCFS)**

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next.




For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

**Advantages:**

- Simple and Fair to all requests
- Every request gets a fair chance.
- No indefinite postponement.

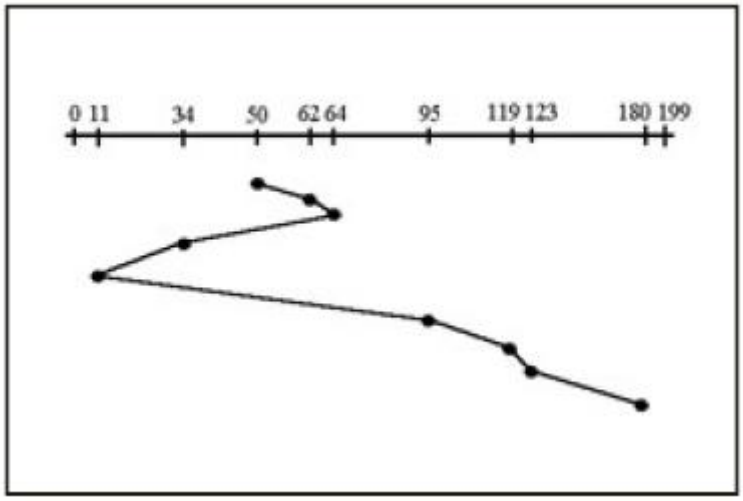
**Disadvantage:**

- Not efficient, because the average seek time is very high. It suffers from zigzag effect.

- Does not try to optimize seek time.
- May not provide the best possible service.

**2. Shortest Seek Time First (SSTF)**

Selects the request with the minimum seek time from the current head position. Also called Shortest Seek Distance First (SSDF) – It's easier to compute distances. It's biased in favour of the middle cylinder's requests. SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.




In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the processes are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.

**Advantages:**

- More efficient than FCFS.
- Average Response time decreases.
- Throughput increases.

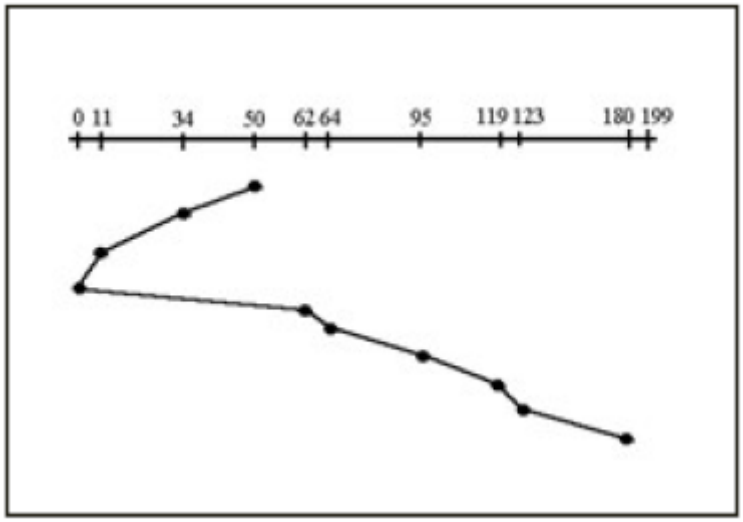
**Disadvantages:**

- Starvation is possible for requests involving longer seek time.
- Overhead to calculate seek time in advance.
- Can cause starvation for a request if it has higher seek time as compared to
- incoming requests?
- High variance of response time as SSTF favours only some requests.

**Elevator Algorithms**

Theses algorithms are based on the common elevator principle. Four combinations of Elevator algorithms: SCAN, LOOK, C-SCAN and C-LOOK. They service in both directions or in only one direction. They operate until last cylinder or until last I/O request encountered.

3. **Elevator (SCAN)**
   In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.




If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.

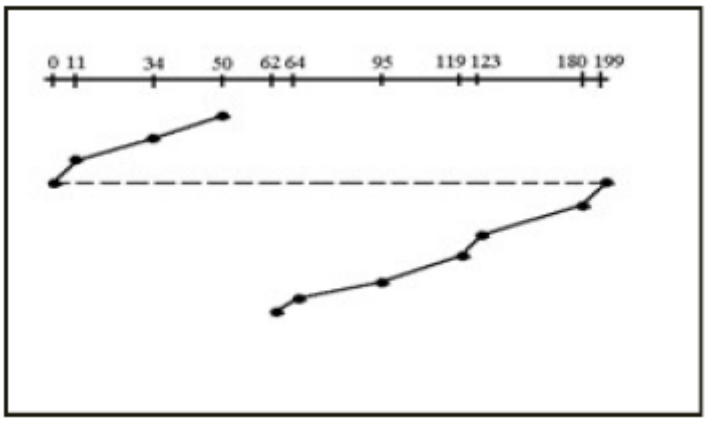
**Advantages:**

- More efficient than FCFS.
- Also, there is no starvation for any requests.
- High throughput.
- Low variance of response time.
- Average response time.

**Disadvantages:**

- Require extra head movement between two extreme points. For example, after servicing 5th cylinder there is no need to visit the 0th cylinder. Though, this algorithm visits the end points.
- Not so fair; cylinders which are just behind head will wait longer.
- Long waiting time for requests for locations just visited by disk arm.

**4. Circular SCAN (C-SCAN)**



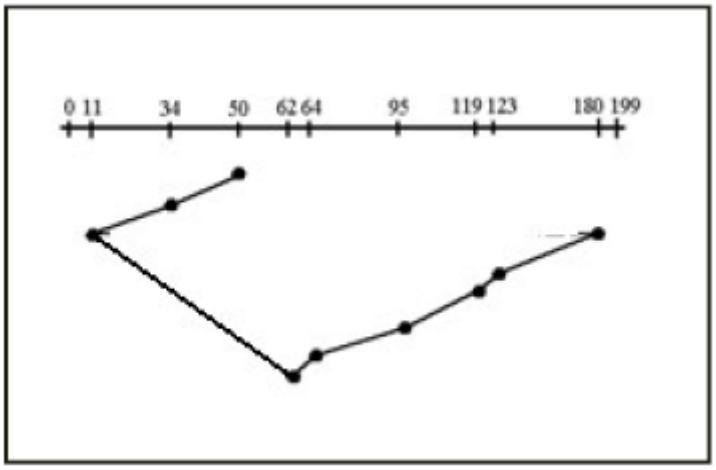

Circular scanning works just like the elevator to some extent. The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. It treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It provides a more uniform wait time than SCAN; it treats all cylinders in the same manner. It begins its scan toward the nearest end and works its way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 tracks, but still, this isn't the more sufficient.

**Advantages:**

- Provides more uniform wait time compared to SCAN.

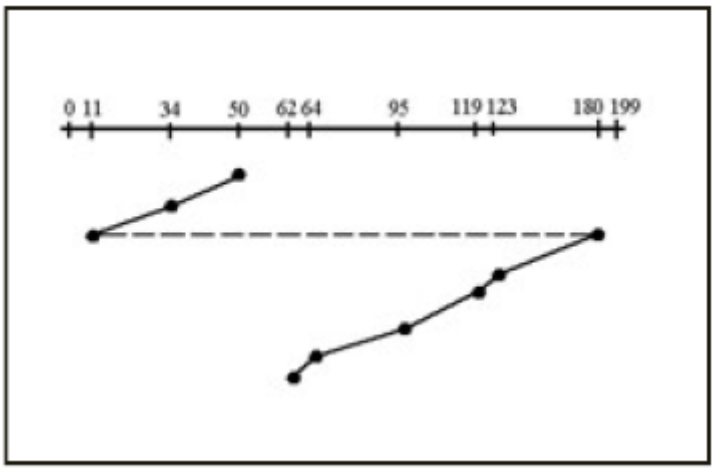
**5. LOOK**

The disk arm starts at the first I/O request on the disk, and moves toward the last I/O request on the other end, servicing requests until it gets to the other extreme I/O request on the disk, where the head movement is reversed and servicing continues. It moves in both directions until both last I/O requests; more inclined to serve the middle cylinder requests. In this example head will move from 11 to 62 rather going to 0 from 11.



## 6. C-LOOK

It is an enhanced 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. Scan versions have a larger total seek time than the corresponding Look versions.




This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks. From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.

**Selecting Disk Scheduling Algorithms**

Here some factors are given which affect the performance of a disk scheduling algorithm as shown below:

1. **Number and types of Requests:**

- Performance of a disk scheduling algorithm heavily depends upon the number and type of request.
- For example, if the device queue contains only one choice is available, all algorithms will behave like FCFS.

2. **File Allocation Method:**

- File allocation method also has good contribution in deciding performance.
- For example, when a contiguously allocated file is accessed, the disk requests are normally close to each other. Whereas for a linked or indexed file, disk blocks are scattered over disk, and disk head requires great movements.

3. Location of directories and index blocks:

- Every file must be opened before use. For this, there is a need to search directory structure to determine location of files. So, directories are accessed frequently.

- If the directory entry is on the first cylinder, and file's data are on the final cylinder, then disk head required to move the entire width of the disk.

These are the main three factors that must be observed at the time of selecting the algorithm. An average seek time with FCFS is very high. SCAN required unnecessary head movements between two end points. So, in default case, either SSTF or LOOK is
reasonable choice as a disk scheduling algorithm.