NAME : Aniket Sriwastva

Reg.No : 11712763

Email : [aniket.sriwastva@gmail.com](mailto:aniket.sriwastva@gmail.com)

Github Link : <https://github.com/anikcode/OS_SIMULATION>

Description:

Q31. Write a C program to solve the following problem:

Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders)that the disk arm moves to satisfy all the pending requests for each of the SSTF disk-scheduling algorithms? (for conceptual clarity refer the textbook)

CONDITION:

* This algorithm services that request next which requires least number of head movements from its current position regardless of the direction.
* It breaks the tie in the direction of head movement.

INPUT:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

CODE:

#include<stdio.h>

#include<conio.h>

struct sstf

{

int num;

int flag;

};

int main()

{

struct sstf d[20];

int disk\_queue;

int arr[20],a[20];

printf("\t\t\t\t\t\t\*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\n");

printf("\t\t\t\t\t\t\* \* \* \*\n");

printf("\t\t\t\t\t\t\*\*\*\*\*\* \*\*\*\*\*\* \* \*\*\*\*\n");

printf("\t\t\t\t\t\t \* \* \* \*\n");

printf("\t\t\t\t\t\t\*\*\*\*\*\* \*\*\*\*\*\* \* \* \*\n");

printf("\t\t\t\t\t\t==================================\n");

printf("\t\t\t\t\t\t ||=========================||\n");

printf("\t\t\t\t\t\t ||Operating System Project ||\n");

printf("\t\t\t\t\t\t ||Made By Aniket Sriwastva ||\n");

printf("\t\t\t\t\t\t ||Submitted To Richa Sharma||\n");

printf("\t\t\t\t\t\t ||=========================||\n");

int i,j,sum=0,n,min,location,x,y;

printf("Enter size of the queue :-> ");

scanf("%d",&n);

printf("Enter Position of head :-> ");

scanf("%d",&disk\_queue);

printf("Enter elements of disk queue\n : ");

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

{

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

d[i]. flag=0;

}

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

{

x=0; min=0;location=0;

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

{

if(d[j].flag==0)

{

if(x==0)

{

arr[j]=disk\_queue-d[j].num;

if(arr[j]<0){ arr[j]=d[j].num-disk\_queue;}

min=arr[j];location=j;x++; }

else

{

arr[j]=disk\_queue-d[j].num;

if(arr[j]<0){ arr[j]=d[j].num-disk\_queue;}

}

if(min>arr[j]){ min=arr[j]; location=j;}

}

}

d[location].flag=1;

a[i]=d[location].num-disk\_queue;

if(a[i]<0)

{

a[i]=disk\_queue-d[location].num;

}

disk\_queue=d[location].num;

}

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

{

sum=sum+a[i];

}

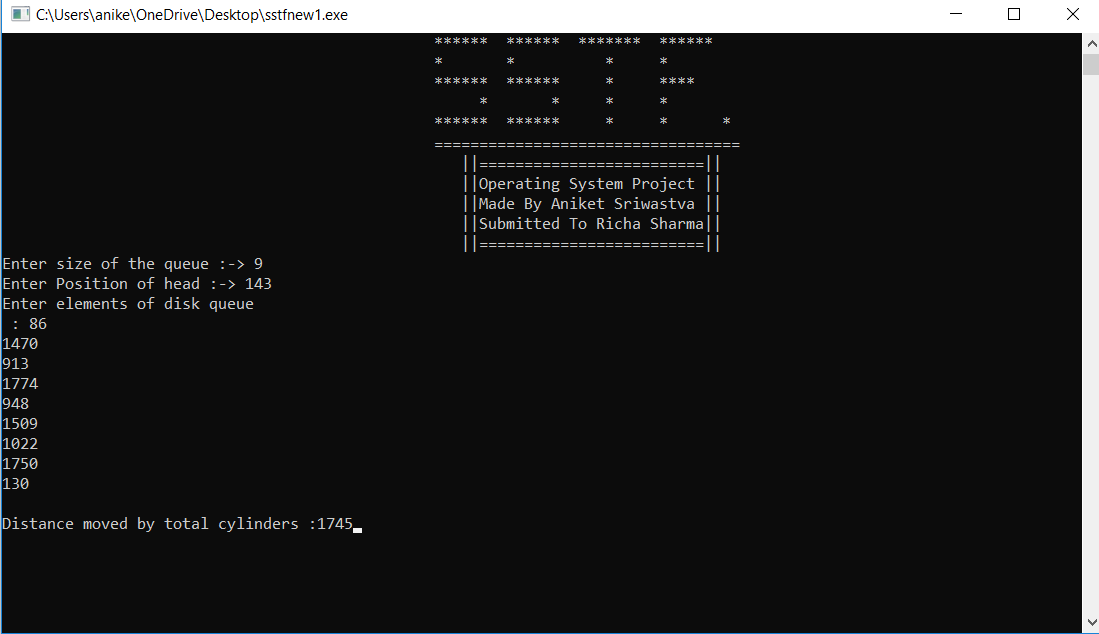
printf("\nDistance moved by total cylinders :%d",sum);

getch();

return 0;

}

OUTPUT:



Q 4. Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as : Priority = 1+ Waiting time / Estimated run time Write a program to implement such an algorithm.

|  |  |  |
| --- | --- | --- |
| -Process | Arrival time | Burst time |
| P1 | 0 | 20 |
| P2 | 5 | 36 |
| P3 | 13 | 19 |
| P4 | 17 | 42 |

CONDITIONS :

* We interchange the given processes by their arrival times in the ascending order
* And also check weather the processes arrival time is less than the previous process burst time we interchange again in them
* we calculate waiting time from than turnaround time by turnaroundtime=waiting time + burst time
* At last we calculate the priority according to the given formula
* : Priority = 1+ Waiting time / Estimated run time

INPUT:

0

20

5

36

13

19

CODE :

#include<stdio.h>

#include<conio.h>

struct Process

{

int process;

int burst\_time;

int arrival\_time;

int pr;

int wt;

int tat;

};

int main()

{

printf("\t\t\t\t\t\t \*\*\*\*\*\* \*\*\*\*\*\* \*\*\*\*\*\*\* \n");

printf("\t\t\t\t\t\t \* \* \* \n");

printf("\t\t\t\t\t\t \*\*\*\*\*\* \* \*\*\*\* \n");

printf("\t\t\t\t\t\t \* \* \* \* \n");

printf("\t\t\t\t\t\t \*\*\*\*\*\* \*\*\* \* \*\n");

printf("\t\t\t\t\t\t ================================\n");

printf("\t\t\t\t\t\t ||=========================||\n");

printf("\t\t\t\t\t\t ||Operating System Project ||\n");

printf("\t\t\t\t\t\t ||Made By Aniket Sriwastva ||\n");

printf("\t\t\t\t\t\t ||Submitted To Richa Sharma||\n");

printf("\t\t\t\t\t\t ||=========================||\n");

int a,i,j;

printf("Enter no.of processers\n");

scanf("%d",&a);

struct Process value[a];

struct Process x[a];

struct Process y[a];

printf("Enter processer details\n1.Processes number\n2.Burst time\n3.Arrival time\n");

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

{

value[i].process=i;

printf("Process %d \n",i);

printf("Burst time\n");

scanf("%d",&value[i].burst\_time);

if(value[i].burst\_time<0)

{

return 0;

}

printf("Arrival time\n");

scanf("%d",&value[i].arrival\_time);

if(value[i].arrival\_time<0)

return 0;

}

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

{

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

{

if(value[i].arrival\_time>value[j].arrival\_time)

{

x[i]=value[i];

value[i]=value[j];

value[j]=x[i];

}

}

}

value[0].wt=0;

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

{

value[i].wt=0;

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

{

value[i].wt+=value[j].burst\_time;

value[i].wt=value[i].wt-value[i].arrival\_time;

}

}

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

{

if(value[i].wt<0)

{

value[i].wt=0;

}

}

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

{

value[i].tat=value[i].wt+value[i].burst\_time;

}

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

{

y[i].pr=value[i].wt/value[i].burst\_time;

y[i].pr=1+y[i].pr;

value[i].pr=y[i].pr;

}

printf("============================================================================================\n");

printf("1.Process\t2.Burst time\t3.Arrival time\t4.Priority\t5.Waiting\t6.Turnaroundtime\n");

printf("============================================================================================\n");

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

{

printf("%d\t\t\t",value[i].process);

printf("%d\t\t",value[i].burst\_time);

printf("%d\t\t",value[i].arrival\_time);

printf("%d\t\t",value[i].pr);

printf("%d\t\t",value[i].wt);

printf("%d\t\t\n",value[i].tat);

}

printf("============================================================================================");

}

OUTPUT :

