**Operating Systems**

**Experiment 10**

**Name: Kartik Jolapara SAP ID: 60004200107**

**Div.: B1 Branch: Computer Engineering**

# Aim -

# To implement Disk scheduling algorithm SSTF, SCAN.

# Theory -

Disk Scheduling Algorithms are needed because a process can make multiple I/O requests and multiple processes run at the same time. The requests made by a process may be located at different sectors on different tracks. Due to this, the seek time may increase more. These algorithms help in minimizing the seek time by ordering the requests made by the processes.

**Shortest Seek Time First (SSTF):** In this algorithm, the shortest seek time is checked from the current position and those requests which have the shortest seek time is served first. In simple words, the closest request from the disk arm is served first.

**SCAN:** In this algorithm, the disk arm moves in a particular direction till the end and serves all the requests in its path, then it returns to the opposite direction and moves till the last request is found in that direction and serves all of them.

## Code –

#include <iostream>

using namespace std;

void sort(int *disk*[], int *n*)

{

    for (int i = 1; i < *n*; i++)

    {

        for (int j = 0; j < *n* - 1; j++)

        {

            int temp;

            if (*disk*[j] > *disk*[j + 1])

            {

                temp = *disk*[j];

*disk*[j] = *disk*[j + 1];

*disk*[j + 1] = temp;

            }

        }

    }

}

void sstf(int *disk*[], int *head*, int *n*)

{

    int sequence[*n*];

    for (int k = 0; k < *n*; k++)

    {

        sequence[k] = 0;

    }

    int track = 0;

    int new\_postion = *head*;

    for (int j = 0; j < *n*; j++)

    {

        int min = 999;

        int idx = 0;

        for (int i = 0; i < *n*; i++)

        {

            int temp = 0;

            if (*disk*[i] > new\_postion)

            {

                temp = *disk*[i] - new\_postion;

            }

            else

            {

                temp = new\_postion - *disk*[i];

            }

            if (temp < min && !sequence[i])

            {

                min = temp;

                idx = i;

            }

        }

        cout << "Request procced " << *disk*[idx] << endl;

        if (*disk*[idx] > new\_postion)

        {

            track += *disk*[idx] - new\_postion;

        }

        else

        {

            track += new\_postion - *disk*[idx];

        }

        new\_postion = *disk*[idx];

        sequence[idx] = 1;

    }

    cout << "Track movement is " << track << endl;

}

void scan(int *disk*[], int *head*, int *n*)

{

    int sequence[*n*];

    int pos = *head*;

    for (int i = 0; i < *n*; i++)

    {

        sequence[i] = 0;

    }

    int track = 0;

    sort(*disk*, *n*);

    for (int j = 0; j < *n*; j++)

    {

        if (*disk*[j] > 50 && !sequence[j] && pos != 199)

        {

            cout << "Request processed" << *disk*[j] << endl;

            sequence[j] = 1;

        }

    }

    if (pos != 199)

    {

        track = 199 - 50;

        pos = 199;

    }

    int new\_postion = 199;

    for (int k = *n* - 1; k >= 0; k--)

    {

        int sum = 0;

        if (*disk*[k] < 199 && !sequence[k])

        {

            cout << "Request processeds" << *disk*[k] << endl;

            sequence[k] = 1;

            sum = new\_postion - *disk*[k];

            track += sum;

            new\_postion = *disk*[k];

        }

    }

    cout << "Track moment " << track << endl;

}

int main()

{

    int n;

    cout << "No of request" << endl;

    cin >> n;

    int disk[n];

    cout << "Enter postion of head" << endl;

    int head;

    cin >> head;

    cout << "Enter request" << endl;

    for (int i = 0; i < n; i++)

    {

        cin >> disk[i];

    }

    cout << "Enter 1 for SSTF and 2 for SCAN" << endl;

    int o;

    cin >> o;

    if (o == 1)

        sstf(disk, head, n);

    if (o == 2)

        scan(disk, head, n);

}

## Output –





# Conclusion

Disk Scheduling Algorithms like SSTF and SCAN are successfully executed.