OSL Assignment 8

Name: Kumar Deshmukh

Roll No: 33143

Batch: M9

Title: Disk Scheduling Algorithms

Problem Statement:

Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

Theory:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus, other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

Seek Time:
Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So, the disk scheduling algorithm that gives minimum average seek time is better.

Rotational Latency:

Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

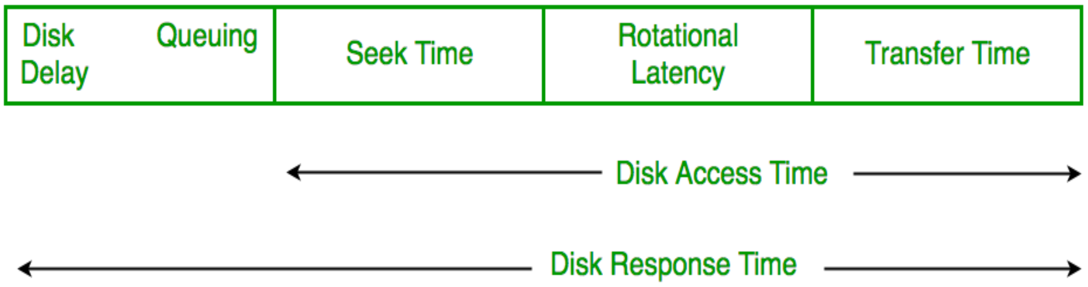
Transfer Time:

Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time:

Disk Access Time is:

Disk Access Time = (Seek Time) + (Rotational Latency) + (Transfer Time)



Disk Response Time:

Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So, the disk scheduling algorithm that gives minimum variance response time is better.

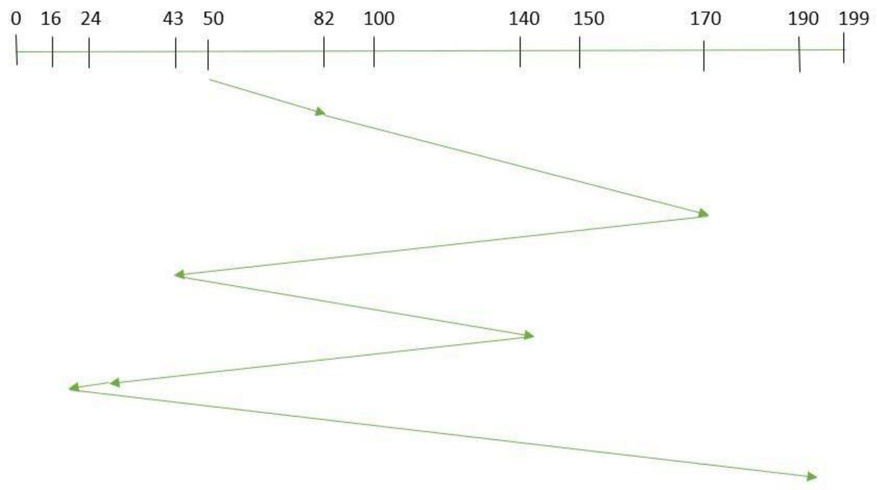
Disk Scheduling Algorithms

1) FCFS:

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.
Let us understand this with the help of an example.

Example:

Suppose the order of request is: (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50



So, total seek time:
= (82-50) +(170-82) +(170-43) +(140-43) +(140-24) +(24-16) +(190-16)
= 642

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

2) SSTF:

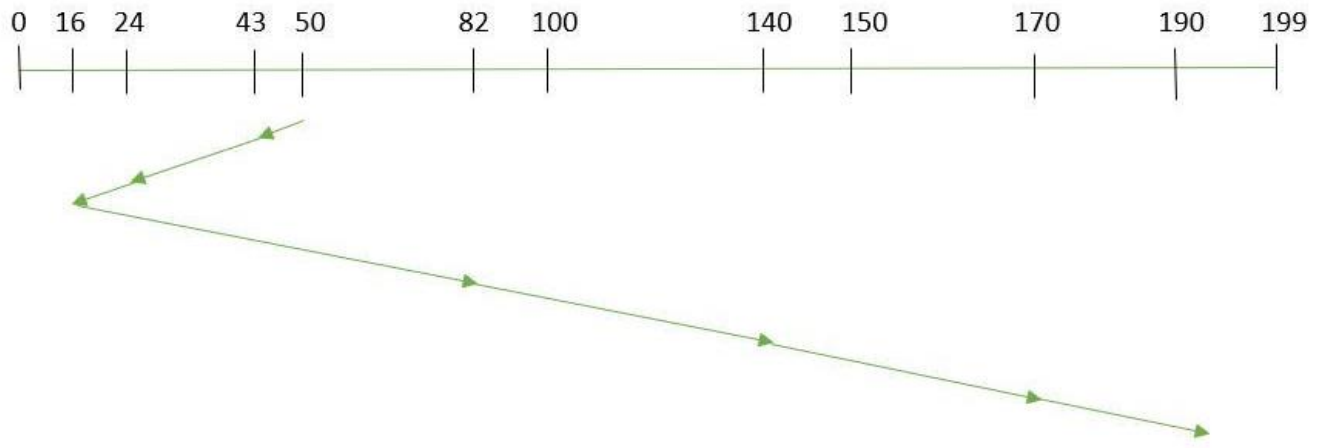
In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Let us understand this with the help of an example.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)

And current position of Read/Write head is : 50



So, total seek time:

= (50-43) +(43-24) +(24-16) +(82-16) +(140-82) +(170-40) +(190-170)

= 208

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

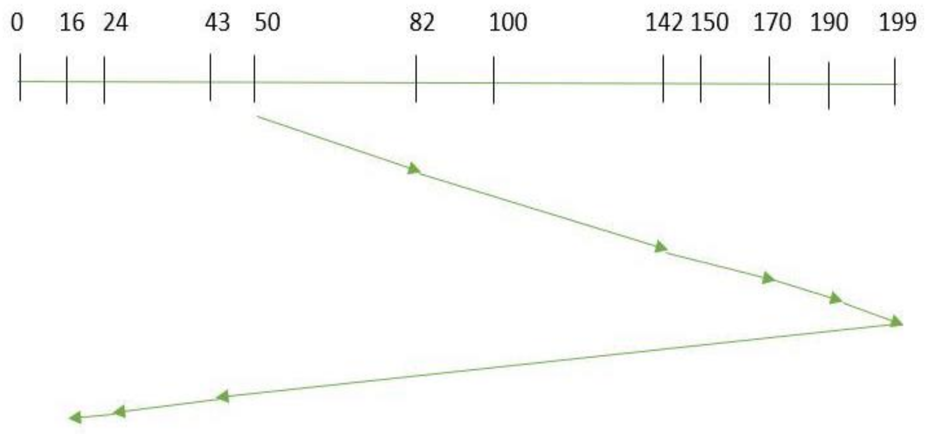
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

3) SCAN:

In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"Towards the larger value".**



Therefore, the seek time is calculated as:

= (199-50) + (199-16)
= 332

Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

4) C-SCAN:

In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

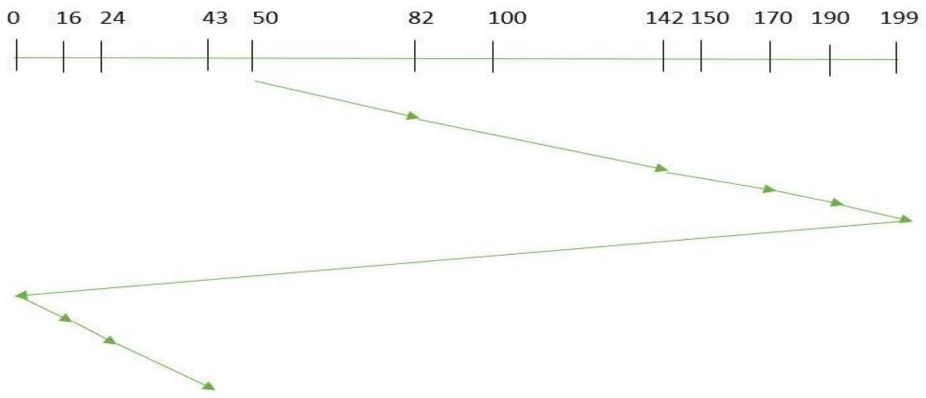
These situations are avoided in *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move

"Towards the larger value".

Seek time is calculated as:



= (199-50)+(199-0)+(43-0)

= 391

Advantages:

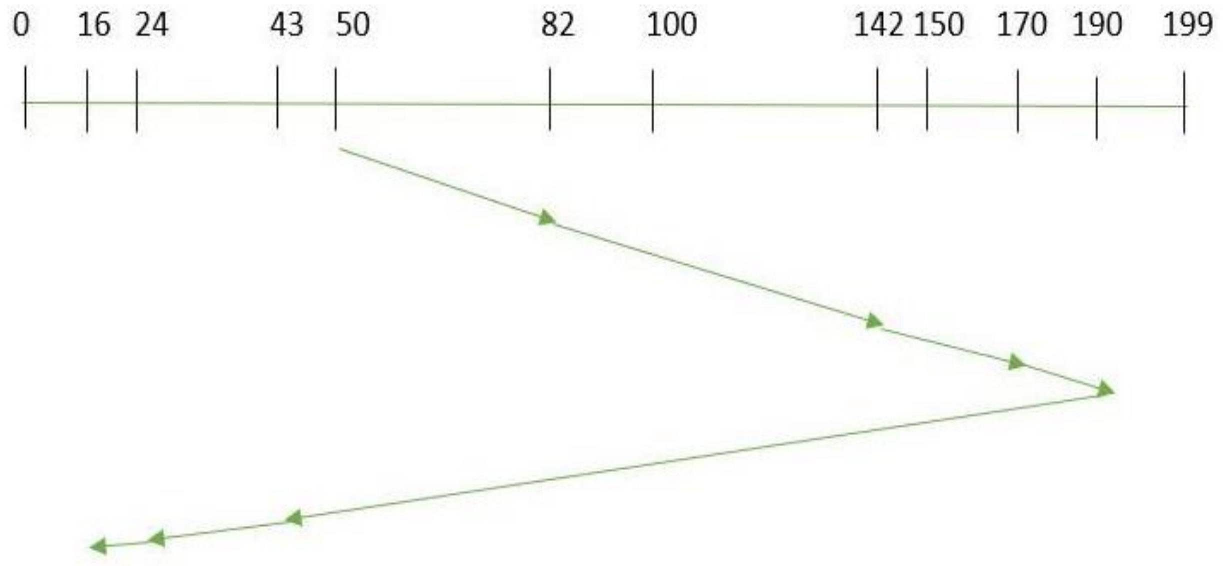
- Provides more uniform wait time compared to SCAN

5) LOOK:

It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"Towards the larger value".**



So, the seek time is calculated as:

= (190-50)+(190-16)

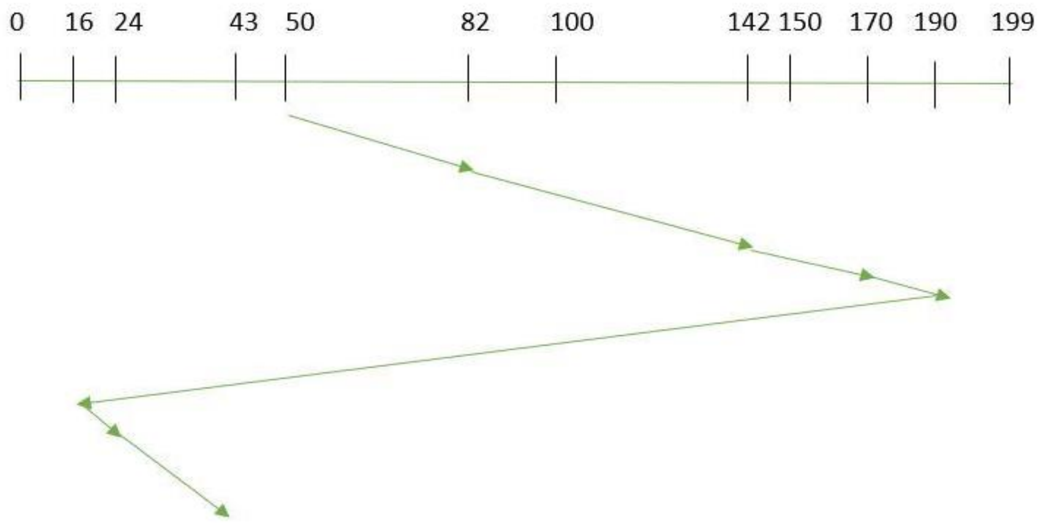
= 314

6) C-LOOK:

As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"Towards the larger value"**



So, the seek time is calculated as:

= (190-50) +(190-16) +(43-16)

= 341

Code:

```
#include <bits/stdc++.h>
#define INF INT_MAX

using namespace std;

vector<int> arr;
int length, n, current;

void SSTF()
{
    int total_track_movement = 0;
    int tempCurrent = current;
    vector<int> isVisited(arr.size(), 0);
    cout << "\nSSTF order::" << endl;
    cout << tempCurrent;
    int min_diff = INF;
    int closest_index;

    for (int j = 0; j < n; j++)
    {
        for (int i = 0; i < n; i++)
        {
            if (isVisited[i] == 0)
            {
                if (abs(tempCurrent - arr[i]) < min_diff)
                {
                    closest_index = i;
                    min_diff = abs(tempCurrent - arr[i]);
                }
            }
        }

        isVisited[closest_index] = 1;
        total_track_movement += abs(tempCurrent - arr[closest_index]);
        tempCurrent = arr[closest_index];
        min_diff = INF;
        cout << " " << tempCurrent;
    }
    cout << "\nTotal Track movement = " << total_track_movement << "\n";
}

void SCAN()
{

    int total_track_movement = 0;
    sort(arr.begin(), arr.end());
    total_track_movement = length - 1 - current;

    if (current > arr[0])
```

```
        total_track_movement += length - 1 - arr[0];

    cout << "\nSCAN order::" << endl;
    cout << current;

    for (int i = 0; i < n; i++)
    {
        if (arr[i] > current)
            cout << " " << arr[i];
    }

    for (int i = n - 1; i >= 0; i--)
    {
        if (arr[i] < current)
            cout << " " << arr[i];
    }
    cout << "\nTotal Track movement = " << total_track_movement << "\n";
}

void CLOOK()
{
    int total_track_movement = 0;
    sort(arr.begin(), arr.end());
    total_track_movement = arr[n - 1] - current;

    if (arr[0] < current)
    {
        total_track_movement += arr[n - 1] - arr[0];
    }
    cout << "\nC-LOOK order::" << endl;
    cout << current;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] > current)
            cout << " " << arr[i];
    }

    int i;

    for (i = 0; arr[i] < current; i++)
    {
        cout << " " << arr[i];
    }

    if (arr[0] < current)
        total_track_movement += arr[i - 1] - arr[0];

    cout << "\nTotal Track movement = " << total_track_movement << "\n";
}

int main()
{
    int choice;
    cout << "Total number of Requests: ";
    cin >> n;
    cout << "Enter Requests:" << endl;
    for (int i = 0; i < n; i++)
    {
        int m;
        cin >> m;
        arr.push_back(m);
    }

    cout << "Total length of the track: ";
    cin >> length;

    cout << "Current position of read-write head: ";
    cin >> current;

    do
    {
        cout << "\nMENU ";
        cout << "\t(1) SSTF" << endl;
        cout << "\t(2) SCAN" << endl;
        cout << "\t(3) C-LOOK" << endl;
        cout << "\t(4) EXIT" << endl;

        cout << "\nChoice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            SSTF();
            break;
        case 2:
            SCAN();
            break;
        case 3:
            CLOOK();
            break;
        case 4:
            break;
        }
    } while (choice != 4);
    cout << "Exited Program" << endl;
    return 0;
}
```

Output:

```
(base) kumar@pop-os:~/Desktop/OS/Assignments$ g++ Assig8.cpp
(base) kumar@pop-os:~/Desktop/OS/Assignments$ ./a.out
Total number of Requests: 7
Enter Requests:
72 160 33 130 14 6 180
Total length of the track: 200
Current position of read-write head: 50

MENU    (1) SSTF
        (2) SCAN
        (3) C-LOOK
        (4) EXIT

Choice: 1

SSTF order::
50 33 14 6 72 130 160 180
Total Track movement = 218

MENU    (1) SSTF
        (2) SCAN
        (3) C-LOOK
        (4) EXIT

Choice: 2

SCAN order::
50 72 130 160 180 33 14 6
Total Track movement = 342

MENU    (1) SSTF
        (2) SCAN
        (3) C-LOOK
        (4) EXIT

Choice: 3

C-LOOK order::
50 72 130 160 180 6 14 33
Total Track movement = 331

MENU    (1) SSTF
        (2) SCAN
        (3) C-LOOK
        (4) EXIT

Choice: 4
Exited Program
(base) kumar@pop-os:~/Desktop/OS/Assignments$
```

Conclusion:

We studied and implemented 3 Disk Scheduling Algorithms:

SSTF, SCAN & C-Look