DISK SCHEDULING

Access Time = Seek Time + Rotational latency

Seek Time: The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.

Rotational latency: The rotational latency is the additional time for the disk to rotate the desired sector to the disk head.

The disk bandwidth is the total no:of bytes transferred, divided by the total time b/w the first request for service and the completion of the last transfer. We can improve both the access time and bandwidth by managing the 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 OS. The request specifies:

- → 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 no:of sectors to be transferred is

If desired disk drive and controller are available, the request is serviced immediately otherwise any new request for service will be placed in queue of pending requests for that drive. When request completed, OS chooses pending request to service next.

Following are the algorithms used:

1. FCFS scheduling algorithm
2. SSTF Scheduling algorithm
3. SCAN Scheduling algorithm
4. C-SCAN Scheduling algorithm
5. LOOK Scheduling algorithm

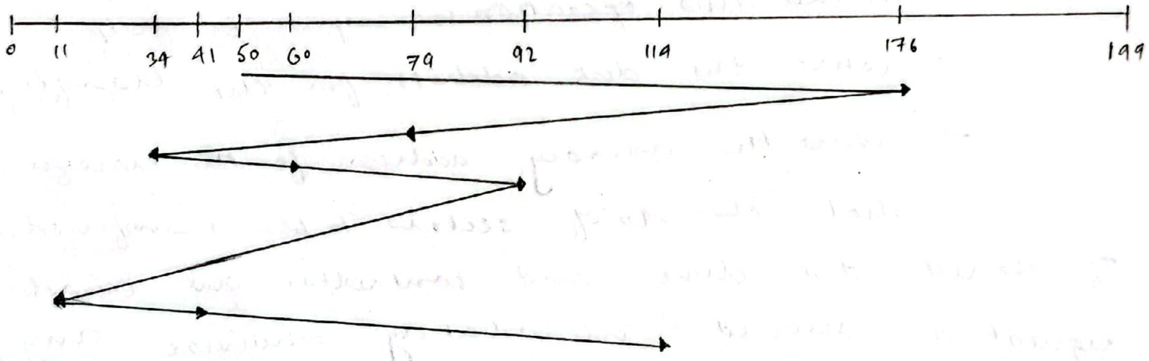
1. FCFS Scheduling Algorithm

FCFS is the simplest disk scheduling algorithm. The algori entertains requests in the order they arrive in the disk queue. Appears to be fair and does not have starvation..

Algorithm

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head
2. Take ~~one~~ tracks one by one in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance
4. Currently serviced track position now becomes new head posn.
5. Go to step 2 until all tracks in request array have not been serviced.

eg: Sequence : {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position : 50



Total seek count = (176 - 50) + (176 - 79) + (79 - 34) + (60 - 34) + (92 - 60) + (92 - 11) + (41 - 11) + (114 - 41) = 510

II. SSTF Algorithm

Basic idea is the tracks which are closer to current disk head position should be serviced first in order to minimize the seek operations.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested. 'head' is the position of diskhead
2. Find the positive distance of all tracks in the request array from head.
3. Find a track from the requested array which is not accessed / serviced yet and has minimum distance

from head.

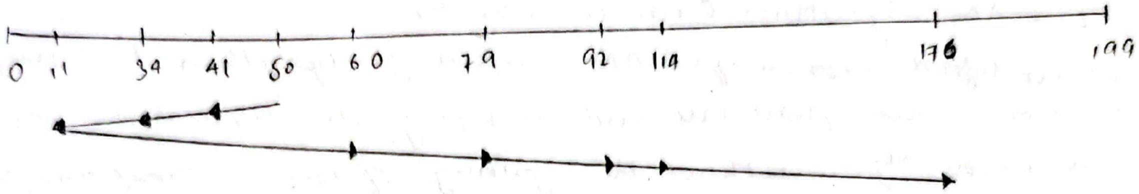
4. Increment total seek count with this distance.

5. currently serviced track position now becomes new head posn.

6. Go to step 2 until all tracks in request array have been not serviced.

eg: Sequence . { 176, 79, 34, 60, 92, 11, 41, 114 }

Initial head position: 50



Seek count: (50-41)+(41-34) + (34-11) +(60-11) +(79-60) +(92-79) + (114-92) + (176-114) = 204.

(iii) SCAN scheduling Algorithm (Elevator)

→ head starts from one end of the disk and moves towards the other end, servicing requests in b/w one by one and reach the other end. Then direction of head is reversed and the process continues as head continuously scan back and forth to access the disk. Requests at midrange are serviced more and those arriving behind the disk arm will have to wait.

Algorithm

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.

2. Let direction represents whether the head is moving towards left or right.

3. In the direction in which head is moving service all tracks one by one

4. Calculate the absolute distance of the track from the head.

5. Increment the total seek count with this distance.

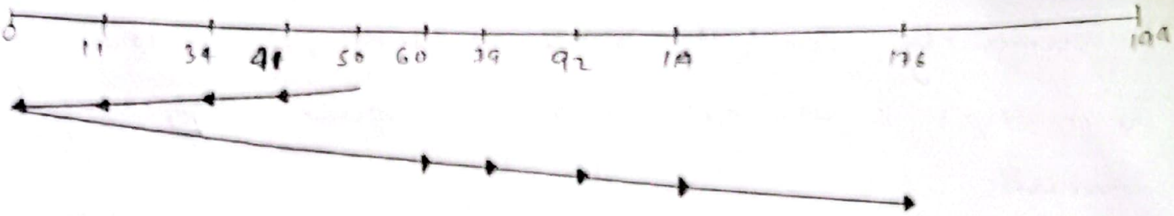
6. currently serviced track position now becomes new head posn.

7. Go to step 3 until we reach at one of the ends of the disk.

8. If we reach at the ends of the disk

8. If we reach the end of the disk reverse the direction and go to step 2 untill all tracks in request array have not been serviced.

eg: Sequence: {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position: 50   Start direction: Left



Total seek count: (50 - 41) + (41 - 34) + (34 - 11) + (11 - 0) + (60 - 0) + (79 - 60) + (92 - 79) + (114 - 92) + (176 - 114) = 226

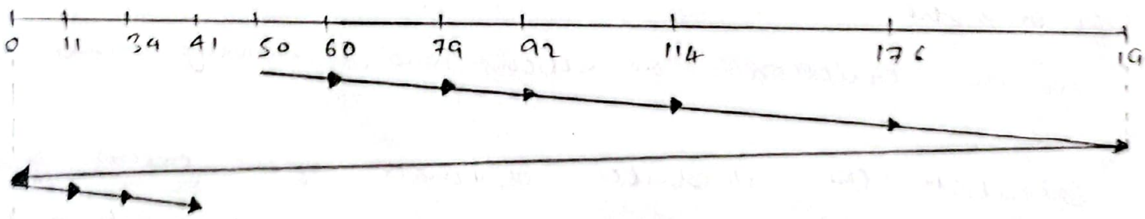
IV C-SCAN Algorithm (Circular Elevator)

→ modified version of SCAN scheduling algorithm that improves efficiency
→ moves head from one end servicing all request to one end and immediately returns to beginning of disk without servicing requests on return trip and starts again once reaches beginning.

Algorithm

1. Initialize request array and head.
2. Head services only in right direction from 0 to size of disk
3. While moving left, no requests are serviced.
4. When we reach beginning, reverse direction.
5. While moving right direction, it services tracks one by one. and while moving right direction calculate absolute distance of track from the head.
6. Increment total seek count and set currently serviced position as new head.
7. Go to step 6 until we reach right end of disk.
8. If we reach right end, reverse direction and go to step 3.

eg: Sequence: {176, 79, 34, 60, 92, 11, 41, 114}
Initial head: 50   Direction: right.



Seek count - (60-50) + (79-60) + (92-79) + (114-92) + (176-114) + (199-176) + (199-0) + (11-0) + (34-11) + (41-34) : 389

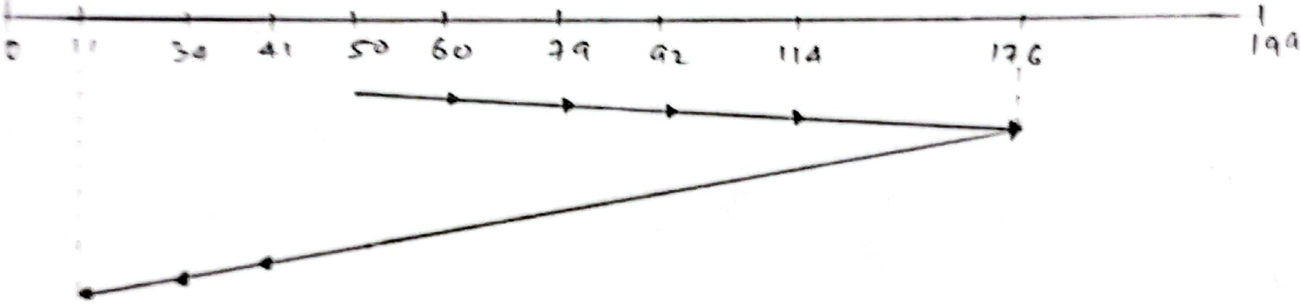
V LOOK Algorithm

→ variation of SCAN algorithm is used to reduce amount of time to access data on hard disk. Operates by scanning disk in a specific direction but instead of going all the way to the end, it reverses direction as soon as it reaches current direction.

Algorithm

1. Determine the initial direction of disk head movement.
2. Sort the pending disk requests in order in which they will be serviced.
3. Scan the disk in the chosen direction, servicing requests as they are encountered.
4. When the last request is serviced, reverse direction and continue scanning until all requests have been serviced.

Sequence: {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position: 50 Direction: right



Seek count: (60-50) + (79-60) + (92-79) + (114-92) + (176-114) + (176-41) + (41-34) + (34-11) = 291.

* CLOOK is a modified algorithm of LOOK algorithm.