OS LAB :-

**1.First Come First Serve Scheduling Algorithm**

#include<stdio.h>

void waitTime(int process[],int n,int bt[],int wt[]) //to find waiting time of each process

{

int i;

wt[0]=0; //waiting time of first proc is 0

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

{

wt[i]=wt[i-1]+bt[i-1];

}

}

void turnAroundTime(int proces[],int n,int bt[],int wt[],int tat[]) //find turnaround time of each proc

{

int i;

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

{

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

}

}

void avgTime(int process[],int n,int bt[])

{

int wt[n],tat[n],i,tot\_wt=0,tot\_tat=0;

float avgWt,avgTat;

waitTime(process,n,bt,wt);

turnAroundTime(process,n,bt,wt,tat);

printf("\n\tPROCESS\tBT\tWT\tTAT");

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

{

tot\_wt+=wt[i];

tot\_tat+=tat[i];

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

}

avgWt=(tot\_wt/n);

avgTat=(tot\_tat/n);

printf("\n\n\t Average Waiting Time = %f",avgWt);

printf("\n\n\t Average Turn Around Time = %f",avgTat);

}

void main()

{

int process[10],n,Bt[10],i;

printf("\n\n Enter the number of process : ");

scanf("%d",&n);

printf("\n Enter the burst time of the process : -");

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

{

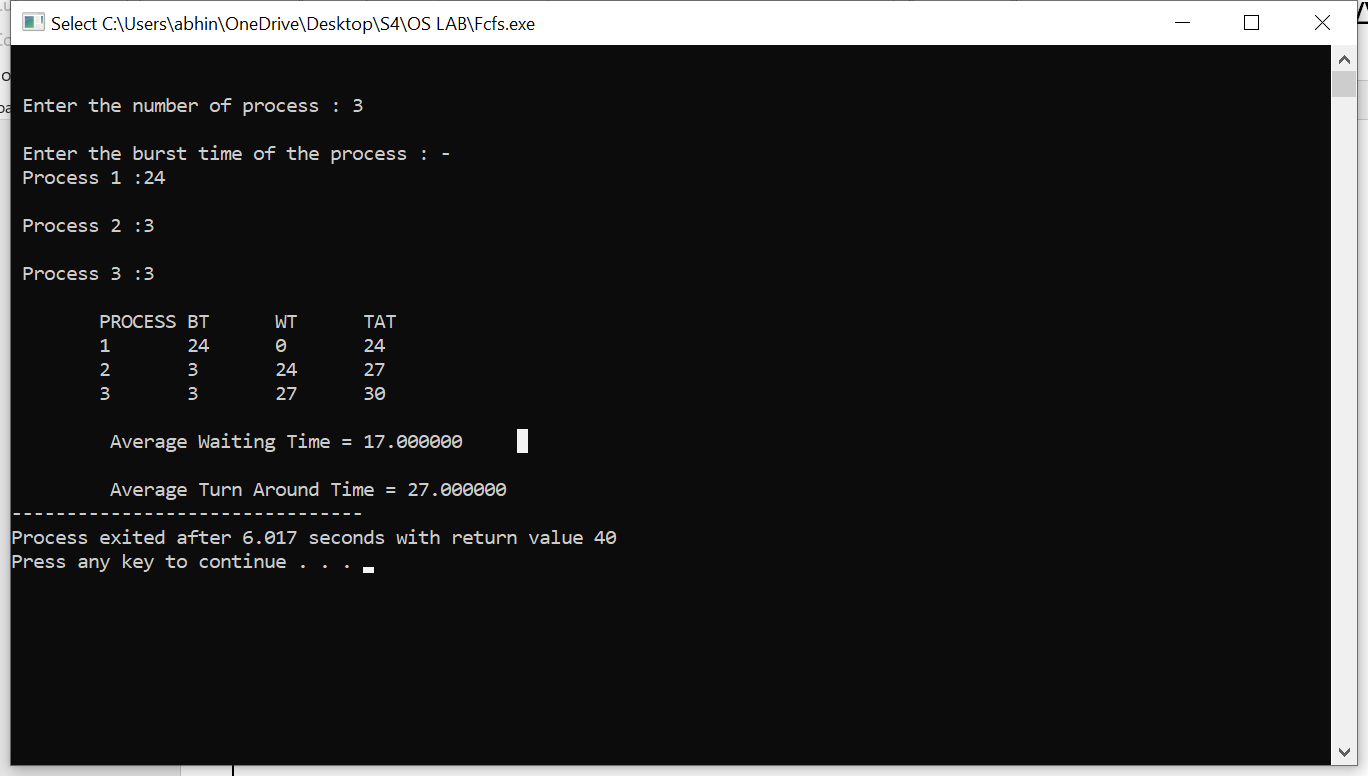
printf("\n Process %d :",(i+1));

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

}

avgTime(process,n,Bt);

}



**2.Shortest Job First Scheduling Algorithm**

//Shortest Job First

#include<stdio.h>

int process[10],bt[10],wt[10],tat[10];

int n,totWt=0,totTat=0;

float avgWt,avgTat;

void waitTime(int process[],int n)

{

int i;

wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1];

totWt+=wt[i];

}

}

void turnAroundTime(int process[],int n)

{

int i;

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

{

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

totTat+=tat[i];

}

}

void main()

{

int i,j,temp;

printf("\n Enter the number of processs : ");

scanf("%d",&n);

printf("\n Enter the burst time of process : ");

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

{

printf("\n Process %d :",(i+1));

process[i]=i+1;

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

}

//Sorting the processes in ascending order of burst time

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

{

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

{

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

{

temp=bt[j];

bt[j]=bt[j+1];

bt[j+1]=temp;

temp=process[j]; //while sorting, process are also sorted

process[j]=process[j+1];

process[j+1]=temp;

}

}

}

waitTime(process,n);

turnAroundTime(process,n);

avgWt=totWt/n;

avgTat=totTat/n;

printf("\n\tPROCESS\tBT\tWT\tTAT");

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

{

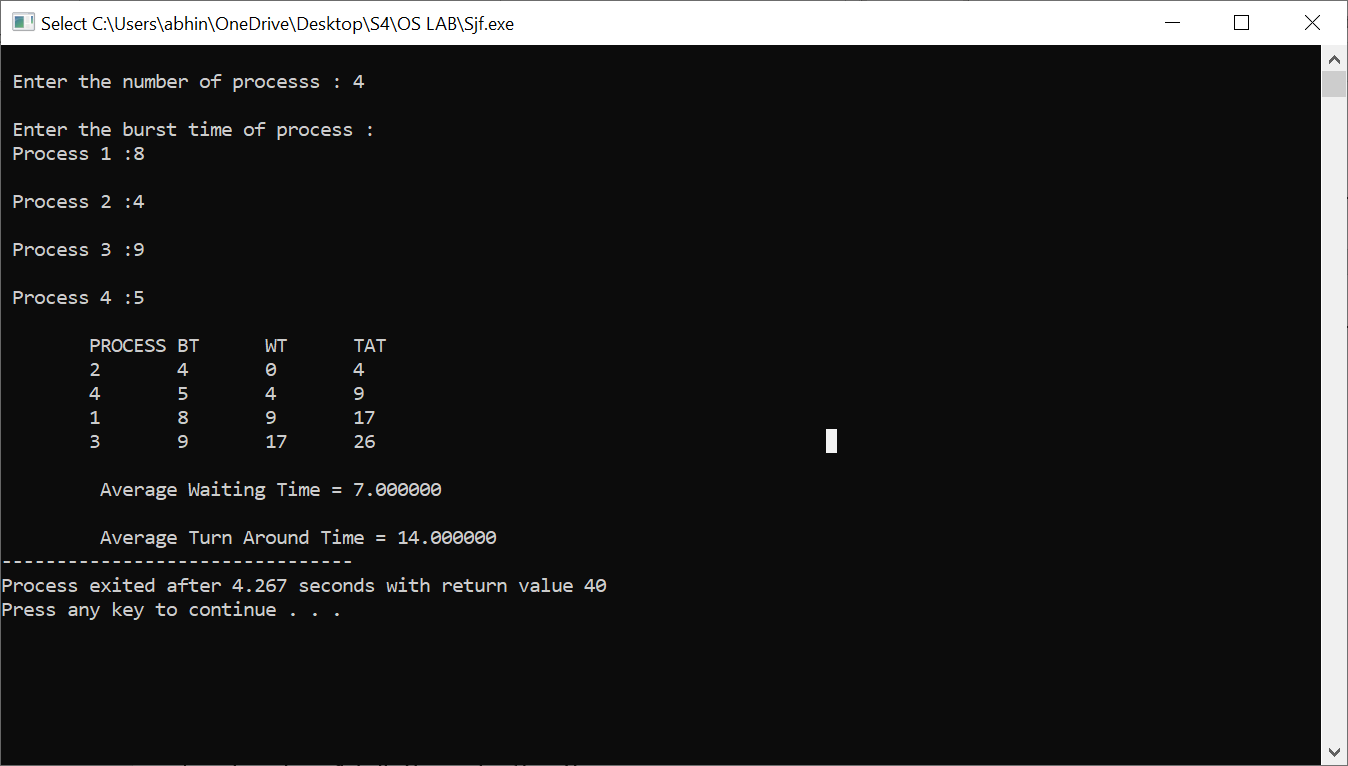
printf("\n\t%d\t%d\t%d\t%d",process[i],bt[i],wt[i],tat[i]);

}

printf("\n\n\t Average Waiting Time = %f",avgWt);

printf("\n\n\t Average Turn Around Time = %f",avgTat);

}



**3.Consumer Producer**

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return (++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\n Produced Item %d ",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\n Consumed Item %d ",x);

x--;

mutex=signal(mutex);

}

void main()

{

int n;

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\n Enter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

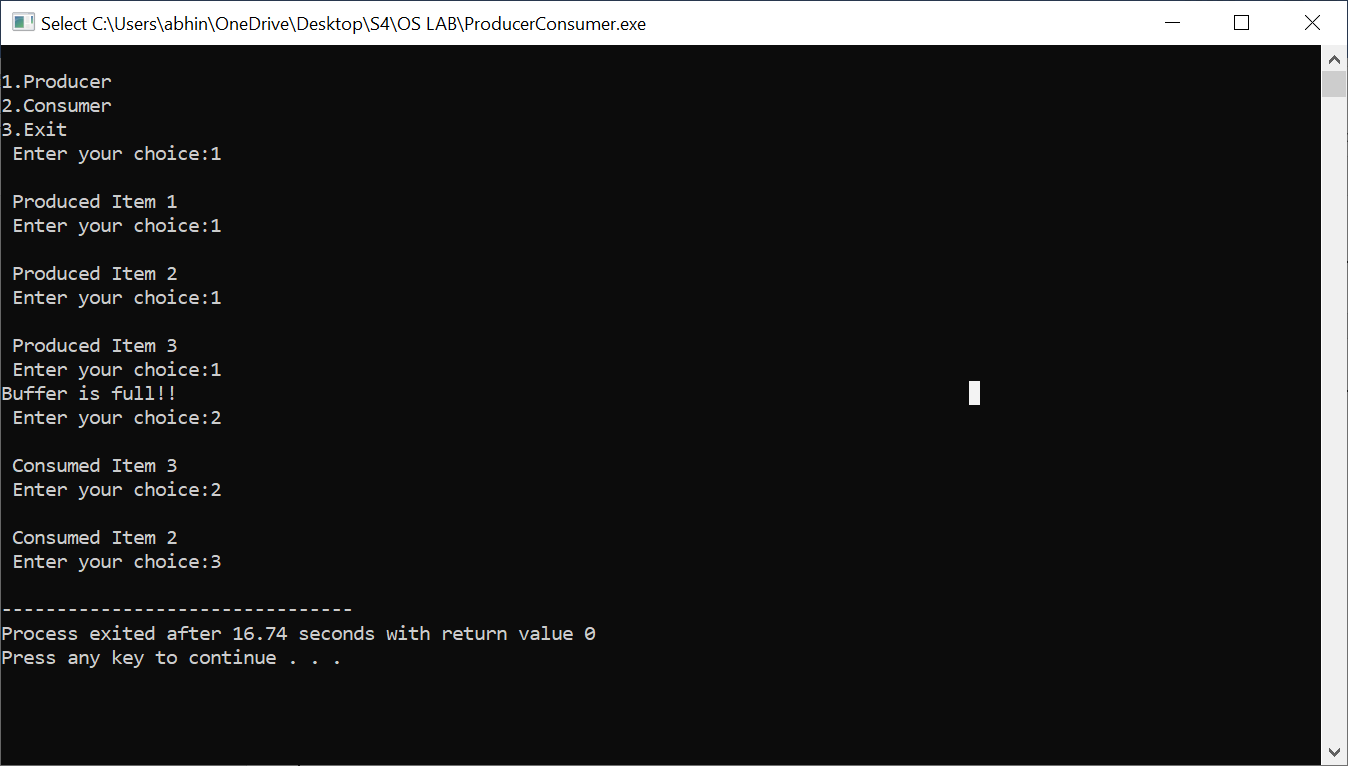
case 3: exit(0);

break;

}

}

}



**First Fit Memory Allocation**

#include<stdio.h>

#define size 10

void main()

{

int block[size],file[size],bno,fno,flag[size],alloc[size],frag[size],i,j,temp;

printf("\n Enter the number of blocks : ");

scanf("%d",&bno);

printf("\n Enter the number of files : ");

scanf("%d",&fno);

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

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

{

printf("\n Block %d : ",i);

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

}

printf("\n Enter the size of files : ");

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

{

printf("\n File %d : ",i);

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

}

//Initialising flag[] as 0 and Alloc[] as -1

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

flag[i]=0;

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

alloc[i]=-1;

// First fit code

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

{

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

{

if(flag[j]==0)

{

temp=block[j]-file[i];

if(temp>=0)

{

alloc[i]=j;

flag[i]=1;

break;

}

}

}

frag[i]=temp;

}

printf("\n First Fit Allocation : - \n");

printf("\n\tFno.\tFsize\tBno.\t\tBsize\t\tFrag\n");

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

{

printf("\n\t%d\t%d\t%d\t\t%d\t\t%d",i,file[i],alloc[i],block[alloc[i]],frag[i]);

}

}

**Best Fit Memory Allocation**

#include<stdio.h>

#define size 10

void main()

{

int block[size],file[size],bno,fno,flag[size],alloc[size],frag[size]

i,j,temp,lowest =10000;

printf("\n Enter the number of blocks : ");

scanf("%d",&bno);

printf("\n Enter the number of files : ");

scanf("%d",&fno);

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

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

{

printf("\n Block %d : ",i);

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

}

printf("\n Enter the size of files : ");

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

{

printf("\n File %d : ",i);

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

}

//Initialising flag[] as 0 and Alloc[] as -1

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

flag[i]=0;

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

alloc[i]=-1;

//Best fit code

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

{

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

{

if(flag[j]==0)

{

temp=block[j]-file[i];

if(temp>=0)

{

if(lowest>temp)

{

alloc[i]=j;

lowest=temp;

}

}

}

}

frag[i]=lowest;

flag[alloc[i]]=1;

lowest=100000;

}

printf("\n Best Fit Allocation : - \n");

printf("\n\tFno.\tFsize\tBno.\t\tBsize\t\tFrag\n");

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

{

printf("\n\t%d\t%d\t%d\t\t%d\t\t%d",i,file[i],alloc[i],block[alloc[i]],frag[i]);

}

}

**Worst Fit Memory Allocation**

#include<stdio.h>

#define size 10

void main()

{

int block[size],file[size],bno,fno,flag[size],alloc[size],frag[size],i,j,temp,highest=0;

printf("\n Enter the number of blocks : ");

scanf("%d",&bno);

printf("\n Enter the number of files : ");

scanf("%d",&fno);

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

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

{

printf("\n Block %d : ",i);

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

}

printf("\n Enter the size of files : ");

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

{

printf("\n File %d : ",i);

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

}

//Initialising flag[] as 0 and alloc[] as -1

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

flag[i]=0;

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

alloc[i]=-1;

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

{

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

{

if(flag[j]==0)

{

temp=block[j]-file[i];

if(temp>=0)

{

if(highest<temp)

{

alloc[i]=j;

highest=temp;

}

}

}

}

frag[i]=highest;

flag[alloc[i]]=1;

highest=0;

}

printf("\n Worst Fit Allocation : - \n");

printf("\n\tFno.\tFsize\tBno.\t\tBsize\t\tFrag\n");

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

{

printf("\n\t%d\t%d\t%d\t\t%d\t\t%d",i,file[i],alloc[i],block[alloc[i]],frag[i]);

}

}

**Page Replacement FIFO**

#include<stdio.h>

void main()

{

int page[20],frame[20],np,nf,i,j,k,avail,count=0;

printf("\n Enter the number of pages : ");

scanf("%d",&np);

printf("\n Enter the page numbers : ");

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

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

printf("\n Enter the number of frames : ");

scanf("%d",&nf);

j=0;

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

frame[i]=-1;

printf("\n Ref String\tPage Numbers");

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

{

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

avail=0;

for(k=0;k<nf;k++)

{

if(frame[k]==page[i])

{

avail=1;

for(k=0;k<nf;k++)

printf(" %d ",frame[k]);

}

if(avail==0)

{

frame[j]=page[i];

j=(j+1)%nf;

count++;

for(k=0;k<nf;k++)

printf(" %d ",frame[k]);

}

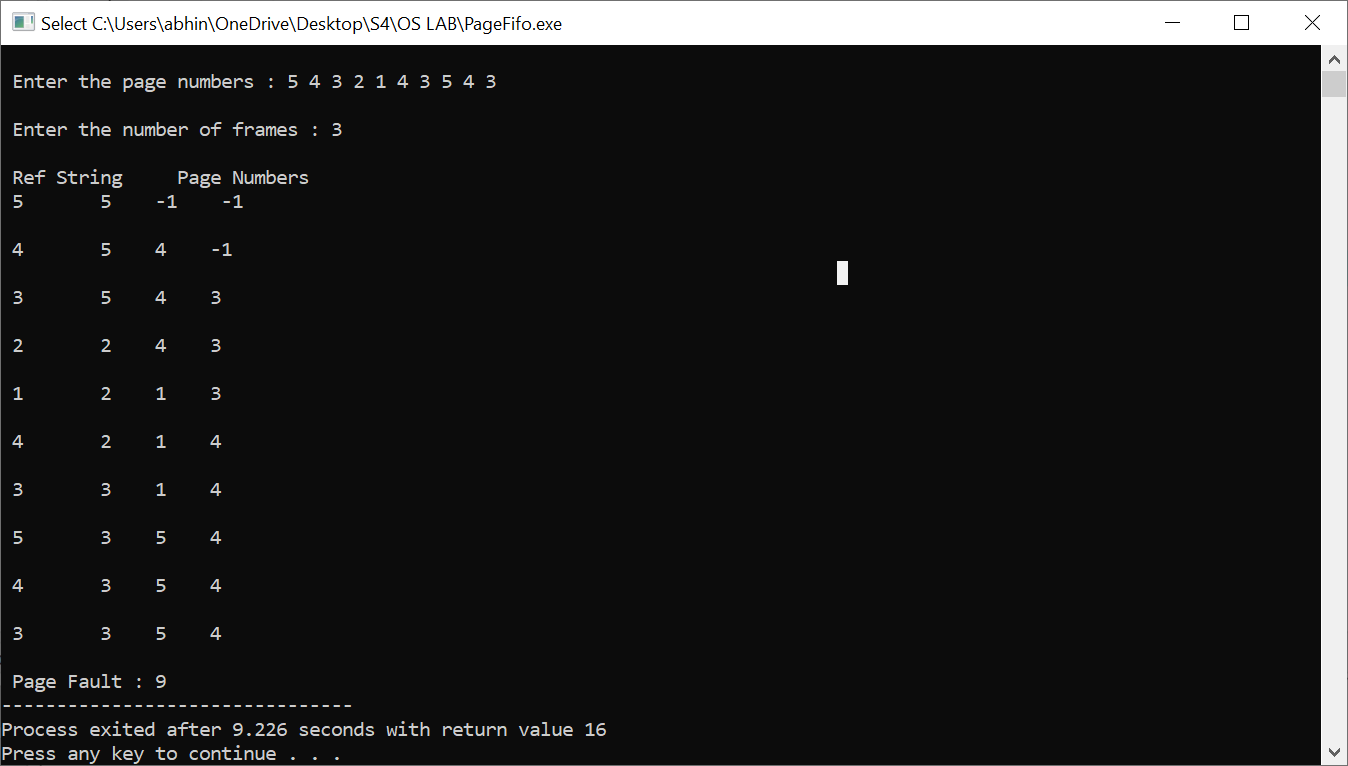
printf("\n");

}

}

printf("\n Page Fault : %d",count);

}



**Replacement LRU**

#include<stdio.h>

int Lru(int time[],int n)

{

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

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

{

if(time[i]<min)

{

min=time[i];

pos=i;

}

}

return pos;

}

void main()

{

int page[20],frame[20],time[20],np,nf,i,j,pos,flag1,flag2,count=0,fault=0;

printf("\n Enter the number of pages : ");

scanf("%d",&np);

printf("\n Enter the page numbers : ");

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

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

printf("\n Enter the number of frames : ");

scanf("%d",&nf);

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

frame[i]=-1;

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

{

flag1=flag2=0;

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

{

if(frame[j]==page[i])

{

count++;

time[j]=count;

flag1=flag2=1;

break;

}

}

if(flag1==0)

{

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

{

if(frame[j]==-1)

{

count++;

fault++;

time[j]=count;

frame[j]=page[i];

flag2=1;

break;

}

}

}

if(flag2==0)

{

pos=Lru(time,nf);

count++;

fault++;

frame[pos]=page[i];

time[pos]=count;

}

printf("\n");

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

{

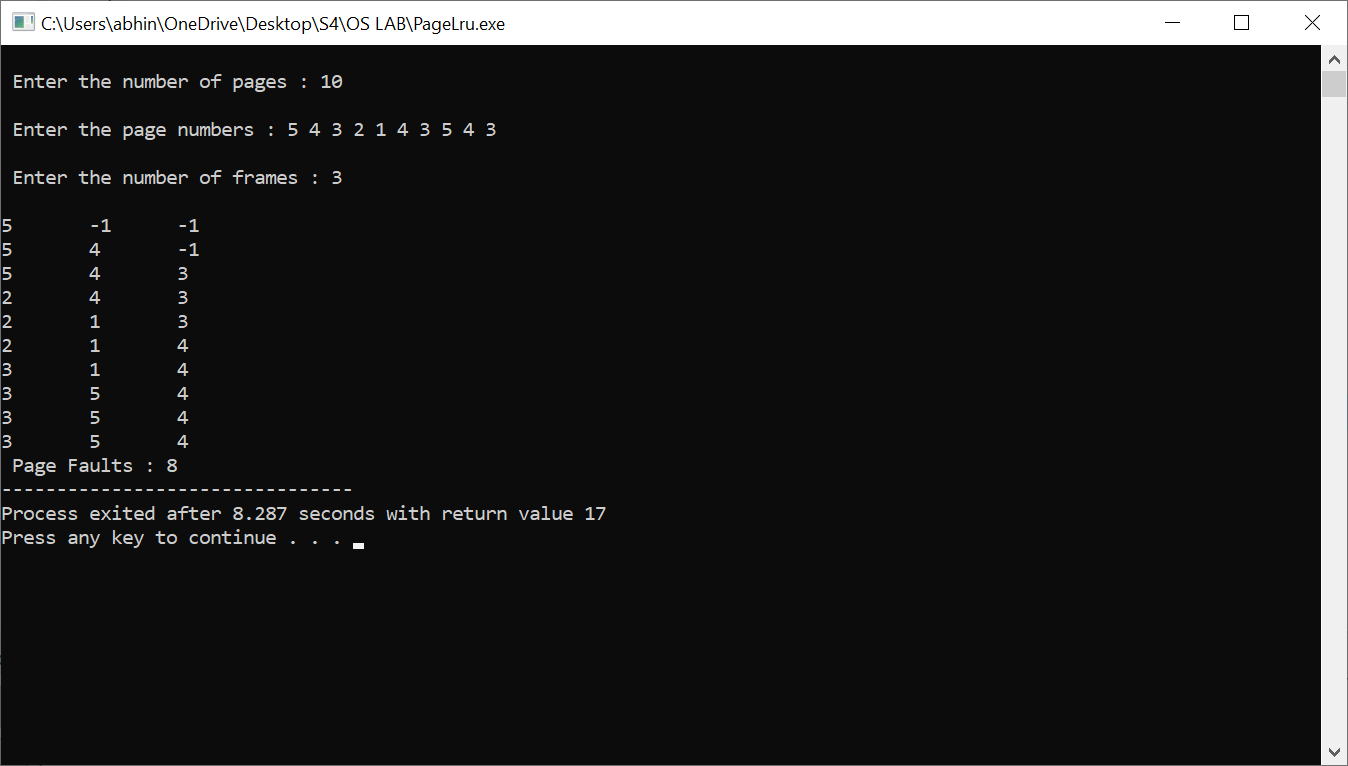
printf("%d\t",frame[j]);

}

}

printf("\n Page Faults : %d",fault);

}



**Page Replacement algorithm Optimal**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int page[20],frame[20],temp[20],np,nf,i,j,k,pos,flag1,flag2,flag3,max,faults=0;

printf("\n Enter the number of pages : ");

scanf("%d",&np);

printf("\n Enter the page numbers : ");

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

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

printf("\n Enter the number of frames : ");

scanf("%d",&nf);

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

frame[i]=-1;

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

{

flag1=flag2=0;

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

{

if(frame[j]==page[i])

{

flag1=flag2=1;

break;

}

}

if(flag1==0)

{

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

{

if(frame[j]==-1)

{

faults++;

frame[j]=page[i];

flag2=1;

break;

}

}

}

if(flag2==0)

{

flag3=0;

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

{

temp[j]=-1;

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

{

if(frame[j]==page[k])

{

temp[j]=k;

break;

}

}

}

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

{

if(temp[j]==-1)

{

pos=j;

flag3=1;

break;

}

}

if(flag3==0)

{

max=temp[0];

pos=0;

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

{

if(temp[j]>max)

{

max=temp[j];

pos=j;

}

}

}

frame[pos]=page[i];

faults++;

}

printf("\n");

for(j = 0; j < nf; ++j){

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

}

}

printf("\n\n Page faults : %d ",faults);

}

