UNIVERSITY OF ASIA PACIFIC

Course Title: Operating System Lab
Course Code: CSE 406

Date of Submission: 19th September,2025

Submitted by:

Name: Md.Sarjil Hasan

Reg: 22101168

Sec : D

Submitted to:

Atia Rahman Orthi

Lecturer

University of Asia Pacific

Lab 7:
SCAN Disk Scheduling Algorithm.

Problem Statement:

In modern computer systems, multiple processes often request disk I/O operations simultaneously. Since the disk arm can only service one request at a time, an efficient scheduling algorithm is required to minimize seek time and improve throughput. The SCAN algorithm (also called the Elevator Algorithm) is designed to optimize the disk head movement by reducing unnecessary back-and-forth seeking.

Objective:

1.To understand the working principle of the SCAN disk scheduling algorithm.
2.To implement SCAN for servicing disk I/O requests.
3.To calculate and compare total head movement with other scheduling algorithms (e.g., FCFS, SSTF).
4.To analyze advantages and disadvantages of SCAN in real-time systems.

Disk Scheduling Basics:

Seek Time: Time taken for the disk arm to move to the desired cylinder.
Rotational Latency: Delay while waiting for the disk platter to rotate.
Access Time: Combination of seek time and latency.

SCAN Algorithm:

1.The disk arm starts at a given initial head position and moves in one fixed direction (say towards higher tracks).
2.While moving, it services all pending requests in its path.
3.When the arm reaches the last track in that direction, it reverses and starts servicing requests on its way back.
4.This movement resembles an elevator, hence it is also known as the Elevator Algorithm.

Code:

ScanAlgo.cpp > main()

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

void scanDiskScheduling(vector<int> requests, int head, int diskSize, string direction) {
    vector<int> left, right;
    vector<int> seekSequence;
    int seekCount = 0;


    // if (direction == "left") {
    //     left.push_back(0);
    // } else if (direction == "right") {
    //     right.push_back(diskSize - 1);
    // }


    for (int req : requests) {
        if (req < head)
            left.push_back(req);
        else
            right.push_back(req);
    }


    sort(left.begin(), left.end());
    sort(right.begin(), right.end());

```

```
    if (direction == "left") {
        for (int i = left.size() - 1; i >= 0; i--) {
            seekSequence.push_back(left[i]);
            seekCount += abs(head - left[i]);
            head = left[i];
        }
        for (int i = 0; i < right.size(); i++) {
            seekSequence.push_back(right[i]);
            seekCount += abs(head - right[i]);
            head = right[i];
        }
    } else if (direction == "right") {
        for (int i = 0; i < right.size(); i++) {
            seekSequence.push_back(right[i]);
            seekCount += abs(head - right[i]);
            head = right[i];
        }
        for (int i = left.size() - 1; i >= 0; i--) {
            seekSequence.push_back(left[i]);
            seekCount += abs(head - left[i]);
            head = left[i];
        }
    }


    cout << "Seek Sequence: ";
    for (int pos : seekSequence) {
        cout << pos << " ";
    }
    cout << "\nTotal Seek Operations: " << seekCount << endl;
}
```

```
int main() {
    vector<int> requests = {14,20,29,40,50,110};
    int head = 29;
    int diskSize = 200;
    string direction = "right";

    scanDiskScheduling(requests, head, diskSize, direction);

    return 0;
}
```

Output:

For right:

```
PS C:\Users\Sarjil\Desktop\lab7&8> cd "c:\Users\Sarjil\Desktop\lab7&
Seek Sequence: 29 40 50 110 20 14
Total Seek Operations: 177
PS C:\Users\Sarjil\Desktop\lab7&8>
```

For left:

```
PS C:\Users\Sarjil\Desktop\lab7&8> cd "c:\Users\Sarjil\Desk
Seek Sequence: 20 14 29 40 50 110
Total Seek Operations: 111
PS C:\Users\Sarjil\Desktop\lab7&8>
```

Discussion:

The SCAN disk scheduling algorithm works on the principle of moving the disk head in one direction at a time, similar to the motion of an elevator. The head starts from its initial position and services all the requests along its path until it reaches the end of the disk. After reaching the end, it reverses direction and continues servicing the remaining requests. This approach helps in reducing the overall seek time compared to the First-Come-First-Serve (FCFS) algorithm, as the head does not move back and forth randomly but follows a systematic path. Unlike the Shortest Seek Time First (SSTF) algorithm, SCAN ensures fairness because no request is left unserved indefinitely, eliminating the problem of starvation. However, requests located at the extreme ends of the disk may experience longer waiting times, since the head has to complete servicing in one direction before reaching them. Overall, SCAN provides a balanced trade-off between efficiency and fairness, making it suitable for general-purpose disk scheduling.

Advantages:

- Provides better performance compared to FCFS by reducing total head movement.
- Avoids starvation (unlike SSTF), since every request is eventually serviced.
- Fair scheduling, as requests are serviced in the order of their cylinder position.

Disadvantages

- Response time may be high for requests at the extreme ends of the disk.
- Direction reversal increases average waiting time.
- Less efficient if requests are uniformly distributed across cylinders.

Conclusion:

The **SCAN disk scheduling algorithm** is efficient for minimizing seek time and ensuring fairness among processes. It balances performance between **FCFS** and **SSTF**, making it suitable for general-purpose systems. However, in systems requiring low latency (e.g., real-time), SCAN may not always be optimal.

GIT-HUB Link:

https://github.com/Sarjil-SarZzz/CSE406-LAB.git