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.

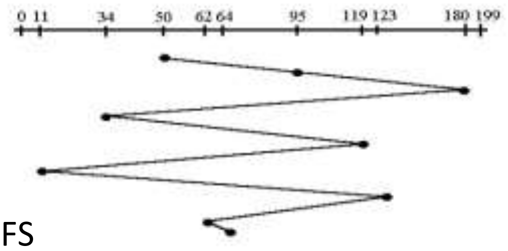
Objective:

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:

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

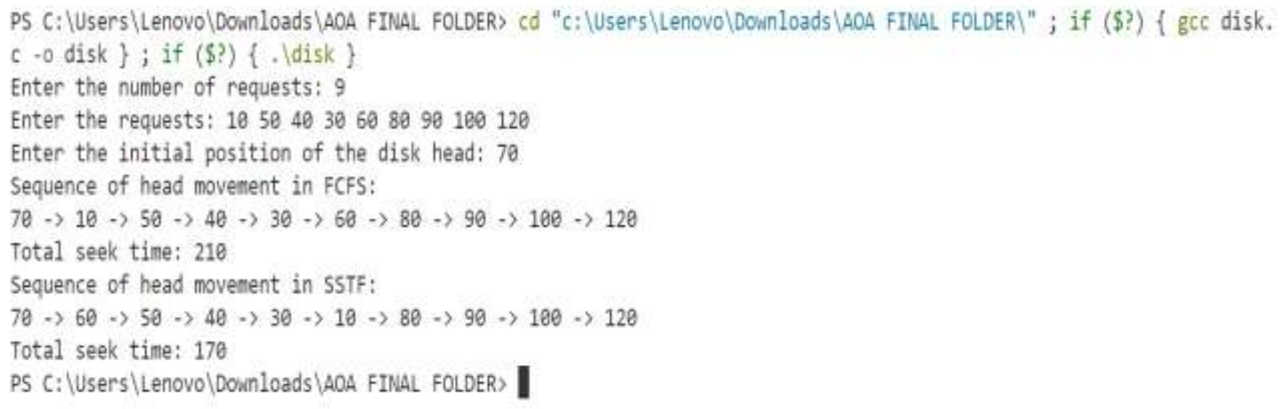
First Come -First Serve (FCFS)

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.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// Function to implement FCFS scheduling algorithm
void fcfs(int *requests, int n, int start) {
   int total_seek = 0;
   printf("Sequence of head movement in FCFS:\n");
   printf("%d -> ", start);
   total_seek += abs(start - requests[0]);
   printf("%d", requests[0]);
   for (int i = 1; i < n; ++i) {
      total_seek += abs(requests[i] - requests[i - 1]);
      printf(" -> %d", requests[i]);
   }
   printf("\nTotal seek time: %d\n", total_seek);
}

// Function to implement SSTF scheduling algorithm
void sstf(int *requests, int n, int start) {
   int total_seek = 0;
   int visited[n];
   for (int i = 0; i < n; ++i)
      visited[i] = 0;
   printf("Sequence of head movement in SSTF:\n");
   printf("%d -> ", start);
   for (int i = 0; i < n; ++i) {
      int min_seek = INT_MAX;
      int next_index = -1;
      for (int j = 0; j < n; ++j) {
         if (!visited[j]) {
            int seek = abs(start - requests[j]);
            if (seek < min_seek) {
               min_seek = seek;
               next_index = j;
            }
            }
        }
        total_seek += min_seek;
        printf("%d", requests[next_index]);
        visited[next_index] = 1;
        start = requests[next_index];
        if (i != n - 1)
            printf(" -> ");
    }
    printf("\nTotal seek time: %d\n", total_seek);
}

int main() {
    int n;
    printf("Enter the number of requests: ");
    scanf("%d", &n);

    int requests[n];
    printf("Enter the requests: ");
    for (int i = 0; i < n; ++i)
        scanf("%d", &requests[i]);

    int start;
    printf("Enter the initial position of the disk head: ");
    scanf("%d", &start);

    fcfs(requests, n, start);
    sstf(requests, n, start);

    return 0;
}
```

Output:


```
PS C:\Users\Lenovo\Downloads\AOA FINAL FOLDER> cd "c:\Users\Lenovo\Downloads\AOA FINAL FOLDER\" ; if ($?) { gcc disk.
c -o disk } ; if ($?) { .\disk }
Enter the number of requests: 9
Enter the requests: 10 50 40 30 60 80 90 100 120
Enter the initial position of the disk head: 70
Sequence of head movement in FCFS:
70 -> 10 -> 50 -> 40 -> 30 -> 60 -> 80 -> 90 -> 100 -> 120
Total seek time: 210
Sequence of head movement in SSTF:
70 -> 60 -> 50 -> 40 -> 30 -> 10 -> 80 -> 90 -> 100 -> 120
Total seek time: 170
PS C:\Users\Lenovo\Downloads\AOA FINAL FOLDER> █
```

Conclusion:
In conclusion, the implementation of disk scheduling algorithms such as First-Come, First-Served (FCFS) and Shortest Seek Time First (SSTF) presents distinct advantages and limitations, each tailored to specific system requirements and workload characteristics.
FCFS, being the simplest form of disk scheduling, ensures fairness in accessing the disk by servicing requests in the order they arrive. However, it suffers from poor performance in scenarios where there are significant variations in seek times, leading to high average response and turnaround times, especially with long seek distances or heavy loads.

Why is disk scheduling important?

Disk scheduling is crucial for efficient data access and retrieval in computer systems, particularly those with hard disk drives (HDDs). Here's why it's important:

Optimizing Disk Access Time: Disk scheduling algorithms determine the order in which disk I/O requests are serviced. By arranging these requests in an efficient manner, disk scheduling minimizes the time required to access data on the disk. This optimization is vital for improving overall system performance and responsiveness.
Reducing Disk Head Movement: Traditional HDDs consist of spinning platters with read/write heads that move across the disk's surface to access data. Disk scheduling algorithms aim to minimize the movement of these read/write heads by grouping or ordering disk requests logically. Minimizing head movement reduces seek time, which

is the time it takes for the read/write heads to reach the desired track on the disk, leading to faster data retrieval.

Fair Resource Allocation: In multi-user or multi-tasking environments, disk scheduling ensures fair access to disk resources among competing processes or users. By employing appropriate scheduling policies, the system can prioritize certain requests while preventing any single process from monopolizing disk access, thereby improving system responsiveness and fairness.

Preventing Starvation: Disk scheduling algorithms prevent starvation, ensuring that all pending disk I/O requests eventually get serviced. This is important to maintain system stability and prevent certain processes from being indefinitely delayed in accessing the disk.

Enhancing Throughput: Effective disk scheduling can improve the overall throughput of the system by efficiently managing disk I/O operations. By minimizing idle time and maximizing the utilization of disk resources, scheduling algorithms contribute to higher system throughput and better overall performance.