**DISK SCHEDULING ALGORITHM**

**SHORTEST SEEK TIME FIRST DISK SCHEDULING**

**EXPT NO:** 7  **DATE:** 7/12/2022

**AIM:** To implement Disk Scheduling Algorithm using SSTF

**THEORY:**

**Disc Scheduling**: The Operating System performs a disc scheduling process to schedule I/O requests that arrive at the disc. Disc scheduling is important since-Many I/O requests may arrive from different processes, and the disc controller can only serve one I/O request at a time. As a result, other I/O requests need to wait in the waiting queue and get scheduled. The operating system needs to manage the hardware efficiently to reduce seek time. (Seek time is the time for the disc arm to move the heads to the cylinder containing the desired sector.)

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It appears reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the Shortest Seek Time First (SSTF) Algorithm. The SSTF algorithm selects the request having the minimum distance from the current head position. Since distance increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

Algorithm

To understand the SSTF disk scheduling algorithm, let us assume a disc queue with requests for I/O. ‘head’ is the position of the disc head. We will now apply the SSTF algorithm-

1. Arrange all the I/O requests in ascending order.
2. The head will find the nearest request (which has a minimum distance from the head) present in any direction (left or right) and will move to that request. Total head movement is calculated as
3. Current request - previous request (if the current request is greater)
4. Previous request - current request (if the previous request is greater)
5. Then the head will move another nearest request which has not been serviced present in any direction.
6. This process is repeated until all the requests are served and we get total head movement.

**Example**:

Consider a disc queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64. The head is initially at cylinder number 50. We will now use the SSTF algorithm to serve these I/O requests.

Input: I/O requests - {95, 180, 34, 119, 11, 123, 62, 64}

Initial head position - 50

Output:

Arranging all the requests in ascending order, we get - {11, 34, 62, 64, 95, 119, 123, 180}. The Head will start moving from 50 to the nearest request, i.e., 62. After 62 head will move to 64, and similarly head will follow the sequence - 34, 11, 95, 119, 123, 180.

The total head movement is calculated as:

= (62 - 50) + (64 - 62) + (64 - 34) + (34 - 11) + (95 - 11) + (119 - 95) + (123 - 119) + (180 - 123) = 12 + 2 + 30 + 23 + 84 + 24 + 4 + 57

= 236

Thus, the order in which requests are served is- 62, 64, 34, 11, 95, 119, 123, 180 with a total head movement of 236.

**Advantage**:

1. Seek time reduces as compared to the FCFS 2. Less waiting time and response time 3. Increase throughput.

**Disadvantage:**

1. It may cause starvation.

(For example- there are two requests - 14 and 186. While 14 is being served, a new request near 14 arrives. This new request will be served next, making the request at 186 wait. While this request is being served, another request close to 14 could arrive. In this way, the request at 186 may starve.)

1. There can be switching of directions repeatedly, which can make the process slow.
2. The variance between the response time and waiting time is very high.

Finding out the closest request adds additional overhead to the entire process

**CODE**

#include<iostream>

#include<algorithm>

#include<vector>

#include<math.h>

#define MAX\_REQUESTS 15

using namespace std;

int CURRENT\_RW,PREVIOUS\_RW;

int N,seekC=0;

struct REQUEST\_DIFFERNCE{

int REQUEST;

int DIFF;

};

vector <REQUEST\_DIFFERNCE>R;

vector <REQUEST\_DIFFERNCE>:: iterator IT;

int COMP(struct REQUEST\_DIFFERNCE A,struct REQUEST\_DIFFERNCE B)

{

return(A.DIFF<B.DIFF);

}

void HEADER\_UPDATER()

{

PREVIOUS\_RW=CURRENT\_RW;

for(IT=R.begin();IT!=R.end();++IT)

{

(\*IT).DIFF=abs((\*IT).REQUEST-CURRENT\_RW);

}

IT=min\_element(R.begin(),R.end(),COMP);

CURRENT\_RW=(\*IT).REQUEST;

R.erase(IT);

seekC+=(abs(CURRENT\_RW-PREVIOUS\_RW));

cout<<CURRENT\_RW<<" ";

}

void SSTF()

{

for(int i=0;i<N;i++)

{

HEADER\_UPDATER();

}

}

int main()

{

int t;

cout<<"ENTER CURRENT R/W HEADER POSITION: ";

cin>>CURRENT\_RW;

cout<<"ENTER NUMBER OF REQUESTS: ";

cin>>N;

cout<<"ENTER THE REQUESTS: ";

for(int i=0;i<N;i++)

{

cin>>t;

R.push\_back(REQUEST\_DIFFERNCE());

R[i].REQUEST=t;

}

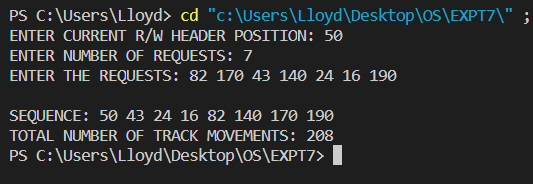
cout<<"\nSEQUENCE: "<<CURRENT\_RW<<" ";

SSTF();

cout<<"\nTOTAL NUMBER OF TRACK MOVEMENTS: "<<seekC;

}

**OUTPUT**

****

**CONCLUSION**

The given problem statement was successfully implemented and executed.