Department of Computer Science and Application

LAB Manual

Operating System

II – BCA - A

PERIYAR MANIAMMAI
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University)
Established Under Sec. 3 of UGC Act, 1956 • NAAC Accredited
think • innovate • transform

Academic Year: 2023 - 2024

Table of contents

Simulate the FCFS - CPU Scheduling Algorithms

Aim:

To develop a program to simulate the FCFS in CPU scheduling algorithm

Algorithm

Step: 1 start the program
Step: 2 declare the array size
Step: 3 get the number of elements to be in inserted
Step: 4 select the process that first arrived in the ready queue
Step: 5 make the average waiting the length of the next process
Step: 6 start with first process from its selection as above and let other process to be in the queue
Step: 7 calculate the total no of burst time
Step: 8 display the values
Step: 9 stop the process

Coding

```
#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;    //waiting time for first process is 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;
}
```

Output:

```
Enter total number of processes(maximum 20):2

Enter Process Burst Time
P[1]:10
P[2]:20

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            10              0               10
P[2]            20              10              30

Average Waiting Time:5
Average Turnaround Time:20

...Program finished with exit code 0
Press ENTER to exit console.
```

EXPERIMENT -2

Simulate the SJF - CPU Scheduling Algorithms.

Aim:

To develop a program to simulate the SJF in CPU scheduling algorithm

Algorithm

Step: 1 start the program
Step: 2 declare the array size
Step: 3 get the number of elements to be in inserted
Step: 4 select the process that first arrived in the ready queue
Step: 5 make the average waiting the length of the next process
Step: 6 start with first process from its selection as above and let other process to be in the queue
Step: 7 calculate the total no of execution time and priority
Step: 8 display the values
Step: 9 stop the process

Coding

```
#include<stdio.h>
 void main()
{
   int bt[20], p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);
   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;          //contains process number
   }
   //sorting burst time in ascending order using selection sort
   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;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
     wt[0]=0;            //waiting time for first process will be zero
     //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
         total+=wt[i];
    }
     avg_wt=(float)total/n;      //average waiting time
    total=0;
     printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
     avg_tat=(float)total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
Enter number of process:4

Enter Burst Time:
p1:2
p2:4
p3:6
p4:5

Process     Burst Time          Waiting Time    Turnaround Time
p1                2                   0                   2
p2                4                   2                   6
p4                5                   6                   11
p3                6                   11                  17

Average Waiting Time=4.750000
Average Turnaround Time=9.000000
```

Experiment -3

Simulate the Priority - CPU Scheduling Algorithms.

Aim:

To develop a program to simulate the priority in CPU scheduling algorithm

Algorithm

Step: 1 start the program
Step: 2 declare the array size
Step: 3 get the number of elements to be in inserted
Step: 4 select the process that first arrived in the ready queue
Step: 5 make the average waiting the length of the next process
Step: 6 start with first process from its selection as above and let other process to be in the queue
Step: 7 calculate the total no of execution time and waiting time
Step: 8 display the values
Step: 9 stop the process

Coding

```
#include<stdio.h>
 int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);
    printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;          //contains process number
   }
    //sorting burst time, priority and process number in ascending order using selection sort
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }
       temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;
      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;
      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;     //waiting time for first process is zero
```

```
    //calculate waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
       total+=wt[i];
   }
    avg_wt=total/n;      //average waiting time
   total=0;
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
  {
      tat[i]=bt[i]+wt[i];     //calculate turnaround time
      total+=tat[i];
      printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
    avg_tat=total/n;     //average turnaround time
   printf("\n\nAverage Waiting Time=%d",avg_wt);
   printf("\nAverage Turnaround Time=%d\n",avg_tat);
         return 0;
}
```

Output:

```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:2
Priority:4

P[2]
Burst Time:4
Priority:6

P[3]
Burst Time:6
Priority:8

P[4]
Burst Time:8
Priority:10

Process     Burst Time          Waiting Time    Turnaround Time
P[1]              2                 0                 2
P[2]              4                 2                 6
P[3]              6                 6                 12
P[4]              8                 12                20

Average Waiting Time=5
Average Turnaround Time=10
```

Experiment - 4

Simulate the Round Robin –b CPU Scheduling Algorithms

Aim:

To develop a program to simulate the Round Robin in CPU scheduling algorithm

Algorithm

Step: 1 start the program
Step: 2 declare the array size
Step: 3 get the number of elements to be in inserted
Step: 4 get the value
Step: 5 set the time sharing system with preemption
Step: 6 define quantum is defined from 10 to 100 ms
Step: 7 declare the queue as a circular
Step: 8 make the CPU scheduler goes around the ready queue allocating CPU to each process for the time interval specified
Step: 9 make the CPU scheduler picks the first process and sets time to interrupt after quantum expired dispatched the process
Step: 10 if the process has burst less than the time quantum than the process releases the CPU

Coding

```
#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;
}
```

Output :

```
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0 9
Enter Arrival Time and Burst Time for Process Process Number 2 :1 5
Enter Arrival Time and Burst Time for Process Process Number 3 :2 7
Enter Arrival Time and Burst Time for Process Process Number 4 :3 8
Enter Time Quantum:     5


Process |Turnaround Time|Waiting Time

P[2]    |        9       |       4
P[1]    |        24      |       15
P[3]    |        24      |       17
P[4]    |        26      |       18

Average Waiting Time= 13.500000
Avg Turnaround Time = 20.750000
```

Experiment -5.1

Simulate MVT

Aim:

To develop a program to simulate MVT

Algorithm

Step: 1 start the program
Step: 2 declare the variable
Step: 3 enter the total memory size
Step: 4 read the number of process
Step: 5 allocate the memory for OS
Step: 6 read the size of each process
Step: 7 calculate available memory by subtracting the memory from OS from the total memory
Step: 8 if available memory >= size of process the allocate the memory to that process
Step: 9 display the wastage of memory
Step: 10 stop the program

Coding

```
#include<stdio.h>
int main()
{
 int tm,om,n,i;
 printf("Enter total memory size,memory for OS and no of processes:\n");
 scanf("%d%d%d",&tm,&om,&n);
 int process[n];
 for (i = 0; i < n; ++i)
 {
  printf("Enter process %d size :\n",i);
  scanf("%d",&process[i]);
 }
 tm = tm - om;
 for (i = 0; i < n; ++i)
 {
  if(tm >= process[i]){
   printf("Allocated memory to process :%d\n",i+1);
   tm = tm - process[i];
  }else{
   printf("Process %d is blocked\n",i+1);
  }
 }
 printf("External fragmentation is %d.\n",tm);
 return 0;
}
```

Output:

```
Enter total memory size,memory for OS and # of processes:
50
10
5
Enter process 0 size :
5
Enter process 1 size :
5
Enter process 2 size :
5
Enter process 3 size :
5
Enter process 4 size :
5
Allocated memory to process :1
Allocated memory to process :2
Allocated memory to process :3
Allocated memory to process :4
Allocated memory to process :5
External fragmentation is 15.
```

Experiment -5.2

Simulate MFT

Aim:

To develop a program to simulate MFT

Algorithm

Step: 1 start the program
Step: 2 declare the variable
Step: 3 enter the total memory size
Step: 4 read the number of partitions to be divided
Step: 5 allocate the memory for OS
Step: 6 calculate available memory by subtracting the memory from OS from the total memory
Step: 7 calculate the size of each partition by dividing available memory with no. of partitions
Step: 8 read the number of processes and the size of each process
Step: 9 if available memory <= size of process the allocate the memory to that process
Step: 10 display the wastage of memory
Step: 11 stop the program

Coding

```
#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");
        }
}
```

Output

```
Enter no. of blocks: 4

Enter size of each block: 10
9
9
8

Enter no. of processes: 4

Enter size of each process: 12
11
10
9

Block no.       size            process no.             size
1               10              3                       10
2               9               4                       9
3               9               Not allocated
4               8               Not allocated
```

Simulate Bankers algorithm for Deadlock Avoidance

Aim:
To develop a program to simulate the bankers algorithm for deadlock avoidance
Algorithm
Step: 1 start the program
Step: 2 get the values of resources and process
Step: 3 get the avail value
Step: 4 after allocation find the need value
Step: 5 check whether it's possible to allocate
Step: 6 if it is possible than the system is in safe state
Step: 7 else system is not in safety state
Step: 8 if the new request comes then check that the system is in safety
Step: 9 or not if we allow the request
Step: 10 stop the program
Coding

```
#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("\n Deadlock will occur\n");
                        break;
        }
                else
                {
                        printf("\nThe processes are in unsafe state");
                        printf("\nAvailable vector:");

                        for (i = 0; i < resources; i++)
                        {
                        printf("\t%d", available[i]);
                        }

                    printf("\n");
        }
        }
        return 0;
}
```

Output:

```
Enter number of processes: 2
Enter number of resources: 2
Enter Claim Vector:
2
3
Enter Allocated Resource Table:
4
3
2
4
5
Enter Maximum Claim Table:
6
4
3
2
1
The Claim Vector is:  2        3
The Allocated Resource Table:
        3       2
        4       5

The Maximum Claim Table:
        4       3
        2       1
Allocated resources:  7        7
Available resources:  -5       -4

Deadlock will occur
```

Experiment – 7

Simulate Bankers algorithm for Deadlock prevention

Aim:
To develop a program to simulate the bankers algorithm for deadlock prevention
Algorithm
Step: 1 start the program
Step: 2 attacking mutex condition: never grant exclusive access but this may not be possible for several resources
Step: 3 attacking preemption: not something you want to do
Step: 4 attacking hold and wait condition: make a process hold at the most 2 resources at a time make the entire request at beginning all or nothing policy. If you feel retry eg: 2-phase locking 3 4
Step: 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 resources graph. Resources numbered 1……n resources can be requested only in increasing order i.e. you cannot request a resource where no is less than any you may be holding
Step: stop the program

Coding

```
#include<stdio.h>
void main()
{
int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
int pno,rno,i,j,prc,count,t,total;
count=0;
printf("\n Enter number of process:");
scanf("%d",&pno);
printf("\n Enter number of resources:");
scanf("%d",&rno);
for(i=1;i<=pno;i++)
{
flag[i]=0;
}
printf("\n Enter total numbers of each resources:");
for(i=1;i<= rno;i++)
scanf("%d",&tres[i]);
printf("\n Enter Max resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&max[i][j]);
}
printf("\n Enter allocated resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&allocated[i][j]);
}
printf("\n available resources:\n");
for(j=1;j<= rno;j++)
{
avail[j]=0;
total=0;
for(i=1;i<= pno;i++)
{
total+=allocated[i][j];
}
avail[j]=tres[j]-total;
work[j]=avail[j];
printf(" %d \t",work[j]);
}
do
{
for(i=1;i<= pno;i++)
{
for(j=1;j<= rno;j++)
{
```

```
need[i][j]=max[i][j]-allocated[i][j];
}
}
printf("\n Allocated matrix Max need");
for(i=1;i<= pno;i++)
{
printf("\n");
for(j=1;j<= rno;j++)
{
printf("%4d",allocated[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",max[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",need[i][j]);
}
}
prc=0;
for(i=1;i<= pno;i++)
{
if(flag[i]==0)
{
prc=i;
for(j=1;j<= rno;j++)
{
if(work[j]< need[i][j])
{
prc=0;
break;
}
}
}
if(prc!=0)
break;
}
if(prc!=0)
{
printf("\n Process %d completed",i);
count++;
printf("\n Available matrix:");
for(j=1;j<= rno;j++)
{
work[j]+=allocated[prc][j];
allocated[prc][j]=0;
max[prc][j]=0;
flag[prc]=1;
```

```
printf(" %d",work[j]);
}
}
}while(count!=pno&&prc!=0);
if(count==pno)
printf("\nThe system is in a safe state!!");
else
printf("\nThe system is in an unsafe state!!");
getch();
}
```

Output:

```
Enter number of process:3

Enter number of resources:1 2

Enter total numbers of each resources:10 7
Enter Max resources for each process:
for process 1:9 2 3
for process 2:1 2
for process 3:1 3
Enter allocated resources for each process:
for process 1:2 3
for process 2:1 2
for process 3:1 1
available resources:
6       1
Allocated matrix Max need
  2   3|   2   3|   0   0
  1   2|   1   2|   0   0
  1   1|   1   3|   0   2
Process 1 completed
Available matrix: 8 4
Allocated matrix Max need
  0   0|   0   0|   0   0
  1   2|   1   2|   0   0
  1   1|   1   3|   0   2
Process 2 completed
Available matrix: 9 6
Allocated matrix Max need
  0 0| 0 0| 0 0
  0   0|   0   0|   0   0
  1   1|   1   3|   0   2
Process 3 completed
Available matrix: 10 7
The system is in a safe state!!
```

Simulate FIFO Page Replacement Algorithms

Aim:

To develop a program to simulate the FIFO page replacement algorithm

Algorithm

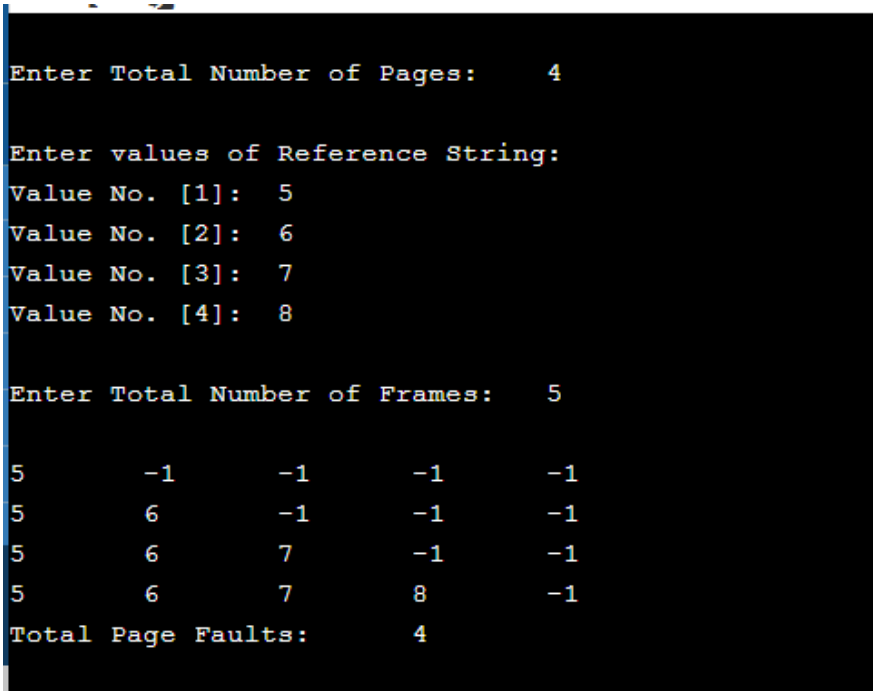
Step: 1 start the program
Step: 2 read the number of frames
Step: 3 read the number of pages
Step: 4 read the page number
Step: 5 initialize the values in frames to -1
Step: 6 allocate the pages into frames in first in first out order
Step: 7 display the number of pages faults
Step: 8 stop the program

Coding

```
#include<stdio.h>
int main()
{
    int reference_string[10], page_faults = 0, m, n, s, pages, frames;
    printf("\nEnter Total Number of Pages:\t");
    scanf("%d", &pages);
    printf("\nEnter values of Reference String:\n");
    for(m = 0; m < pages; m++)
    {
        printf("Value No. [%d]:\t", m + 1);
        scanf("%d", &reference_string[m]);
    }
    printf("\nEnter 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(reference_string[m] == temp[n])
            {
                s++;
                page_faults--;
            }
        }
        page_faults++;
        if((page_faults <= frames) && (s == 0))
        {
            temp[m] = reference_string[m];
```

```
        }
        else if(s == 0)
        {
            temp[(page_faults - 1) % frames] = reference_string[m];
        }
        printf("\n");
        for(n = 0; n < frames; n++)
        {
            printf("%d\t", temp[n]);
        }
    }
    printf("\nTotal Page Faults:\t%d\n", page_faults);
    return 0;
}
```

Output:



Experiment – 9

Simulate LRU Page Replacement Algorithms

Aim:

To develop a program to simulate the LRU page replacement algorithm

Algorithm

Step: 1 start the program
Step: 2 declare the size
Step: 3 get the number of pages to be inserted
Step: 4 get the value
Step: 5 declare the counter and stack
Step: 6 select the least recently used page by counter value
Step: 7 stack them according the selection
Step: 8 display the values

Step: 9 stop the program

Coding

```
#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;
}
```

Output:

```
Enter number of frames: 4
Enter number of pages: 6
Enter reference string: 5
6
4
1
2
3

5       -1      -1      -1
5       6       -1      -1
5       6       4       -1
5       6       4       1
2       6       4       1
2       3       4       1

Total Page Faults = 6
```

Simulate Optimal Page Replacement Algorithms

Aim:

To develop a program to simulate the optimal page replacement algorithm

Algorithm

Step: 1 start the program
Step: 2 read the number of frames
Step: 3 read the number of pages
Step: 4 read the page number
Step: 5 initialize the values in frames to -1
Step: 6 allocate the pages into frames by selecting the page that will not be used for the longest period of time
Step: 7 display the number of page faults
Step: 8 stop the program

Coding

```
#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;
   }
```

Output:

```
Enter number of frames: 3
Enter number of pages: 6
Enter page reference string: 6
5
4
3
2
1

6       -1      -1
6       5       -1
6       5       4
3       5       4
2       5       4
1       5       4

Total Page Faults = 6
```

Experiment – 11

Simulate Paging Technique of Memory Management

Aim:

To develop a program to simulate paging technique of memory management

Algorithm

Step: 1 start the program
Step: 2 read the number of pages
Step: 3 read the page size
Step: 4 allocate the memory to the pages dynamically in non – contiguous location
Step: 5 display the pages and their addresses
Step: 6 stop the program

Coding

```
#include<stdio.h>
#include<conio.h>
int main()
{
int np,ps,i;
int *sa;
printf("\n enter how many pages:");
scanf("%d",&np);
printf("\n enter the page size:");
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: