**OPERATING SYSTEMS**

LAB EXPERIMENT - 10

Submitted To: Prashant Giridhar Shambharkar Sir.

Submitted By: Manav Agrawal (2K19/CO/216)

---------------------------------------------------------------------

Aim:

Write a C program to simulate disk scheduling algorithms. a) First Come First Serve (FCFS) b) SCAN

Introduction:

Disk Scheduling Algorithms:

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:

1. 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.
2. Two or more request may be far from each other so can result in greater disk arm movement.
3. 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 = 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.

The following are the disk scheduling algorithms which we will be implementing:

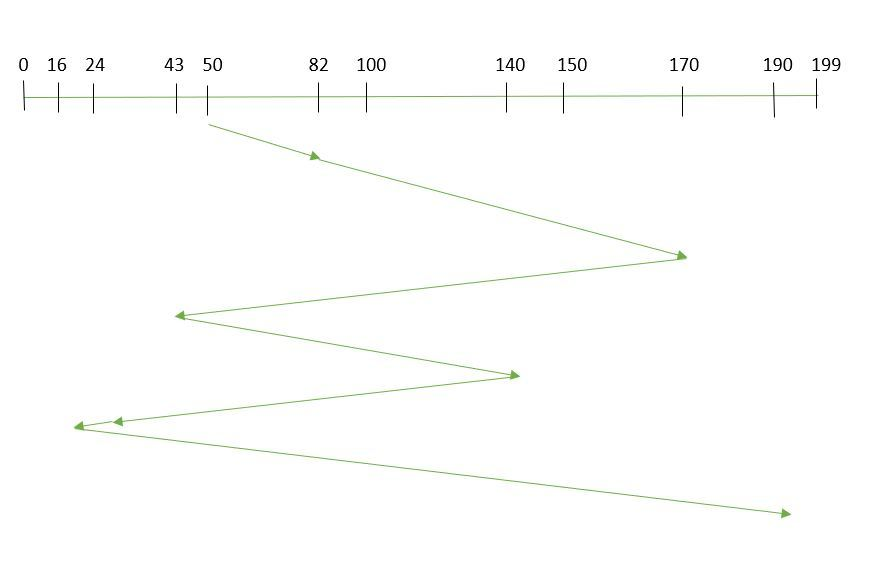
### First Come First Serve (FCFS):

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)

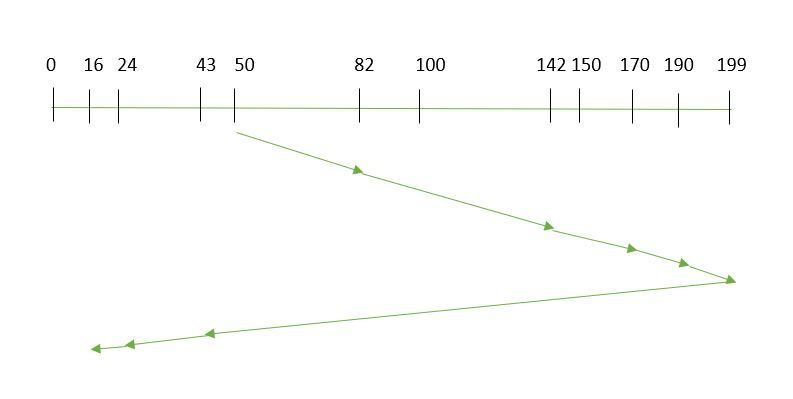
=642

### 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

Algorithms:

1. **First Come First Serve (FCFS):**

1- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.

2- Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.

3- Increment the total seek count with this distance.

4- Currently serviced track position now becomes the new head position.

5- Go to step 2 until all tracks in request array have not been serviced.

1. **SCAN:**

1- Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.

2- Let direction represents whether the head is moving towards left or right.

3- In the direction in which head is moving service all tracks one by one.

4- Calculate the absolute distance of the track from the head.

5- Increment the total seek count with this distance.

6- Currently serviced track position now becomes the new head position.

7- Go to step 3 until we reach at one of the ends of the disk.

8- If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

Implementation:

**A) First Come First Serve (FCFS):**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

void FCFS(int arr[], int head)

{

int seek\_count = 0;

int distance, cur\_track;

for (int i = 0; i < size; i++) {

cur\_track = arr[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < size; i++) {

cout << arr[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

FCFS(arr, head);

return 0;

}

**B) SCAN:**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void SCAN(int arr[], int head, string direction)

{

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

if (direction == "left")

left.push\_back(0);

else if (direction == "right")

right.push\_back(disk\_size - 1);

for (int i = 0; i < size; i++) {

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

int run = 2;

while (run--) {

if (direction == "left") {

for (int i = left.size() - 1; i >= 0; i--) {

cur\_track = left[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "right";

}

else if (direction == "right") {

for (int i = 0; i < right.size(); i++) {

cur\_track = right[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "left";

}

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

string direction = "left";

SCAN(arr, head, direction);

return 0;

}

Output:

**A) First Come First Serve (FCFS):**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

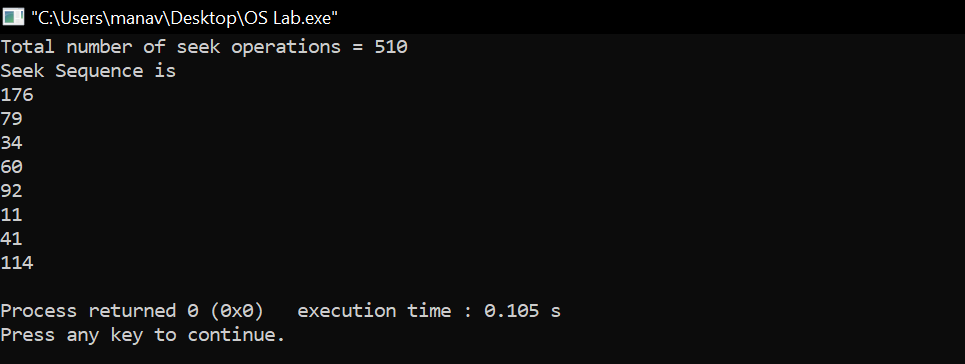
Initial head position = 50

**Output:**

The total seek count is calculated as:

= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41)

= 510



**B) SCAN:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

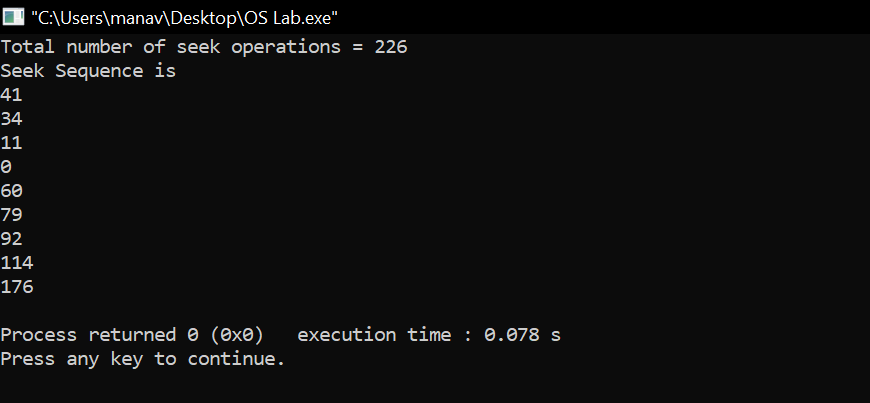
Direction = left (We are moving from right to left)

**Output:**

The total seek count is calculated as:

= (50-41)+(41-34)+(34-11)+(11-0)+(60-0)+(79-60)+(92-79)+(114-92)+(176-114)

= 226



Learning From The Experiment:

In FCFS algorithm every request gets a fair chance and there is no indefinite postponement. However, it does not try to optimize seek time. Also, FCFS may not provide the best possible service. FCFS being a nonpreemptive scheduling algorithm, the short processes which are at the back of the queue have to wait for the long process at the front to finish. The throughput of FCFS is not very efficient.

SCAN algorithm is simple and easy to understand. SCAN algorithm have no starvation. This algorithm is better than FCFS Scheduling algorithm. However, it is more complex algorithm to implement. This algorithm is not fair because it cause long waiting time for the cylinders just visited by the head.It causes the head to move till the end of the disk in this way the requests arriving ahead of the arm position would get immediate service but some other requests that arrive behind the arm position will have to wait for the request to complete.

***THANK YOU!***