Experiment No.10
Implement disk scheduling algorithms SCAN.
Date of Performance:
Date of Submission:

Aim: To study and implement disk scheduling algorithms SCAN

Objective: The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

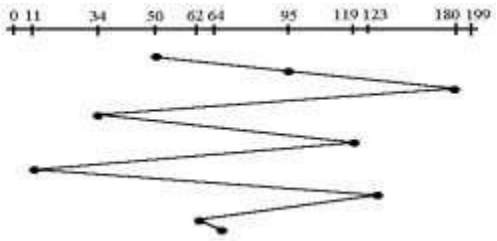
Theory:

TYPES OF DISK SCHEDULING ALGORITHMS

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms:

1. First Come-First Serve (FCFS)
2. Shortest Seek Time First (SSTF)
3. Elevator (SCAN)
4. Circular SCAN (C-SCAN)
5. C-LOOK

Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

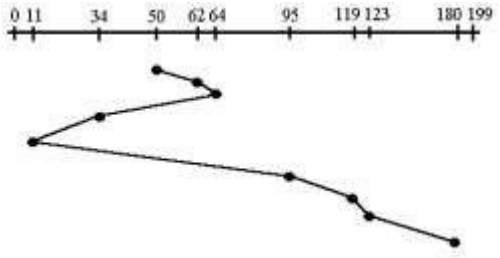


FCFS

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.



SSTF

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. SCAN

In the SCAN Disk Scheduling Algorithm, the head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reaching the other end. Then the direction of the head is reversed and the process continues as the head continuously scans back and forth to access the disk. So, this algorithm works as an elevator and is hence also known as the elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Algorithm

Step 1: Let the 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 the disk head.
Step 2: Let direction represents whether the head is moving towards left or right.
Step 3: In the direction in which the head is moving, service all tracks one by one.
Step 4: Calculate the absolute distance of the track from the head.
Step 5: Increment the total seek count with this distance.
Step 6: Currently serviced track position now becomes the new head position.
Step 7: Go to step 3 until we reach one of the ends of the disk.
Step 8: If we reach the end of the disk reverse the direction and go to step 2 until all tracks in the request array have not been serviced.

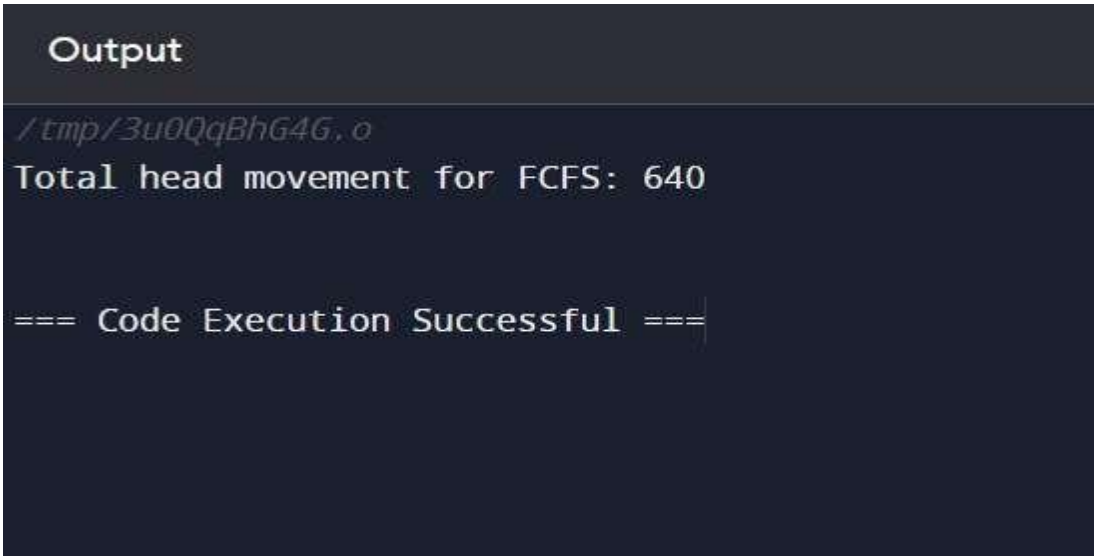
Result:

FCFS:-

```
#include <stdio.h>
 #include <stdlib.h>
 // Function to simulate FCFS disk scheduling
 void fcfs(int tracks[], int n, int head) {
 int total_head_movement = 0;
 int current_track = head;
 // Traverse the tracks in the order they appear
 for (int i = 0; i < n; i++) {
```

```
// Calculate the absolute distance to the next track
total_head_movement += abs(tracks[i]- current_track);
current_track = tracks[i];
}
printf("Total head movement for FCFS: %d\n", total_head_movement);
}
int main() {
int tracks[] = { 98, 183, 37, 122, 14, 124, 65, 67 }; // Example tracks
int n = sizeof(tracks) / sizeof(tracks[0]);
int head = 53; // Initial head position
// Simulate FCFS disk scheduling
fcfs(tracks, n, head);
return 0;
}
```

OUTPUT

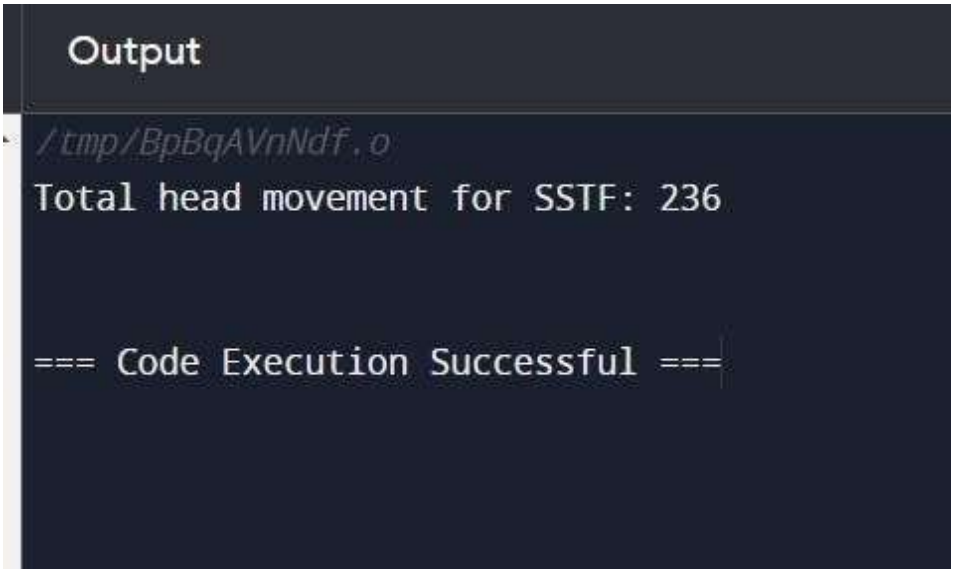


SSTF

```
#include <stdio.h>
 #include <stdlib.h>
 // Function to simulate SSTF disk scheduling
 void sstf(int tracks[], int n, int head) {
 int total_head_movement = 0;
 int current_track = head;
 // Repeat until all tracks are visited
while (n > 0) {
 int min_distance = abs(tracks[0]- current_track);
 int index = 0;
 // Find the track with the shortest seek time
 for (int i = 1; i < n; i++) {
 int distance = abs(tracks[i]- current_track);
 if (distance < min_distance) {
 min_distance = distance;
 index = i;
 }
 }
 // Update total head movement and current track
 total_head_movement += min_distance;
 current_track = tracks[index];
 // Remove the selected track from the array
 for (int j = index; j < n- 1; j++) {
```

```
 tracks[j] = tracks[j + 1];
 }
 n--;
 }
 printf("Total head movement for SSTF: %d\n", total_head_movement);
 }
int main() {
 int tracks[] = { 98, 183, 37, 122, 14, 124, 65, 67 }; // Example tracks
 int n = sizeof(tracks) / sizeof(tracks[0]);
 int head = 53; // Initial head position
 // Simulate SSTF disk scheduling
 sstf(tracks, n, head);
 return 0;
 }
```

OUTPUT



Conclusion: SCAN disk scheduling starts at one end, servicing requests along the way, and reverses direction. C SCAN disk scheduling adds predictability by ignoring requests while returning, reducing variability in waiting times. These algorithms are efficient, ensuring all requests eventually get served.