University of Asia Pacific

Department of Computer Science & Engineering

Lab Report 6

Course Title: Operating Systems Lab

Course Code : CSE 406

SUBMITTED BY

Md. Atikul Islam Atik

Section - B , Reg.ID – 21201063

Department of Computer Science & Engineering

(University of Asia Pacific)

Submission Date : 14 March, 2025

SUBMITTED TO

Atia Rahman Orthi

LECTURER

Department of Computer Science & Engineering

(University of Asia Pacific)

Problem Statement: Shortest Seek Time First Disk Scheduling Algorithm

Request Sequence: {176, 79, 34, 60, 92, 11, 41, 114}

Head Position = 50

In an operating system, disk scheduling is crucial for optimizing read/write operations. The Shortest Seek Time First (SSTF) algorithm selects the disk request that is closest to the current head position, reducing the total seek time. Given a sequence of disk requests and an initial head position, the goal is to determine the order of execution and calculate the total seek operations required.

Algorithm Steps:

The SSTF (Shortest Seek Time First) algorithm works by selecting the request that is closest to the current head position. It continues this process until all requests are served. The key idea is to reduce seek time by minimizing the movement of the disk head.

Initialize variables:

- Store the given request sequence.
- Set the initial position of the disk head.
- Initialize the total seek time to 0.

Process the requests:

- Find the request in the sequence that has the shortest seek time from the current head position.
- Move the disk head to that position.
- Add the seek time (absolute difference between current and next request) to the total seek count.
- Remove the processed request from the sequence.

Repeat until all requests are served.

Output the total seek operations.

Source Code:

```python
#Shortest Seek Time First: SSTF disk scheduling

def sstf(request_sequence, initial_head):
    current_head = initial_head
    sequence = [current_head]
    total_seektime = 0

    while request_sequence !=[]:
        request_sequence = sorted(request_sequence, key= lambda x: abs(x-current_head))
        next_request = request_sequence.pop(0)
        sequence.append(next_request)
        total_seektime += abs(next_request - current_head)
        current_head = next_request
    return total_seektime, sequence

def take_input():
    head = int(input("Enter the initial head position: "))
    n = int (input("Enter the total number of sequene: "))
    req_sequences = []

    for i in range(n):
        req = int(input(f'Enter sequence number  {i+1} : '))
        req_sequences.append(req)
    return req_sequences, head

def print_sequence(sequences):
    for i in range(len(sequences)):
        if i < len(sequences)-1:
            print(sequences[i], end=" ---> ")
        else:
            print(sequences[i], end="")

if __name__ == "__main__":
    inputs = take_input()
    req_sequences = inputs[0]
    head = inputs[1]

    res = sstf(req_sequences, head)
    print("Total Seek Operation", res[0])
    print_sequence(res[1])
```

Code LINK

Output Console:

```
 atik  os/lab5  main !?  v3.13.2  11:11 PM
 python -u "/home/atik/Codes/python/os/lab5/sstf.py"
Enter the initial head position: 50
Enter the total number of sequene: 8
Enter sequence number  1 : 176
Enter sequence number  2 : 79
Enter sequence number  3 : 34
Enter sequence number  4 : 60
Enter sequence number  5 : 92
Enter sequence number  6 : 11
Enter sequence number  7 : 41
Enter sequence number  8 : 114
Total Seek Operation 204
50 ---> 41 ---> 34 ---> 11 ---> 60 ---> 79 ---> 92 ---> 114 ---> 176

 atik  os/lab5  main !?  v3.13.2  11:12 PM
```

Advantages of Shortest Seek Time First Disk Scheduling Algorithms

Faster than FCFS:

- Unlike First Come First Serve (FCFS), SSTF reduces seek time significantly by choosing the nearest request.

Efficient for small request queues:

- Works well when the number of requests is small and uniformly distributed.

Reduces average seek time:

- Prioritizing close requests minimizes disk head movement.

Disadvantages of Shortest Seek Time First Disk Scheduling Algorithms

Starvation problem:

- Some requests may be delayed indefinitely if closer requests keep appearing.

Unfair scheduling:

- It does not guarantee fairness since some requests may be processed much later than others.

Not optimal for real-time systems:

- In real-time applications where fairness is crucial, SSTF may not be the best choice.

Conclusion:

The SSTF disk scheduling algorithm improves efficiency compared to FCFS by reducing seek time. However, it can lead to starvation issues if new requests constantly arrive close to the head's current position. Despite this, it is a widely used algorithm for optimizing disk performance, especially in systems with moderate disk request loads. For scenarios requiring fairness, algorithms like SCAN or CLOOK may be better alternatives.