**PROGRAM CODE:**

#include<stdio.h>

int safety(int p,int r,int finish[],int available[],int allocation[][r],int request[][r])

{

int j,k,flag,cnt=0;

while(cnt!=p)

{

flag=0;

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

{

if (finish[k]==0)

{

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

if(request[k][j]>available[j])

break;

if(j==r)

{

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

available[j]+=allocation[k][j];

finish[k]=1;

cnt++;

flag=1; //there exists a process that can be satisfied

}

} //end if finish

}//end for

if(flag==0)

return flag;

} //end while

return flag;

}

void main()

{

int p,r,safe,i,j,k;

printf("\t\tDEADLOCK DETECTION \n");

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

scanf("%d",&p);

printf("\nEnter the no.of resources: ");

scanf("%d",&r);

int available[r],request[p][r],allocation[p][r],finish[p];

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

finish[k]=0;

printf("\nEnter the currently available no.of instances for each resource: \n");

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

{

printf("Resource %c: ",i+65);

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

}

printf("\nEnter the allocated no.of resources for each process: ");

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

{

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

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

{

printf("Resource %c: ",j+65);

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

}

}

printf("\nEnter the request for each process: ");

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

{

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

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

{

printf("Resource %c: ",j+65);

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

}

}

safe=safety(p,r,finish,available,allocation,request);

if(safe==0)

{

printf("\nSystem is in deadlocked state");

printf("\nProcesses involved in deadlock are: ");

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

if(finish[k]==0)

printf("P%d ",k);

printf("\n");

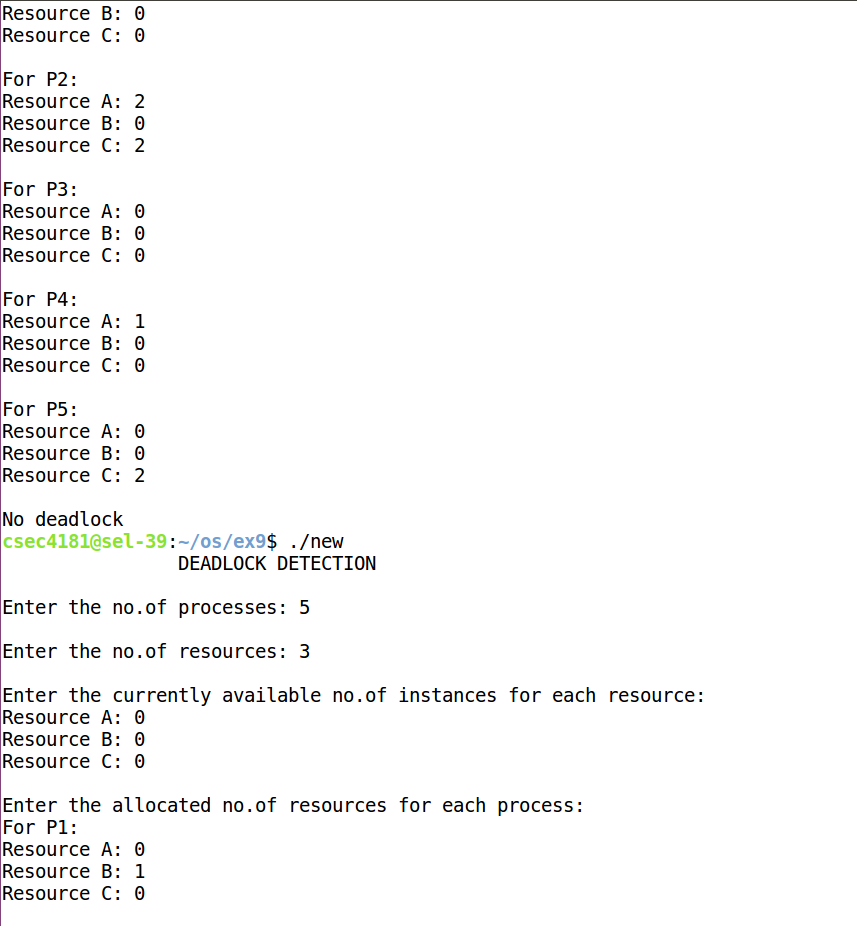
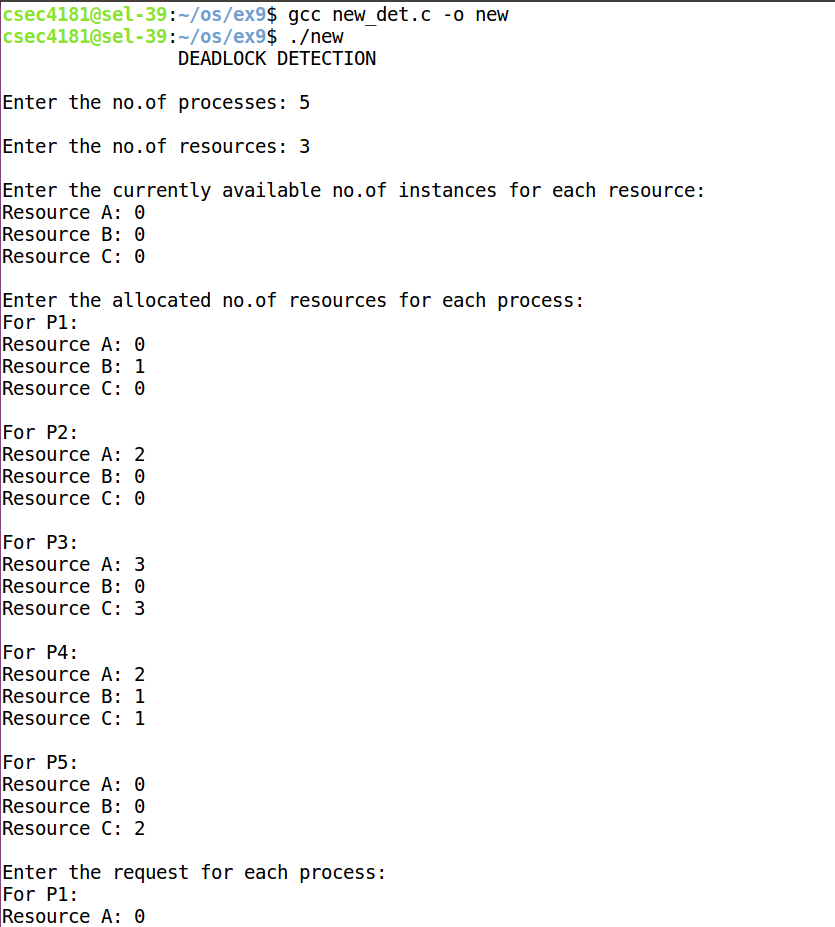
}

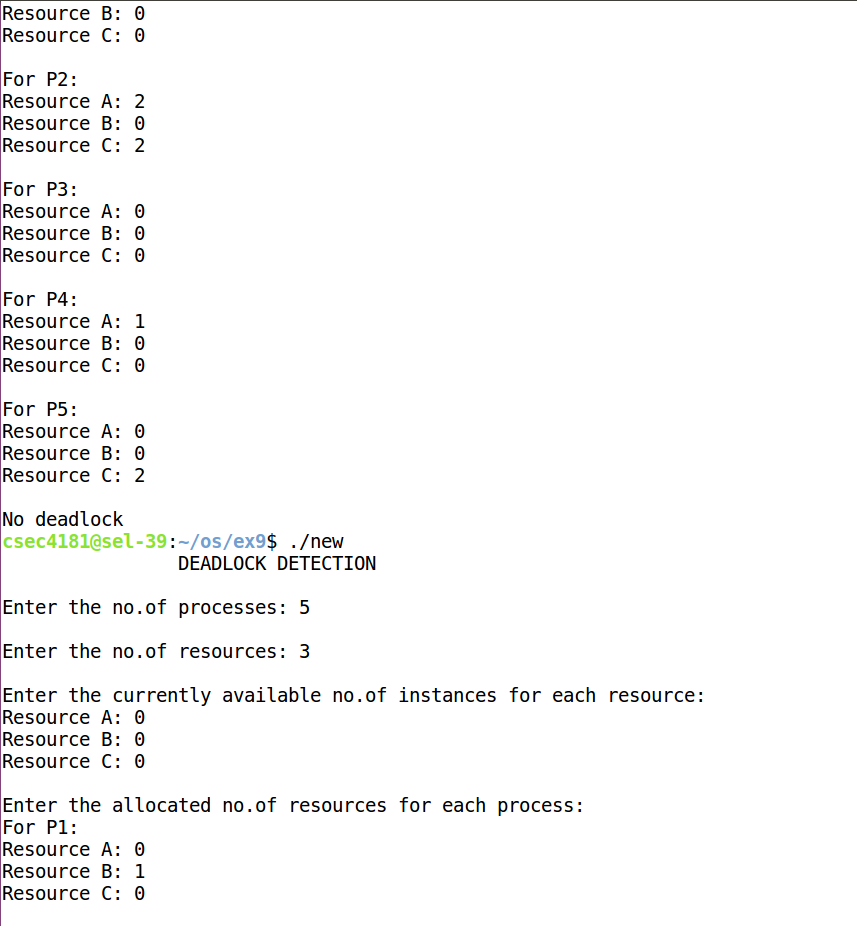
else

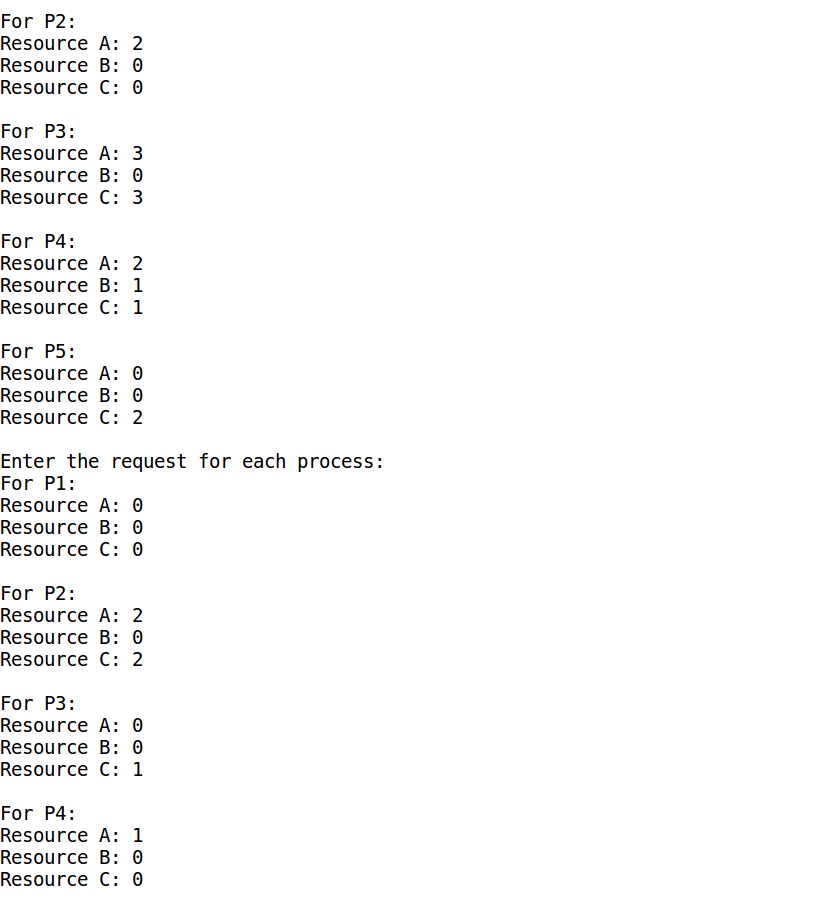
printf("\nNo deadlock\n");

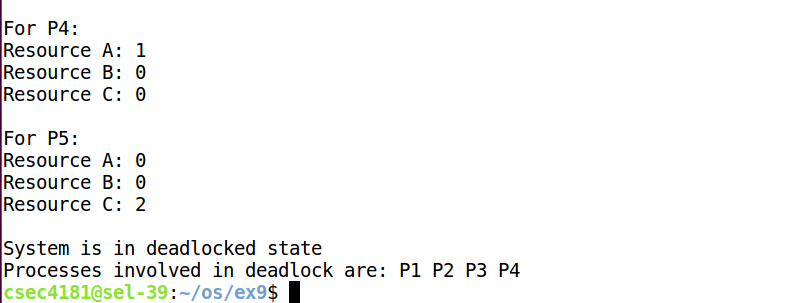
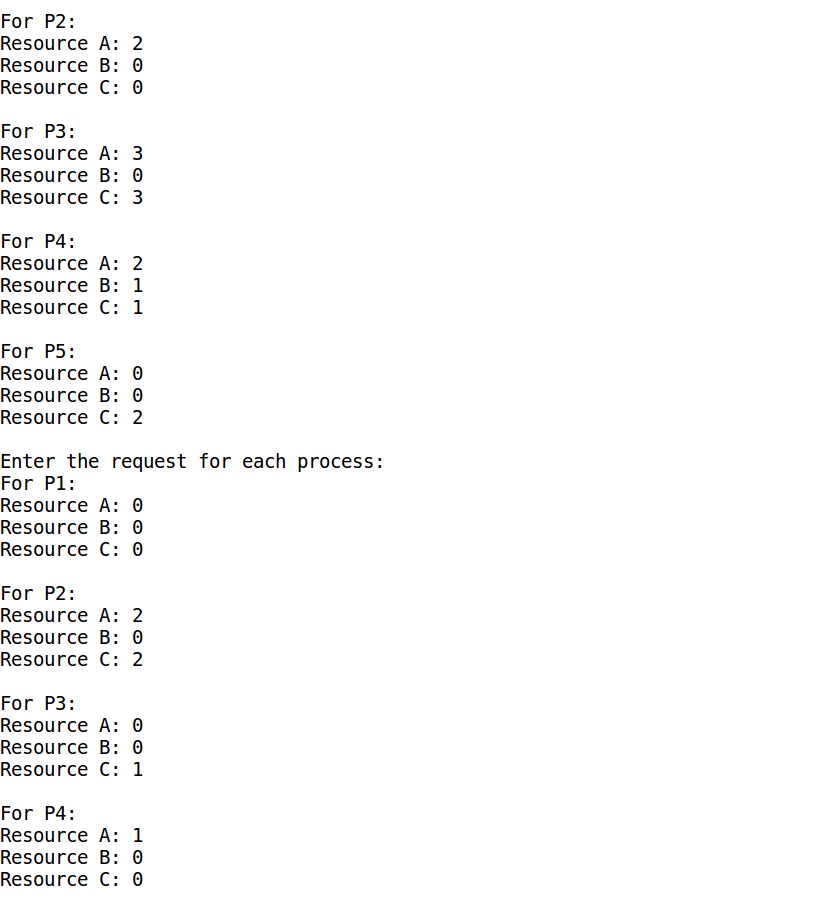
}

**OUTPUT:**









**PROGRAM CODE:**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct input

{

char \*\*data;

int count;

}in;

float avg=0.0,median,sd;

int min,max; /\* this data is shared by the thread(s) \*/

void \*worker\_avg(void \*param)

{

struct input \*t=(struct input \*)param;

int i,sum = 0;

for (i = 0; i < t->count; i++)

sum += atoi(t->data[i]);

avg=(float)sum/(t->count-1);

pthread\_exit(0);

}

void \*worker\_min(void \*param)

{

struct input \*t=(struct input \*)param;

int i;

min = atoi(t->data[1]);

for (i = 2; i < t->count; i++)

if (atoi(t->data[i])<min)

min=atoi(t->data[i]);

pthread\_exit(0);

}

void \*worker\_max(void \*param)

{

struct input \*t=(struct input \*)param;

int i;

max = atoi(t->data[1]);

for (i = 2; i < t->count; i++)

if (atoi(t->data[i])>max)

max=atoi(t->data[i]);

pthread\_exit(0);

}

void \*worker\_median(void \*param)

{

struct input \*t=(struct input \*)param;

int i,j,n=t->count;

char \* temp;

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

{

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

{

if(atoi(t->data[j]) > atoi(t->data[j+1]))

{

temp=t->data[j];

t->data[j]=t->data[j+1];

t->data[j+1]=temp;

}

}

}

if((n-1)%2==0)

{ median=(float) ( atof(t->data[n/2]) + atof(t->data[(n+1)/2]) ) /2; }

else

median=atof(t->data[n/2]);

pthread\_exit(0);

}

void \*worker\_sd(void \*param)

{

struct input \*t=(struct input \*)param;

int arr[t->count-1] , i;

float average,sum=0;

for(i=0;i<t->count-1;i++)

arr[i]=atoi(t->data[i+1]);

for (i = 0; i < t->count-1; i++)

sum += arr[i];

average=(float)sum/(t->count-1);

sum=0;

for (i = 0; i < t->count-1; i++)

sum+=(float)pow((arr[i]-average),2);

sd=(float)sqrt(sum/(t->count-1));

pthread\_exit(0);

}

void main(int argc, char \*argv[])

{

pthread\_t tid1,tid2,tid3,tid4,tid5; /\* the thread identifier \*/

pthread\_attr\_t attr; /\* set of thread attributes \*/

/\* get the default attributes \*/

pthread\_attr\_init(&attr);

/\* create the thread \*/

in.data=argv;

in.count=argc;

pthread\_create(&tid1,&attr,worker\_avg,&in);

pthread\_create(&tid2,&attr,worker\_min,&in);

pthread\_create(&tid3,&attr,worker\_max,&in);

pthread\_create(&tid4,&attr,worker\_median,&in);

pthread\_create(&tid5,&attr,worker\_sd,&in);

/\* wait for the thread to exit \*/

pthread\_join(tid1,NULL);

pthread\_join(tid2,NULL);

pthread\_join(tid3,NULL);

pthread\_join(tid4,NULL);

pthread\_join(tid5,NULL);

printf("Avg = %.2f\n",avg);

printf("Min = %d\n",min);

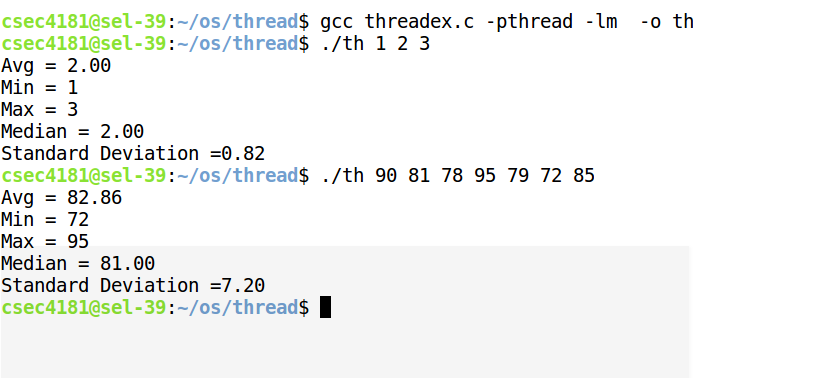
printf("Max = %d\n",max);

printf("Median = %.2f\n",median);

printf("Standard Deviation =%.2f\n",sd);

}

**OUTPUT:**

****

**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

typedef struct PageTable pagetable;

int pgsize, phy\_size;

int no\_of\_frames, noff; //noff-no of free frames

int frames[100];

struct PageTable

{

int pgno;

int fr\_no;

pagetable\* next;

};

pagetable\* process[10];

void display\_table(pagetable \*head)

{

for(pagetable \*temp = head; temp != NULL; temp = temp->next)

printf("Page %d: Frame %d\n", temp->pgno, temp->fr\_no);

printf("\n");

}

void allocate()

{

int i,pid, mem\_req, no\_of\_pages,fr\_idx=0;

pagetable \*head, \*temp, \*prev;

printf("\nEnter the process ID: ");

scanf("%d", &pid);

printf("Enter the memory required for the process in KB: ");

scanf("%d", &mem\_req);

no\_of\_pages = mem\_req/pgsize;

if(mem\_req%pgsize!=0)

no\_of\_pages++;

printf("\nProcess is divided into %d pages.\n", no\_of\_pages);

if(no\_of\_pages>noff)

{

printf("Not available!\n");

return;

}

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

{

temp = (pagetable\*)malloc(sizeof(pagetable));

temp->pgno = i;

while(frames[fr\_idx]==0) fr\_idx++; //frame not free

temp->fr\_no = fr\_idx;

frames[fr\_idx]=0; //frame occupied

fr\_idx++;

temp->next = NULL;

noff--;

if(i==0)

head = temp;

else

prev->next = temp;

prev = temp;

}

process[pid] = head;

printf("Page Table for process ID: %d\n", pid);

display\_table(process[pid]);

}

void deallocate()

{

pagetable \*pd, \*temp, \*d;

int pid;

printf("Enter the process id to deallocate:");

scanf("%d", &pid);

temp = process[pid];

if(temp==NULL)

{

printf("Not allocated!\n");

return;

}

while(temp!=NULL)

{

d = temp;

frames[d->fr\_no]=1; //frame is made free

noff++;

temp = temp->next;

free(d);

}

process[pid] = NULL;

printf("Deallocated!\n");

}

void display\_all()

{

int i;

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

if(process[i] != NULL)

{ printf("Page Table for process ID: %d\n", i);

display\_table(process[i]);

}

}

void display\_free\_frames()

{

int i;

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

if(frames[i]==1) printf("%d ", i);

printf("\n");

}

void addressmap()

{

int i,pid, la, page\_no,offset,frame\_no, pa;

pagetable \*h , \*temp;

printf("Enter the process id: ");

scanf("%d", &pid);

printf("Enter the logical address: ");

scanf("%d", &la);

page\_no = la / (pgsize \* 1024);

offset = la % (pgsize \* 1024);

temp=process[pid];

for(i = 0; i < page\_no; i++) //find the page

temp = temp->next;

frame\_no = temp->fr\_no;

pa = (frame\_no \* pgsize \* 1024 )+ offset; //frame size=page size

printf("Physical address: %d\n", pa);

}

void main()

{

int i,fr,ch=0;

for(i=0;i<100;i++) frames[i]=0; //frame free

printf("\n\tPAGING TECHNIQUE");

printf("\n\nEnter physical memory size in KB: ");

scanf("%d", &phy\_size);

printf("Enter the page size in KB: ");

scanf("%d", &pgsize);

no\_of\_frames = phy\_size / pgsize; //page size =frame size

printf("Physical memory is divided into %d frames.\n", no\_of\_frames);

for(i = 0; i < no\_of\_frames/2; i++) //half of the frames are free

{

fr = random()%no\_of\_frames;

if(frames[fr] != 1)

{

frames[fr] = 1;

noff++;

}

else //if same no is being generated again

i--;

}

do

{

printf("\n1.Process Request\n2.Deallocate\n3.Page Table display for all input process\n4.Free Frame list display\n5.Address Mapping\n6.Exit\n\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

allocate(); break;

case 2:

deallocate(); break;

case 3:

display\_all(); break;

case 4:

display\_free\_frames(); break;

case 5:

addressmap(); break;

case 6:

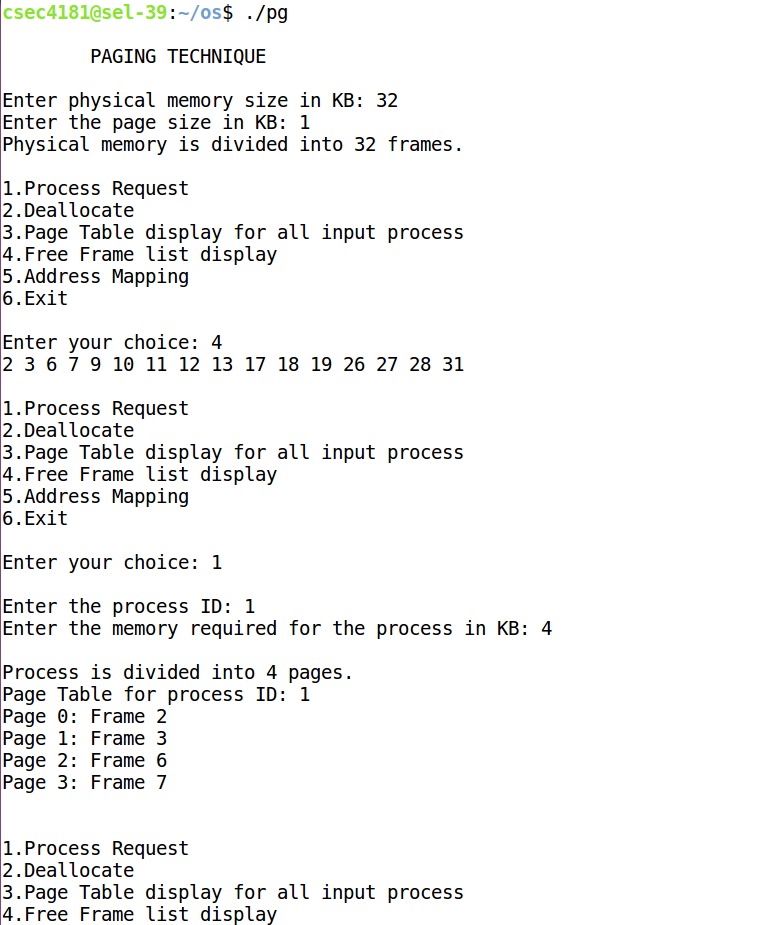
exit(0);

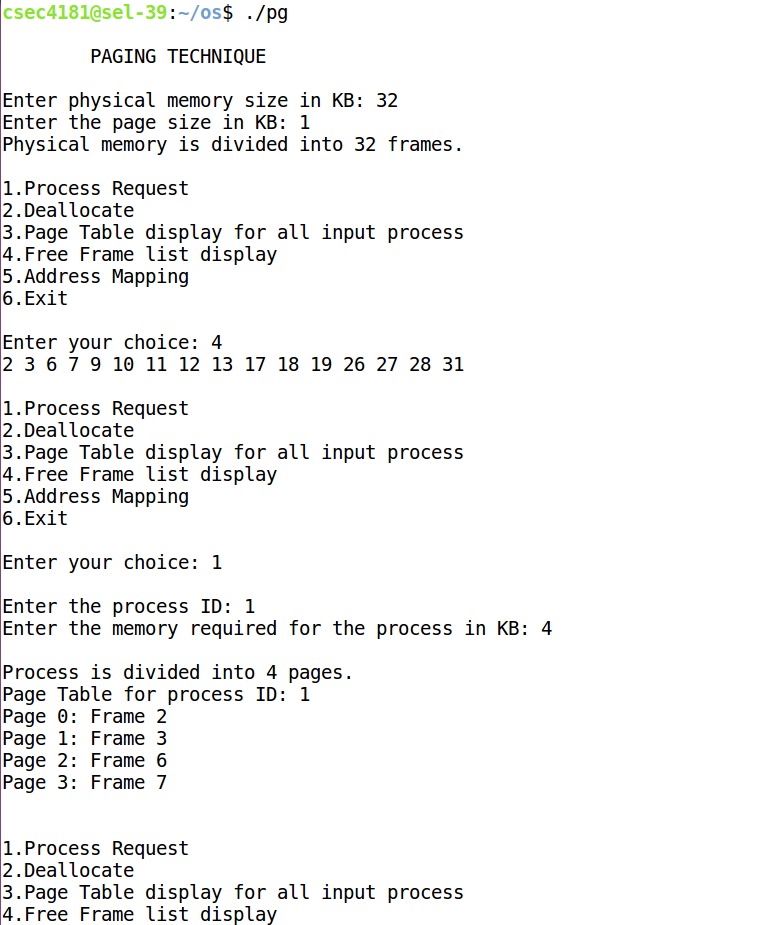
}

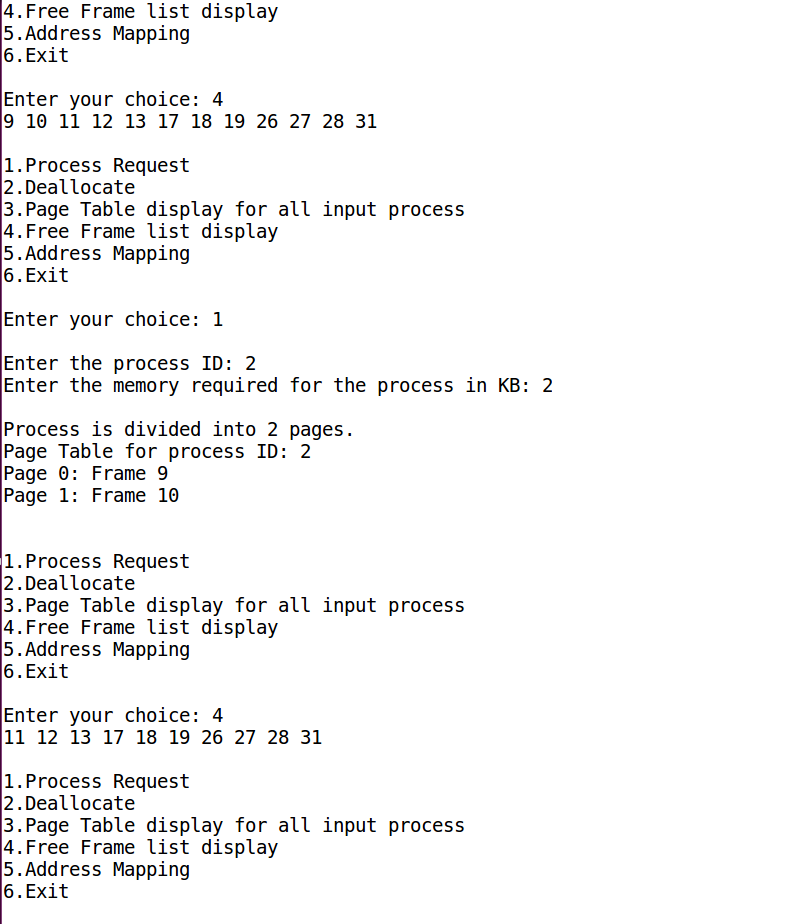
}while(1);

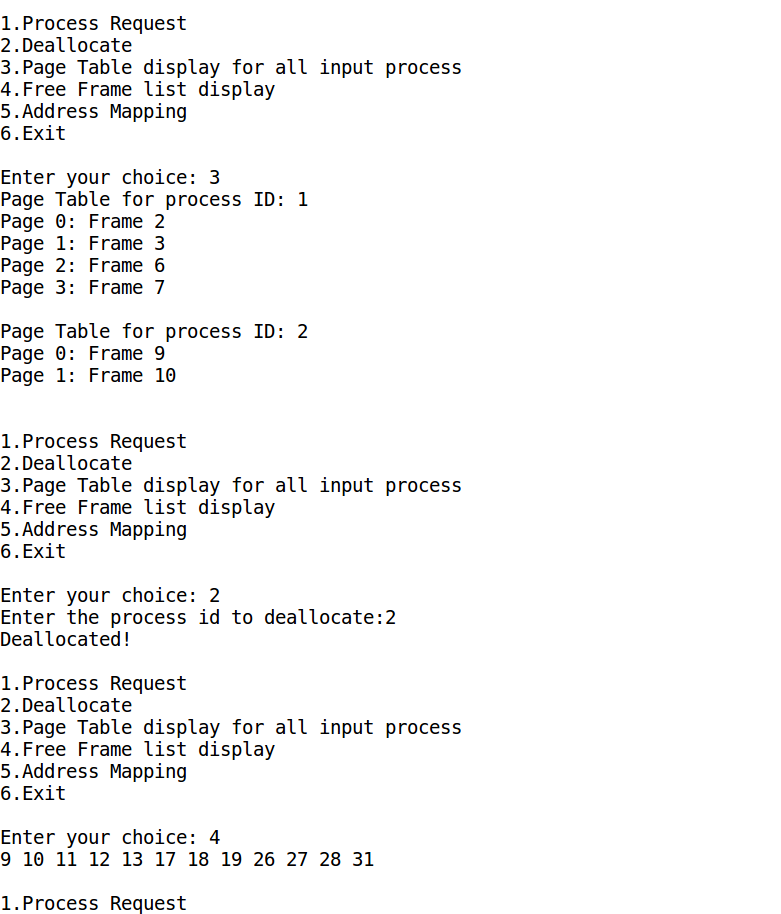
}

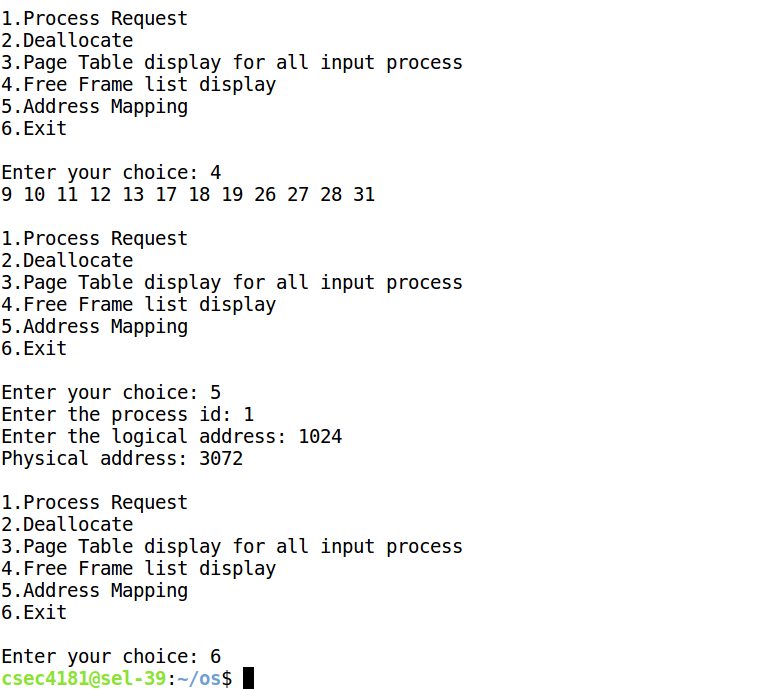
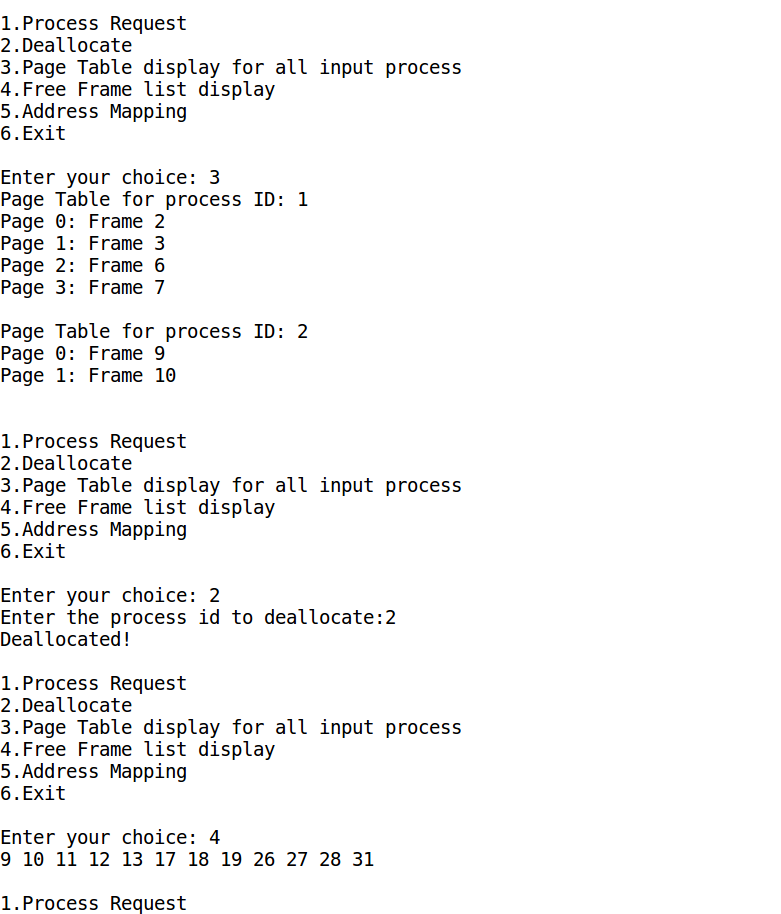
**OUTPUT:**

****

****

****

****

****

**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

int find(int ref\_string, int \*frames, int frames\_req)

{

for(int i=0;i<frames\_req;i++)

{

if(ref\_string==frames[i])

return 1;

}

return 0;

}

int find\_not\_recently\_used(int \*ref\_string, int n,int \*frames, int frames\_req, int curr)

{

int cnt[frames\_req],max,i;

for(i=0;i<frames\_req;i++)

{

cnt[i]=0;

for(int j=curr-1;j>=0;--j)

{

if(frames[i]==ref\_string[j])

break;

++cnt[i];

}

}

max=cnt[0];

curr=0;

i=1;

while(i!=frames\_req)

{

if(cnt[i]>max)

{

max=cnt[i];

curr=i;

}

++i;

}

return curr;

}

int find\_not\_req\_next(int \*ref\_string, int n,int \*frames, int frames\_req, int curr)

{

int cnt[frames\_req],max,i;

for(i=0;i<frames\_req;i++)

{

cnt[i]=0;

for(int j=curr+1;j<n;j++)

{

if(frames[i]==ref\_string[j])

break;

++cnt[i];

}

}

max=cnt[0];

curr=0;

i=1;

while(i!=frames\_req)

{

if(cnt[i]>max)

{

max=cnt[i];

curr=i;

}

++i;

}

return curr;

}

void FIFO(int frames\_req, int \*ref\_string, int n)

{

printf("\n\n\tFIFO ALGORITHM\n\n");

int ptr=-1, pg\_fault=0,cnt,i;

int frames[frames\_req];

for(cnt=0;cnt<frames\_req;cnt++)

{

frames[cnt]=ref\_string[cnt];

++pg\_fault;

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

if(j>cnt)

printf("X ");

else

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

printf("\n");

}

while(cnt!=n)

{

i=find(ref\_string[cnt], frames,frames\_req);

if(i==0)//not available

{

++pg\_fault;

if(ptr+1!=frames\_req)

++ptr;

else

ptr=0;

frames[ptr]=ref\_string[cnt];

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

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

printf("\n");

}

++cnt;

}

printf("\nTotal no of page faults = %d\n",pg\_fault);

}

void optimal(int frames\_req, int \*ref\_string, int n)

{

printf("\n\n\tOPTIMAL ALGORITHM\n\n");

int ptr=0, pg\_fault=0,cnt,i;

int frames[frames\_req];

for(cnt=0;cnt<frames\_req;cnt++)

{

frames[cnt]=ref\_string[cnt];

++pg\_fault;

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

if(j>cnt)

printf("X ");

else

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

printf("\n");

}

while(cnt!=n)

{

i=find(ref\_string[cnt], frames,frames\_req);

if(i==0)//not available

{

++pg\_fault;

ptr=find\_not\_req\_next(ref\_string,n,frames,frames\_req,cnt);

frames[ptr]=ref\_string[cnt];

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

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

printf("\n");

}

++cnt;

}

printf("\nTotal no of page faults = %d\n",pg\_fault);

}

void LRU(int frames\_req, int \*ref\_string, int n)

{

printf("\n\n\tLEAST RECENTLY USED ALGORITHM\n\n");

int ptr=0, pg\_fault=0,cnt,i;

int frames[frames\_req];

for(cnt=0;cnt<frames\_req;cnt++)

{

frames[cnt]=ref\_string[cnt];

++pg\_fault;

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

if(j>cnt)

printf("X ");

else

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

printf("\n");

}

while(cnt!=n)

{

i=find(ref\_string[cnt], frames,frames\_req);

if(i==0)//not available

{

++pg\_fault;

ptr=find\_not\_recently\_used(ref\_string,n,frames,frames\_req,cnt);

frames[ptr]=ref\_string[cnt];

printf("Page fault in inserting page %d - ",ref\_string[cnt]);

for(int j=0;j<frames\_req;j++)

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

printf("\n");

}

++cnt;

}

printf("\nTotal no of page faults = %d\n",pg\_fault);

}

void main()

{

int i,fcnt=-1,noff, frames, ref\_string[20], n=0,choice;

printf("\*\*\*\*\*\*\*\*\*\*\*\* PAGE REPLACEMENT \*\*\*\*\*\*\*\*\*\*\*\*\*");

do

{

printf("\n\n\tMENU:\n1.Read Input\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the number of free frames: ");

scanf("%d",&noff);

printf("Enter the number of frames required by the process: ");

scanf("%d",&frames);

if(noff<frames)

{

printf("No of frames required is greater than free frames available!");

exit(1);

}

printf("Enter the reference string:\n");

do

{

scanf("%d",&ref\_string[++fcnt]);

}while(ref\_string[fcnt]!=-1);

break;

case 2: FIFO(frames,ref\_string,fcnt); break;

case 3: optimal(frames,ref\_string,fcnt); break;

case 4: LRU(frames,ref\_string,fcnt); break;

case 5: break;

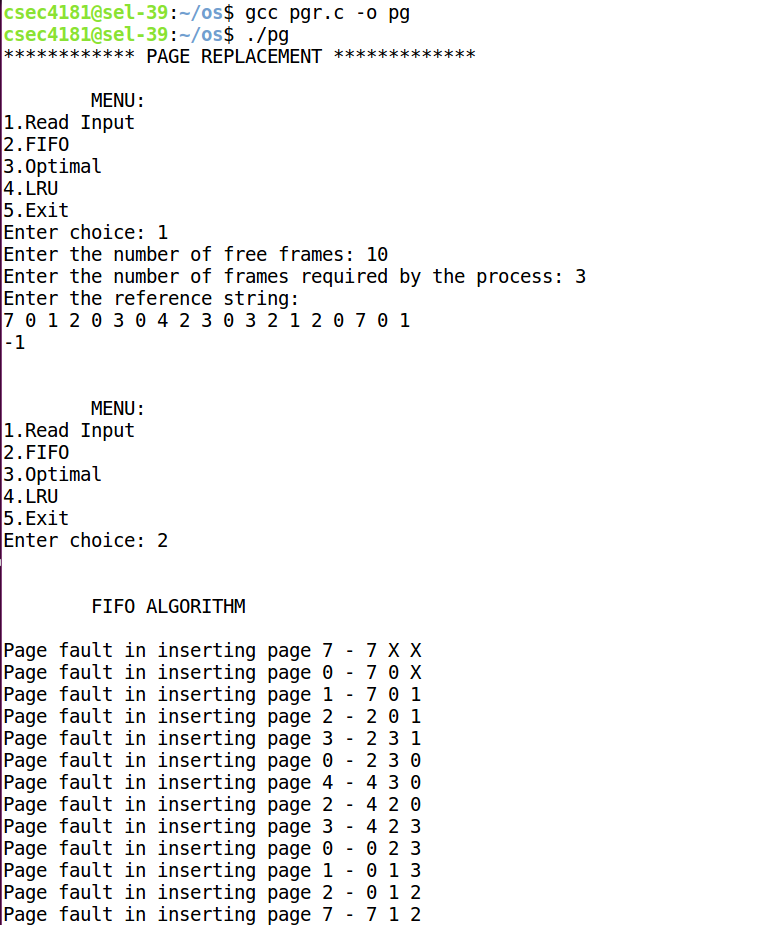
default: printf("\nInvalid choice ");

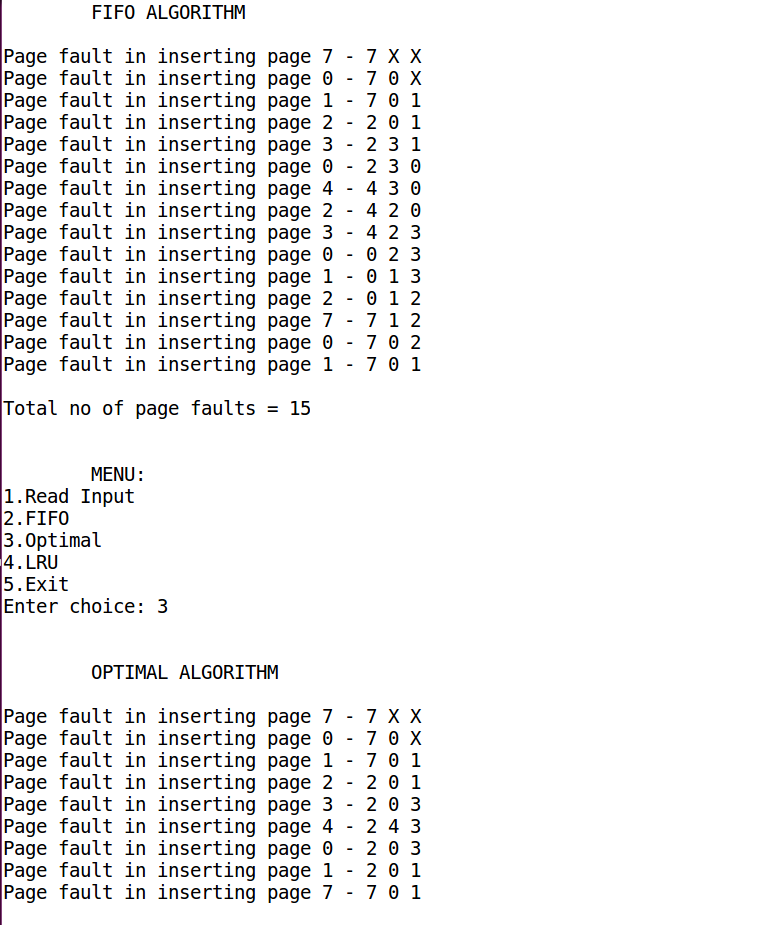
}

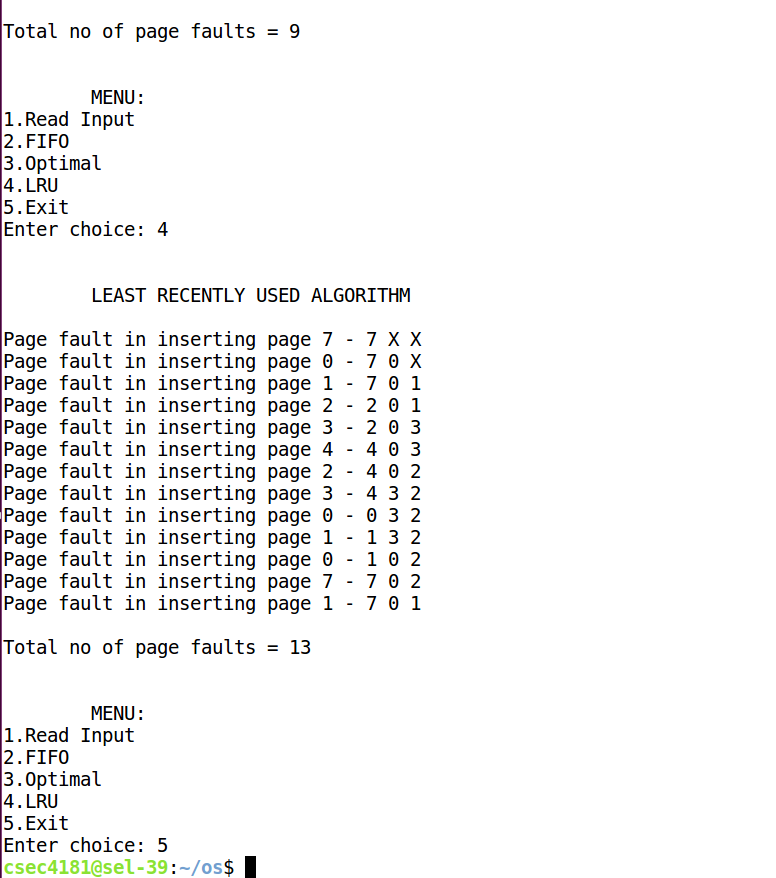
}while(choice!=5);

}

**OUTPUT:**

****

****

****