Name: Brijesh Rameshbhai Rohit

Admission number: U19CS009

OS-ASSIGNMENT-09

1.To implement Shortest Seek Time First (SSTF) Disk scheduling algorithm.

Code:

```
#include <bits/stdc++.h>
using namespace std;


class process
{
private:
    int id;
    int seekloc;


public:
    process(int id, int a)
    {
        this->id = id;
        this->seekloc = a;
    }
    int getSeekloc()
    {
        return this->seekloc;
    }
    void setSeekloc(int loc)
    {
        this->seekloc = loc;
    }
    int getId()
    {
        return this->id;
    }
};


//FUNCTION TO GET THE PROCESS CLOSEST TO THE HEAD
int nextnearprocess(vector<process> &p, int l, int head)
{
    int n = p.size();
    int min = INT_MAX;
```

```
    int index = -1;
    for (int i = l; i < p.size(); i++)
    {
        int dis = abs(p[i].getSeekloc() - head);
        if (dis < min)
        {
            min = dis;
            index = i;
        }
    }
    return index;
}


//FUNCTION TO IMPLEMENT SSTF
void sstf(vector<process> &p, int n, int headpos)
{
    int curr = 0, itr = 0;
    int seekcount = 0, h = headpos; //loop till all processes are completed
    while (curr < n)
    {
        int closestInd = nextnearprocess(p, curr, headpos); //getting Index of
closest process
        if (closestInd == -1)
        {
            break;
        }
        seekcount += abs(p[closestInd].getSeekloc() - headpos);
        headpos = p[closestInd].getSeekloc();
        swap(p[closestInd], p[itr]); //swapping inorder to remove the process
that are completed and then increasing the itr
        itr++;
        curr++;
    }
    cout << endl << "Seek count : " << seekcount << endl;
    cout << endl << "Seek Sequence is : " << endl;
    cout << "Process\tSeekloc" << endl;
    cout << "head \t" << h << endl;
    for (int i = 0; i < n; i++)
    {
        cout << p[i].getId() << "\t" << p[i].getSeekloc() << endl;
    }
}
```

```
int main()
{
    int n;
    cout << "Enter Number of Processes : ";
    cin >> n;

    vector<process> p;

    cout << "Enter disk request sequence : ";

    for (int i = 0; i < n; i++)
    {
        int a;
        cin >> a;
        p.push_back(process(i + 1, a));
    }

    int headpos;
    cout << "Enter the head location : ";
    cin >> headpos;

    sstf(p, n, headpos);

    return 0;
}
```

Output:

```
PS C:\Users\msi\Documents\sem-5\OS\OS-ASSIGN-09> cd "c:\Users\
 { .\u19cs009-sstf }
Enter Number of Processes : 10
Enter disk request sequence : 12 45 23 34 78 56 98 65 32 8
Enter the head location : 60

Seek count : 136

Seek Sequence is :
Process Seekloc
head    60
6       56
8       65
5       78
7       98
2       45
4       34
9       32
3       23
1       12
10      8
PS C:\Users\msi\Documents\sem-5\OS\OS-ASSIGN-09>
```

2. To implement Scan Disk scheduling algorithm.

Code:

```cpp
#include <bits/stdc++.h>
using namespace std;


class process
{
private:
    int id;
    int seekloc;
    int complete;


public:
    process(int id, int a)
    {
        this->id = id;
        this->seekloc = a;
        this->complete = 0;
    }
    int getSeekloc() { return this->seekloc; }
    void setSeekloc(int loc) { this->seekloc = loc; }
```

```
    int getId() { return this->id; }
    void setComplete() { this->complete = 1; }
    int getComplete() { return this->complete; }
};


//FUNCTION TO CALCULATE DISTANCES IN A ONE DIRECTION FROM HEAD TO SELECT
PROCESSES
vector<pair<int, int>> calcd(vector<process> &p, int flag, int head)
{
    int n = p.size();
    if (flag == 0) //LEFT DIRECTION
    {
        vector<pair<int, int>> p1;
        for (int i = 0; i < n; i++)
        {
            if (p[i].getSeekloc() <= head && p[i].getComplete() != 1)
            {
                p1.push_back({i, abs(p[i].getSeekloc() - head)});
            }
        }
        return p1;
    }
    else //RIGHT DIRECTION
    {
        vector<pair<int, int>> p1;
        for (int i = 0; i < n; i++)
        {
            if (p[i].getSeekloc() >= head && p[i].getComplete() != 1)
            {
                p1.push_back({i, abs(p[i].getSeekloc() - head)});
            }
        }
        return p1;
    }
}


bool compare(pair<int, int> p1, pair<int, int> p2)
{
    return p1.second < p2.second;
}


void scan(vector<process> &p, int n, int headloc, char dir)
```

```
{
    int curr = 0, itr = 0;
    int seekcount = 0, h = headloc;
    vector<pair<int, int>> output;
    while (curr < n + 1)
    {
        if (dir == 'l') //IF LEFT
        {
            vector<pair<int, int>> dis;
            dir = 'r'; //ALL THE PROCESSES IN THE LEFT OF THE HEAD
            dis = calcd(p, 0, h);
            sort(dis.begin(), dis.end(), compare);
            for (auto i : dis)
            { //PROCESS INCOMPLETE THEN ALLOCATE
                if (p[i.first].getComplete() != 1)
                {
                    seekcount += abs(p[i.first].getSeekloc() - h);
                    h = p[i.first].getSeekloc();
                    output.push_back({p[i.first].getId(),
p[i.first].getSeekloc()});
                    p[i.first].setComplete();
                }
                curr++;
                if (curr >= n + 1)
                    break; //ALL PROCESS COMPLETE THEN BREAK;
            }
            if (curr >= n + 1)
                break; //ALL PROCESS COMPLETE THEN BREAK;
        }
        else if (dir == 'r') //IF RIGHT
        {
            vector<pair<int, int>> dis;
            dir = 'l'; //ALL THE PROCESSES IN THE RIGHT OF THE HEAD
            dis = calcd(p, 1, h);
            sort(dis.begin(), dis.end(), compare);
            for (auto i : dis)
            { //PROCESS INCOMPLETE THEN ALLOCATE
                if (p[i.first].getComplete() != 1)
                {
                    seekcount += abs(p[i.first].getSeekloc() - h);
                    h = p[i.first].getSeekloc();
                    output.push_back({p[i.first].getId(),
p[i.first].getSeekloc()});
                    p[i.first].setComplete();
                }
```

```
                curr++;
                if (curr >= n + 1)
                    break; //ALL PROCESS COMPLETE THEN BREAK;
            }
            if (curr >= n + 1)
                break; //ALL PROCESS COMPLETE THEN BREAK;
        }
    }
    cout << "\nSeek count : " << seekcount << endl;
    cout << "\nSeek Sequence is : " << endl;
    cout << "Process\tSeekloc" << endl;
    cout << "head \t" << headloc << endl;
    for (int i = 0; i < output.size(); i++)
    {
        cout << output[i].first << "\t" << output[i].second << endl;
    }
}
int main()
{
    int n, size;
    cout << "Enter Number of Processes : ";
    cin >> n;
    cout << "Enter disk size : ";
    cin >> size;


    vector<process> p;
    cout << "Enter disk request sequence : ";
    for (int i = 0; i < n; i++)
    {
        int a;
        cin >> a;
        p.push_back(process(i + 1, a));
    }


    int headloc;
    cout << "Enter the head location : ";
    cin >> headloc;


    char dir;
    cout << "Enter initial direction : " << endl;
    cout << "type \"l\" for left, and \"r\" for right : ";
    cin >> dir;
```

```
    p.push_back(process(-1, 0));

    p.push_back(process(-2, size));

    scan(p, n, headloc, dir);

    return 0;
}
```

Output:

```
PS C:\Users\msi\Documents\sem-5\OS\OS-ASSIGN-09> cd "c:\Users\msi\
Enter Number of Processes : 10
Enter disk size : 100
Enter disk request sequence : 12 45 76 32 54 34 43 65 76 9
Enter the head location : 40
Enter initial direction :
type "l" for left, and "r" for right : l

Seek count : 116

Seek Sequence is :
Process Seekloc
head    40
6       34
4       32
1       12
10      9
-1      0
7       43
2       45
5       54
8       65
3       76
9       76
PS C:\Users\msi\Documents\sem-5\OS\OS-ASSIGN-09>
```

-1 refers to left end of the Disk, meaning at which head location is 0

```
.cpp -o scan } ; if ($?) { .\scan }OS-ASSIGN-09>
Enter Number of Processes : 13
Enter disk size : 200
Enter disk request sequence : 45 110 123 198 43 29 109 167 100 9
12 34 56
Enter the head location : 10
Enter initial direction :
type "l" for left, and "r" for right : r

Seek count : 381

Seek Sequence is :
Process Seekloc
head    10
11      12
6       29
12      34
5       43
1       45
13      56
9       100
7       109
2       110
3       123
8       167
4       198
-2      200
10      9
```

-1 refers to the right end of the disk meaning at which head location would be disk size [here 200].