II DISK STRUCTURE

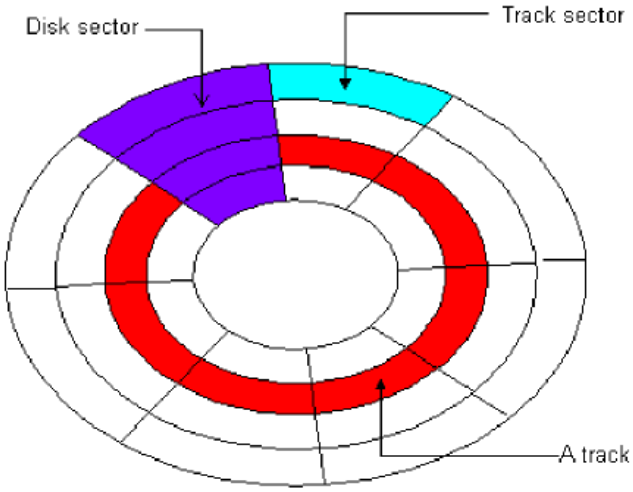
A hard disk is a memory storage device which looks like this:

The disk is divided into **tracks**. Each track is further divided into **sectors**. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity.

This is because the storage density is high in sectors of the inner tracks whereas the bits are sparsely arranged in sectors of the outer tracks.

Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity. Read-Write(R-W) head moves over the rotating hard disk.

It is this Read-Write head that performs all the read and write operations on the disk and hence, position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position.



Some important terms must be noted here:

1. **Seek time –** The time taken by the R-W head to reach the desired track from it's current position.
2. **Rotational latency –** Time taken by the sector to come under the R-W head.
3. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller time –** The processing time taken by the controller.
5. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

In questions, if the seek time and controller time is not mentioned, take them to be zero. If the amount of data to be transferred is not given, assume that no data is being transferred. Otherwise, calculate the time taken to transfer the given amount of data.

The average of rotational latency is taken when the current position of R-W head is not given. Because, the R-W may be already present at the desired position or it might take a whole rotation to get the desired sector under the R-W head. But, if the current position of the R-W head is given then the rotational latency must be calculated.

Example –

Consider a hard disk with:4 surfaces

64 tracks/surface

128 sectors/track

256 bytes/sector

1. **What is the capacity of the hard disk?**

   Disk capacity = surfaces * tracks/surface * sectors/track * bytes/sector

   Disk capacity = 4 * 64 * 128 * 256

   Disk capacity = 8 MB

2. **The disk is rotating at 3600 RPM, what is the data transfer rate?**

   60 sec -> 3600 rotations

   1 sec -> 60 rotations

   Data transfer rate = number of rotations per second * track capacity * number of surfaces (since 1 R-W head is used for each surface)

   Data transfer rate = 60 * 128 * 256 *

   4 Data transfer rate = 7.5 MB/sec

3. **The disk is rotating at 3600 RPM, what is the average access time?**

   Since, seek time, controller time and the amount of data to be transferred is not given, we consider all the three terms as 0.

   Therefore, Average Access time = Average rotational

   delay Rotational latency => 60 sec -> 3600 rotations 1 sec

   -> 60 rotations

   Rotational latency = (1/60) sec = 16.67 msec.

   Average Rotational latency = (16.67)/2

   = 8.33 msec.

   Average Access time = 8.33 msec.

4. **Disk Scheduling and Management**

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

Disk scheduling is important because:

Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled. Two or more request may be far from each other so can result in greater disk arm movement. Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

Seek Time: Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

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

Disk Access Time: Disk Access Time is:

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

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 Algorithms

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Advantages:

Every request gets a fair chance

No indefinite postponement

Disadvantages:

Does not try to optimize seek time

May not provide the best possible service

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in 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. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Advantages:

Average Response Time

Decreases Throughput increases

Disadvantages:

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

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 like 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.

Advantages:

High throughput

Low variance of response time Average response time

Disadvantages:

Long waiting time for requests for locations just visited by disk arm

3. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in *CSAN* 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 SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

Provides more uniform wait time compared to SCAN

4. **LOOK:** It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head 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.
5. **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm inspite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

1. FCFS Scheduling:

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. Consider, for

example, a disk queue with requests for I/O

I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67.

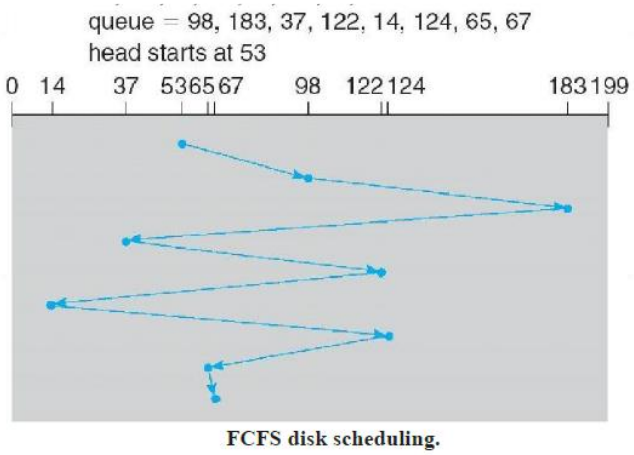
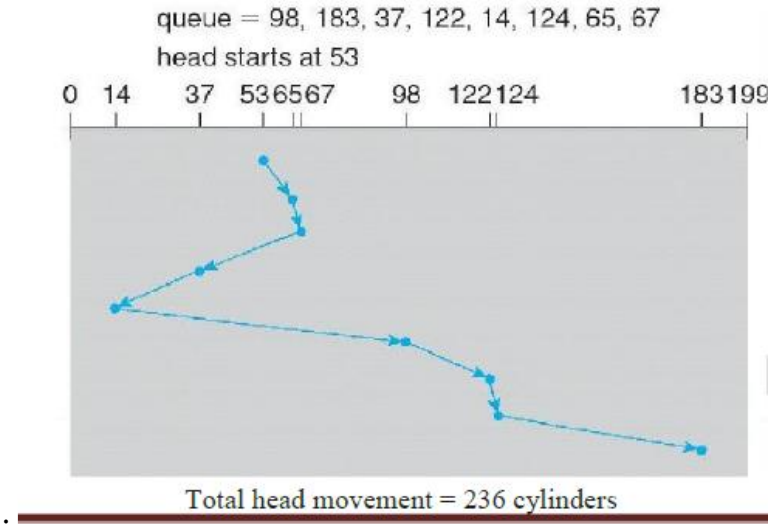


Fig : FCFS disk scheduling.

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, for **a total head movement of 640 cylinders**. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

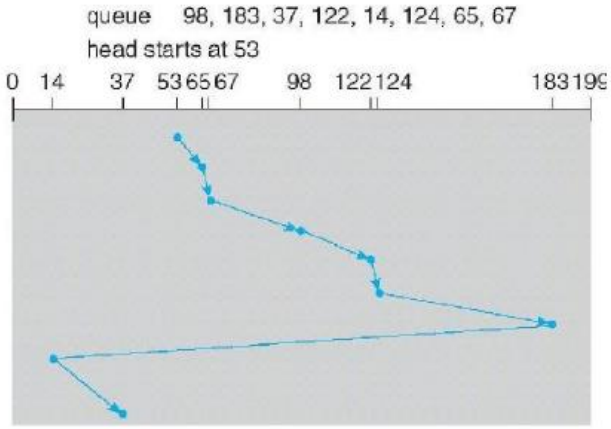
2. SSTF(shortest-seek-time-first)Scheduling

Service all the requests close to the current head position, before moving the head far away to service other requests. That is selects the request with the minimum seek time from the current head position



Total head movement = 236 cylinders

3.SCAN Scheduling

3. 

C-LOOK disk scheduling.

Disk Management

1. Disk Formatting:

Before a disk can store data, the sector is divided into various partitions. This process is called low- level formatting or physical formatting. It fills the disk with a special data structure for each sector. The data structure for a sector consists of

✓ Header,

✓ Data area (usually 512 bytes in size),and

✓ Trailer.

Error-Correcting Code (ECC).

This formatting enables the manufacturer to

Test the disk and

To initialize the mapping from logical block numbers

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps.

(a) The first step is **Partition** the disk into one or more groups of cylinders. Among the partitions, one partition can hold a copy of the OS‘s executable code, while another holds user files.

(b) The second step is **logical formatting** .The operating system 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.

2. Boot Block:

For a computer to start running-for instance, when it is powered up or rebooted-it needs to have an initial program to run. This initial program is called bootstrap program & it should be simple.

It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.

To do its job, the bootstrap program

1. Finds the operating system kernel on disk,
2. Loads that kernel into memory, and
3. Jumps to an initial address to begin the operating system execution.

The bootstrap is stored in read-only memory **(ROM).**

Advantages:

1. ROM needs no initialization.
2. It is at a fixed location that the processor can start executing when powered up or reset.
3. It cannot be infected by a computer virus. Since, ROM is read only.

The full bootstrap program is stored in a partition called the **boot blocks**, at a fixed location on the disk. A disk that has a boot partition is called a **boot disk or system disk**. The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code. **Bootstrap loader**: load the entire operating system from a non-fixed location on disk, and to start the operating system running.

4. Bad Blocks

The disk with defected sector is called as bad block.

Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: "Handled manually''

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: "sector sparing or forwarding"

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

A typical bad-sector transaction might be as follows:

1. The operating system tries to read logical block87.
2. The controller calculates the ECC and finds that the sector is bad.
3. It reports this finding to the operating system.
4. The next time that the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.

5. After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Method 3: "sector slipping"

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

Swap Space Management

Modern systems typically swap out pages as needed, rather than swapping out entire processes. Hence the swapping system is part of the virtual memory management system Managing swap space is obviously an important task for modern OS.

Swap-Space Use

The amount of swap space needed by an OS varies greatly according to how it is used. Some systems require an amount equal to physical RAM; some want a multiple of that; some want an amount equal to the amount by which virtual memory exceeds physical RAM, and some systems use little or none at all!

Some systems support multiple swap spaces on separate disks in order to speed up the virtual memory system.

The interchange of data between virtual memory and real memory is called as swapping and **space** on disk as "**swap space**".

Swap-Space Location

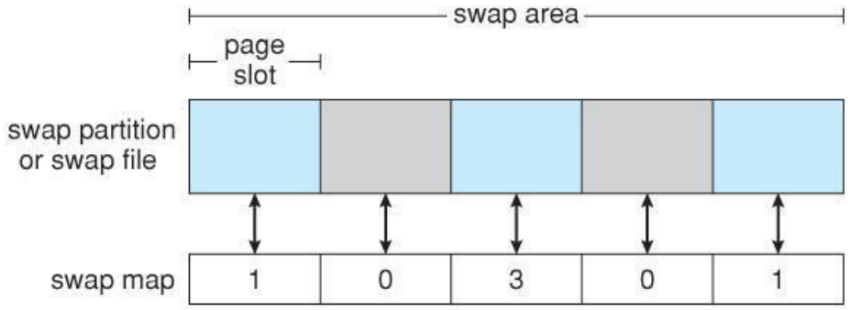
Swap space can be physically located in one of two locations:

As a large file which is part of the regular file system. This is easy to implement, but inefficient. Not only must the swap space be accessed through the directory system, the file is also subject to fragmentation issues. Caching the block location helps in finding the physical blocks, but that is not a complete fix.

As a raw partition, possibly on a separate or little-used disk. This allows the OS more control over swap space management, which is usually faster and more efficient. Fragmentation of swap space is generally not a big issue, as the space is re-initialized every time the system is rebooted. The downside of keeping swap space on a raw partition is that it can only be grown by repartitioning the hard drive.

Swap-Space Management: An Example

Historically OS swapped out entire processes as needed. Modern systems swap out only individual pages, and only as needed. In the mapping system shown below for Linux systems, a map of swap space is kept in memory, where each entry corresponds to a 4K block in the swap space. Zeros indicate free slots and non-zeros refer to how many processes have a mapping to that particular block (>1 for shared pages only.)



The data structures for swapping on Linux systems.