3.2 Disk I/O operations

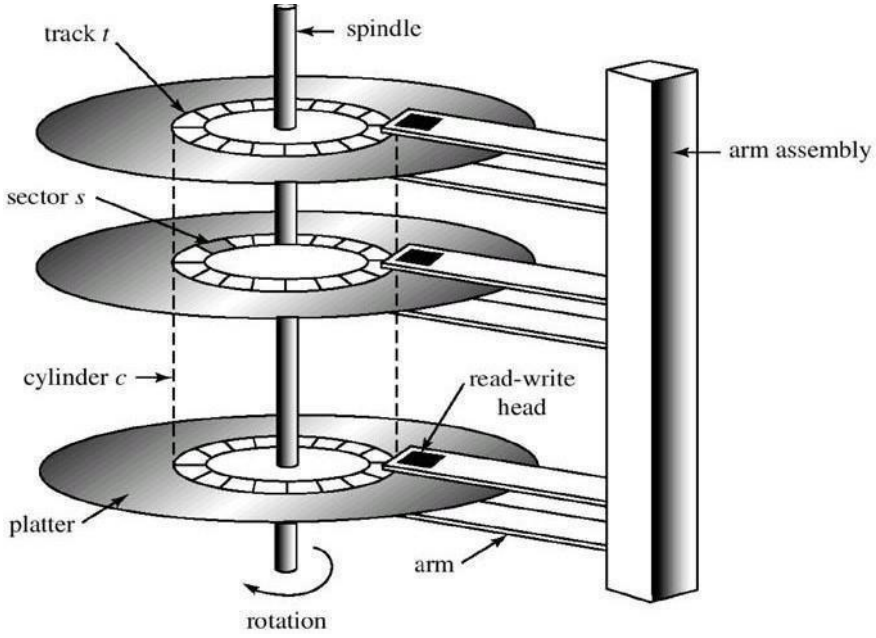
Disk I/O (Input/Output) operations in an operating system refer to process of **reading from or writing to a storage device** such as a hard disk drive (HDD), solid-state drive (SSD), or external disks. These operations are fundamental to how data is accessed, stored, and managed by the OS.

- **Definition**: Disk I/O encompasses all read/write operations between the CPU and a physical disk.
- **Purpose**: It allows programs and users to access files, save data, and retrieve information from storage.
- **Latency**: Disk access is much slower than memory access, making efficient I/O management crucial for performance.

Structure of a Disk

Magnetic disks provide the bulk of secondary storage for modern computer systems. Each disk **platter** has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 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,** which 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 forms 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



A hard disk consists of:

1. **Platters** – Circular magnetic disks stacked vertically.
2. **Surfaces** – Each side of a platter that stores data.
3. **Tracks** – Concentric circles on a surface where data is written.
4. **Sectors** – Smallest addressable unit in a track (commonly 512 bytes or 4 KB).
5. **Clusters (or Blocks)** – A group of sectors (OS reads/writes in blocks).
6. **Cylinders** – All tracks on different platters that are vertically aligned.

Capacity of Magnetic disks(C) = S x T x P x N

Where S= no. of surfaces = 2 x no. of disks, T= no. of tracks in a surface, P= no. of sectors per track, N= size of each sector

Transfer Time: The transfer time to or from the disk depends on the rotation speed of the disk in the following fashion:

$T = b / (r \times N)$

Where *T*= transfer time, b=number of bytes to be transferred, *N*= number of bytes on a track, **r**= rotation speed, in revolutions per second.

Steps in a Disk I/O Operation

When a program requests data from the disk, the OS performs several steps:

(a) Request Initiation

- The OS receives a request from a process (via a system call like `read()` or `write()`).
- The file system determines **where** on the disk the requested data resides.

(b) Translating Logical Address to Physical Address

- The OS translates the **logical block address (LBA)** into a **cylinder-head-sector (CHS)** or SSD page location.
- Uses file allocation tables (FAT, inode mapping, etc.) to locate the block.

(c) Scheduling the Request

- Multiple processes may request disk access.
- The OS uses **disk scheduling algorithms** (e.g., FCFS, SSTF, SCAN) to decide the order of service to minimize seek time.

(d) Seek Operation

- The **disk arm** (HDD) moves the read/write head to the correct **track** (seek time).
- For SSDs, this step is electronic and much faster.

(e) Rotational Latency

- Wait for the desired sector to rotate under the read/write head (HDD-specific; negligible for SSDs).

(f) Data Transfer

- Data is transferred between the disk and **disk buffer (cache)** in the device controller.
- Then it is moved to the main memory (RAM) using **Programmed I/O (PIO)** or **Direct Memory Access (DMA)**.

(g) Completion and Acknowledgment

- The OS updates the file system structures (like inode modification time, journaling logs).
- The process is informed that the I/O operation has completed.

Types of Disk I/O Operations

(a) Read Operation

- Retrieves data from disk storage to memory.
- Commonly involves reading **sequential blocks** (faster) or **random blocks** (slower).

(b) Write Operation

- Stores data from memory to disk.
- May use **write-back caching** or **write-through** policy for performance and reliability.

(c) Seek Operation

- Moving the disk head to the desired track (HDD only; main source of delay).

(d) Control Operations

- Non-data transfers like disk formatting, mounting, or ejecting media.

3.2.1 DISK SCHEDULING ALGORITHMS

Disk scheduling algorithms are good in managing how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed, significantly impacting the speed and efficiency of data access. Disk scheduling is a technique operating systems use to manage the order in which disk I/O (input/output) requests are processed. Disk scheduling is also known as **I/O Scheduling**.

Importance of Disk Scheduling

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.

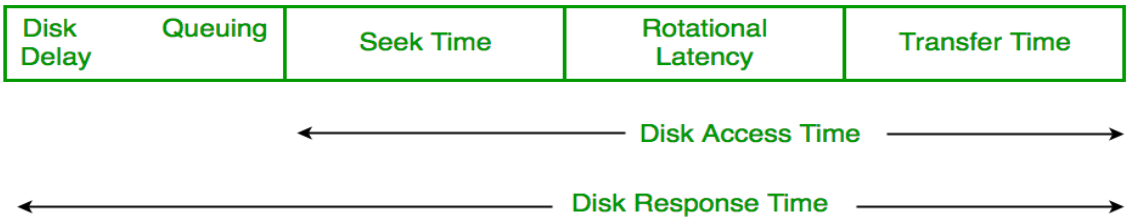
- Two or more requests may be far from each other so this can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Key Terms Associated with Disk Scheduling

- **Seek Time -** It is the time taken by the disk arm to locate the desired track.
- **Rotational Latency -** The time taken by a desired sector of the disk to rotate itself to the position where it can access the Read/Write heads is called Rotational Latency.
- **Transfer Time -** It is the time taken to transfer the data requested by the processes.
- **Disk Access Time -** Disk Access time is the sum of the Seek Time, Rotational Latency, and Transfer Time.

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

*Total Seek Time = Total head Movement * Seek Time*



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

Goal of Disk Scheduling Algorithms

- Minimize Seek Time
- Maximize Throughput
- Minimize Latency
- Fairness
- Efficiency in Resource Utilization

Disk Scheduling Algorithms

There are several Disk Several Algorithms. We will discuss in detail each one of them.

1) FCFS (First Come First Serve)

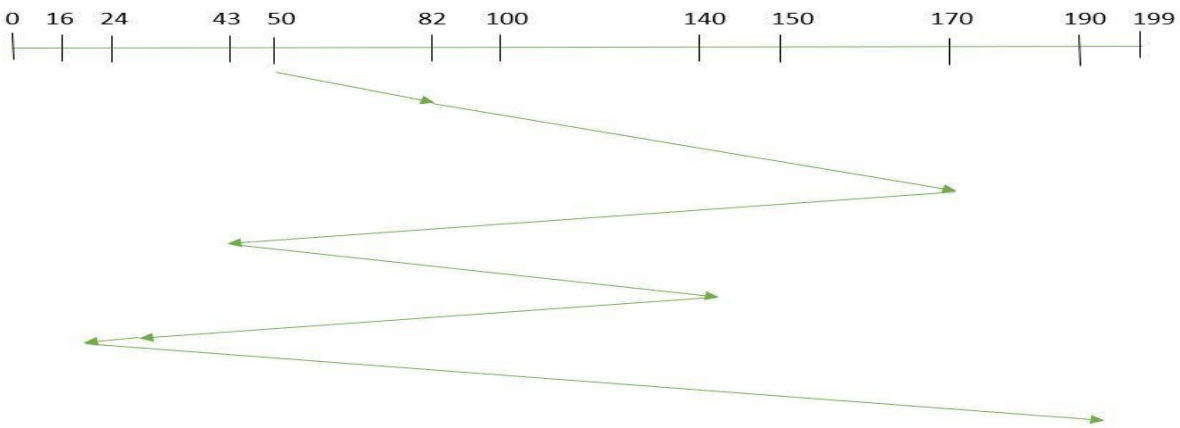
2) SSTF (Shortest Seek Time First)
3) SCAN
4) C-SCAN
5) LOOK
6) C-LOOK

1. FCFS (First Come First Serve)

FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50



So, total overhead movement (total distance covered by the disk arm) = (82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

Advantages

- Simple to implement.
- Fair to all processes (no starvation).

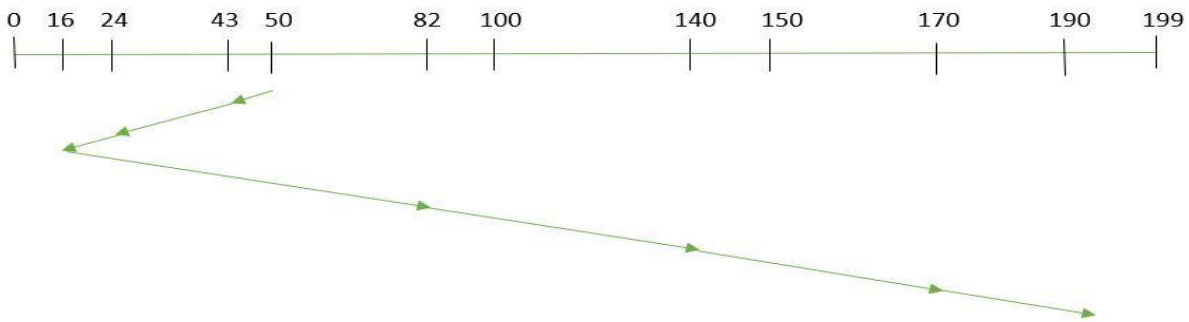
Disadvantages

- High average seek time if requests are far apart.
- No consideration of disk arm movement optimization.
- May not provide the best possible service.

2. SSTF (Shortest Seek Time First)

In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. THE disk arm moves to the request with the shortest seek time from its current position, services it, and then repeats this process until all requests have been serviced. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.

Example:



Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is: 50.

total overhead movement (total distance covered by the disk arm) = (50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

Advantages:

Here are some of the advantages of Shortest Seek Time First.

- The average Response Time decreases
- Throughput increases

Disadvantages:

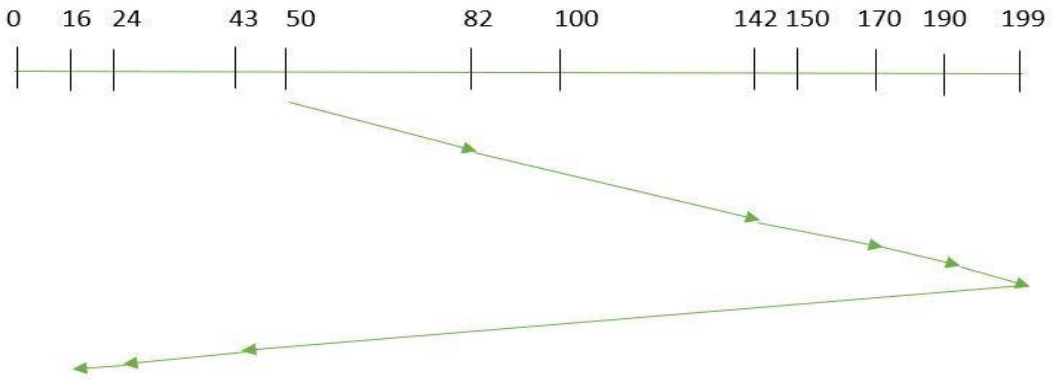
Here are some of the disadvantages of Shortest Seek Time First.

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- Needs continuous tracking of head position.

3. SCAN

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

Example:



Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as

= (199-50) + (199-16) = 332

Advantages:

- High throughput
- Low variance of response time
- Average response time
    - Fair and better performance than FCFS.
    - No starvation.

Disadvantages:

Slightly longer wait time for requests just behind the head.

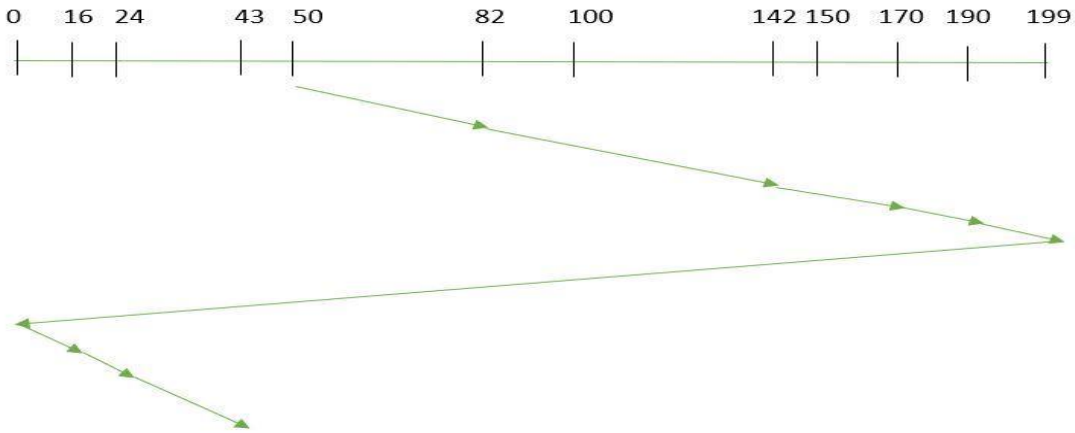
More complex than FCFS.

4. C-SCAN (Circular SCAN)

In the *C-SCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**



So, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50) + (199-0) + (43-0) = 391

Advantages of C-SCAN Algorithm

- Provides more uniform wait time compared to SCAN.

Disadvantages

- Slightly higher seek time than SCAN in some cases.
- Head jump from end to start adds overhead.

5. LOOK

It is similar to SCAN, but instead of going to the physical end, the head reverses at the last request in that direction. and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example: Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

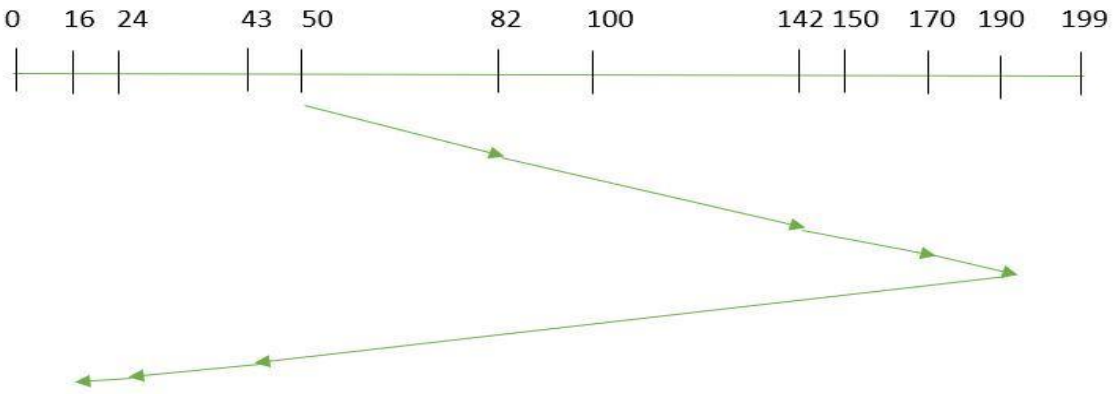
So, the total overhead movement (total distance covered by the disk arm) is calculated as:

= (190-50) + (190-16) = 314

Advantages

- Less unnecessary movement than SCAN.
- Better seek time in many cases.

Disadvantages



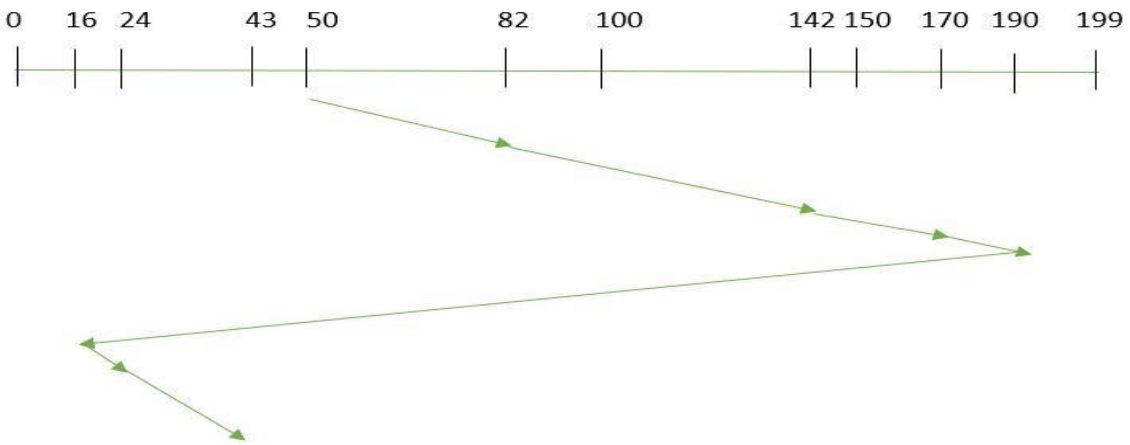
- More complex implementation.
- Still some delay for new requests in the opposite direction.

6. C-LOOK

Similar to C-SCAN, but head jumps back from last request directly to the first request without going to disk's physical end. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example: Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value"** So, the total overhead movement (total distance covered by the disk arm) is calculated as

= (190-50) + (190-16) + (43-16) = 341



Advantages

- Avoids unnecessary movement beyond last request.
- Uniform wait time distribution.

Disadvantages

- Not as simple as FCFS.
- Head jump still wastes time (though less than C-SCAN).