NAME: ALISTAIR SALDANHA
SAPID: 60009200024
BRANCH: K1

Experiment 12

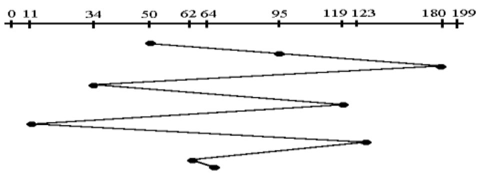
(Disk Scheduling)

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.

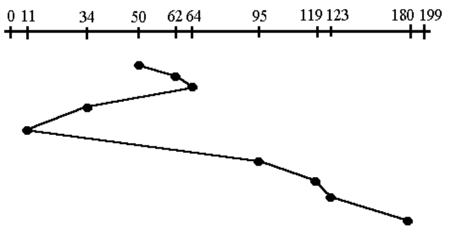
TYPES OF DISK SCHEDULING ALGORITHMS

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



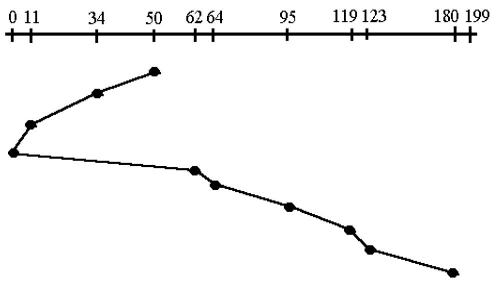
1. **First Come -First Serve (FCFS)**

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

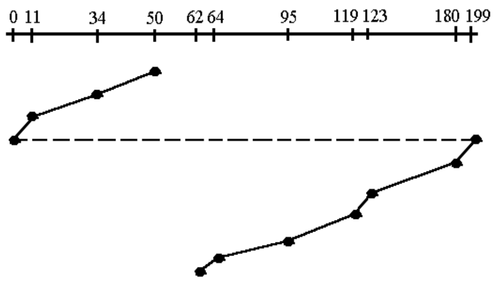


2. Shortest Seek Time First (SSTF)

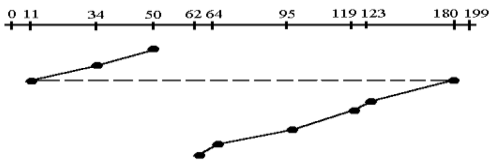
In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case 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 eachother 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.

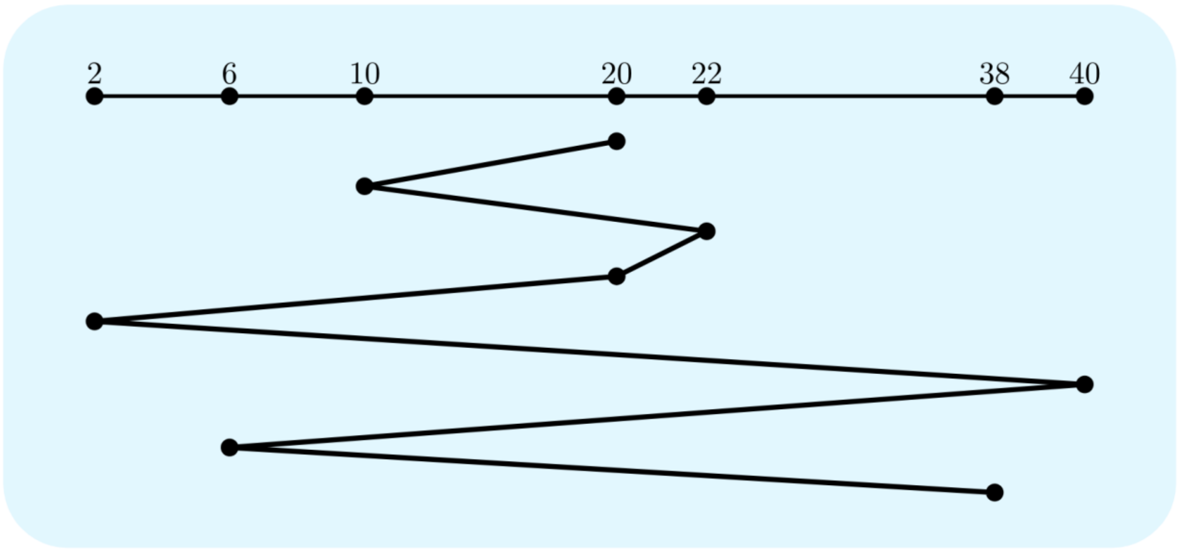


4. Circular Scan (C-SCAN)
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge 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.

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.
Sol: Implement **FCFS** sequence will be ⇒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 =146×6=876msec



FCFS

NAME: ALISTAIR SALDANHA

SAPID: 60009200024

BATCH: K1

EXPERIMENT-8

FCFS

In [25]:

```
def FCFS(req, head_start):
  req.insert(0, head_start)
  seek_time = 0
  print(req)
  for i in range(len(req)-1):
    print(f"Going for: ({req[i+1]} - {req[i]})")
    seek_time += abs(req[i+1] - req[i])
  print(f"Seek Time for FCFS is: {seek_time} ms")
```

SSTF

In [26]:

```
def SSTF(req, head_start):
  # Initialise seek time to 0
  seek_time=0
  # Append the head_start and sort the list in order to find nearest to headstart quickly
.
  req.append(head_start)
  req.sort()
  print(req)
  # i for left iteration, j for right iteration, k for current working process
  k = req.index(head_start)
  i = k-1
  j = k+1
  # Run the loop until i reaches the 0th index or j reaches the last index of req
  while(i > 0 or j != len(req)):
    # Calculate both the absolute values of numbers nearest to curr process and find mini
mum
    temp1 = abs(req[i] - req[k])
    temp2 = abs(req[j] - req[k])
    if(temp1 < temp2):
      print(f"Going for: ({req[i]} - {req[k]})")
      # Adding in seek time
      seek_time += temp1
      # change k pointer
      k = req.index(req[i])
      # Decrement i
      i-=1
    else:
      print(f"Going for: ({req[j]} - {req[k]})")
      seek_time += temp2
      # change k pointer
      k = req.index(req[j])
      # Decrement j
      j+=1
  print(f"Seek Time for SSTF is: {seek_time} ms")
```

SCAN

In [27]:

```
# If initial direction is given then move in that direction else check for nearest endpoi
nt from headstart
def SCAN(req, head_start):
  # Initialise seek time to 0
  seek_time=0
  # Append the head_start and sort the list in order to find nearest to headstart quickly
.
  req.append(head_start)
  req.sort()
  # Extra Points to add (0, 199)
  req.insert(0, 0)
  print(req)
  # i and j for iteration, k for current working process
  k = req.index(head_start)
  i = k-1
  j = k+1
  # Run the loop until i reaches the 0th index
  while(i >= 0):
    seek_time += abs(req[i] - req[k])
    print(f"Going for: ({req[i]} - {req[k]})")
    k = req.index(req[i])
    i-=1
  # Once left iteration is completed
  # Run the loop until j reaches the last index of req
  while(j<len(req)):
    seek_time += abs(req[j] - req[k])
    print(f"Going for: ({req[j]} - {req[k]})")
    k = req.index(req[j])
    j+=1
  print(f"Seek Time for SCAN is: {seek_time} ms")
```

C-SCAN

In [28]:

```
# If initial direction is given then move in that direction else check for nearest endpoi
nt from headstart
# the huge jump doesn't count as a head movement.
def C_SCAN(req, head_start):
  # Initialise seek time to 0
  seek_time=0
  # Append the head_start and sort the list in order to find nearest to headstart quickly
.
  req.append(head_start)
  req.sort()
  # Extra Points to add (0, 199)
  req.insert(0, 0)
  req.append(199)
  print(req)
  # i and j for iteration, k for current working process
  k = req.index(head_start)
  i = k-1
  j = len(req)-2 #2nd last position
  # Run the loop until i reaches the 0th index
  while(i >= 0):
    seek_time += abs(req[i] - req[k])
    print(f"Going for: ({req[i]} - {req[k]})")
    k = req.index(req[i])
    i -= 1
  # Once left iteration is completed shift k to last index of req and again iterate towar
ds left
  k = len(req)-1
  while(j>req.index(head_start)):
    seek_time += abs(req[j] - req[k])
    print(f"Going for: ({req[j]} - {req[k]})")
    k = req.index(req[j])
    j-=1
  print(f"Seek Time for C SCAN is: {seek_time} ms")
```

C-LOOK

In [29]:

```
# If initial direction is given then move in that direction else check for nearest endpoi
nt from headstart
# the huge jump doesn't count as a head movement.
def C_LOOK(req, head_start):
  # Initialise seek time to 0
  seek_time=0
  # Append the head_start and sort the list in order to find nearest to headstart quickly
.
  req.append(head_start)
  req.sort()
  print(req)
  # i and j for iteration, k for current working process
  k = req.index(head_start)
  i = k-1
  j = len(req)-2 #2nd last position
  # Run the loop until i reaches the 0th index
  while(i >= 0):
    seek_time += abs(req[i] - req[k])
    print(f"Going for: ({req[i]} - {req[k]})")
    k = req.index(req[i])
    i -= 1
  # Once left iteration is completed shift k to last index of req and again iterate towar
ds left
  k = len(req)-1
  while(j>req.index(head_start)):
    seek_time += abs(req[j] - req[k])
    print(f"Going for: ({req[j]} - {req[k]})")
    k = req.index(req[j])
    j-=1
  print(f"Seek Time for C LOOK is: {seek_time} ms")
```

INPUT

In [30]:

```
for ch in range(1, 6):
    if(ch == 1):
        print("\nFCFS\n")
        req_string = [95, 180, 34, 119, 11, 123, 62, 64]
        head_start = 50
        FCFS(req_string, head_start)
    elif(ch == 2):
        print("\nSSTF\n")
        req_string = [95, 180, 34, 119, 11, 123, 62, 64]
        head_start = 50
        SSTF(req_string, head_start)
    elif(ch == 3):
        print("\nSCAN\n")
        req_string = [95, 180, 34, 119, 11, 123, 62, 64]
        head_start = 50
        SCAN(req_string, head_start)
    elif(ch == 4):
        print("\nC-SCAN\n")
        req_string = [95, 180, 34, 119, 11, 123, 62, 64]
        head_start = 50
        C_SCAN(req_string, head_start)
    elif(ch == 5):
        print("\nC-LOOK\n")
        req_string = [95, 180, 34, 119, 11, 123, 62, 64]
        head_start = 50
        C_LOOK(req_string, head_start)
```

```
FCFS

[50, 95, 180, 34, 119, 11, 123, 62, 64]
Going for: (95 - 50)
Going for: (180 - 95)
Going for: (34 - 180)
Going for: (119 - 34)
Going for: (11 - 119)
Going for: (123 - 11)
Going for: (62 - 123)
Going for: (64 - 62)
Seek Time for FCFS is: 644 ms

SSTF

[11, 34, 50, 62, 64, 95, 119, 123, 180]
Going for: (62 - 50)
Going for: (64 - 62)
Going for: (34 - 64)
Going for: (11 - 34)
Going for: (95 - 11)
Going for: (119 - 95)
Going for: (123 - 119)
Going for: (180 - 123)
Seek Time for SSTF is: 236 ms

SCAN

[0, 11, 34, 50, 62, 64, 95, 119, 123, 180]
Going for: (34 - 50)
Going for: (11 - 34)
Going for: (0 - 11)
Going for: (62 - 0)
Going for: (64 - 62)
Going for: (95 - 64)
Going for: (119 - 95)
Going for: (123 - 119)
Going for: (180 - 123)
Seek Time for SCAN is: 230 ms

C-SCAN

[0, 11, 34, 50, 62, 64, 95, 119, 123, 180, 199]
Going for: (34 - 50)
Going for: (11 - 34)
Going for: (0 - 11)
Going for: (180 - 199)
Going for: (123 - 180)
Going for: (119 - 123)
Going for: (95 - 119)
Going for: (64 - 95)
Going for: (62 - 64)
Seek Time for C SCAN is: 187 ms

C-LOOK

[11, 34, 50, 62, 64, 95, 119, 123, 180]
Going for: (34 - 50)
Going for: (11 - 34)
Going for: (123 - 180)
Going for: (119 - 123)
Going for: (95 - 119)
Going for: (64 - 95)
Going for: (62 - 64)
Seek Time for C LOOK is: 157 ms
```