Experiment No.10
File Management & I/O Management Implement disk scheduling algorithms FCFS, SSTF.
Date of Performance:
Date of Submission:
Marks:
Sign:

Aim: To study and implement disk scheduling algorithms FCFS.

Objective:

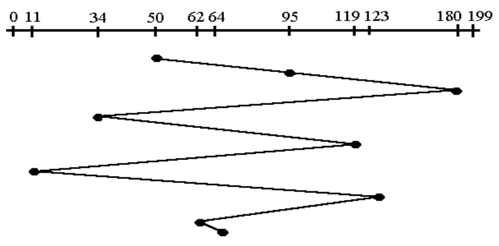
The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

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

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.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <limits.h>

#define NUM_REQUESTS 8

// Function prototypes

void FCFS(int requests[], int n, int initial_position);

void SSTF(int requests[], int n, int initial_position);

int main() {

   int requests[NUM_REQUESTS] = {98, 183, 37, 122, 14, 124, 65, 67};

   int initial_position = 53;

   printf("FCFS Disk Scheduling Algorithm:\n");

   FCFS(requests, NUM_REQUESTS, initial_position);

   printf("\nSSTF Disk Scheduling Algorithm:\n");

   SSTF(requests, NUM_REQUESTS, initial_position);
```

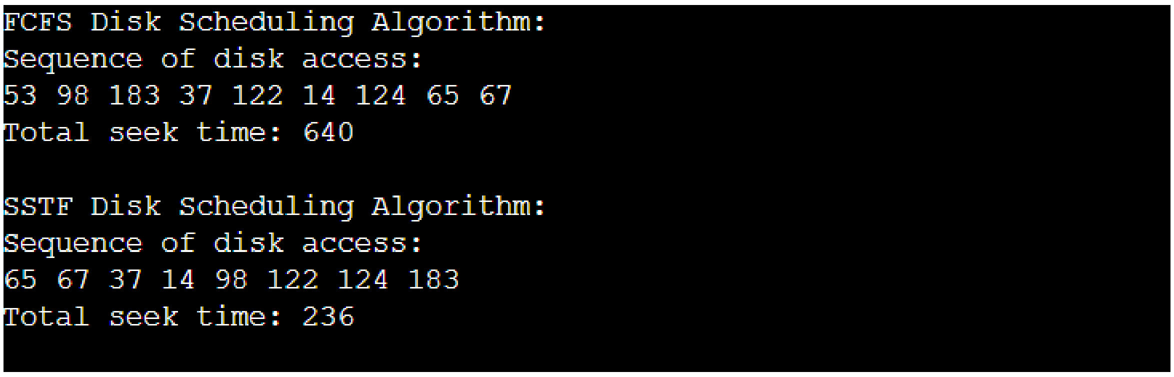
```
    return 0;
}
void FCFS(int requests[], int n, int initial_position) {
    printf("Sequence of disk access:\n");
    printf("%d ", initial_position);
    int total_seek_time = 0;
    for (int i = 0; i < n; i++) {
        total_seek_time += abs(initial_position - requests[i]);
        printf("%d ", requests[i]);
        initial_position = requests[i];
    }
    printf("\nTotal seek time: %d\n", total_seek_time);
}
void SSTF(int requests[], int n, int initial_position) {
    printf("Sequence of disk access:\n");

    bool visited[NUM_REQUESTS] = {false}; // Mark requests as unvisited
    int total_seek_time = 0;
    int current_position = initial_position;

    for (int i = 0; i < n; i++) {
        int min_seek_time = INT_MAX;
        int next_request_index = -1;

        for (int j = 0; j < n; j++) {
            if (!visited[j]) {
                int seek_time = abs(current_position - requests[j]);
                if (seek_time < min_seek_time) {
                    min_seek_time = seek_time;
                    next_request_index = j;
                }
            }
        }
        total_seek_time += min_seek_time;
        visited[next_request_index] = true;
        printf("%d ", requests[next_request_index]);
        current_position = requests[next_request_index];
    }
    printf("\nTotal seek time: %d\n", total_seek_time);
}
```

Output:



```
FCFS Disk Scheduling Algorithm:
Sequence of disk access:
53 98 183 37 122 14 124 65 67
Total seek time: 640

SSTF Disk Scheduling Algorithm:
Sequence of disk access:
65 67 37 14 98 122 124 183
Total seek time: 236
```

Conclusion:

Why is Disk Scheduling important?

Disk scheduling is important because it determines the order in which disk I/O requests are serviced, directly impacting system performance and user experience. Efficient disk scheduling algorithms aim to minimize disk head movement, reduce disk access times, and optimize disk throughput. By organizing disk accesses in a strategic manner, disk scheduling algorithms help improve overall system responsiveness, reduce latency, and enhance the efficiency of I/O operations. Without effective disk scheduling, system performance may suffer from increased disk contention, longer wait times, and inefficient resource utilization, leading to degraded performance and user dissatisfaction. Therefore, disk scheduling plays a crucial role in optimizing I/O performance and ensuring the efficient operation of computer systems.