OPERATING SYSTEMS

EXPERIMENT : 1

NAMEOF THE EXPERIMENT: Simulate the following CPU Scheduling Algorithms

a) FCFS b) SJF c) Round Robin d) Priority

AIM: Using CPU scheduling algorithms find the min & max waiting time.

HARDWARE REQUIREMENTS: Intel based Desktop Pc RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

CPU SCHEDULING

Maximum CPU utilization obtained with multiprogramming

CPU–I/O Burst Cycle – Process execution consists of a *cycle* of

CPU execution and I/O wait

CPU burst distribution

a) First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P1 , P2 , P3

The Gantt Chart for the schedule is:

Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Burst times of processes
5. calculate the waiting time of each process
   wt[i+1]=bt[i]+wt[i]
6. calculate the turnaround time of each process
   tt[i+1]=tt[i]+bt[i+1]
7. Calculate the average waiting time and average turnaround time.
8. Display the values
9. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
clrscr();
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\n,at=%f\n",aw,at);
getch();
}
```

INPUT

```
Enter no of processes
3
enter bursttime
12
8
20
```

EXPECTED OUTPUT

```
bt wt tt
12 0 12
8 12 20
20 20 40
aw=10.666670
at=24.00000
```

VIVA QUESTIONS

1. What is First-Come-First-Served (FCFS) Scheduling?
2. Why CPU scheduling is required?
3. Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?

   1) Time-sharing 2) SPOOLing 3) Preemptive scheduling 4) Multiprogramming
4. CPU performance is measured through _____.

   1) Throughput 2) MHz 3) Flaps 4) None of the above
5. Which of the following is a criterion to evaluate a scheduling algorithm?

   1 CPU Utilization: Keep CPU utilization as high as possible.

   2 Throughput: number of processes completed per unit time.

   3 Waiting Time: Amount of time spent ready to run but not running.

   4 All of the above

EXPERIMENT : 1b)
NAMEOF THE EXPERIMENT: Simulate the following CPU Scheduling Algorithms

b) **SJF**
AIM: Using CPU scheduling algorithms find the min & max waiting time.

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

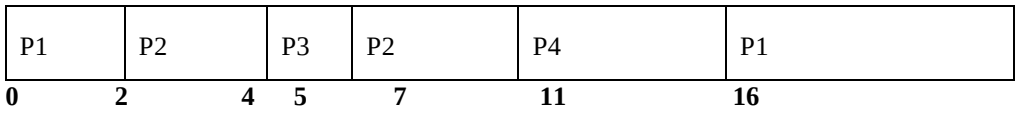
THEORY:
Example of Non Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	3.0	4

P1	**P3**	**P2**	**P4**
0 **7** **8**	**12**	**16**	

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	3.0	4



Average waiting time = (9 + 1 + 0 +2)/4 = 3

ALGORITHM

1. Start

2. Declare the array size

3. Read the number of processes to be inserted

4. Read the Burst times of processes

5. sort the Burst times in ascending order and process with shortest burst time is first executed.

6. calculate the waiting time of each process
   wt[i+1]=bt[i]+wt[i]
7. calculate the turnaround time of each process
   tt[i+1]=tt[i]+bt[i+1]

8. Calculate the average waiting time and average turnaround time.

9. Display the values

10. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
clrscr();
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
for(j=i;j<n;j++)
if(bt[i]>bt[j])
```

```
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
}
}
for(i=0;i<n;i++)
printf("%d",bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\t wt\t tt\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\n,at=%f\n",aw,at);
getch();
}
```

INPUT:

```
enter no of processes
3
enter burst time
12
8
20
```

OUTPUT:

```
bt wt tt
12 8 20
8  0 8
20 20 40
aw=9.33
at=22.64
```

VIVA QUESTIONS:

1) The optimum CPU scheduling algorithm is
   (A)FIFO (B)SJF with preemption.
   (C)SJF without preemption.(D)Round Robin.
2) In terms of average wait time the optimum scheduling algorithm is
   (A)FCFS (B)SJF (C)Priority (D)RR
3) What are the dis-advantages of SJF Scheduling Algorithm?
4) What are the advantages of SJF Scheduling Algorithm?
5) Define CPU Scheduling algorithm?

EXPERIMENT : 1c)
NAMEOF THE EXPERIMENT: Simulate the following CPU Scheduling Algorithms

c) Round Robin

AIM: Using CPU scheduling algorithms find the min & max waiting time.

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

Round Robin:

Example of RR with time quantum=3

Process	**Burst time**
aaa	4
Bbb	3
Ccc	2
Ddd	5
Eee	1

ALGORITHM
1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the burst times of the processes
5. Read the Time Quantum
6. if the burst time of a process is greater than time Quantum then subtract time quantum form the burst time
Else
Assign the burst time to time quantum.
7.calculate the average waiting time and turn around time of the processes.
8. Display the values

9. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
clrscr();
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
awt=(float)swt/n;
```

```
atat=(float)stat/n;
printf("Process_no Burst time Wait time Turn around time");
for(i=0;i<n;i++)
printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
printf("\nAvg wait time is %f Avg turn around time is %f",awt,atat);
getch();
}
```

Input:

```
Enter no of jobs
4
Enter burst time
5
12
8
20
```

Output:

```
Bt wt tt
5 0 5
12 5 13
8 13 25
20 25 45
aw=10.75000
at=22.000000
```

VIVA QUESTIONS:

1.Round Robin scheduling is used in

(A)Disk scheduling. (B)CPU scheduling

(C)I/O scheduling. (D)Multitasking

2. What are the dis-advantages of RR Scheduling Algoritm?

3.What are the advantages of RR Scheduling Algoritm?

4.Super computers typically employ _____.

1 Real time Operating system 2 Multiprocessors OS

3 desktop OS 4 None of the above

5. An optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of processes is _____.

1 FCFS scheduling algorithm 2 Round robin scheduling algorithm

3 Shortest job - first scheduling algorithm 4 None of the above

EXPERIMENT : 1d)
NAMEOF THE EXPERIMENT: Simulate the following CPU Scheduling Algorithms

d) Priority

AIM: Using CPU scheduling algorithms find the min & max waiting time.

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

In Priority Scheduling, each process is given a priority, and higher priority methods are executed first, while equal priorities are executed First Come First Served or Round Robin.

There are several ways that priorities can be assigned:

- Internal priorities are assigned by technical quantities such as memory usage, and file/IO operations.
- External priorities are assigned by politics, commerce, or user preference, such as importance and amount being paid for process access (the latter usually being for mainframes).

ALGORITHM

1. Start

2. Declare the array size

3. Read the number of processes to be inserted

4. Read the Priorities of processes

5. sort the priorities and Burst times in ascending order

5. calculate the waiting time of each process
   wt[i+1]=bt[i]+wt[i]
6. calculate the turnaround time of each process
   tt[i+1]=tt[i]+bt[i+1]

6. Calculate the average waiting time and average turnaround time.

7. Display the values

8. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
 {
  int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
  float aw,at;
  clrscr();
  printf("enter the number of processes:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
  printf("The process %d:\n",i+1);
   printf("Enter the burst time of processes:");
   scanf("%d",&bt[i]);
    printf("Enter the priority of processes %d:",i+1);
    scanf("%d",&prior[i]);
    pno[i]=i+1;
  }
    for(i=0;i<n;i++)
     {
      for(j=0;j<n;j++)
      {
       if(prior[i]<prior[j])
       {
              s=prior[i];
              prior[i]=prior[j];
              prior[j]=s;

               s=bt[i];
               bt[i]=bt[j];
               bt[j]=s;

                s=pno[i];
                pno[i]=pno[j];
                pno[j]=s;
               }
              }
      }
      for(i=0;i<n;i++)
      {
        wt[0]=0;
        tt[0]=bt[0];
             wt[i+1]=bt[i]+wt[i];
             tt[i+1]=tt[i]+bt[i+1];
             w1=w1+wt[i];
             t1=t1+tt[i];
             aw=w1/n;
             at=t1/n;
             }
```

```
        printf(" \n job \t bt \t wt \t tat \t prior\n");
          for(i=0;i<n;i++)
           printf("%d \t %d \t %d\t %d\t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
           printf("aw=%f \t at=%f \n",aw,at);
          getch();
}
```

Input:

Enter no of jobs
4
Enter bursttime
10
2
4
7
Enter priority values
4
2
1
3

Output:

Bt priority wt tt
4 1 0 4
2 2 4 6
7 3 6 13
10 4 13 23
aw=5.750000
at=12.500000

VIVA QUESTIONS:

1. Priority CPU scheduling would most likely be used in a _____ os.
2. Cpu allocated process to _____ priority.
3. calculate avg waiting time=
4. Maximum CPU utilization obtained with _____
5. Using _____algorithms find the min & max waiting time.

EXPERIMENT :2a)

NAME OF EXPERIMENT: Simulate file Allocation strategies:
a) Sequential

AIM: Simulate the file allocation strategies using file allocation methods

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

File Allocation Strategies:

The main problem is how to allocate disk space to the files so that disk space is utilized effectively band files can be accessed quickly. We have 3 space allocation method.

1. **Contiguous allocation (Sequential)**

   It requires each file to occupy a set of contiguous blocks on the hard disk where disk address define a linear ordering on the disk.

   Disadvantages:

   i. Difficult for finding space for a new file.

   ii. Internal and external fragmentation will be occurred.

ALGORITHM:

1. Start
2. Read the number of files
3. For each file, read the number of blocks required and the starting block of the file.
4. Allocate the blocks sequentially to the file from the starting block.
5. Display the file name, starting block , and the blocks occupied by the file.
6. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 clrscr();
 printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
```

```
            printf("Enter no. of blocks occupied by file%d",i+1);
            scanf("%d",&b[i]);
            printf("Enter the starting block of file%d",i+1);
            scanf("%d",&sb[i]);
            t[i]=sb[i];
            for(j=0;j<b[i];j++)
                        c[i][j]=sb[i]++;
            }
 printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)

            printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
      printf("blocks occupiedare:");
       for(i=0;i<n;i++)
       {  printf("fileno%d",i+1);
     for(j=0;j<b[i];j++)
     printf("\t%d",c[i][j]);
     printf("\n");
       }
 getch();
}
```

OUTPUT:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename        Start block     length
 1                  2               4
2                   5               10
```

VIVA QUESTIONS:

1.What file access pattern is particularly suited to chained file allocation on disk?

2. Define Sequential File allocation

4. Why we use file allocation strategies?

5.what are the advantages and dis-advantages of Sequential File allocation?

6. The average waiting time =_____.

EXPERIMENT :2 b)

NAME OF EXPERIMENT: Simulate file Allocation strategies:
b) Indexed

AIM: Simulate the file allocation strategies using file allocation methods

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

Indexed allocation

In linked allocation it is difficult to maintain FAT – so instead of that method indexed allocation method is used. Indexed allocation method solves all the problems in the linked allocation by bringing all the pointers together into one location called index block.

ALGORITHM:

1. Start
2. Read the number of files
3. Read the index block for each file.
4. For each file, read the number of blocks occupied and number of blocks of the file.
5. Link all the blocks of the file to the index block.
6. Display the file name, index block , and the blocks occupied by the file.
7. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,m[20],i,j,ib[20],b[20][20];
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {          printf("Enter index block :",i+1);
            scanf("%d",&ib[i]);
            printf("Enter blocks occupied by file%d:",i+1);
            scanf("%d",&m[i]);
            printf("enter blocks of file%d:",i+1);
            for(j=0;j<m[i];j++)
                        scanf("%d",&b[i][j]);
 } printf("\nFile\t index\tlength\n");
```

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

            printf("%d\t%d\t%d\n",i+1,ib[i],m[i]);
printf("blocks occupiedare:");
      for(i=0;i<n;i++)
      {  printf("fileno%d",i+1);
    for(j=0;j<m[i];j++)
    printf("\t%d--->%d\n",ib[i],b[i][j]);
    printf("\n");
              }
 getch();
 }
```

OUTPUT:

```
Enter no. of files:2
 Enter index block  3
Enter blocks occupied by file1: 4
enter blocks of file1:9
 4  6  7
Enter index block  5
Enter blocks occupied by file2:2
enter blocks of file2: 10   8
  File     index  length
 1       3       4
2       5       2
blocksoccupiedare:
file1
3--->9
3--->4
3--->6
3--->7
file2
5--->10
5--->8
```

VIVA QUESTIONS:

1. What file allocation strategy is most appropriate for random access files?
2.Define File?
3.Define Directory?
4. Why we use file allocation strategies?
5.what are the advantages and dis-advantages Indexed Allocation?

EXPERIMENT : 2 c)

NAME OF EXPERIMENT: Simulate file Allocation strategies:
c) Linked

AIM: Simulate the file allocation strategies using file allocation methods

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

Linked Allocation

Linked allocation of disk space overcomes all the problems of contiguous allocation. In linked allocation each file is a linked list of disk blocks where the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

Disadvantages : Space required to maintain pointers.

ALGORITHM:

1. Start
2. Read the number of files
3. For each file, read the file name, starting block, number of blocks and block numbers of the file.
4. Start from the starting block and link each block of the file to the next block in a linked list fashion.
5. Display the file name, starting block, size of the file , and the blocks occupied by the file.
6. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
 int i,j,n;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
```

```
printf("Enter file name:");
scanf("%s",&f[i].fname);
printf("Enter starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("Enter no.of blocks:");
scanf("%d",&f[i].size);
printf("Enter block numbers:");
for(j=1;j<=f[i].size;j++)
{
        scanf("%d",&f[i].block[j]);
}
}
printf("File\tstart\tsize\tblock\n");
for(i=0;i<n;i++)
{
        printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
        for(j=0;j<f[i].size;j++)
                printf("%d--->",f[i].block[j]);
        printf("%d",f[i].block[j]);
        printf("\n");
}
getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File    start   size    block
venkat  20      6       20--->4--->12--->15--->45--->32--->25
rajesh  12      5       12--->6--->5--->4--->3--->2
```

VIVA QUESTIONS:

1.What file access pattern is particularly suited to chained file allocation on disk?

2. What file allocation strategy is most appropriate for random access files?

3.Mention different file allocation strategies?

4. Why we use file allocation strategies?

5.what are the advantages and dis-advantages of each strategies?

6. The _____contains a pointer to the first and last blocks of the file.

EXPERIMENT : 3a

NAME OF EXPERIMENT: Simulate MFT .

AIM: Simulate Multiple Programming with fixed Number of Tasks (MFT**)**

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

Multiple Programming with fixed Number of Tasks (MFT) Algorithm

Background:

IBM in their Mainframe Operating System OS/MFT implements the MFT concept. *OS/MFT* uses Fixed partitioning concept to load programs into Main memory.

Fixed Partitioning:

- In fixed partitioning concept, RAM is divided into set of fixed partition of equal Size
- Programs having the Size Less than the partition size are loaded into Memory
- Programs Having Size more then the size of Partitions Size is rejected
- The program having the size less than the partition size will lead to internal Fragmentation.
- If all partitions are allocated and a new program is to be loaded, the program that lead to Maximum Internal Fragmentation can be replaced

ALGORITHM:

Step1: start
Step2: Declare variables.
Step3: Enter total memory size.
Step4: Read the no of partitions to be divided.
Step5: Allocate memory for os.
Step6:calculate available memory by subtracting the memory of os from total memory
Step7: calculate the size of each partition by dividing available memory with no of partitions.
Step8: Read the number of processes and the size of each process.
Step9: If size of process<= size of partition then allocate memory to that process.
Step10: Display the wastage of memory.
Step11: Stop .

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int ms,i,ps[20],n,size,p[20],s,intr=0;
        clrscr();
        printf("Enter size of memory:");
        scanf("%d",&ms);
        printf("Enter memory for OS:");
        scanf("%d",&s);
        ms-=s;
        printf("Enter no.of partitions to be divided:");
        scanf("%d",&n);
        size=ms/n;
        for(i=0;i<n;i++)
        {
                printf("Enter process size");
                scanf("%d ",&ps[i]);
                if(ps[i]<=size)
                {
                        intr=intr+size-ps[i];
                        printf("process%d is allocated\n",p[i]);
                }
                else
                        printf("process%d is blocked",p[i]);
        }
        printf("total fragmentation is %d",intr);
        getch();
}
```

OUTPUT:

Enter total memory size : 50

Enter memory for OS :10

Enter no.of partitions to be divided:4

Enter size of page : 10
Enter size of page : 9
Enter size of page : 9
Enter size of page : 8

Internal Fragmentation is = 4

VIVA QUESTIONS

1. The problem of fragmentation arises in _____.

1)Static storage allocation 2) Stack allocation storage

3 Stack allocation with dynamic binding 4 Heap allocation

2.Boundary registers _____.

1 Are available in temporary program variable storage

2 Are only necessary with fixed partitions

3 Track the beginning and ending the program

4 Track page boundaries

3.The principle of locality of reference justifies the use of _____.

1 Virtual Memory 2 Interrupts

3 Main memory 4 Cache memory

4. In memory management , a technique called as paging, physical memory is broken into fixed-sized blocks called _____.

1) Pages 2) Frames 3) Blocks 4) Segments

5.Demand paged memory allocation

1 allows the virtual address space to be independent of the physical memory

2 allows the virtual address space to be a multiple of the physical memory size

3 allows deadlock tobe detected in paging schemes

4 is present only in Windows NT

EXPERIMENT :3 b)

NAME OF EXPERIMENT: multiple Programming with Varible Number of Tasks (MVT) :

AIM: Simulate multiple Programming with Varible Number of Tasks (MVT)

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

multiple Programming with Varible Number of Tasks (MVT) Algorithm

Background:

IBM in their Mainframe Operating 'System OS/MVT implements the MVT concept. OSIMVT uses Dynamic Partition concept to load programs into Main memory.

Dynamic Partitioning:

- Initially RAM is portioned according to the of programs to be loaded into Memory till such time no other program can be loaded.
- The Left over Memory is called a hole which is too small too fit any process.
- When a new program is to be into Memory Look for the partition, Which Leads to least External fragmentation and load the Program.
- The space that is not used in a partition is called as External Fragmentation

ALGORITHM:

Step1: start
Step2: Declare variables.
Step3: Enter total memory size.
Step4: Read the no of processes
Step5: Allocate memory for os.
Step6: read the size of each process
Step7:calculate available memory by subtracting the memory of os from total memory
Step8: If available memory >= size of process then allocate memory to that process.
Step9: Display the wastage of memory.
Step10: Stop .

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i,m,n,tot,s[20];
        clrscr();
        printf("Enter total memory size:");
        scanf("%d",&tot);
        printf("Enter no. of processes:");
        scanf("%d",&n);
        printf("Enter memory for OS:");
        scanf("%d",&m);
        for(i=0;i<n;i++)
        {
                printf("Enter size of process %d:",i+1);
                scanf("%d",&s[i]);
        }
        tot=tot-m;
        for(i=0;i<n;i++)
        {
                if(tot>=s[i])
                {
                        printf("Allocate memory to  process %d\n",i+1);
                        tot=tot-s[i];
                }
                else
                        printf("process p%d is blocked\n",i+1);
        }
        printf("External Fragmentation is=%d",tot);
        getch();
}
```

OUTPUT:

```
Enter total memory size : 50
Enter no.of pages        : 4
Enter memory for OS    :10

Enter size of page  : 10
Enter size of page  : 9
Enter size of page  : 9
Enter size of page  : 10

External Fragmentation is = 2
```

VIVA QUESTIONS:

1. Explain about MFT?
2. Full form of MFT_____
3. Full form of MVT_____
4. differentiate MFT and MVT?
5. The _____ Memory is called a hole.
6. OSIMVT uses _____concept to load programs into Main memory.

7.OS/MFT uses _____ concept to load programs into Main memory.

EXPERIMENT :4

NAME OF EXPERIMENT: Simulate Banker's Algorithm for Deadlock Avoidance.

AIM: Simulate Banker's Algorithm for Deadlock Avoidance to find whether the system is in safe state or not.

HARDWARE REQUIREMENTS: Intel based Desktop Pc

RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

DEAD LOCK AVOIDANCE

To implement deadlock avoidance & Prevention by using Banker's Algorithm.

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
- Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,
  Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   - Finish[i] =False
   - Need<=Work

   If no such I exists go to step 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
   Available=Available-Request I;
   Allocation I =Allocation+Request I;
   Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct da {
int max[10],al[10],need[10],before[10],after[10];
}p[10];
void main() {
int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
clrscr();
printf("\n Enter the no of processes:");
scanf("%d",&n);
printf("\n Enter the no of resources:");
scanf("%d",&r);
for(i=0;i<n;i++) {
printf("process %d \n",i+1);
for(j=0;j<r;j++) {
printf("maximum value for resource %d:",j+1);
scanf("%d",&p[i].max[j]);
}
for(j=0;j<r;j++) {
printf("allocated from resource %d:",j+1);
scanf("%d",&p[i].al[j]);
```

```
p[i].need[j]=p[i].max[j]-p[i].al[j];
}
}
for(i=0;i<r;i++) {
printf("Enter total value of resource %d:",i+1);
scanf("%d",&tot[i]);
}
for(i=0;i<r;i++) {
for(j=0;j<n;j++)
temp=temp+p[j].al[i];
av[i]=tot[i]-temp;
temp=0;
}
printf("\n\t max allocated needed total avail");
for(i=0;i<n;i++) {
printf("\n P%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].max[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].al[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].need[j]);
printf("\t");
for(j=0;j<r;j++)
{
if(i==0)
printf("%d",tot[j]);
}
printf(" ");
for(j=0;j<r;j++) {
if(i==0)
printf("%d",av[j]);
}
}
printf("\n\n\t AVAIL BEFORE \t AVAIL AFTER");
for(l=0;l<n;l++)
{
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
if(p[i].need[j]>av[j])
cn++;
if(p[i].max[j]==0)
cz++;
}
if(cn==0 && cz!=r)
{
```

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

p[i].before[j]=av[j]-p[i].need[j];
p[i].after[j]=p[i].before[j]+p[i].max[j];
av[j]=p[i].after[j];
p[i].max[j]=0;
}
printf("\n p%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].before[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].after[j]);
cn=0;
cz=0;
c++;
break;
}
else {
cn=0;cz=0;
}
}
}
if(c==n)
printf("\n the above sequence is a safe sequence");
else
printf("\n deadlock occured");
getch();
}
```

OUTPUT:

```
//TEST CASE 1:

 ENTER THE NO. OF PROCESSES:4

 ENTER THE NO. OF RESOURCES:3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:0
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1:6
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
```

PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTER TOTAL VALUE OF RESOURCE 1:9
ENTER TOTAL VALUE OF RESOURCE 2:3
ENTER TOTAL VALUE OF RESOURCE 3:6

RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1 322	100	222	936	112
P2 613	511	102		
P3 314	211	103		
P4 422	002	420		

	AVAIL BEFORE	AVAIL AFTER
P 2	010	623
P 1	401	723
P 3	620	934
P 4	514	936

THE ABOVE SEQUENCE IS A SAFE SEQUENCE

VIVA QUESTIONS:

1. Differentiate deadlock avoidance and fragmentation

2.Tell me the real time example where this deadlock occurs?

3.How do we calculate the need for process?

4.What is the name of the algorithm to avoid deadlock?

5.Banker's algorithm for resource allocation deals with

(A)Deadlock prevention. (B)Deadlock avoidance.

(C)Deadlock recovery. (D)Mutual exclusion

EXPERIMENT :5

NAME OF EXPERIMENT: Simulate Algorithm for Deadlock prevention.

AIM: Simulate Algorithm for Deadlock prevention .

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

Deadlock Definition:
A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause (including itself).Waiting for an event could be:

- waiting for access to a critical section
- waiting for a resource Note that it is usually a non-preemptable (resource).

Conditions for Deadlock :

•Mutual exclusion: resources cannot be shared.
•Hold and wait:processes request resources incrementally, and hold on to What they've got.
•No preemption: resources cannot be forcibly taken from processes.
•Circular wait: circular chain of waiting, in which each process is waiting for a resource held by the next process in the chain.

Strategies for dealing with Deadlock :
•ignore the problem altogether
•detection and recovery
•avoidance by careful resource allocation
•prevention by structurally negating one of the four necessary conditions.

Deadlock Prevention :
Difference from avoidance is that here, the system itself is built in such a way that there are no deadlocks. Make sure atleast one of the 4 deadlock conditions is never satisfied. This may however be even more conservative than deadlock avoidance strategy.
Algorithm:

1.Start
2.Attacking Mutex condition : never grant exclusive access. but this may not be possible for several resources.
3..Attacking preemption: not something you want to do.
4.Attacking hold and wait condition : make a process hold at the most 1 resource at a time.make all the requests at the beginning. All or nothing policy. If you feel,retry. eg. 2-phase locking

5.Attacking circular wait: Order all the resources. Make sure that the requests are issued in the correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n. Resources can be requested only in increasing
order. ie. you cannot request a resource whose no is less than any you may be holding.
6.Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int max[10][10],alloc[10][10],need[10][10],avail[10],i,j,p,r,finish[10]={0},flag=0;
main( )
{
clrscr( );
printf("\n\nSIMULATION OF DEADLOCK PREVENTION");
printf("Enter no. of processes, resources");
scanf("%d%d",&p,&r);printf("Enter allocation matrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("enter max matrix");
for(i=0;i<p;i++) /*reading the maximum matrix and availale matrix*/
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf("enter available matrix");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
for(i=0;i<p;i++)
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
fun(); /*calling function*/
if(flag==0)
{i
f(finish[i]!=1)
{
printf("\n\n Failing :Mutual exclusion");
for(j=0;j<r;j++)
{ /*checking for mutual exclusion*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}fun();
printf("\n By allocating required resources to process %d dead lock is prevented ",i);
printf("\n\n lack of preemption");
for(j=0;j<r;j++)
{
if(avail[j]<need[i][j])
avail[j]=need[i][j];
alloc[i][j]=0;
}
fun( );
printf("\n\n daed lock is prevented by allocating needed resources");
```

```
printf(" \n \n failing:Hold and Wait condition ");
for(j=0;j<r;j++)
{ /*checking hold and wait condition*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}
fun( );
printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");
}
}
getch( );
}
fun()
{
while(1)
{
for(flag=0,i=0;i<p;i++)
{
if(finish[i]==0)
{
for(j=0;j<r;j++)
{
if(need[i][j]<=avail[j])
continue;
elsebreak;
}
if(j==r)
{
for(j=0;j<r;j++)
avail[j]+=alloc[i][j];
flag=1;
finish[i]=1;
}
}
}
if(flag==0)
break;
}
}
```

Output:

```
SIMULATION OF DEADLOCK PREVENTION
Enter no. of processes, resources 3, 2
enter allocation matrix 2 4 5
                        3 4 5
Enter max matrix4 3 4
                    5 6 1
Enter available matrix2
```

5

Failing : Mutual Exclusion
by allocating required resources to process dead is prevented
Lack of no preemption deadlock is prevented by allocating needed resources
Failing : Hold and Wait condition

VIVA QUESTIONS:

1. The Banker's algorithm is used for _____.

2._____ is the situation in which a process is waiting on another process,which is also waiting on another process ... which is waiting on the first process. None of the processes involved in this circular wait are making progress.

3.what is safe state?

4.What are the conditions that cause deadlock?

5.How do we calculate the need for process?

EXPERIMENT : 6a

NAME OF EXPERIMENT: 6) Simulate page replacement algorithms:

a) FIFO

AIM: Simulate FIFO page replacement algorithms.

HARDWARE REQUIREMENTS: Intel based Desktop Pc

RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

FIFO algorithm:

The simpler page replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replace, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue when a page is brought into memory; we insert it at the tail of the queue.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2
	0	0	0
		1	1

2	2	4	4	4	0
3	3	3	2	2	2
4	0	0	0	3	3

0	0
1	1
3	2

7	7	7
1	0	0
2	2	1

ALGORITHM:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
 clrscr();
 printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
 printf("\n Enter no.of frames....");
 scanf("%d",&nof);
 printf("Enter number of Pages.\n");
 scanf("%d",&nor);
 printf("\n Enter the Page No...");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
 printf("\nThe given Pages are:");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
 frm[i]=-1;
 printf("\n");
 for(i=0;i<nor;i++)
 {
   flag=0;
   printf("\n\t page no %d->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
     if(frm[j]==ref[i])
     {
       flag=1;
       break;
     }}
      if(flag==0)
   {
     pf++;
     victim++;
     victim=victim%nof;
     frm[victim]=ref[i];
     for(j=0;j<nof;j++)
     printf("%4d",frm[j]);
   }
 }
 printf("\n\n\t\t No.of pages faults...%d",pf);
 getch();
}
```

OUTPUT:

FIFO PAGE REPLACEMENT ALGORITHM

Enter no.of frames....4
Enter number of reference string..
6

Enter the reference string..
5 6 4 1 2 3

The given reference string:
..... 5 6 4 1 2 3

Reference np5-> 5 -1 -1 -1
Reference np6-> 5 6 -1 -1
Reference np4-> 5 6 4 -1
Reference np1-> 5 6 4 1
Reference np2-> 2 6 4 1
Reference np3-> 2 3 4 1

No.of pages faults...6

VIVA QUESTIONS:

1.Define FIFO?

2.Which of the following statement is not true?

a)Multiprogramming implies multitasking

b)Multi-user does not imply multiprocessing

c) Multitasking does not imply multiprocessing

d)Multithreading implies multi-user

3.Define page?

4.Define Frame?

5.Write advantages and dis-advantages of FIFO?

EXPERIMENT :6b)

NAME OF EXPERIMENT: 6) Simulate page replacement algorithms:
b) LRU

AIM: Simulate LRU page replacement algorithms

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB
SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

ALGORITHM :

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();

void main()
{
  clrscr();
  printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
  printf("\n Enter no.of Frames....");
  scanf("%d",&nof);

  printf(" Enter no.of reference string..");
  scanf("%d",&nor);

  printf("\n Enter reference string..");
  for(i=0;i<nor;i++)
  scanf("%d",&ref[i]);

  printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
```

```
 printf("\n\t The given reference string:");
 printf("\n……..");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
 {
  frm[i]=-1;
  lrucal[i]=0;
 }

 for(i=0;i<10;i++)
  recent[i]=0;
  printf("\n");
  for(i=0;i<nor;i++)
  {
    flag=0;
    printf("\n\t Reference NO %d->\t",ref[i]);
    for(j=0;j<nof;j++)
    {



      if(frm[j]==ref[i])
      {
            flag=1;
            break;
      }
    }

    if(flag==0)
    {
      count++;
      if(count<=nof)
      victim++;
      else
      victim=lruvictim();
      pf++;
      frm[victim]=ref[i];
      for(j=0;j<nof;j++)
      printf("%4d",frm[j]);
    }
    recent[ref[i]]=i;
  }
  printf("\n\n\t No.of page faults...%d",pf);
  getch();
}
int lruvictim()
{
 int i,j,temp1,temp2;
 for(i=0;i<nof;i++)
```

```
 {
  temp1=frm[i];
  lrucal[i]=recent[temp1];
 }
 temp2=lrucal[0];
 for(j=1;j<nof;j++)
 {
  if(temp2>lrucal[j])
  temp2=lrucal[j];
 }
 for(i=0;i<nof;i++)
 if(ref[temp2]==frm[i])
 return i;
 return 0;
}
```

OUTPUT:

```
                LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3
 Enter no.of reference string..6

 Enter reference string..
6 5 4 2 3 1


          LRU PAGE REPLACEMENT ALGORITHM
      The given reference string:
       ..... 6   5   4   2   3   1

      Reference NO 6->        6  -1  -1
      Reference NO 5->        6   5  -1
      Reference NO 4->        6   5   4
      Reference NO 2->        2   5   4
      Reference NO 3->        2   3   4
      Reference NO 1->        2   3   1

      No.of page faults...6
```

VIVA QUESTIONS:

1.In which of the following page replacement policies, Bolady's anomaly occurs?

(A)FIFO (B)LRU (C)LFU (D)SRU

2. Explain the difference between FIFO and LRU?

3. The operating system manages _____.

1 Memory 2 Processor

3 Disk and I/O devices 4 All of the above

4. A program at the time of executing is called _____.

1 Dynamic program 2 Static program

3 Binded Program 4 A Process

5.The principle of locality of reference justifies the use of _____.

1 Virtual Memory 2 Interrupts

3 Main memory 4 Cache memory

EXPERIMENT : 6c

NAME OF EXPERIMENT: 6) Simulate page replacement algorithms:
c)LFU

AIM: Simulate LFU page replacement algorithms .

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

ALGORITHM:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n.....");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string");
  scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
    scanf("%d",&ref[i]);
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n.....");
  printf("\nThe given string");
  printf("\n.....\n");
  for(i=0;i<nor;i++)
```

```
        printf("%4d",ref[i]);
    for(i=0;i<nof;i++)
    {
        frm[i]=-1;
        optcal[i]=0;
    }
    for(i=0;i<10;i++)
        recent[i]=0;
    printf("\n");
    for(i=0;i<nor;i++)
    {
      flag=0;
      printf("\n\tref no %d ->\t",ref[i]);
      for(j=0;j<nof;j++)
      {
            if(frm[j]==ref[i])
            {
              flag=1;
              break;
            }
      }
      if(flag==0)
      {
            count++;
            if(count<=nof)
                victim++;
            else
                victim=optvictim(i);
            pf++;
            frm[victim]=ref[i];
            for(j=0;j<nof;j++)
              printf("%4d",frm[j]);
      }
    }
    printf("\n Number of page faults: %d",pf);
    getch();
}
int optvictim(int index)
{
    int i,j,temp,notfound;
    for(i=0;i<nof;i++)
    {
      notfound=1;
      for(j=index;j<nor;j++)
            if(frm[i]==ref[j])
            {
                notfound=0;
                optcal[i]=j;
                break;
            }
```

```
    if(notfound==1)
            return i;
  }
  temp=optcal[0];
  for(i=1;i<nof;i++)
    if(temp<optcal[i])
            temp=optcal[i];
  for(i=0;i<nof;i++)
    if(frm[temp]==frm[i])
            return i;
 return 0;
}
```

OUTPUT:

```
 OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3
Enter no.of reference string..6

Enter reference string..
6 5 4 2 3 1

        OPTIMAL PAGE REPLACEMENT ALGORITHM
    The given reference string:
     ……. 6  5  4  2  3  1

    Reference NO 6->      6  -1  -1
    Reference NO 5->      6   5  -1
    Reference NO 4->      6   5   4
    Reference NO 2->      2   5   4
    Reference NO 3->      2   3   4
    Reference NO 1->      2   3   1

    No.of page faults...6
```

VIVA QUESTIONS:

1. What is the full form of LRU?
2. Explain when page replacement occurs?
3. Which is the best page replacement alg ?why?
4. FIFO scheduling is _____..
5. Explain various page replacement algorithms?
6. what do u mean by page fault?

EXPERIMENT :7

NAME OF EXPERIMENT: Simulate Paging Technique of memory management

AIM: Simulate Paging Technique of memory management.

HARDWARE REQUIREMENTS: Intel based Desktop Pc

RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

PAGING

Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available

Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8,192 bytes)

Divide logical memory into blocks of same size called pages

Keep track of all free frames

To run a program of size *n* pages, need to find *n* free frames and load program

ALGORITHM:

1. Start
2. Read the number of pages
3. Read the page size
4. Allocate the memory to the pages dynamically in non contiguous locations.
5. Display the pages and their addresses.
6. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{

   int np,ps,i;
   int *sa;
   clrscr();
   printf("enter no of pages");
```

```
scanf("%d",&np);
printf("enter the page size\n");
scanf("%d",&ps);
sa=(int*)malloc(2*np);
for(i=0;i<np;i++)
 {
   sa[i]=(int)malloc(ps);
   printf("page%d\t address%u\n",i+1,sa[i]);
 }
 getch();
}
```

OUTPUT:

```
enter no of pages: 5
 enter the page size:4
page1 address:1894
page2 address:1902
page3 address:1910
page4 address:1918
page5address:1926
```

VIVA QUESTIONS:

1. The mechanism that bring a page into memory only when it is needed is called _____

2.What are the advantages and dis-advantages of paging?

3.Define external fragmentation?

4._____ into blocks of same size called pages.

5._____ space of a process can be noncontiguous

6. What is page table?