**EXP NO :12 DISK SCHEDULING ALGORITHM**

**FIRST COME FIRST SERVE**

**AIM:**

To implement the disk scheduling algorithm - first come first serve

for the service queue.

**ALGORITHM :**

STEP 1 : Start

STEP 2 : Input the value of the request sequence in the given order and the initial positional value as head.

STEP 3 : The positive distance of the each value in the request array are calculated in the order of the sequence.

STEP 4 : The value of seek count is incremented by one at each step.

STEP 5 : These steps are repeated until the request sequence is completed.

STEP 6 : Thus the total positive distance is displayed at last along with the order of execution.

STEP 7 : Stop

**CODE :**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int head,size,req[20],i,sum=0,diff;

printf("Enter the initial head position:");

scanf("%d",&head);

printf("Enter the total size of request sequence:”);

scanf("%d",&size);

printf("Enter the request sequence elements:");

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

scanf("%d",&req[i]);

sum=req[0]-head;

for(i=0;i<size-1;i++)

{

diff=abs(req[i]-req[i+1]);

sum=sum+diff;

}

printf("\nTotal no of seek operations:%d\n",sum);

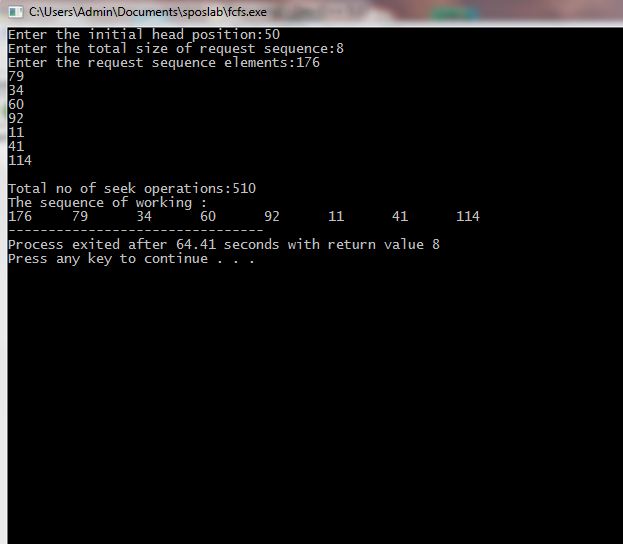
printf(“The sequence of working :\n");

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

printf("%d\t",req[i]);

}

**OUTPUT :**



**SHORTEST SEEK TIME FIRST ALGORITHM**

**AIM :**

To implement the disk scheduling algorithm – shortest seek time firstfor the service queue.

**ALGORITHM :**

STEP 1 : Start.

STEP 2 : Input the value of the request sequence and the head position .

STEP 3 : The positive distance of all tracks in the request array from the head are calculated.

STEP 4 : A track from the requested array which has not been accessed yet and has a minimum distance from the head are calculated.

STEP 5 : The total seek count are incremented with this distance.

STEP 6 : Assign the head position as currently serviced track position.

STEP 7 : Go to step 4 until all tracks in request array have not been serviced.

STEP 8 : Stop.

**CODE :**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int queue[100],t[100],head,seek=0,n,i,j,temp;

float avg;

printf("Enter the size of Queue\t");

scanf("%d",&n);

printf("Enter the Queue\t");

for(i=0;i<n;i++)

{

scanf("%d",&queue[i]);

}

printf("Enter the initial head position\t");

scanf("%d",&head);

for(i=1;i<n;i++)

t[i]=abs(head-queue[i]);

for(i=0;i<n;i++)

{

for(j=i+1;j<n;j++)

{

if(t[i]>t[j])

{

temp=t[i];

t[i]=t[j];

t[j]=temp;

temp=queue[i];

queue[i]=queue[j];

queue[j]=temp;

}

}

}

for(i=1;i<n-1;i++)

{

seek=seek+abs(head-queue[i]);

head=queue[i];

}

printf("\nTotal Seek Time is%d\t",seek);

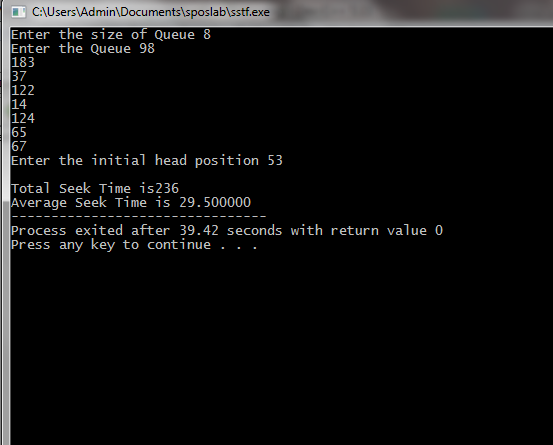
avg=seek/(float)n;

printf("\nAverage Seek Time is %f\t",avg);

return 0;

}

**OUTPUT :**



**CLOOK ALGORITHM**

**AIM :**

To implement the disk scheduling algorithm – Circular LOOK algorithm for the service queue.

**ALGORITHM :**

STEP 1 : Start.

STEP 2 : Input the initial position of head.

STEP 3 : Input the service request queue.

STEP 4 : The initial direction is given and starts servicing the requests in that direction one by one.

STEP 5 : All the requests in that direction is served.

STEP 6 : Calculate the (current track – head).

STEP 7 : Change head = current track.

STEP 8 : Repeat step 5-6 until last request.

STEP 9 : Reverse the direction and reach the last request in that direction without servicing in between.

STEP 10 : Go to step 4 and repeat till all the requests were serviced.

STEP 11 : Display the service sequence and seek time

STEP 12 : Stop.

**CODE :**

#include<bits/stdc++.h>

using namespace std;

class look\_algo

{

private:

int i,j,hpos;

string dir;

int arr[20];

int dsize=200;

int d, cp, seektime, head;

vector<int> left, right;

vector<int> ss;

public:

look\_algo()

{

cout<<"----C-LOOK----"<<endl;

cout<<"enter initial head position\n";

cin>>hpos;

i=0;

cout<<"enter sequence with space and press enter when completed: ";

do

{

cin>>arr[i++];

}while(cin.peek()!='\n');

cout<<i<<endl;

}

void clook()

{

seektime=0;

for(j=0;j<i;j++)

{

if(arr[j]<hpos)

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

else

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

}

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

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

//servicing requests in right side

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

{

cp = right[i];

ss.push\_back(cp);

d = abs(cp - hpos);

seektime += d;

hpos= cp;

}

cout<<endl;

//right side service id finished

seektime+=abs(hpos-left[0]);

hpos = left[0];

// Now service the requests in left

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

{

cp = left[i];

ss.push\_back(cp);

d = abs(cp - hpos);

seektime += d;

hpos= cp;

}

cout << "CLOOK seektime: "<< seektime << endl;

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

cout << ss[i] << endl;

}

};

main()

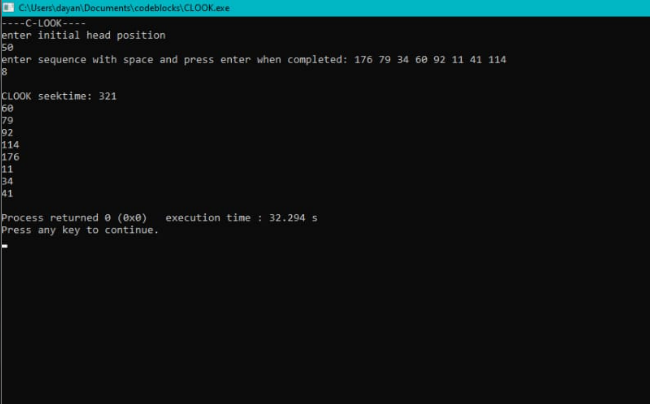
{

look\_algo a;

a.clook();

}

**OUTPUT :**



**SCAN AND CSCAN ALGORITHM**

**AIM :**

To implement the disk scheduling algorithm – SCAN AND CSCAN algorithm for the service queue.

**ALGORITHM :**

STEP 1 : Start.

STEP 2 : Input the initial position of head.

STEP 3 : Input the request array according to their arrival.

STEP 4 : For SCAN algorithm ,create an instance

1) Input the direction as left | right for head.

2) Service all requests in the given direction one by one.

3) Calculate the absolute distance between the track and the head.

4) Currently serviced track becomes new head.

5) Repeat 2-4 to service all requests in that direction until the end of the disk in that particular direction is reached.

6) Reverse the direction and service the remaining requests.

STEP 5 : For CSCAN algorithm,create an instance,

1) The head services only in the right direction from 0 to disk size.

2) While moving left , no service is done.

3) Reverse the direction once the end is reached.

4) Service the tracks one by one by moving right.

5) Calculate the (track – head).

6) Current track becomes head.

7) Repeat 5 until right end of the disk.

8) The same process is repeated 2-7 to service remaining requests.

STEP 6 : The service sequence and total seek time is displayed.

STEP 7 : Stop.

**CODE :**

#include<bits/stdc++.h>

using namespace std;

class scan\_algo

{

private:

int i,j,hpos;

string dir;

int arr[20];

int dsize=200;

int d, cp, seektime, head;

vector<int> left, right;

vector<int> ss;

public:

scan\_algo()

{

cout<<"enter initial head position\n";

cin>>hpos;

i=0;

cout<<"enter sequence with space and press enter when completed: ";

do

{

cin>>arr[i++];

}while(cin.peek()!='\n');

cout<<i<<endl;

}

void push()

{

for(j=0;j<i;j++)

{

if(arr[j]<hpos)

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

else

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

}

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

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

}

void display()

{

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

cout << ss[i] << endl;

}

void scan()

{

seektime=0;

cout<<"direction: "<<dir;

cin>>dir;

if(dir=="left")

left.push\_back(0);

else

right.push\_back(dsize-1);

push();

for(int t=1;t<=2;t++)

{

if(dir=="right")

{

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

{

cp=right[j];

ss.push\_back(cp);

d=abs(cp-hpos);

seektime+=d;

hpos=cp;

}

dir="left";

}

else if(dir=="left")

{

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

{

cp=left[j];

ss.push\_back(cp);

d=abs(cp-hpos);

seektime+=d;

hpos=cp;

}

dir="right";

}

}

cout<<"SCAN Seektime : "<<seektime<<endl;

display();

}

void cscan()

{

seektime=0;

left.push\_back(0);

right.push\_back(dsize-1);

push();

//servicing requests in right side

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

{

cp = right[i];

ss.push\_back(cp);

d = abs(cp - hpos);

seektime += d;

hpos= cp;

}

cout<<endl;

//right side service id finished

hpos = 0;

// Now service the requests in left

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

cp = left[i];

ss.push\_back(cp);

d = abs(cp - hpos);

seektime += d;

hpos= cp;

}

cout << "CSCAN seektime: "<< seektime << endl;

display();

}

};

main()

{

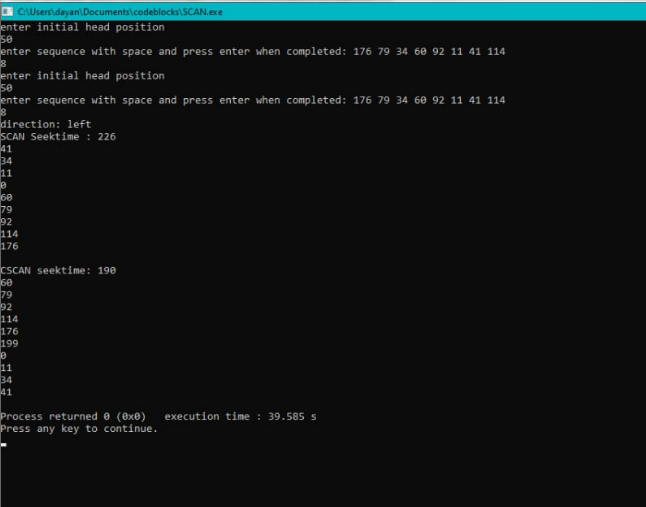
scan\_algo a;

//a.scan();

a.cscan();

}

**OUTPUT :**

****

**RESULT :**

Thus the program for the implementation of the disk scheduling algorithm are executed and their outputs are verified.