HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT
Hamidpur, Delhi-110036

(An iso 9001:2008 certified, AICTE approved & GGSIP university affiliated institute)

OPERATING SYSTEMS LAB

Submitted To:	Submitted By:
Mr Rakesh	Nitin Tyagi
Assistant Professor	(41113303118)
(IT Department)	(Btech(IT))

HMR INSTITUTE OF TECHNOLOGY AND MANAGEMENT

HAMIDPUR, DELHI 110036

Affiliated to

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY Sector - 16C Dwarka, Delhi - 110075, India 2018-2022

HMR INSTITUTE OF TECHNOLOGY & MANAGEMENT
Hamidpur, Delhi-110036

(An iso 9001:2008 certified, AICTE approved & GGSIP university affiliated institute)

INDEX

S.No.	Experiment Description	Page No.	Experiment Date	Submission Date	Remarks
1	Write a program to implement CPU scheduling for first come first serve.	**1**	**26-03-2021**	**19-04-2021**	
2	Write a program to implement CPU scheduling for shortest job first.	**4**	**26-03-2021**	**19-04-2021**	
3	Write a program to perform priority scheduling.	**6**	**19-04-2021**	**21-05-2021**	
4	Write a program to implement CPU scheduling for Round Robin.	**10**	**19-04-2021**	**21-05-2021**	
5	Write a program for page replacement policy using a) LRU b) FIFO c) Optimal	**15**	**21-05-2021**	**28-05-2021**	
6	Write a program to implement first fit, best fit and worst fit algorithm for memory management.	**24**	**28-05-2021**	**04-06-2021**	
7	Write a program to implement reader/writer problem using semaphore.	**31**	**04-06-2021**	**11-06-2021**	
8	Write a program to implement Banker's algorithm for deadlock avoidance.	**34**	**04-06-2021**	**11-06-2021**	

Experiment - 1

Aim :-

Write a program to implement CPU scheduling for first come first serve.

PROGRAM :-

```
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

bool compareArrival(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compareID(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
```

```
    int total_idle_time = 0;
    float throughput;

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;

    for(int i = 0; i < n; i++) {
        cout<<"Enter arrival time of process "<<i+1<<": ";
        cin>>p[i].arrival_time;
        cout<<"Enter burst time of process "<<i+1<<": ";
        cin>>p[i].burst_time;
        p[i].pid = i+1;
        cout<<endl;
    }

    sort(p,p+n,compareArrival);

    for(int i = 0; i < n; i++) {
        p[i].start_time = (i == 0)?p[i].arrival_time:max(p[i-1].completion_time,p[i].arrival_time);
        p[i].completion_time = p[i].start_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        p[i].response_time = p[i].start_time - p[i].arrival_time;

        total_turnaround_time += p[i].turnaround_time;
        total_waiting_time += p[i].waiting_time;
        total_response_time += p[i].response_time;
        total_idle_time += (i == 0)?(p[i].arrival_time):(p[i].start_time - p[i-1].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((p[n-1].completion_time - total_idle_time) / (float) p[n-
1].completion_time)*100;
    throughput = float(n) / (p[n-1].completion_time - p[0].arrival_time);

    sort(p,p+n,compareID);

    cout<<endl;
    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

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

```
cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"<<p[i]
.completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].response_t
ime<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
    cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;


}
```

OUTPUT:-

```
Enter the number of processes: 4
Enter arrival time of process 1: 0
Enter burst time of process 1: 2

Enter arrival time of process 2: 1
Enter burst time of process 2: 2

Enter arrival time of process 3: 5
Enter burst time of process 3: 3

Enter arrival time of process 4: 6
Enter burst time of process 4: 4


#P      AT      BT      ST      CT      TAT     WT      RT

1       0       2       0       2       2       0       0

2       1       2       2       4       3       1       1

3       5       3       5       8       3       0       0

4       6       4       8       12      6       2       2
```

```
Average Turnaround Time = 3.50
Average Waiting Time = 0.75
Average Response Time = 0.75
CPU Utilization = 91.67%
Throughput = 0.33 process/unit time


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

Experiment - 2

Aim :-

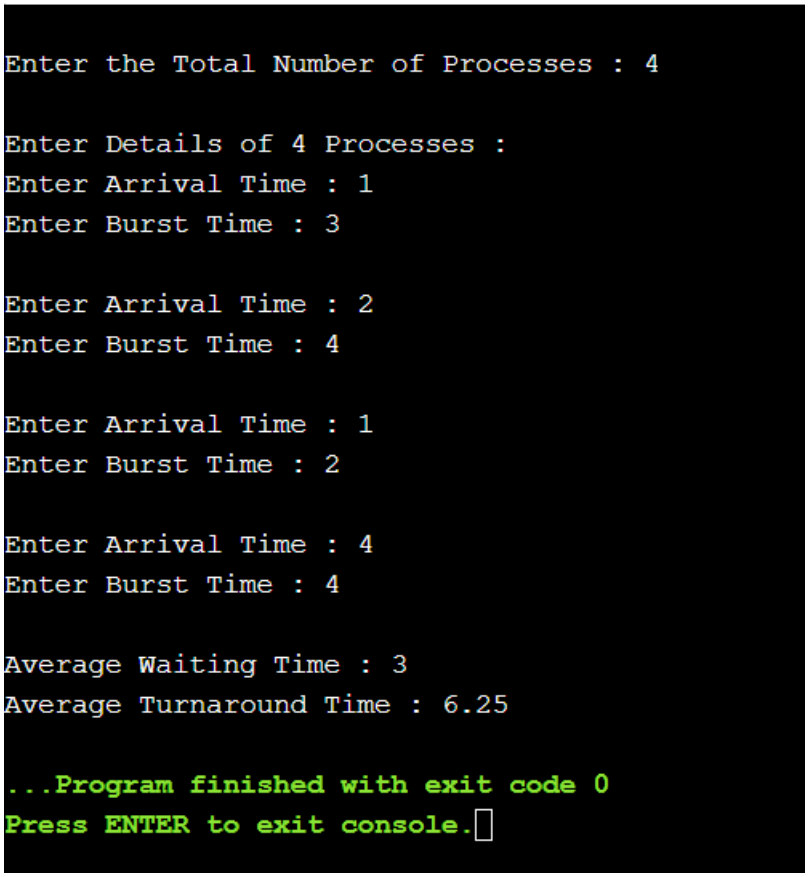
Write a program to implement CPU scheduling for shortest job first.

PROGRAM :-

```
#include<iostream>
using namespace std;
int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    cout<<endl<<"Enter the Total Number of Processes : ";
    cin>>limit;
    cout<<endl<<"Enter Details of "<<limit<<" Processes : ";
    for(i=0;i<limit;i++)
    {
        cout<<endl<<"Enter Arrival Time : ";
        cin>>arrival_time[i];
        cout<<"Enter Burst Time : ";
        cin>>burst_time[i];
        temp[i]=burst_time[i];
    }
    burst_time[9]=9999;
    for(time=0;count!=limit;time++)
    {
        smallest=9;
        for(i=0;i<limit;i++)
        {
            if(arrival_time[i]<=time&&burst_time[i]<burst_time[smallest]&&burst_time[i]>0)
            {
                smallest=i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest]==0)
        {
            count++;
            end=time+1;
            wait_time=wait_time+end-arrival_time[smallest]-temp[smallest];
            turnaround_time=turnaround_time+end-arrival_time[smallest];
```

```
        }
    }
    average_waiting_time=wait_time/limit;
    average_turnaround_time=turnaround_time/limit;
    cout<<"\nAverage Waiting Time : "<<average_waiting_time;
    cout<<"\nAverage Turnaround Time : "<<average_turnaround_time;
    return 0;
}
```

OUTPUT:-



Experiment - 3

Aim :-

Write a program to perform priority scheduling.

PROGRAM :-

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int is_completed[100];
    memset(is_completed,0,sizeof(is_completed));

    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;
```

```
for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    cout<<"Enter priority of the process "<<i+1<<": ";
    cin>>p[i].priority;
    p[i].pid = i+1;
    burst_remaining[i] = p[i].burst_time;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mx = -1;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].priority > mx) {
                mx = p[i].priority;
                idx = i;
            }
            if(p[i].priority == mx) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mx = p[i].priority;
                    idx = i;
                }
            }
        }
    }

    if(idx != -1) {
        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time;
            total_idle_time += p[idx].start_time - prev;
        }
        burst_remaining[idx] -= 1;
        current_time++;
        prev = current_time;

        if(burst_remaining[idx] == 0) {
            p[idx].completion_time = current_time;
```

```
            p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;

            is_completed[idx] = 1;
            completed++;
          }
        }
        else {
            current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;

    cout<<endl<<endl;

    cout<<"PID\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"\n"<<endl;

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

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].priority<<"\t"<<p[i].st
art_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"
\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
}
```

OUTPUT:-

```
Enter priority of the process 2: 20

Enter arrival time of process 3: 2
Enter burst time of process 3: 2
Enter priority of the process 3: 30

Enter arrival time of process 4: 4
Enter burst time of process 4: 1
Enter priority of the process 4: 40



PID     AT      BT      PRI     ST      CT      TAT     WT

1       0       5       10      0       12      12      7

2       1       4       20      1       8       7       3

3       2       2       30      2       4       2       0

4       4       1       40      4       5       1       0

Average Turnaround Time = 5.50
Average Waiting Time = 2.50


...Program finished with exit code 0
```

Experiment - 4

Aim :-

Write a program to implement CPU scheduling for Round Robin.

PROGRAM :-

```
#include<iostream>
#include<cstdlib>
#include<queue>
#include<cstdio>
using namespace std;

/* C++ Program to Round Robin*/

typedef struct process
{
  int id,at,bt,st,ft,pr;
  float wt,tat;
}process;

 process p[10],p1[10],temp;
 queue<int> q1;

int accept(int ch);
void turnwait(int n);
void display(int n);
void ganttrr(int n);

int main()
{
  int i,n,ts,ch,j,x;

  p[0].tat=0;
  p[0].wt=0;

  n=accept(ch);
  ganttrr(n);
  turnwait(n);

 display(n);

 return 0;
}
```

```
int accept(int ch)
{
  int i,n;

  printf("Enter the Total Number of Process: ");
  scanf("%d",&n);

 if(n==0)
  {
    printf("Invalid");
    exit(1);
  }

  cout<<endl;

  for(i=1;i<=n;i++)
  {
    printf("Enter an Arrival Time of the Process P%d: ",i);
    scanf("%d",&p[i].at);
    p[i].id=i;
  }

 cout<<endl;

  for(i=1;i<=n;i++)
  {
    printf("Enter a Burst Time of the Process P%d: ",i);
    scanf("%d",&p[i].bt);
  }

  for(i=1;i<=n;i++)
   {
    p1[i]=p[i];
 }
  return n;
}

void ganttrr(int n)
{
  int i,ts,m,nextval,nextarr;

  nextval=p1[1].at;
  i=1;
```

```
 cout<<"\nEnter the Time Slice or Quantum: ";
 cin>>ts;

 for(i=1;i<=n && p1[i].at<=nextval;i++)
  {
   q1.push(p1[i].id);
}

 while(!q1.empty())
 {
   m=q1.front();
   q1.pop();

   if(p1[m].bt>=ts)
    {
     nextval=nextval+ts;
}
   else
    {
     nextval=nextval+p1[m].bt;
   }
   if(p1[m].bt>=ts)
    {
     p1[m].bt=p1[m].bt-ts;
}
   else
    {
     p1[m].bt=0;
}

   while(i<=n&&p1[i].at<=nextval)
   {
     q1.push(p1[i].id);
     i++;
   }

   if(p1[m].bt>0)
    {
     q1.push(m);
}
   if(p1[m].bt<=0)
    {
     p[m].ft=nextval;
    }
}
```

```
}

void turnwait(int n)
{
  int i;

  for(i=1;i<=n;i++)
  {
    p[i].tat=p[i].ft-p[i].at;
    p[i].wt=p[i].tat-p[i].bt;
    p[0].tat=p[0].tat+p[i].tat;
    p[0].wt=p[0].wt+p[i].wt;
  }

  p[0].tat=p[0].tat/n;
  p[0].wt=p[0].wt/n;
}

void display(int n)
{
  int i;

/*
Here
at = Arrival time,
bt = Burst time,
time_quantum= Quantum time
tat = Turn around time,
wt = Waiting time
*/

  cout<<"\n=====================================================\n";
 cout<<"\n\nHere AT = Arrival Time\nBT = Burst Time\nTAT = Turn Around Time\nWT =
Waiting Time\n";

  cout<<"\n===================TABLE=============================\n";
  printf("\nProcess\tAT\tBT\tFT\tTAT\t\tWT");

  for(i=1;i<=n;i++)
 {
  printf("\nP%d\t%d\t%d\t%d\t%f\t%f",p[i].id,p[i].at,p[i].bt,p[i].ft,p[i].tat,p[i].wt);
 }
  cout<<"\n=====================================================\n";
   printf("\nAverage Turn Around Time: %f",p[0].tat);
   printf("\nAverage Waiting Time: %f\n",p[0].wt);
```

}

OUTPUT:-

```
Enter the Total Number of Process: 4

Enter an Arrival Time of the Process P1: 0
Enter an Arrival Time of the Process P2: 1
Enter an Arrival Time of the Process P3: 2
Enter an Arrival Time of the Process P4: 4

Enter a Burst Time of the Process P1: 5
Enter a Burst Time of the Process P2: 4
Enter a Burst Time of the Process P3: 2
Enter a Burst Time of the Process P4: 1

Enter the Time Slice or Quantum: 2

=========================================================



Here AT = Arrival Time
BT = Burst Time
TAT = Turn Around Time
WT = Waiting Time
```

```
==================TABLE================================

Process AT      BT      FT      TAT             WT
P1      0       5       12      12.000000       7.000000
P2      1       4       11      10.000000       6.000000
P3      2       2       6       4.000000        2.000000
P4      4       1       9       5.000000        4.000000
=========================================================

Average Turn Around Time: 7.750000
Average Waiting Time: 4.750000


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

Experiment - 5

Aim :-

Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.

PROGRAM :-

```
#include<iostream>
using namespace std;
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;

void getData()
{
   printf("\nEnter length of page reference sequence:");
   scanf("%d",&n);
   printf("\nEnter the page reference sequence:");
   for(i=0; i<n; i++)
      scanf("%d",&in[i]);
   printf("\nEnter no of frames:");
   scanf("%d",&nf);
}

void initialize()
{
   pgfaultcnt=0;
   for(i=0; i<nf; i++)
      p[i]=9999;
}

int isHit(int data)
{
   hit=0;
   for(j=0; j<nf; j++)
   {
      if(p[j]==data)
```

```
        {
            hit=1;
            break;
        }

    }

    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
            printf(" %d",p[k]);
    }

}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}

void fifo()
{
```

```
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];

            p[k]=in[i];
            pgfaultcnt++;
            dispPages();
        }
        else
            printf("No page fault");
    }
    dispPgFaultCnt();
}


void optimal()
{
    initialize();
    int near[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i; k<n; k++)
                {
                    if(pg==in[k])
```

```
                {
                   near[j]=k;
                   found=1;
                   break;
                }
                else
                   found=0;
             }
             if(!found)
                near[j]=9999;
          }
          int max=-9999;
          int repindex;
          for(j=0; j<nf; j++)
          {
             if(near[j]>max)
             {
                max=near[j];
                repindex=j;
             }
          }
          p[repindex]=in[i];
          pgfaultcnt++;

          dispPages();
       }
       else
          printf("No page fault");
    }
    dispPgFaultCnt();
}

void lru()
{
    initialize();

    int least[50];
    for(i=0; i<n; i++)
    {

       printf("\nFor %d :",in[i]);
```

```
    if(isHit(in[i])==0)
    {

        for(j=0; j<nf; j++)
        {
            int pg=p[j];
            int found=0;
            for(k=i-1; k>=0; k--)
            {
                if(pg==in[k])
                {
                    least[j]=k;
                    found=1;
                    break;
                }
                else
                    found=0;
            }
            if(!found)
                least[j]=-9999;
        }
        int min=9999;
        int repindex;
        for(j=0; j<nf; j++)
        {
            if(least[j]<min)
            {
                min=least[j];
                repindex=j;
            }
        }
        p[repindex]=in[i];
        pgfaultcnt++;

        dispPages();
    }
    else
        printf("No page fault!");
}
dispPgFaultCnt();
```

```
}
int main()
{
   int choice;
   while(1)
   {
      printf("\n\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");
      scanf("%d",&choice);
      switch(choice)
      {
      case 1:
         getData();
         break;
      case 2:
         fifo();
         break;
      case 3:
         optimal();
         break;
      case 4:
         lru();
         break;
      default:
         return 0;
         break;
      }
   }
}
```

OUTPUT:-

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice: 1

Enter length of page reference sequence: 20

Enter the page reference sequence: 1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

Enter no of frames: 3


Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice: 2

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 2 3 4
For 2 :No page fault
For 1 : 3 4 1
For 5 : 4 1 5
For 6 : 1 5 6
For 2 : 5 6 2
For 1 : 6 2 1
For 2 :No page fault
For 3 : 2 1 3
For 7 : 1 3 7
For 6 : 3 7 6
For 3 :No page fault
For 2 : 7 6 2
For 1 : 6 2 1
For 2 :No page fault
For 3 : 2 1 3
For 6 : 1 3 6
Total no of page faults:16
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice: 3

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 1 2 4
For 2 :No page fault
For 1 :No page fault
For 5 : 1 2 5
For 6 : 1 2 6
For 2 :No page fault
For 1 :No page fault
For 2 :No page fault
For 3 : 3 2 6
For 7 : 3 7 6
For 6 :No page fault
For 3 :No page fault
For 2 : 3 2 6
For 1 : 3 2 1
For 2 :No page fault
For 3 :No page fault
For 6 : 6 2 1
Total no of page faults:11
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice: 4

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 4 2 3
For 2 :No page fault!
For 1 : 4 2 1
For 5 : 5 2 1
For 6 : 5 6 1
For 2 : 5 6 2
For 1 : 1 6 2
For 2 :No page fault!
For 3 : 1 3 2
For 7 : 7 3 2
For 6 : 7 3 6
For 3 :No page fault!
For 2 : 2 3 6
For 1 : 2 3 1
For 2 :No page fault!
For 3 :No page fault!
For 6 : 2 3 6
Total no of page faults:15
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice: 5


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

Experiment - 6

Aim :-

Write a program to implement first fit, best fit and worst fit algorithm for memory management.

PROGRAM :-

- **First fit**

```
#include<bits/stdc++.h>
using namespace std;
void First_Fit(int block_size[], int total_blocks, int process_size[], int total_process) {
  int allocation[total_process];
  memset(allocation, -1, sizeof(allocation));
  for (int i = 0; i < total_process; i++) {
    for (int j = 0; j < total_blocks; j++) {
      if (block_size[j] >= process_size[i]) {
        allocation[i] = j;
        block_size[j] -= process_size[i];
        break;
      }
    }
  }
  cout << "\nProcess No.\tProcess Size\tBlock no.\n";
  for (int i = 0; i < total_process; i++) {
    cout << " " << i+1 << "\t\t" << process_size[i] << "\t\t";
    if (allocation[i] != -1)
      cout << allocation[i] + 1;
    else
      cout << "Not Allocated";
      cout << endl;
  }
}
int main() {
  int no_of_process=0;
  cout<<"Enter Number Of Process : ";
  cin>>no_of_process;
  cout<<endl;
  int process_size[no_of_process];
  for(int i=0;i<no_of_process;i++)
  {
```

```
    cout<<"Enter Process Size of process "<<i+1<<" : ";
    cin>>process_size[i];
  }
  cout<<endl;
  int no_of_blocks;
  cout<<endl<<"Enter No. Of Blocks:";
  cin>>no_of_blocks;
  int block_size[no_of_blocks];
  for(int i=0;i<no_of_blocks;i++)
  {
    cout<<"Enter Block Size of block "<<i+1<<" : ";
    cin>>block_size[i];
  }
  int total_blocks = sizeof(block_size) / sizeof(block_size[0]);
  int total_process = sizeof(process_size) / sizeof(process_size[0]);
  First_Fit(block_size, total_blocks, process_size, total_process);
  return 0 ;
}
```

- **Best fit**

```
#include <bits/stdc++.h>
#include <memory>
using namespace std;
void bestfit(int bsize[], int m, int psize[], int n) {
  int alloc[n];
   memset(alloc, -1, sizeof(alloc));
  for (int i=0; i<n; i++) {
    int bestIdx = -1;
    for (int j=0; j<m; j++) {
      if (bsize[j] >= psize[i]) {
        if (bestIdx == -1)
          bestIdx = j;
        else if (bsize[bestIdx] > bsize[j])
          bestIdx = j;
      }
    }
    if (bestIdx != -1) {
      alloc[i] = bestIdx;
      bsize[bestIdx] -= psize[i];
    }
  }
```

```
  cout << "\nProcess No.\tProcess Size\tBlock no.\n";
  for (int i = 0; i < n; i++) {
    cout << " " << i+1 << "\t\t" << psize[i] << "\t\t ";
    if (alloc[i] != -1)
      cout << alloc[i] + 1;
    else
      cout << "Not Allocated";
      cout << endl;
  }
}
int main() {
  int no_of_process=0;
  cout<<"Enter Number Of Process : ";
  cin>>no_of_process;
  cout<<endl;
  int psize[no_of_process];
  for(int i=0;i<no_of_process;i++)
  {
     cout<<"Enter Process Size of process "<<i+1<<" : ";
     cin>>psize[i];
  }
  cout<<endl;
  int no_of_blocks;
  cout<<endl<<"Enter No. Of Blocks:";
  cin>>no_of_blocks;
  int bsize[no_of_blocks];
  for(int i=0;i<no_of_blocks;i++)
  {
     cout<<"Enter Block Size of block "<<i+1<<" : ";
     cin>>bsize[i];
  }
  int m = sizeof(bsize)/sizeof(bsize[0]);
  int n = sizeof(psize)/sizeof(psize[0]);
  bestfit(bsize, m, psize, n);
  return 0 ;
}
```

- **Worst fit**

```
#include<bits/stdc++.h>
using namespace std;
void worstFit(int blockSize[], int m, int processSize[],int n)
```

```
{
    int allocation[n];
   memset(allocation, -1, sizeof(allocation));
    for (int i=0; i<n; i++)
    {
       int wstIdx = -1;
       for (int j=0; j<m; j++)
       {
          if (blockSize[j] >= processSize[i])
          {
             if (wstIdx == -1)
                wstIdx = j;
             else if (blockSize[wstIdx] < blockSize[j])
                wstIdx = j;
          }
       }

       if (wstIdx != -1)
       {
          allocation[i] = wstIdx;
          blockSize[wstIdx] -= processSize[i];
       }
    }

    cout << "Process No.\tProcess Size.\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
       cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";
       if (allocation[i] != -1)
          cout << allocation[i] + 1;
       else
          cout << "Not Allocated";
       cout << endl;
    }
}
int main()
{
   int no_of_process=0;
   cout<<"Enter Number Of Process : ";
   cin>>no_of_process;
   cout<<endl;
```

```
    int processSize[no_of_process];
    for(int i=0;i<no_of_process;i++)
    {
       cout<<"Enter Process Size of process "<<i+1<<" : ";
       cin>>processSize[i];
    }
    cout<<endl;
    int no_of_blocks;
    cout<<endl<<"Enter No. Of Blocks:";
    cin>>no_of_blocks;
    int blockSize[no_of_blocks];
    for(int i=0;i<no_of_blocks;i++)
    {
       cout<<"Enter Block Size of block "<<i+1<<" : ";
       cin>>blockSize[i];
    }
     cout<<endl;
     int m = sizeof(blockSize)/sizeof(blockSize[0]);
     int n = sizeof(processSize)/sizeof(processSize[0]);

     worstFit(blockSize, m, processSize, n);

     return 0 ;
}
```

OUTPUT :-

- **First fit**

```
Enter Number Of Process : 5

Enter Process Size of process 1 : 200
Enter Process Size of process 2 : 47
Enter Process Size of process 3 : 212
Enter Process Size of process 4 : 426
Enter Process Size of process 5 : 10


Enter No. Of Blocks:5
Enter Block Size of block 1 : 300
Enter Block Size of block 2 : 50
Enter Block Size of block 3 : 200
Enter Block Size of block 4 : 350
Enter Block Size of block 5 : 70

Process No.     Process Size    Block no.
 1              200             1
 2              47              1
 3              212             4
 4              426             Not Allocated
 5              10              1


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

- **Best fit**

```
Enter Number Of Process : 4

Enter Process Size of process 1 : 112
Enter Process Size of process 2 : 518
Enter Process Size of process 3 : 110
Enter Process Size of process 4 : 526


Enter No. Of Blocks:5
Enter Block Size of block 1 : 100
Enter Block Size of block 2 : 500
Enter Block Size of block 3 : 200
Enter Block Size of block 4 : 300
Enter Block Size of block 5 : 400

Process No.     Process Size    Block no.
 1              112              3
 2              518              Not Allocated
 3              110              4
 4              526              Not Allocated


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

- **Worst fit**

```
Enter Number Of Process : 4

Enter Process Size of process 1 : 212
Enter Process Size of process 2 : 417
Enter Process Size of process 3 : 112
Enter Process Size of process 4 : 426


Enter No. Of Blocks:5
Enter Block Size of block 1 : 100
Enter Block Size of block 2 : 500
Enter Block Size of block 3 : 200
Enter Block Size of block 4 : 300
Enter Block Size of block 5 : 600

Process No.     Process Size.   Block no.
   1            212             5
   2            417             2
   3            112             5
   4            426             Not Allocated


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

Experiment - 7

Aim :-

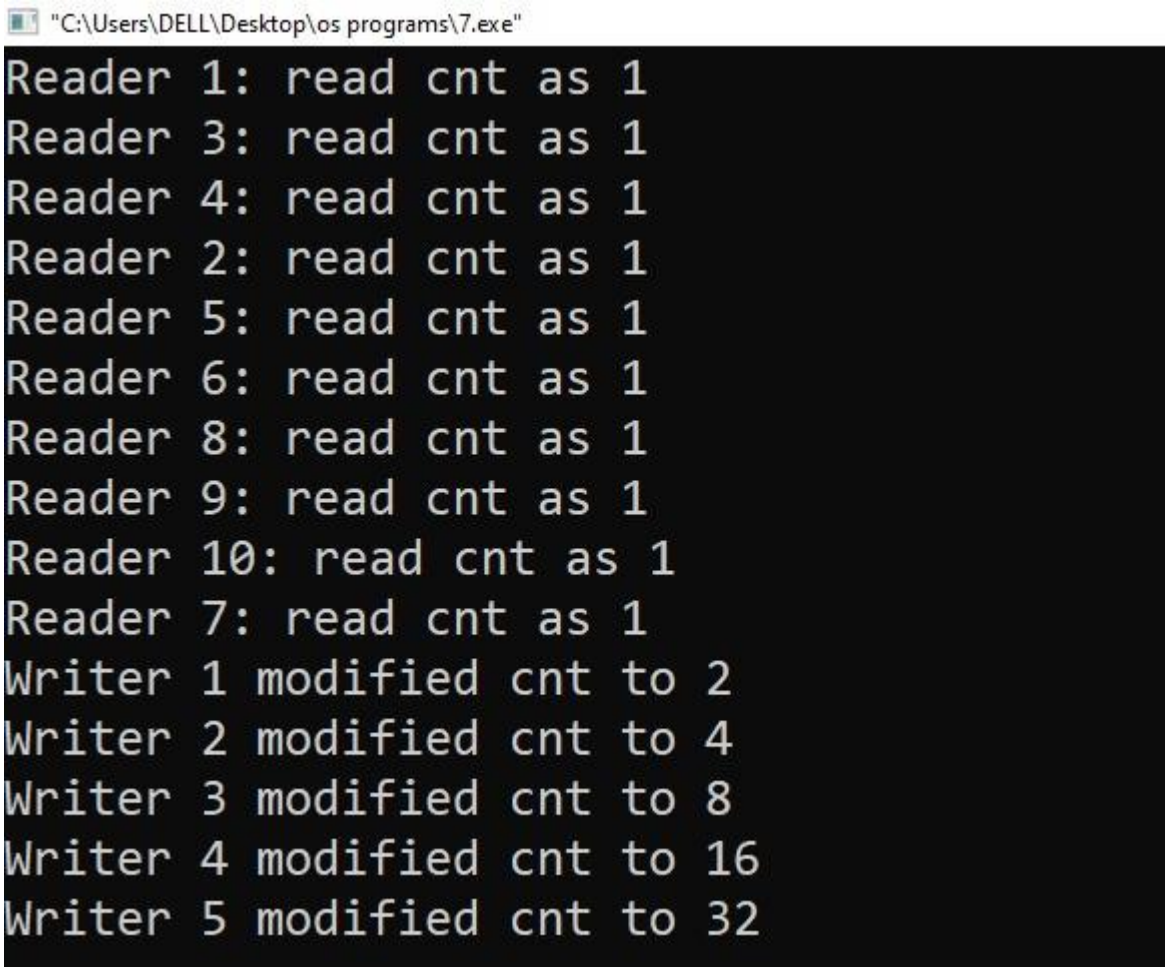
Write a program to implement reader/writer problem using semaphore.

PROGRAM :-

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
   sem_post(&wrt);

}
void *reader(void *rno)
{
   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt);    }
   pthread_mutex_unlock(&mutex);
   printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);

   pthread_mutex_lock(&mutex);
   numreader--;
   if(numreader == 0) {
      sem_post(&wrt);
   }
   pthread_mutex_unlock(&mutex);
}
```

```
int main()
{

   pthread_t read[10],write[5];
   pthread_mutex_init(&mutex, NULL);
   sem_init(&wrt,0,1);

   int a[10] = {1,2,3,4,5,6,7,8,9,10};

   for(int i = 0; i < 10; i++) {
      pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
   }
   for(int i = 0; i < 5; i++) {
      pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
   }

   for(int i = 0; i < 10; i++) {
      pthread_join(read[i], NULL);
   }
   for(int i = 0; i < 5; i++) {
      pthread_join(write[i], NULL);
   }

   pthread_mutex_destroy(&mutex);
   sem_destroy(&wrt);

   return 0;
}
```

OUTPUT :-


"C:\Users\DELL\Desktop\os programs\7.exe"
Reader 1: read cnt as 1
Reader 3: read cnt as 1
Reader 4: read cnt as 1
Reader 2: read cnt as 1
Reader 5: read cnt as 1
Reader 6: read cnt as 1
Reader 8: read cnt as 1
Reader 9: read cnt as 1
Reader 10: read cnt as 1
Reader 7: read cnt as 1
Writer 1 modified cnt to 2
Writer 2 modified cnt to 4
Writer 3 modified cnt to 8
Writer 4 modified cnt to 16
Writer 5 modified cnt to 32

Experiment - 8

Aim :-

Write a program to implement Banker's algorithm for deadlock avoidance.

PROGRAM :-

```
#include <iostream>
using namespace std;
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()
{
        cout<<"\nEnter number of processes: ";
   cin>>processes;
   for(i=0;i<processes;i++)
        {
      running[i] = 1;
      counter++;
   }
   cout<<"\nEnter number of resources: ";
   cin>>resources;
   cout<<"\nEnter Claim Vector:";
   for(i=0;i<resources;i++)
         cin>>maxres[i];
   cout<<"\nEnter Allocated Resource Table:\n";
   for(i=0;i<processes;i++)
      for(j=0;j<resources;j++)
                           cin>>current[i][j];
    cout<<"\nEnter Maximum Claim Table:\n";
    for(i=0;i<processes;i++)
      for(j = 0; j < resources; j++)
                  cin>>maximum_claim[i][j];
        cout<<"\nThe Claim Vector is: ";
    for(i=0;i<resources;i++)
          cout<<"\t"<<maxres[i];
        cout<<"\nThe Allocated Resource Table:\n";
    for(i=0;i<processes;i++)
```

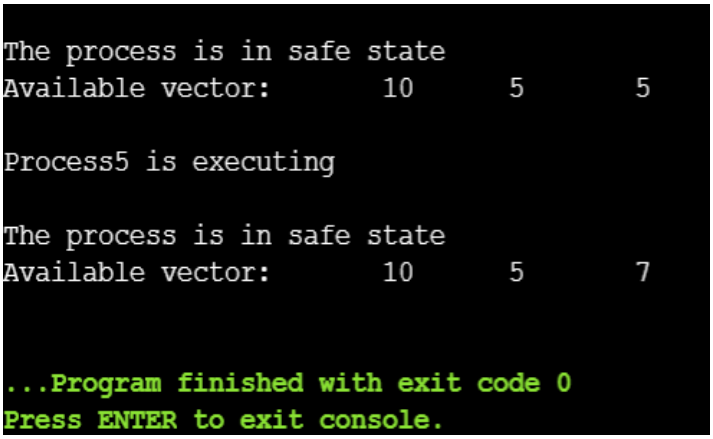
```
{
   for(j=0;j<resources;j++)
                cout<<"\t"<<current[i][j];
   cout<<endl;
}
cout<<"\nThe Maximum Claim Table:\n";
for(i=0;i<processes;i++)
{
   for (j = 0; j < resources; j++)
     cout<<"\t"<<maximum_claim[i][j];
   cout<<endl;
}
      for(i=0;i<processes;i++)
  for(j=0;j<resources;j++)
      allocation[j]+=current[i][j];

cout<<"\nAllocated resources:";
for(i=0;i<resources;i++)
  cout<<"\t"<<allocation[i];
for(i=0;i<resources;i++)
  available[i] = maxres[i] - allocation[i];
cout<<"\nAvailable resources:";
for(i=0;i<resources;i++)
       cout<<"\t"<<available[i];
cout<<endl;
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)
                                    {
              cout<<"\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)
                      {
            cout<<"\nThe processes are in unsafe state.\n";
            break;
         }
        else
                      {
          cout<<"\nThe process is in safe state";
          cout<<"\nAvailable vector:";
          for (i = 0; i < resources; i++)
             cout<<"\t"<<available[i];
                       cout<<endl;
         }
        }
           return 0;
}
```

OUTPUT :-

```
Enter number of processes: 5

Enter number of resources: 3

Enter Claim Vector: 10 5 7

Enter Allocated Resource Table:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Maximum Claim Table:
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3

The Claim Vector is:    10      5       7
The Allocated Resource Table:
        0       1       0
        2       0       0
        3       0       2
        2       1       1
        0       0       2
```

```
The Maximum Claim Table:
        7       5       3
        3       2       2
        9       0       2
        4       2       2
        5       3       3

Allocated resources:    7       2       5
Available resources:    3       3       2

Process2 is executing

The process is in safe state
Available vector:       5       3       2

Process4 is executing

The process is in safe state
Available vector:       7       4       3

Process1 is executing

The process is in safe state
Available vector:       7       5       3

Process3 is executing
```


The process is in safe state
Available vector: 10 5 5
Process5 is executing
The process is in safe state
Available vector: 10 5 7
...Program finished with exit code 0
Press ENTER to exit console.