Experiment – 9

NAME : Madhuram Brijeshkumar Modi

ROLL NO. : 21BCP102

DIV, GROUP : 2,G3

Disc Scheduling

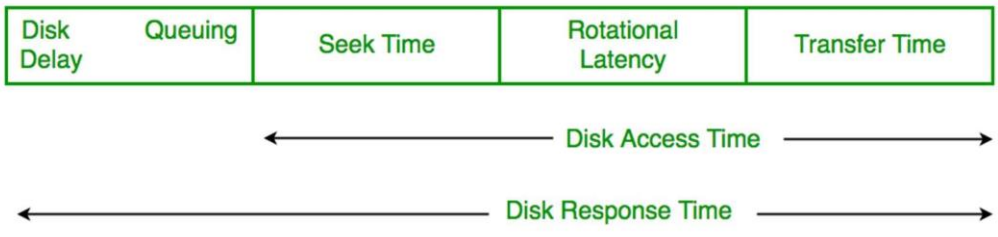
Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. Disk scheduling is important because:

- 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 requests 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.

1. First Come First Serve (FCFS)

FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Algorithm:

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.

Example:

Input:

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

Initial head position = 50

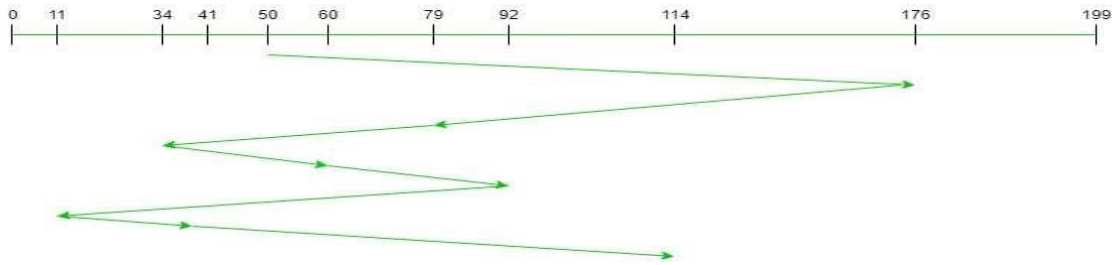
Output:

Total number of seek operations = 510

Seek Sequence is

--> 176, 79, 34, 60, 92, 11, 41, 114

The following chart shows the sequence in which requested tracks are serviced using FCFS.



Therefore, the total seek count is calculated as:

= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41) = 510

 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

Input:

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i, n, req[50], mov = 0, cp;
    printf("enter the current position\n");
    scanf("%d", &cp);
    printf("enter the number of requests\n");
    scanf("%d", &n);
    printf("enter the request order\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &req[i]);
    }

    mov = mov + abs(cp - req[0]);
    printf("%d -> %d", cp, req[0]);

    for(i = 1; i < n; i++)
    {
        mov = mov+abs(req[i] - req[i - 1]);
        printf(" ->%d", req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n", mov);
}
```

OUTPUT

```
PS D:\vs code> cd "d:\vs code\CP\" ; if ($?) { g++ trial2.cpp -o trial2 } ; if ($?) { .\trial2 }
enter the current position
55
enter the number of requests
5
enter the request order
12 15 32 56 84
55 -> 12 ->15 ->32 ->56 ->84
total head movement = 115
PS D:\vs code\CP>
```

1. Shortest Seek Time First (SSTF)

Basic idea is the tracks which are closer to current disk head position should be serviced first in order to minimise the seek operations.

Advantages of Shortest Seek Time First (SSTF) – 1. Better performance

than FCFS scheduling algorithm.

2. It provides better throughput.
3. This algorithm is used in Batch Processing system where throughput is more important.
4. It has less average response and waiting time.

Disadvantages of Shortest Seek Time First (SSTF) –

1. Starvation is possible for some requests as it favours easy to reach request and ignores the far away processes.

2. There is lack of predictability because of high variance of response time.
3. Switching direction slows things down.

Algorithm –

1. Let Request array represents an array storing indexes of tracks that have been requested. 'Head' is the position of disk head.

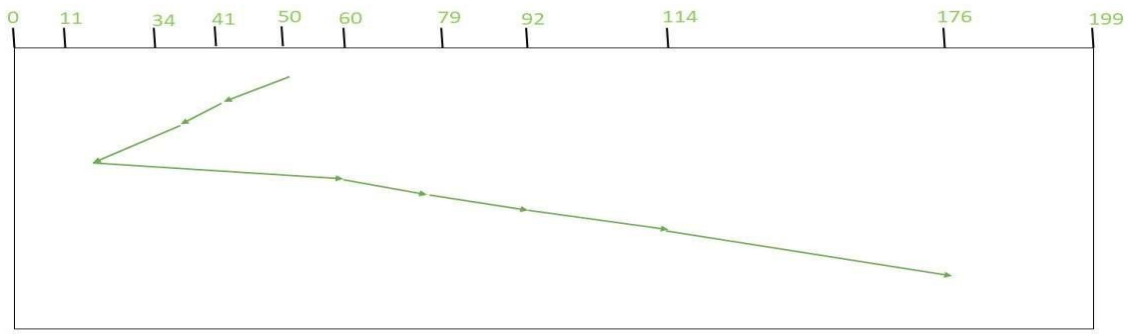
Find the positive distance of all tracks in the request array from head.

2. Find a track from requested array which has not been accessed/serviced yet and has minimum distance from 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.

Example –

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114} Initial head position = 50

The following chart shows the sequence in which requested tracks are serviced using SSTF.



Input:

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int i,j,k,n,m,sum=0,x,y,h;
    cout<<"Enter the size of disk\n";
    cin>>m;
    cout<<"Enter number of requests\n";
    cin>>n;
    cout<<"Enter the requests\n";

    vector <int> a(n),b;

    map <int, int> mp;

    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }

    for(i=0;i<n;i++)
    {
        if(a[i]>m)
        {
            cout<<"Error, Unknown position "<<a[i]<<"\n";
            return 0;
        }
    }

    cout<<"Enter the head position\n";
    cin>>h;
    int temp=h;
    int ele;
    b.push_back(h);
    int count=0;
```

```cpp
    while(count<n)
    {
        int diff=999999;
        for (auto q:mp)
        {
            if(abs(q.first-temp)<diff)
            {
                ele=q.first;
                diff=abs(q.first-temp);
            }
        }

    mp[ele]--;
    if(mp[ele]==0)
    {
        mp.erase(ele);
    }

        b.push_back(ele);
        temp=ele;
        count++;
    }

    cout<<b[0];
    temp=b[0];
    for(i=1;i<b.size();i++)
    {
        cout<<" ->"<<b[i];
        sum+= abs(b[i]-temp);
        temp=b[i];
    }

    cout<<'\n';
    cout<<"Total head movements = "<< sum<<'\n';
    cout<<"Average head movement = "<<(float)sum/n<<'\n';
    return 0;
}
```

OUTPUT-

```
PS D:\vs code> cd "d:\vs code\CP\" ; if ($?) { g++ trial1.cpp -o trial1 } ; if ($?) { .\trial1 }
Enter the size of disk
199
Enter number of requests
5
Enter the requests
98 183 37 122 14 124 65 67
Enter the head position
124 ->1967222677 ->1967222677 ->1967222677 ->1967222677
Total head movements = 1967222553
Average head movement = 3.93445e+008
PS D:\vs code\CP>
```