SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

A.Y.: 2022-23   Class: S.Y.B.Tech   Sub: System Fundamentals

Experiment 10

(Disk Scheduling)

Aim: Implement Disk Scheduling Algorithms.

Theory: In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request.

TYPES OF 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

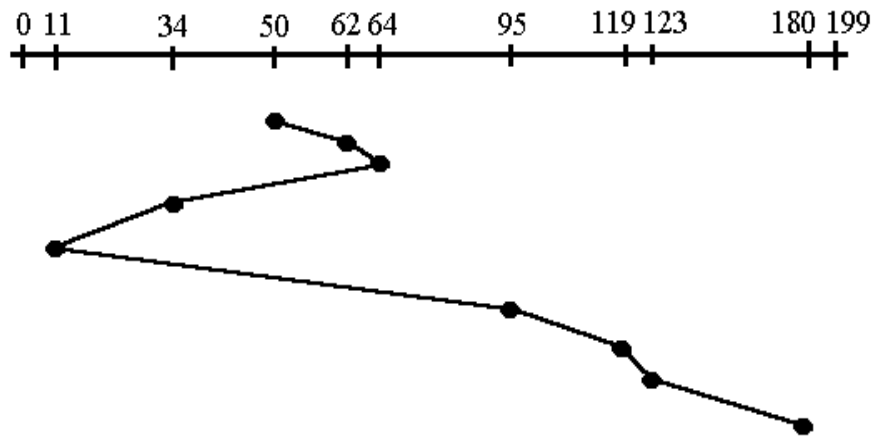 1. **First Come -First Serve (FCFS)**
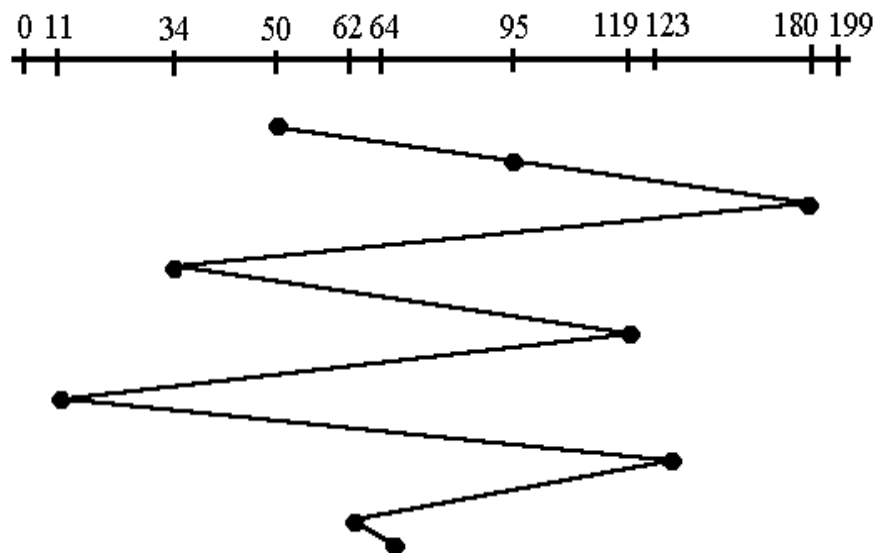
All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request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

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

A.Y.: 2022-23   **Class: S.Y.B.Tech**   **Sub:** System Fundamentals

soon see, this is the worse algorithm that one can use.

2. Shortest Seek Time First (SSTF)

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 process 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.

3. Elevator (SCAN)
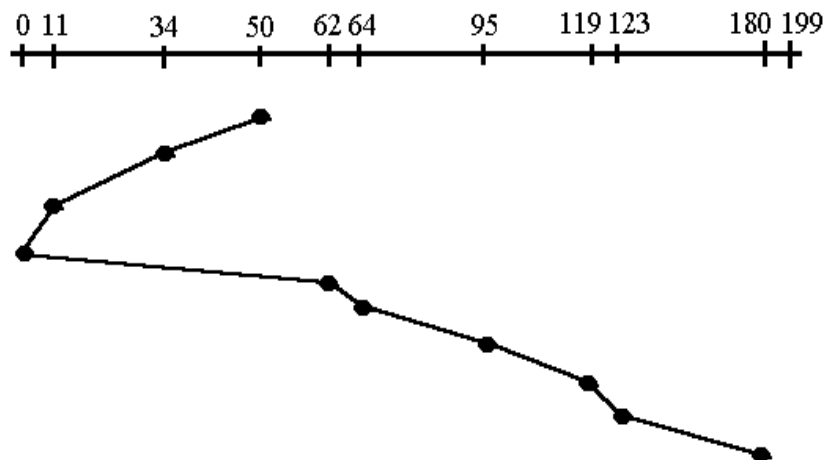
This approach works like an elevator does. It scans down towards the nearest end and

then when it hits the bottom it scans up servicing the requests that it didn't get going down. 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.

4. Circular Scan (C-SCAN)
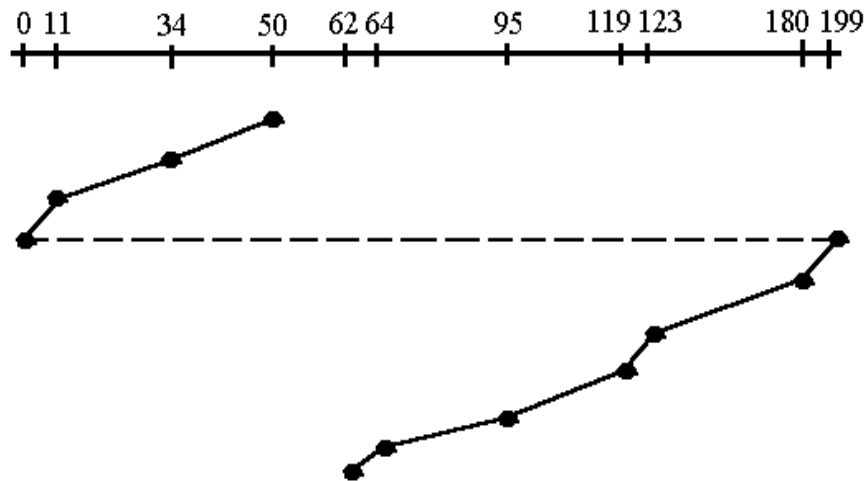
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.

5. C-LOOK
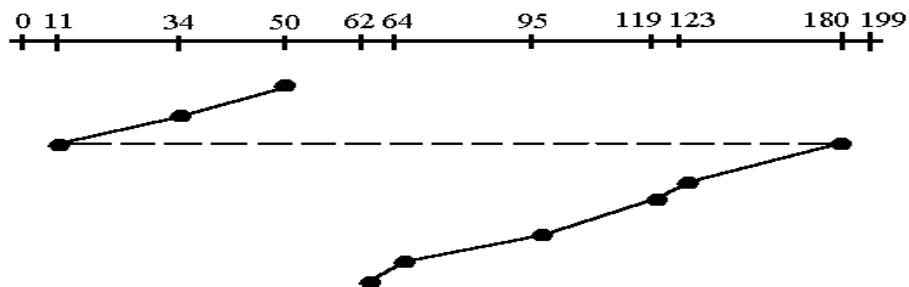
This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 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.

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

A.Y.: 2022-23   **Class: S.Y.B.Tech**   **Sub:** System Fundamentals

Lab Assignments to complete in this session

1. Disk requests come to disk driver for cylinders 10,22,20,2,40,6 and 38, in that order at a time when the disk drive is reading from cylinder 20. The seek time is 6 msec per cylinder. Compute the total seek time if the disk arm scheduling algorithm is. First come first served, SCAN, LOOK and C-SCAN.

Sol: Implement **FCFS** sequence will be $\Rightarrow$ 20,10,22,20,2,40,6,38

Total movement:$|20-10|+|10-22|+|22-20|+|20-2|+|2-40|+|40-6|+|6-38|=146$

So total seek time $=146\times6=876$msec

FCFS
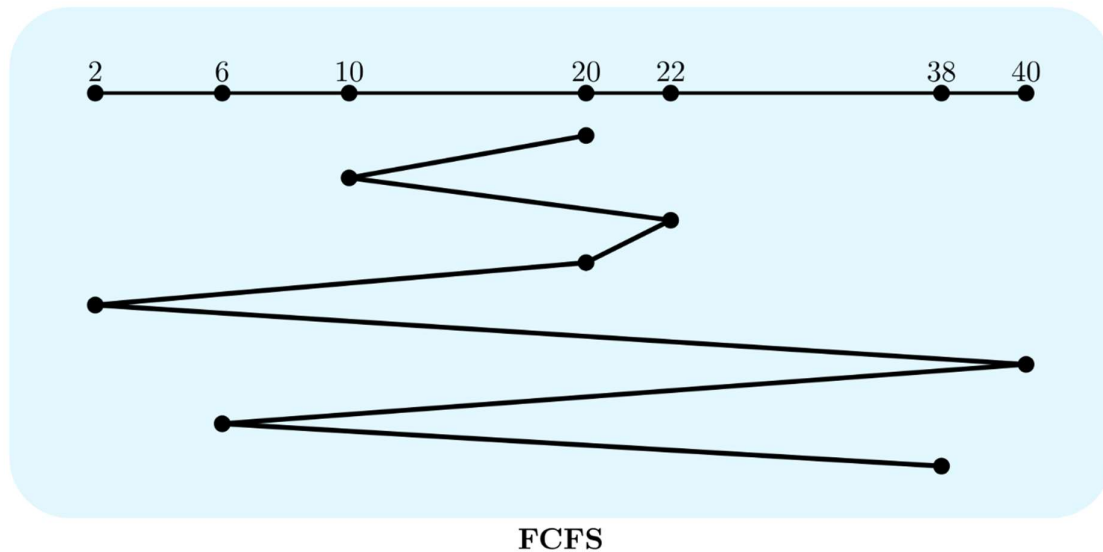
```python
n=int(input("Enter no. of requests:"))
req=[]
for i in range(n):
    x=int(input("Enter position on cylinder:"))
    req.append(x)
start=int(input("Enter the initial position of cyclinder:"))
```

```
Enter no. of requests:7
Enter position on cylinder:10
Enter position on cylinder:22
Enter position on cylinder:20
Enter position on cylinder:2
Enter position on cylinder:40
Enter position on cylinder:6
Enter position on cylinder:38
Enter the initial position of cyclinder:20
```

```python
# FCFS
seek=0
current_position = start
for i in range(n):
    seek_time = abs(current_position - req[i])
    seek += seek_time
    current_position = req[i]

# Print
print("Total Head Movements: ",seek)
print("Average seek time is: ",seek/n)
```

```
Total Head Movements:  146
Average seek time is:  20.857142857142858
```

```python
# SSTF
seek = 0
current_position = start
visited = [False] * n
for i in range(n):
    shortest_seek_time = float('inf')
    shortest_seek_index = 0
    for j in range(n):
        if not visited[j]:
            seek_time = abs(current_position - req[j])
            if seek_time < shortest_seek_time:
                shortest_seek_time = seek_time
                shortest_seek_index = j

    visited[shortest_seek_index] = True
    seek += shortest_seek_time
    current_position = req[shortest_seek_index]

print("Total Head Movements: ", seek)
print("Average seek time is: ",seek/n)
```

```
Total Head Movements:  60
Average seek time is:  8.571428571428571
```

```
#scan
direction = "right"
seek = 0
current_position = start
req.sort()
while req:
    if direction == "right":
        pending_req = [r for r in req if r >= current_position]
    else:
        pending_req = [r for r in req if r <= current_position]

    if not pending_req:
        direction = "right" if direction == "left" else "left"
        continue

    if direction == "right":
        next_req = min(pending_req)
    else:
        next_req = max(pending_req)

    seek += abs(next_req - current_position)
    current_position = next_req
    req.remove(next_req)

print("Total Head Movements: ", seek)
print("Average seek time is: ",seek/n)
```

```
Total Head Movements:  58
Average seek time is:  8.285714285714286
```

```python
def CSCAN(arr, head):
    seek_count = 0
    distance = 0
    cur_track = 0
    left = []
    right = []
    left.append(0)
    right.append(disk_size - 1)

    for i in range(size):
      if (arr[i] < head):
        left.append(arr[i])
      if (arr[i] > head):
        right.append(arr[i])
    left.sort()
    right.sort()

    for i in range(len(right)):
      cur_track = right[i]
      distance = abs(cur_track - head)
      seek_count += distance
      head = cur_track
    head = 0
    seek_count += (disk_size - 1)

    for i in range(len(left)):
      cur_track = left[i]
      distance = abs(cur_track - head)
      seek_count += distance
      head = cur_track

    print("Total number of seek operations =",seek_count)
    print("Average seek time",seek_count/size)
CSCAN(arr, head)
```

```
Total number of seek operations = 388
Average seek time 55.42857142857143
```

```python
#look
direction = "right"
def LOOK(arr, head, direction):

    seek_count = 0
    distance = 0
    cur_track = 0
    left = []
    right = []

    for i in range(size):
        if (arr[i] < head):
            left.append(arr[i])
        if (arr[i] > head):
            right.append(arr[i])

    left.sort()
    right.sort()
```

```python
    run = 2
    while (run):
        if (direction == "left"):
            for i in range(len(left) - 1, -1, -1):
                cur_track = left[i]
                distance = abs(cur_track - head)
                seek_count += distance
                head = cur_track
            direction = "right"

        elif (direction == "right"):
            for i in range(len(right)):
                cur_track = right[i]
                distance = abs(cur_track - head)
                seek_count += distance
                head = cur_track
            direction = "left"
        run -= 1

    print("Total number of seek operations =", seek_count)

LOOK(arr, head, direction)
```

```
Total number of seek operations = 58
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

A.Y.: 2022-23   **Class: S.Y.B.Tech**   **Sub:** System Fundamentals

```python
def CLOOK(arr, head):
    seek_count = 0
    distance = 0
    cur_track = 0
    left = []
    right = []

    for i in range(size):
      if (arr[i] < head):
        left.append(arr[i])
      if (arr[i] > head):
        right.append(arr[i])
    left.sort()
    right.sort()

    for i in range(len(right)):
      cur_track = right[i]
      distance = abs(cur_track - head)
      seek_count += distance
      head = cur_track
    seek_count += abs(head - left[0])
    head = left[0]
    for i in range(len(left)):
      cur_track = left[i]
      distance = abs(cur_track - head)
      seek_count += distance
      head = cur_track

    print("Total number of seek operations =",
      seek_count)

CLOOK(arr, head)
```

```
Total number of seek operations = 66
```