**FIFO Scheduling**

#include<stdio.h>

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes(maximum 20):"); scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

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

{

printf("P[%d]:",i+1);

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

}

wt[0]=0;

//calculating waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

//calculating turnaround time

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

{

tat[i]=bt[i]+wt[i];

avwt+=wt[i];

avtat+=tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]); }

avwt/=i;

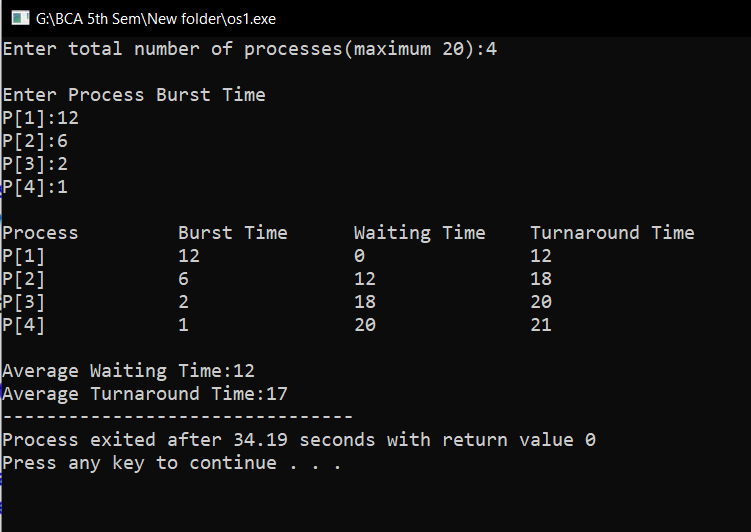
avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

****

**SJF Scheduling.**

#include<stdio.h>

int main()

{

int bt[20], p[20], wt[20], tat[20], i,j,n, total=0, pos,temp; float avg\_wt, avg\_tat;

printf("Enter No of Processes");

scanf("%d", &n);

printf("\n Enter Brust Time");

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

{

printf("p%d:",i+1);

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

p[i]=i+1;

}

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

{

pos=i;

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

}

if(bt[j]<bt[pos])

pos=j;

{

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_tat=(float)total/n;

total=0;

printf("\n Process \t brust time \t waiting time \t TAT"); for(i=0; i<n; i++)

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

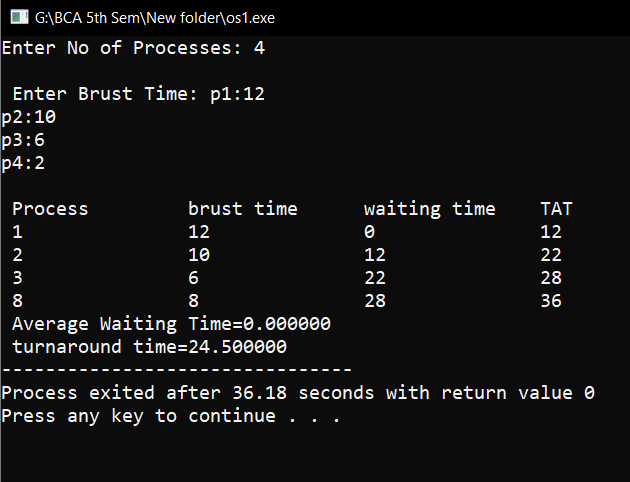
printf("\n %d\t\t %d\t\t %d\t\t %d", p[i],bt[i], wt[i],tat[i]); }

avg\_tat= (float)total/n;

printf("\n Average Waiting Time=%f",avg\_wt);

printf("\n turnaround time=%f",avg\_tat);

}



**Round-Robin Scheduling.**

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10]; printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

printf("Enter Time Quantum:\t");

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n"); for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]- bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

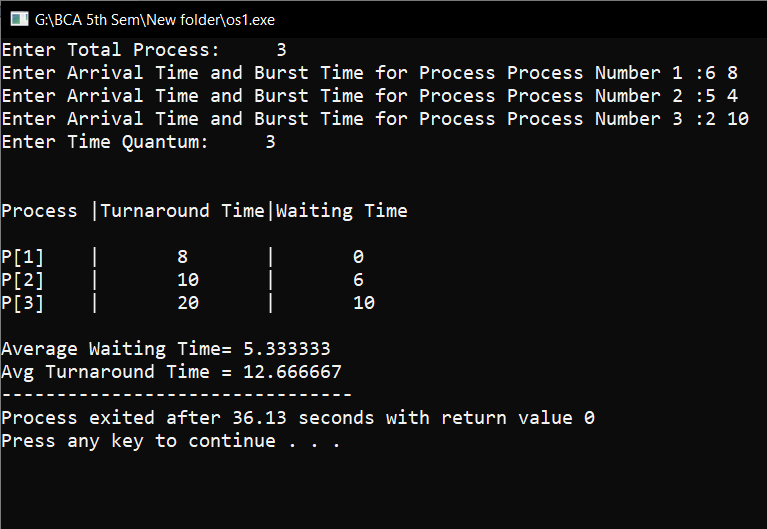
count=0;

}

printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n); printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}



**Banker’s Algorithm Simulation.**

#include <stdio.h>

int current[5][5], maximum\_claim[5][5], available[5]; int allocation[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe = 0; int counter = 0, i, j, exec, resources, processes, k = 1;

int main()

{

printf("\nEnter number of processes: ");

scanf("%d", &processes);

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

{

running[i] = 1;

counter++;

}

printf("\nEnter number of resources: ");

scanf("%d", &resources);

printf("\nEnter Claim Vector:"); for (i = 0; i < resources; i++)

{

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

}

printf("\nEnter Allocated Resource Table:\n"); for (i = 0; i < processes; i++)

{

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

{

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

}

}

printf("\nEnter Maximum Claim Table:\n"); for (i = 0; i < processes; i++)

{

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

{

scanf("%d", &maximum\_claim[i][j]);

}

}

printf("\nThe Claim Vector is: "); for (i = 0; i < resources; i++)

{

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

}

printf("\nThe Allocated Resource Table:\n"); for (i = 0; i < processes; i++)

{

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

{

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

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n"); for (i = 0; i < processes; i++)

{

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

{

printf("\t%d", maximum\_claim[i][j]);

}

printf("\n");

}

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

{

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

{

allocation[j] += current[i][j];

}

}

printf("\nAllocated resources:");

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

{

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

}

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

{

available[i] = maxres[i] - allocation[i];

}

printf("\nAvailable resources:");

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

{

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

}

printf("\n");

while (counter != 0)

{

safe = 0;

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

{

if (running[i])

{

exec = 1;

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

{

if (maximum\_claim[i][j] - current[i][j] > available[j]) {

exec = 0;

break;

}

}

if (exec)

{

printf("\nProcess%d is executing\n", i + 1); running[i] = 0;

counter--;

safe = 1;

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

{

available[j] += current[i][j];

}

break;

}

}

}

if (!safe)

{

printf("\nThe processes are in unsafe state.\n"); break;

}

else

{

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

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

{

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

}

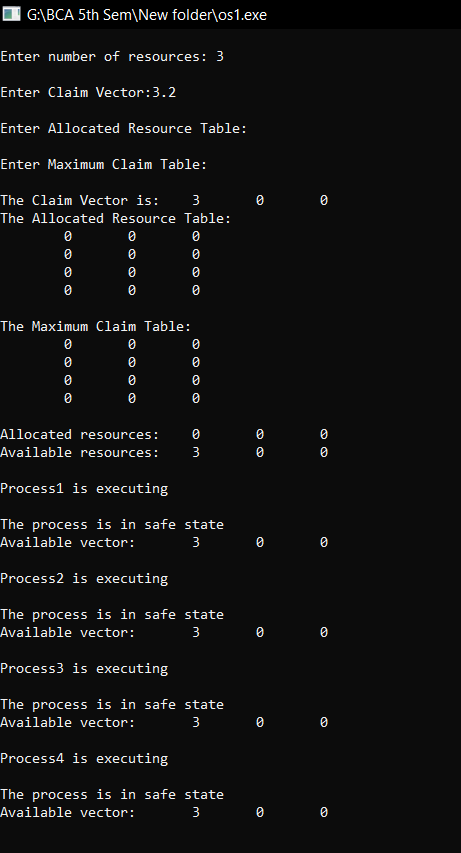
printf("\n");

}

}

return 0;

}



**Continuous Memory Allocation Techniques.**

**a) Worst Fit**

#include<stdio.h>

int main()

{

int n,n1,i;

printf("enter the number of processes:"); scanf("%d",&n);

int process[n];

printf("\n enter the size of processes:\n"); for(i=0;i<n;i++)

{

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

}

printf("enter the no of memoryblocks:"); scanf("%d",&n1);

int blocks[n1];

printf("\n enter the size of blocks:\n");

int total=0;

for(i=0;i<n1;i++)

{

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

total=total+blocks[i];

}

int process1[n1];

int job[n1];

int frag[n1];

int check[n1];

for(i=0;i<n1;i++)

{

check[i]=0;

}

int j,used=0;

i=0;

while(i<n)

{

int max=-1,j1=-1,k=-1,max1;

for(j=0;j<n1;j++)

{

max1=blocks[j];

if(max1>=max&&check[j]==0&&max1>=process[i]) {

max=max1;

j1=j;

}

else

{

if(check[j]==0)

{

process1[j]=0;

job[j]=0;

frag[j]=blocks[j];

}

}

}

if(k!=j1)

{

process1[j1]=process[i];

job[j1]=i+1;

frag[j1]=blocks[j1]-process[i];

used=used+process[i];

check[j1]=1;

int l;

}

i++;

}

printf("blocksize\tprocess size\tprocessno\tfragmentation\n"); for(i=0;i<n1;i++)

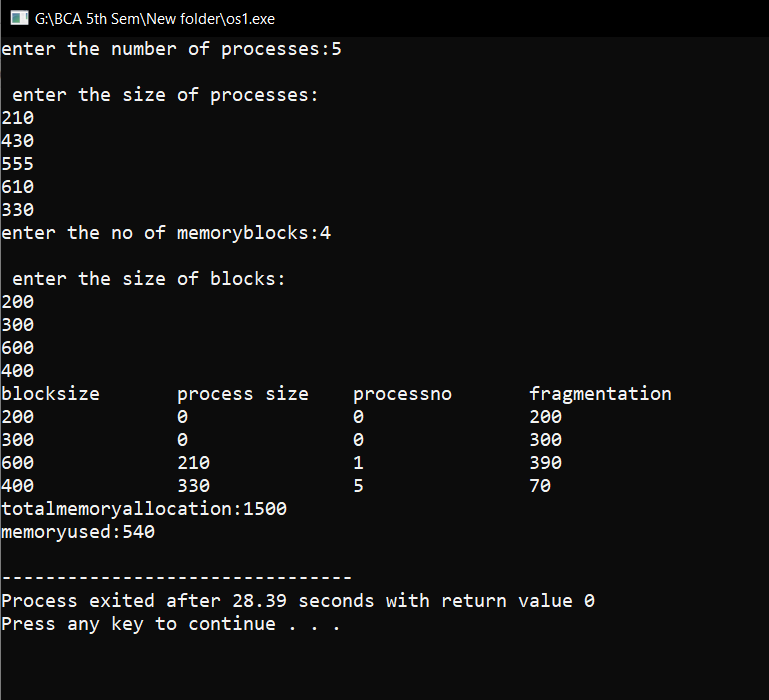
{

printf("%d\t\t%d\t\t%d\t\t%d\n",blocks[i],process1[i],job[i],frag[i]); }

printf("totalmemoryallocation:%d\n",total);

printf("memoryused:%d\n",used);

}



**b) Best Fit**

#include<stdio.h>

int main()

{

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999; static int barray[20],parray[20];

printf("\n\t\t\tMemory Management Scheme - Best Fit"); printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of processes:");

scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block no.%d:",i);

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

}

printf("\nEnter the size of the processes :-\n");

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

{

printf("Process no.%d:",i);

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

}

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

{

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

{

if(barray[j]!=1)

{

temp=b[j]-p[i];

if(temp>=0)

if(lowest>temp)

{

parray[i]=j;

lowest=temp;

}

}

}

fragment[i]=lowest;

barray[parray[i]]=1;

lowest=10000;

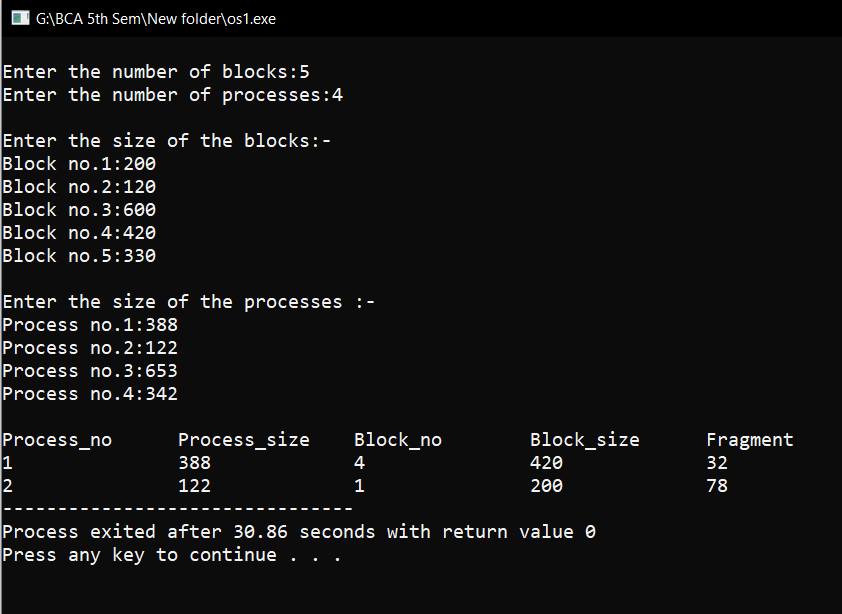
}

printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment"); for(i=1;i<=np && parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);

return 0;

}



**c) First Fit**

#include<stdio.h>

void main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

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

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

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

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

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

for(i = 0; i < pno; i++) //allocation as per first fit for(j = 0; j < bno; j++)

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

//display allocation details

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize"); for(i = 0; i < bno; i++)

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

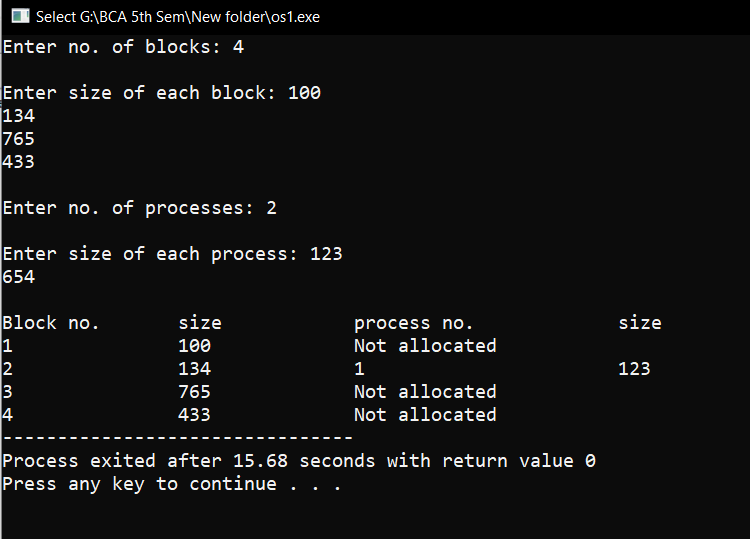
if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]); else

printf("Not allocated");

}

}



**FIFO Page Replacement.**

#include <stdio.h>

int main()

{

int referenceString[10], pageFaults = 0, m, n, s, pages, frames; printf("\nEnter the number of Pages:\t");

scanf("%d", &pages);

printf("\nEnter reference string values:\n"); for( m = 0; m < pages; m++)

{

printf("Value No. [%d]:\t", m + 1); scanf("%d", &referenceString[m]);

}

printf("\n What are the total number of frames:\t"); {

scanf("%d", &frames);

}

int temp[frames];

for(m = 0; m < frames; m++)

{

temp[m] = -1;

}

for(m = 0; m < pages; m++)

{

s = 0;

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

{

if(referenceString[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0)) {

temp[m] = referenceString[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = referenceString[m]; }

printf("\n");

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

{

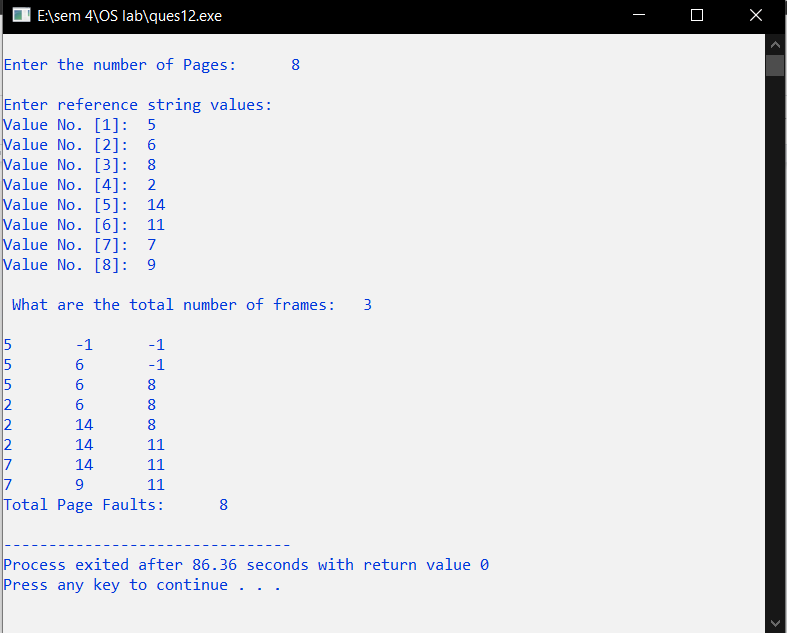
printf("%d\t", temp[n]);

}

}

printf("\nTotal Page Faults:\t%d\n", pageFaults); return 0;

}



**Optimal Page Replacement.**

#include<stdio.h>

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j < no\_of\_frames; ++j){ temp[j] = -1;

for(k = i + 1; k < no\_of\_pages; ++k){ if(frames[j] == pages[k]){

temp[j] = k;

break;

}

}

}

for(j = 0; j < no\_of\_frames; ++j){ if(temp[j] == -1){

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

pos = 0;

for(j = 1; j < no\_of\_frames; ++j){ if(temp[j] > max){

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){ printf("%d\t", frames[j]);

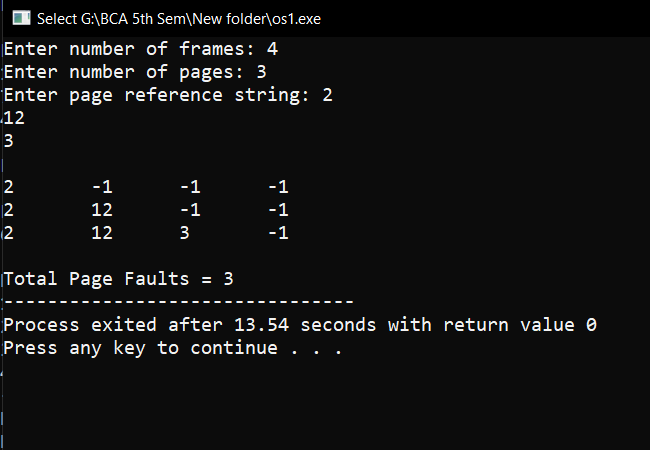
}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}



**LRU Page Replacement.**

#include<stdio.h>

int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;

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

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

printf("%d\t", frames[j]);

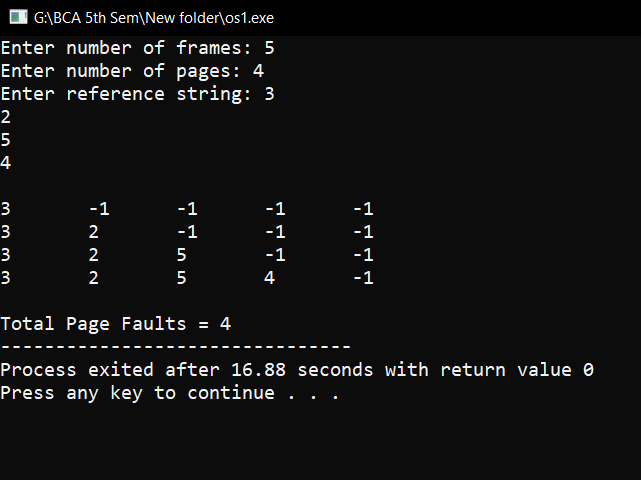
}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}



**SCAN Disk Scheduling.**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n"); scanf("%d",&move);

// logic for Scan disk scheduling

/\*logic for sort the request array \*/

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

{

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

{

if(RQ[j]>RQ[j+1])

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

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

{

if(initial<RQ[i])

{

index=i;

break;

}

}

// if movement is towards high value if(move==1)

{

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

// last movement for max size

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1); initial = size-1;

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

}

// if movement is towards low value else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

// last movement for min size

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0); initial =0;

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

{

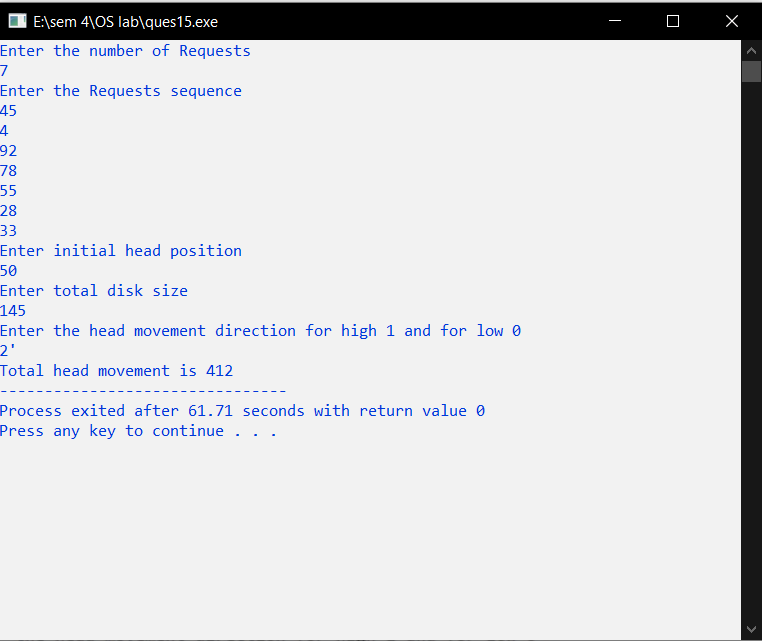
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial); initial=RQ[i];

}

}

printf("Total head movement is %d",TotalHeadMoment); return 0;

}



**.**

**SSJF Disk Scheduling.**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial,count=0; printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

// logic for sstf disk scheduling

/\* loop will execute until all process is completed\*/ while(count!=n)

{

int min=1000,d,index;

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

{

d=abs(RQ[i]-initial);

if(min>d)

{

min=d;

index=i;

}

}

TotalHeadMoment=TotalHeadMoment+min; initial=RQ[index];

// 1000 is for max

// you can use any number

RQ[index]=1000;

count++;

}

printf("Total head movement is %d",TotalHeadMoment); return 0;

}

