Experiment:7

7. Simulate the FIFO page replacement algorithm

Sourcecode:

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf(""No. of Page Faults: %d",count);
return 0;
}
```

Output:

C:\Users\admin\Desktop\Untitled5.exe

```
Enter the no of pages:
12
Enter page numbers
7 0 1 2 0 3 0 4 2 3 0 3
Enter the no of frames
3
refstring       page frames
7               7       -1      -1
0               7       0       -1
1               7       0       1
2               2       0       1
0
3               2       3       1
0               2       3       0
4               4       3       0
2               4       2       0
3               4       2       3
0               0       2       3
3
No of pages faluts are 10
--------------------------------
Process exited after 35.2 seconds with return value 0
Press any key to continue . . .
```

Experiment:8

8. Simulate the LRU page replacement algorithm

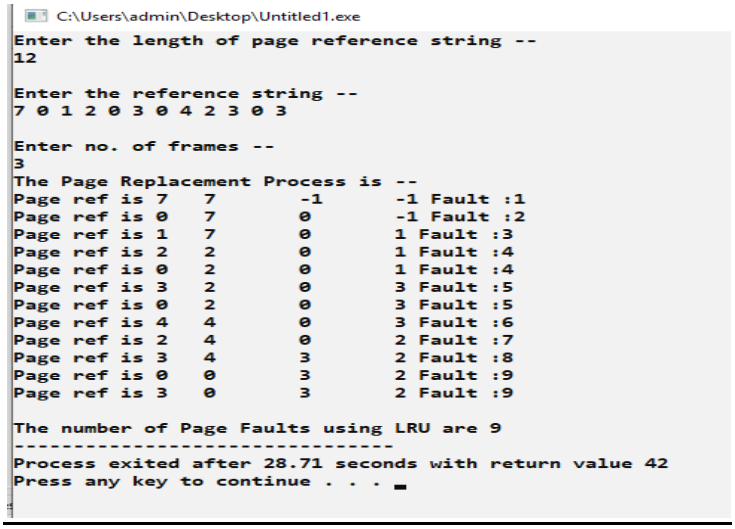
Sourcecode:

```
#include <stdio.h>
int main()
{
int i, j, k, f,max,p=10, pf=0, count[10], pageref[25], fp[10], n,flag[10];
printf("\n Enter the length of page reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&pageref[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
fp[i]=-1;count[i]=0;flag[i]=0;
}
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(count[k]==0)
{
fp[k]=pageref[i];
pf++;
count[k]=1;p=k;flag[k]=1;
break;
}
else if(fp[k]==pageref[i])     //required page found
{

count[k]=1;p=k;flag[k]=1;
break;
}
}
if(k==f) //LRU replacement
{
max=0;
for(j=0;j<f;j++)
```

```
{
if( count[j]>max)
{
max=count[j];
p=j;
}
}
fp[p]=pageref[i];
count[p]=1;
flag[p]=1;
pf++;
}
printf("Page ref is %d",pageref[i]);
for(j=0;j<f;j++)
{
if(j==p || count[j]==0)
continue;
count[j]=count[j]+1;
}
for(j=0;j<f;j++)
{
printf("\t%d ",fp[j]);
}
printf("Fault :%d",pf);
printf("\n");
}
printf("\n The number of Page Faults using LRU are %d",pf);
}
```

Output:



```
C:\Users\admin\Desktop\Untitled1.exe
Enter the length of page reference string --
12

Enter the reference string --
7 0 1 2 0 3 0 4 2 3 0 3

Enter no. of frames --
3
The Page Replacement Process is --
Page ref is 7   7       -1      -1 Fault :1
Page ref is 0   7       0       -1 Fault :2
Page ref is 1   7       0       1 Fault :3
Page ref is 2   2       0       1 Fault :4
Page ref is 0   2       0       1 Fault :4
Page ref is 3   2       0       3 Fault :5
Page ref is 0   2       0       3 Fault :5
Page ref is 4   4       0       3 Fault :6
Page ref is 2   4       0       2 Fault :7
Page ref is 3   4       3       2 Fault :8
Page ref is 0   0       3       2 Fault :9
Page ref is 3   0       3       2 Fault :9

The number of Page Faults using LRU are 9
--------------------------------
Process exited after 28.71 seconds with return value 42
Press any key to continue . . .
```

Experiment:

Q. Simulate the LFU page replacement algorithm

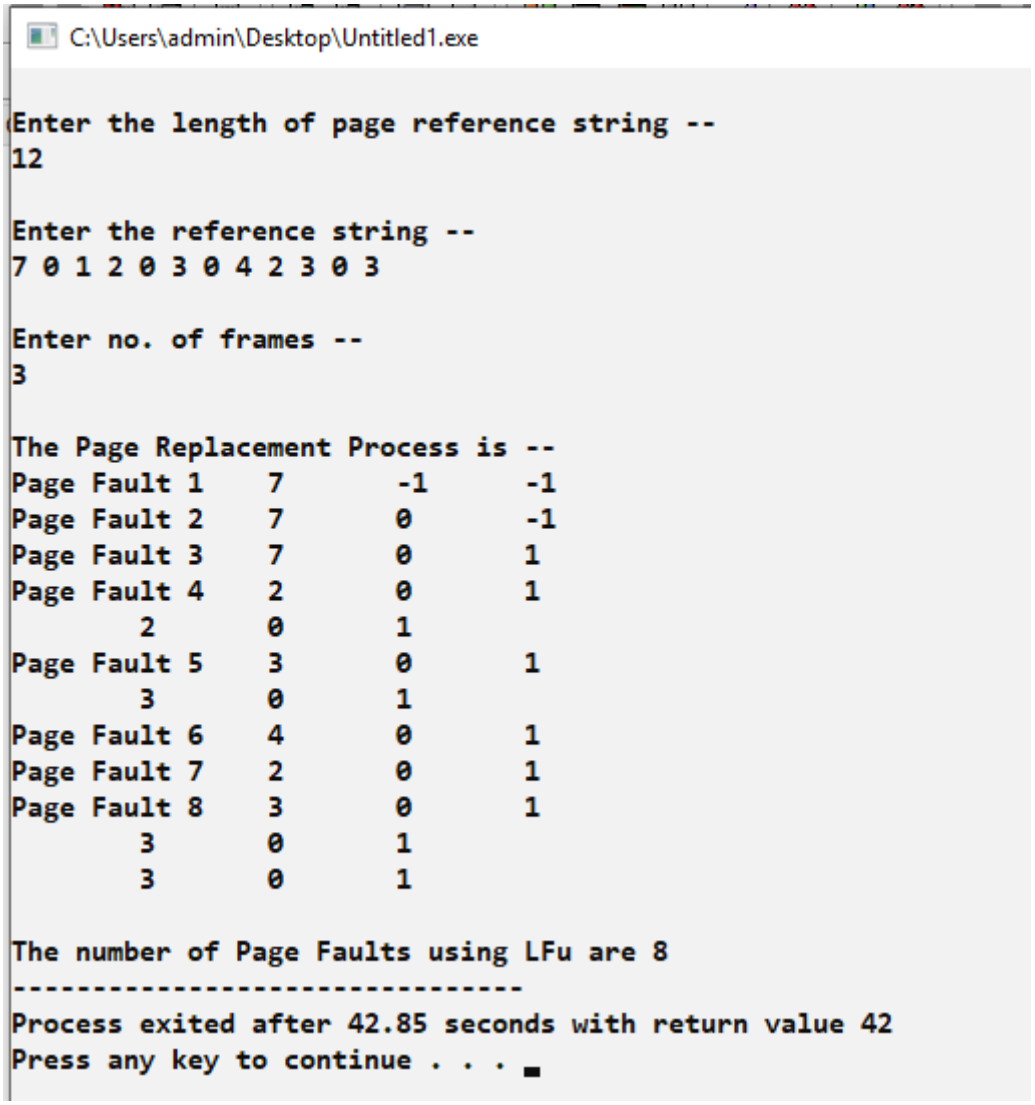
Sourcecode:

```
#include<stdio.h>
int main()
{
int i, j, k, f,min,p, pf=0, count[10], pageref[25], fp[10], n;
printf("\nEnter the length of page reference string -- ");
scanf("%d",&n);
printf("\nEnter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&pageref[i]);
printf("\nEnter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
                    fp[i]=-1;
                    count[i]=0;
}
printf("\nThe Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(fp[k]==pageref[i])
{
count[k]++;
break;
}
}
if(k==f)
{
min=100;
for(j=0;j<f;j++)
{
if( count[j]<min)
{
min=count[j];
p=j;
}
}
```

```
fp[p]=pageref[i];
count[p]=1;
pf++;
printf("Page Fault %d",pf);
}
for(j=0;j<f;j++)
printf("\t%d",fp[j]);
printf("\n");

}
printf("\nThe number of Page Faults using LFu are %d",pf);
 }
```

Output:



Experiment:6

6. Write a program to implementBankers Algorithm for Deadlock Avoidance.

Sourcecode:

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void show();
void cal();
int main()
{
        int i,j;
        printf("********** Banker's Algorithm ***********\n");
        int i,j;
        printf("Enter the no of Processes\t");
        scanf("%d",&n);
        printf("Enter the no of resources instances\t");
        scanf("%d",&r);
        printf("Enter the Max Matrix\n");
        for(i=0;i<n;i++)
        {
        for(j=0;j<r;j++)
        {
        scanf("%d",&max[i][j]);
        }
        }
        printf("Enter the Allocation Matrix\n");
        for(i=0;i<n;i++)
        {
        for(j=0;j<r;j++)
        {
        scanf("%d",&alloc[i][j]);
        }
        }
        printf("Enter the available Resources\n");
        for(j=0;j<r;j++)
        {
        scanf("%d",&avail[j]);
```

```
	}
	show();
	cal();
	getch();
	return 0;
}

void show()
{
	int i,j;
	printf("Process\t Allocation\t Max\t Available\t");
	for(i=0;i<n;i++)
	{
		printf("\nP%d\t ",i);
		for(j=0;j<r;j++)
		{
		printf("%d ",alloc[i][j]);
		}
		printf("\t");
		for(j=0;j<r;j++)
		{
		printf("%d ",max[i][j]);
		}
		printf("\t");
		if(i==0)
		{
		for(j=0;j<r;j++)
		printf("%d ",avail[j]);
		}
	}
}
void cal()
{
	int finish[100], temp, need[100][100],flag=1,k,c1=0;
	int safe[100];
	int i,j;
	for(i=0;i<n;i++)
	{
	finish[i]=0;
	}
	//find need matrix
```

```
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j] = max[i][j] -alloc[i][j];
}
}
printf("\n");
while(flag)
{
	flag=0;
	for(i=0;i<n;i++)
	{
	int c=0;
	for(j=0;j<r;j++)
	{
	if((finish[i]==0)&&(need[i][j]<=avail[j]))
	{
	c++;
	if(c==r)
	{
	for(k=0;k<r;k++)
	{
	avail[k]+=alloc[i][j];
	finish[i]=1;
	flag=1;
	}
	printf("P%d->",i);
	if(finish[i]==1)
	{
	i=n;
	}
	}
	}
	}
	}
}
for(i=0;i<n;i++)
{
	if(finish[i]==1)
	{
	c1++;
	}
	else
```

```
                {
                printf("P%d->",i);
                }
        }
        if(c1==n)
        {
        printf("\n The system is in safe state");
        }
        else
        {
        printf("\n Process are in dead lock");
        printf("\n System is in unsafe state");
        }
}
```

Output:

```
Select C:\Users\admin\Desktop\Untitled2.exe
********* Banker's Algorithm ***********
Enter the no of Processes       5
Enter the no of resources instances     3
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available Resources
3 3 2
Process Allocation      Max     Available
P0       0 1 0          7 5 3   3 3 2
P1       2 0 0          3 2 2
P2       3 0 2          9 0 2
P3       2 1 1          2 2 2
P4       0 0 2          4 3 3
P1->P3->P4->P2->P0->
 The system is in safe state
```