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 8:
C-SCAN Disk Scheduling Algorithm.

Problem Statement:

In disk scheduling, multiple I/O requests arrive for different tracks on the disk. Efficient scheduling is required to minimize seek time and improve performance. The Circular SCAN (C-SCAN) disk scheduling algorithm is designed to provide a more uniform wait time by servicing requests in one direction only and then jumping back to the beginning of the disk without servicing any requests on the return.

Objective:

- To implement and analyze the C-SCAN disk scheduling algorithm.
- To calculate the total head movement and average seek time.
- To understand how C-SCAN differs from other algorithms like FCFS, SSTF, and SCAN.

Algorithm (Steps)

1. Take the number of requests, initial head position, and list of track requests.
2. Sort all the requests.
3. Depending on the head position and chosen direction (left or right):
    - If moving right: Service all requests greater than the head, then move to the maximum track, jump to track 0, and continue servicing the remaining requests.
    - If moving left: Service all requests less than the head, then move to the minimum track (0), jump to the maximum track, and continue servicing the remaining requests.
4. Calculate total head movement as the sum of all head jumps.
5. Compute average seek time = total head movement / number of requests.

Code:

CscanDisk.cpp > main()

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, head;
    cout << "Enter number of requests: ";
    cin >> n;

    cout << "Enter head position: ";
    cin >> head;

    cout << "Enter direction (left/right): ";
    string direction;
    cin >> direction;

    vector<int> requests(n);
    cout << "Enter requests: ";
    for (int i = 0; i < n; i++) {
        cin >> requests[i];
    }


    sort(requests.begin(), requests.end());

    vector<int> order;
    int seek_count = 0;
    int pos = lower_bound(requests.begin(), requests.end(), head) - requests.begin();

    if (direction == "right") {

        for (int i = pos; i < n; i++) {
            order.push_back(requests[i]);
            seek_count += abs(head - requests[i]);
            head = requests[i];
        }

```

```
        if (pos > 0) {
            seek_count += abs(head - requests[0]);
            head = requests[0];

            for (int i = 0; i < pos; i++) {
                order.push_back(requests[i]);
                seek_count += abs(head - requests[i]);
                head = requests[i];
            }
        }

    } else {
        for (int i = pos - 1; i >= 0; i--) {
            order.push_back(requests[i]);
            seek_count += abs(head - requests[i]);
            head = requests[i];
        }


        if (pos < n) {
            seek_count += abs(head - requests[n - 1]);
            head = requests[n - 1];

            for (int i = n - 1; i >= pos; i--) {
                order.push_back(requests[i]);
                seek_count += abs(head - requests[i]);
                head = requests[i];
            }
        }
    }
```

```
    cout << "\nSeek Sequence: ";
    for (int r : order) cout << r << " ";
    cout << "\nTotal Seek Operations: " << seek_count << "\n";

    return 0;
}
```

Output:

```
PS C:\Users\Sarjil\Desktop\lab7&8> cd "c:\Users\Sarjil\Des
Enter number of requests: 6
Enter head position: 29
Enter direction (left/right): right
Enter requests: 14 20 29 40 50 110

Seek Sequence: 29 40 50 110 14 20
Total Seek Operations: 183
PS C:\Users\Sarjil\Desktop\lab7&8>
```

Discussion:

In this experiment, the C-SCAN disk scheduling algorithm was implemented and analyzed. Unlike FCFS or SSTF, which may cause uneven waiting times, C-SCAN provides fairness by servicing requests only in one direction and then wrapping around to the beginning of the disk. This ensures that all requests are treated in a cyclic manner, which makes the waiting time more uniform. From the execution example, it was observed that the head first services all requests on one side of the head, then moves to the end of the disk, and jumps back to the start without servicing any requests during the return. This additional jump may increase the total head movement compared to SCAN, but it avoids starvation and ensures that requests near both ends of the disk are serviced regularly. Therefore, C-SCAN offers a balanced trade-off between efficiency and fairness, making it suitable for real-time systems where consistent response times are more important than minimizing every single seek operation.

Advantages:

- Provides **more uniform wait times** compared to SCAN.
- Reduces the chance of starvation for requests at the edges.
- Fair scheduling because all tracks are serviced in cyclic order.

Disadvantages:

- Slightly more **head movement** compared to SCAN due to the jump.
- Less efficient if requests are highly concentrated in one region.
- May waste time in the jump when few requests are pending on one side.

Conclusion:

The C-SCAN disk scheduling algorithm ensures fairness by servicing requests in one direction and wrapping around to the start, preventing starvation. Although it may cause slightly higher seek time compared to SCAN, it provides more consistent response times across requests.

GIT-HUB Link:

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