Department of Computer Science & Engineering,
University of Asia Pacific.

Course Title : **Operating System Lab**
Course Code : **CSE 406**
Lab Report : **03**

Experiment Name : SSTF Disk Scheduling
Submission Date : 12.04.25

Submitted To:
Atia Rahman Orthi
Lecturer,
Department of CSE, UAP

Submitted By:
Debodipto Samadder
Reg: 21201199
Sec: B2

Problem Statement:

Implement the **Shortest Seek Time First (SSTF) Disk Scheduling** Algorithm.

Request Sequence = [0 14 41 53 65 67 98 122 124 183 199]

Head = 53

Solving Process Steps:

1. Set the current head position using initial_head.
2. Prepare an empty list seek_sequence to record the order of service requests.
3. Set total_seek to zero.
4. Create a copy of the requests list to preserve the original.
5. Loop while there are still remaining requests.
6. Use min() with a lambda function to find the request that has the shortest seek time (i.e., smallest absolute distance from the current head).
7. Determine how far the head must move.
8. Add this distance to total_seek.
9. Append the selected request to seek_sequence.
10. Move the head to the serviced request position.
11. Remove the serviced request from the remaining list.
12. After all requests are serviced, return the seek_sequence and total_seek.

Source Code:

```cpp
#include <iostream>
#include <cmath>
#include <limits.h>

using namespace std;

int main() {
    int requests[] = {0, 14, 41, 53, 65, 67, 98, 122, 124, 183, 199};
    int n = sizeof(requests) / sizeof(requests[0]);
    int visited[100] = {0};
    int head = 53;
    int total_seek = 0;

    for (int i = 0; i < n; i++) {
        int min_diff = INT_MAX;
        int index = -1;

        for (int j = 0; j < n; j++) {
            if (!visited[j]) {
                int diff = abs(head - requests[j]);
                if (diff < min_diff) {
                    min_diff = diff;
                    index = j;
                }
            }
        }

        visited[index] = 1;
        total_seek += abs(head - requests[index]);
        head = requests[index];
    }

    cout << "Total number of seek operations = " << total_seek << endl;
    return 0;
}
```

Output:

```
Total number of seek operations = 369

Process returned 0 (0x0)   execution time : 4.185 s
Press any key to continue.
```

Github Link:

https://github.com/debodipto/OS-lab-7/blob/main/OS%20Lab-7.cpp

Discussion:

Advantages:

- **Minimized Seek Time**: SSTF significantly reduces the average seek time compared to FCFS by always choosing the nearest request.
- **Improved Efficiency**: It enhances overall system throughput, especially when disk requests are non-uniform and clustered.
- **Simple Logic**: The algorithm is relatively simple to implement with a greedy approach.

Disadvantages:

- **Starvation Risk**: Requests located far from the current head position may suffer indefinite delays if closer requests keep arriving.
- **Lack of Real-Time Guarantee**: SSTF does not ensure predictable response times, making it less suitable for real-time systems.
- **Static Input Assumption**: The basic implementation works best when all disk requests are known in advance. Handling dynamically arriving requests would require additional logic.

Conclusion:

The SSTF (Shortest Seek Time First) disk scheduling algorithm improves performance by servicing the nearest request to the current head position, thus minimizing each seek operation. This results in reduced total head movement and better efficiency for systems with static or moderately scattered disk access patterns. However, its limitations, such as starvation and poor handling of dynamic inputs, make it less ideal for real-time or fairness-critical systems. For scenarios where all requests are known beforehand, SSTF remains a practical and effective choice.