VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

(23CS4PCOPS)

Submitted by

MADHUSHREE S SHETTY(1BM22CS141)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Apr-2024 to Aug-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "OPERATING SYSTEMS – 23CS4PCOPS" carried out by **MADHUSHREE S SHETTY(1BM22CS141),** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS - (23CS4PCOPS)** work prescribed for the said degree.

Rajeshwari Madli	**Dr. Jyothi S Nayak**
Assistant Professor Department of CSE BMSCE, Bengaluru	Professor and Head Department of CSE BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System
CO2	Analyze various Operating system strategies and techniques
CO3	Demonstrate the different functionalities of Operating System
CO4	Conduct practical experiments to implement the functionalities of Operating system

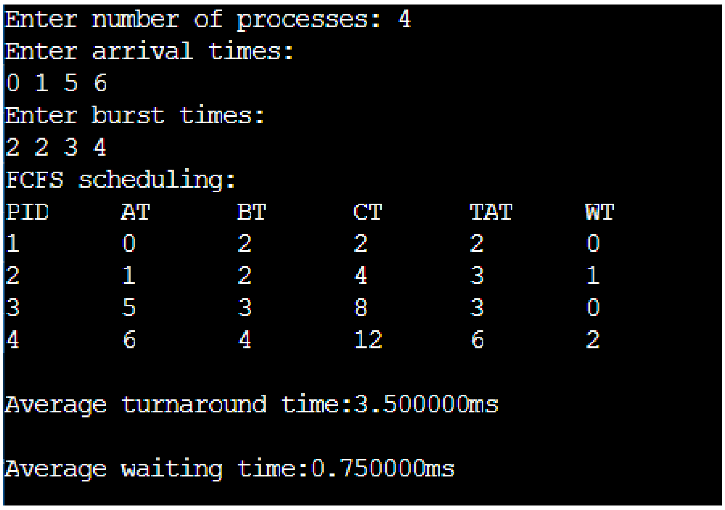
Program 1:

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

FCFS

```
#include<stdio.h>

void sort(int proc_id[],int at[],int bt[],int n)
{
    int min=at[0],temp=0;
    for(int i=0;i<n;i++)
    {
        min=at[i];
        for(int j=i;j<n;j++)
        {
            if(at[j]<min)
            {
                temp=at[i];at[i]=at[j];at[j]=temp;
                temp=bt[j];bt[j]=bt[i];bt[i]=temp;
                temp=proc_id[i];proc_id[i]=proc_id[j];proc_id[j]=temp;
            }
        }
    }
}

void main()
{
    int n,c=0;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n];
    double avg_tat=0.0,ttat=0.0,avg_wt=0.0,twt=0.0;
    for(int i=0;i<n;i++)
        proc_id[i]=i+1;
    printf("Enter arrival times:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&at[i]);
    printf("Enter burst times:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&bt[i]);

    sort(proc_id,at,bt,n);
    //completion time
    for(int i=0;i<n;i++)
    {
        if(c>=at[i])
            c+=bt[i];
        else
            c+=at[i]-ct[i-1]+bt[i];
        ct[i]=c;
    }
    //turnaround time
    for(int i=0;i<n;i++)
        tat[i]=ct[i]-at[i];
    //waiting time
    for(int i=0;i<n;i++)
        wt[i]=tat[i]-bt[i];

    printf("FCFS scheduling:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<n;i++)

printf("%d\t%d\t%d\t%d\t%d\t%d\n",proc_id[i],at[i],bt[i],ct[i],tat[i],wt[i]
);

    for(int i=0;i<n;i++)
    {
        ttat+=tat[i];twt+=wt[i];
    }
    avg_tat=ttat/(double)n;
    avg_wt=twt/(double)n;
    printf("\nAverage turnaround time:%lfms\n",avg_tat);
    printf("\nAverage waiting time:%lfms\n",avg_wt);
}
```

Output:

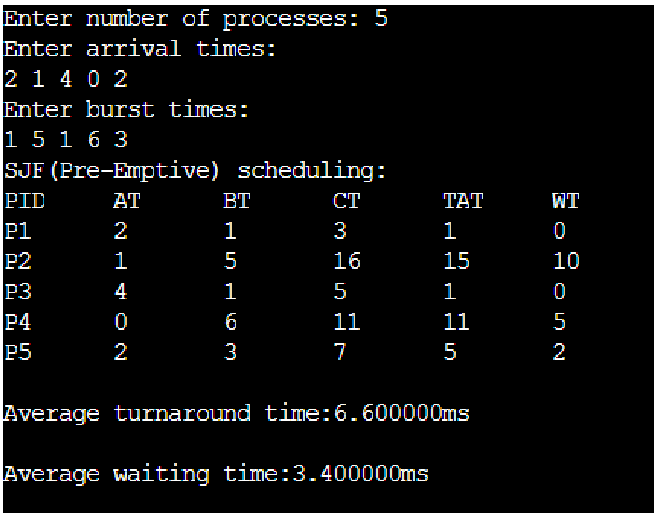


```
Enter number of processes: 4
Enter arrival times:
0 1 5 6
Enter burst times:
2 2 3 4
FCFS scheduling:
PID     AT      BT      CT      TAT     WT
1       0       2       2       2       0
2       1       2       4       3       1
3       5       3       8       3       0
4       6       4       12      6       2

Average turnaround time:3.500000ms

Average waiting time:0.750000ms
```

SJF (Pre-emptive)

```
#include<stdio.h>

void main()
{
    int n,c=0;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n],m[n],b[n];
    double avg_tat=0.0,ttat=0.0,avg_wt=0.0,twt=0.0;
    for(int i=0;i<n;i++)
    {    proc_id[i]=i+1;m[i]=0;}
    printf("Enter arrival times:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&at[i]);
    printf("Enter burst times:\n");
    for(int i=0;i<n;i++)
    {    scanf("%d",&bt[i]);b[i]=bt[i];}

    //completion time
    int count=0,mb,p=0,min=0;
    while(count<n)
    {
        min=b[0];mb=0;
        for(int i=0;i<n;i++)
        {
            if(at[i]<=c && m[i]!=1)
            {
                min=b[i];mb=i;
                for(int k=0;k<n;k++)
                {
                    if(b[k]<=min && at[k]<=c && m[k]!=1)
                    {
                        min=b[k];mb=k;
                    }
                }
                if(b[mb]==1)
                {m[mb]=1;count++;}
                if(c>=at[mb])
                {    c++;b[mb]--;}
                else
                    c+=at[mb]-ct[p];
                if(b[mb]==0)
                ct[mb]=c;
            }
            p=mb;
            if(count==n)
            break;
        }
    }

    //turnaround time
    for(int i=0;i<n;i++)
        tat[i]=ct[i]-at[i];
    //waiting time
    for(int i=0;i<n;i++)
        wt[i]=tat[i]-bt[i];

    printf("SJF(Pre-Emptive) scheduling:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<n;i++)
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",proc_id[i],at[i],bt[i],ct[i],tat[i],wt[i]);

    for(int i=0;i<n;i++)
    {
        ttat+=tat[i];twt+=wt[i];
    }
    avg_tat=ttat/(double)n;
    avg_wt=twt/(double)n;
    printf("\nAverage turnaround time:%lfms\n",avg_tat);
    printf("\nAverage waiting time:%lfms\n",avg_wt);
}
```

Output:

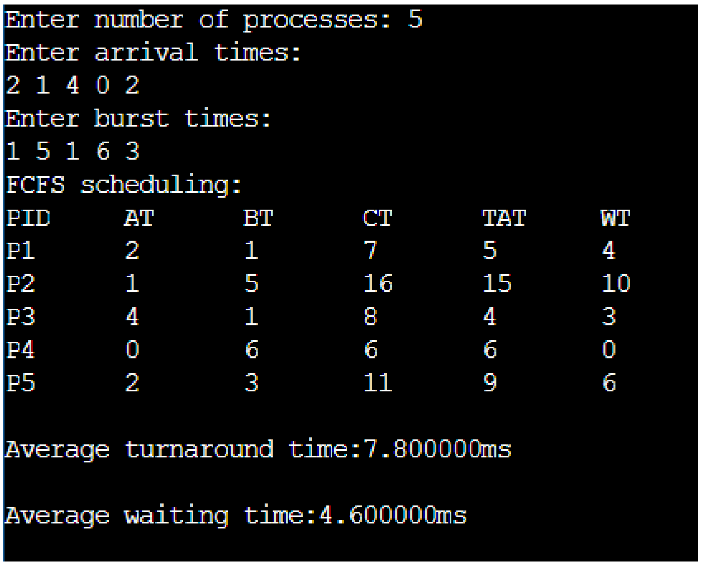


```
Enter number of processes: 5
Enter arrival times:
2 1 4 0 2
Enter burst times:
1 5 1 6 3
SJF(Pre-Emptive) scheduling:
PID     AT      BT      CT      TAT     WT
P1      2       1       3       1       0
P2      1       5       16      15      10
P3      4       1       5       1       0
P4      0       6       11      11      5
P5      2       3       7       5       2

Average turnaround time:6.600000ms

Average waiting time:3.400000ms
```

SJF(Non-preemptive)

```
#include<stdio.h>

void main()
{
    int n,c=0;
    printf("Enter number of processes: ");
    scanf("%d",&n);
    int proc_id[n],at[n],bt[n],ct[n],tat[n],wt[n],m[n];
    double avg_tat=0.0,ttat=0.0,avg_wt=0.0,twt=0.0;
    for(int i=0;i<n;i++)
    {    proc_id[i]=i+1;m[i]=0;}
    printf("Enter arrival times:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&at[i]);
    printf("Enter burst times:\n");
    for(int i=0;i<n;i++)
        scanf("%d",&bt[i]);

    //completion time
    int count=0,mb,p=0,min=0;
    while(count<n)
    {
        min=bt[0];mb=0;
        for(int i=0;i<n;i++)
        {
            if(at[i]<=c && m[i]!=1)
            {
                min=bt[i];mb=i;
                for(int k=0;k<n;k++)
                {
                    if(bt[k]<min && at[k]<=c && m[k]!=1)
                    {
                        min=bt[k];mb=k;
                    }
                }
                m[mb]=1;count++;
                if(c>=at[mb])
                    c+=bt[mb];
                else
                    c+=at[mb]-ct[p]+bt[mb];
                ct[mb]=c;
            }
            p=mb;
            if(count==n)
            break;
        }
    }
    //turnaround time
    for(int i=0;i<n;i++)
        tat[i]=ct[i]-at[i];
    //waiting time
    for(int i=0;i<n;i++)
        wt[i]=tat[i]-bt[i];

    printf("FCFS scheduling:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<n;i++)
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",proc_id[i],at[i],bt[i],ct[i],tat[i],wt[i
]);
    for(int i=0;i<n;i++)
    {
        ttat+=tat[i];twt+=wt[i];
    }
    avg_tat=ttat/(double)n;
    avg_wt=twt/(double)n;
    printf("\nAverage turnaround time:%lfms\n",avg_tat);
    printf("\nAverage waiting time:%lfms\n",avg_wt);
}
```

Output:



```
Enter number of processes: 5
Enter arrival times:
2 1 4 0 2
Enter burst times:
1 5 1 6 3
FCFS scheduling:
PID     AT      BT      CT      TAT     WT
P1      2       1       7       5       4
P2      1       5       16      15      10
P3      4       1       8       4       3
P4      0       6       6       6       0
P5      2       3       11      9       6

Average turnaround time:7.800000ms

Average waiting time:4.600000ms
```

Program 2:

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

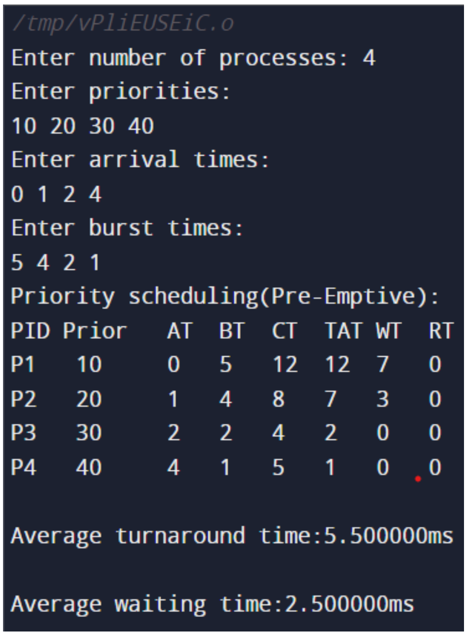
Priority (Preemptive)

```
#include<stdio.h>
void sort (int proc_id[], int p[], int at[], int bt[], int b[], int n)
{
  int min = p[0], temp = 0;
  for (int i = 0; i < n; i++)
      {
        min = p[i];
        for (int j = i; j < n; j++)
            {
              if (p[j] < min)
                  {
                    temp = at[i];
                    at[i] = at[j];
                    at[j] = temp;
                    temp = bt[j];
                    bt[j] = bt[i];
                    bt[i] = temp;
                    temp = b[j];
                    b[j] = b[i];
                    b[i] = temp;
                    temp = p[j];
                    p[j] = p[i];
                    p[i] = temp;
                    temp = proc_id[i];
                    proc_id[i] = proc_id[j];
                    proc_id[j] = temp;
                  }
            }
      }
}
void main ()
{
  int n, c = 0;
  printf ("Enter number of processes: ");
  scanf ("%d", &n);
   int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], m[n], b[n], rt[n],
p[n];
  double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
  for (int i = 0; i < n; i++)
```

```
   {
    proc_id[i] = i + 1;
    m[i] = 0;
   }
 printf ("Enter priorities:\n");
 for (int i = 0; i < n; i++)
   scanf ("%d", &p[i]);
 printf ("Enter arrival times:\n");
 for (int i = 0; i < n; i++)
   scanf ("%d", &at[i]);
 printf ("Enter burst times:\n");
 for (int i = 0; i < n; i++)
   {
    scanf ("%d", &bt[i]);
    b[i] = bt[i];
    m[i] = -1;
    rt[i] = -1;
   }

 sort (proc_id, p, at, bt, b, n);
//completion time
 int count = 0, pro = 0, priority = p[0];
 int x = 0;
 c = 0;
 while (count < n)
   {for (int i = 0; i < n; i++)
      {if (at[i] <= c && p[i] >= priority && b[i] > 0 && m[i] != 1)
         {x = i;
          priority = p[i];
         }
      }
    if (b[x] > 0)
      { if (rt[x] == -1)
         rt[x] = c - at[x];b[x]--;
       c++;
      }
    if (b[x] == 0)
      {count++;
       ct[x] = c;
       m[x] = 1;
       while (x >= 1 && b[x] == 0)
         priority = p[--x];
      }
    if (count == n)
```

```
            break;
      }

  //turnaround time and RT
  for (int i = 0; i < n; i++)
      tat[i] = ct[i] - at[i];
  //waiting time
  for (int i = 0; i < n; i++)
      wt[i] = tat[i] - bt[i];

  printf ("Priority scheduling(Pre-Emptive):\n");
  printf ("PID\tPrior\tAT\tBT\tCT\tTAT\tWT\tRT\n");
  for (int i = 0; i < n; i++)
      printf  ("P%d\t  %d\t\t%d\t%d\t%d\t%d\t%d\t%d\n",  proc_id[i],  p[i],
at[i],
                  bt[i], ct[i], tat[i], wt[i], rt[i]);
for (int i = 0; i < n; i++)
      { ttat += tat[i];
        twt += wt[i];
      }
  avg_tat = ttat / (double) n;
  avg_wt = twt / (double) n;
  printf ("\nAverage turnaround time:%lfms\n", avg_tat);
  printf ("\nAverage waiting time:%lfms\n", avg_wt);
}
```

Output:



```
/tmp/vPliEUSEiC.o
Enter number of processes: 4
Enter priorities:
10 20 30 40
Enter arrival times:
0 1 2 4
Enter burst times:
5 4 2 1
Priority scheduling(Pre-Emptive):
PID Prior   AT  BT  CT  TAT WT  RT
P1   10     0   5   12  12  7   0
P2   20     1   4   8   7   3   0
P3   30     2   2   4   2   0   0
P4   40     4   1   5   1   0   0

Average turnaround time:5.500000ms

Average waiting time:2.500000ms
```

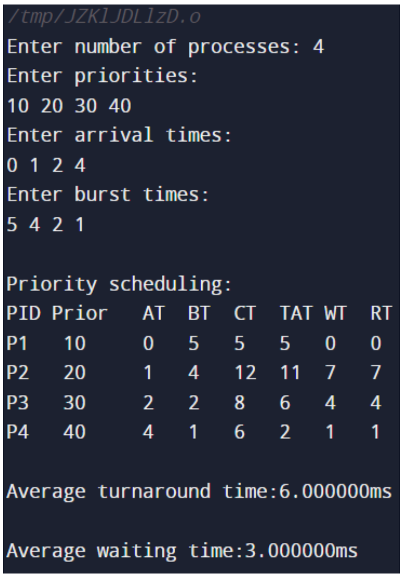
Priority (Non-Preemptive)

```
#include<stdio.h>
void sort (int proc_id[], int p[], int at[], int bt[], int n)
{
  int min = p[0], temp = 0;
  for (int i = 0; i < n; i++)
      {
        min = p[i];
        for (int j = i; j < n; j++)
            {
              if (p[j] < min)
                  {
                    temp = at[i];
                    at[i] = at[j];
                    at[j] = temp;
                    temp = bt[j];
                    bt[j] = bt[i];
                    bt[i] = temp;
                    temp = p[j];
                    p[j] = p[i];
                    p[i] = temp;
                    temp = proc_id[i];
                    proc_id[i] = proc_id[j];
                    proc_id[j] = temp;
                  }
            }
      }
}
void main ()
{
  int n, c = 0;
  printf ("Enter number of processes: ");
  scanf ("%d", &n);
  int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], m[n], rt[n], p[n];
  double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
  for (int i = 0; i < n; i++)
      {
        proc_id[i] = i + 1;
        m[i] = 0;
      }
  printf ("Enter priorities:\n");
  for (int i = 0; i < n; i++)
      scanf ("%d", &p[i]);
```

```
 printf ("Enter arrival times:\n");
 for (int i = 0; i < n; i++)
     scanf ("%d", &at[i]);
 printf ("Enter burst times:\n");
 for (int i = 0; i < n; i++)
     {
       scanf ("%d", &bt[i]);
       m[i] = -1;
       rt[i] = -1;
     }
sort (proc_id, p, at, bt, n);
//completion time
 int count = 0, pro = 0, priority = p[0];
 int x = 0;
 c = 0;
 while (count < n)
     {
       for (int i = 0; i < n; i++)
           {
             if (at[i] <= c && p[i] >= priority && m[i] != 1)
                 {
                   x = i;
                   priority = p[i];
                 }
           }
       if (rt[x] == -1)
           rt[x] = c - at[x];
       if (at[x] <= c)
           c += bt[x];
       else
           c += at[x] - c + bt[x];

       count++;
       ct[x] = c;
       m[x] = 1;
       while (x >= 1 && m[--x] != 1)
           {
             priority = p[x];
             break;
           }
       x++;
       if (count == n)
           break;
     }
```

```
//turnaround time and RT
  for (int i = 0; i < n; i++)
      tat[i] = ct[i] - at[i];
  //waiting time
  for (int i = 0; i < n; i++)
      wt[i] = tat[i] - bt[i];

  printf ("\nPriority scheduling:\n");
  printf ("PID\tPrior\tAT\tBT\tCT\tTAT\tWT\tRT\n");
  for (int i = 0; i < n; i++)
      printf  ("P%d\t  %d\t\t%d\t%d\t%d\t%d\t%d\t%d\n",  proc_id[i],  p[i],
at[i],
                  bt[i], ct[i], tat[i], wt[i], rt[i]);
 for (int i = 0; i < n; i++)
      {
        ttat += tat[i];
        twt += wt[i];
      }
  avg_tat = ttat / (double) n;
  avg_wt = twt / (double) n;
  printf ("\nAverage turnaround time:%lfms\n", avg_tat);
  printf ("\nAverage waiting time:%lfms\n", avg_wt);
}
```

Output:



```
/tmp/JZKlJDLlzD.o
Enter number of processes: 4
Enter priorities:
10 20 30 40
Enter arrival times:
0 1 2 4
Enter burst times:
5 4 2 1

Priority scheduling:
PID Prior   AT  BT  CT  TAT WT  RT
P1   10     0   5   5   5   0   0
P2   20     1   4   12  11  7   7
P3   30     2   2   8   6   4   4
P4   40     4   1   6   2   1   1

Average turnaround time:6.000000ms

Average waiting time:3.000000ms
```

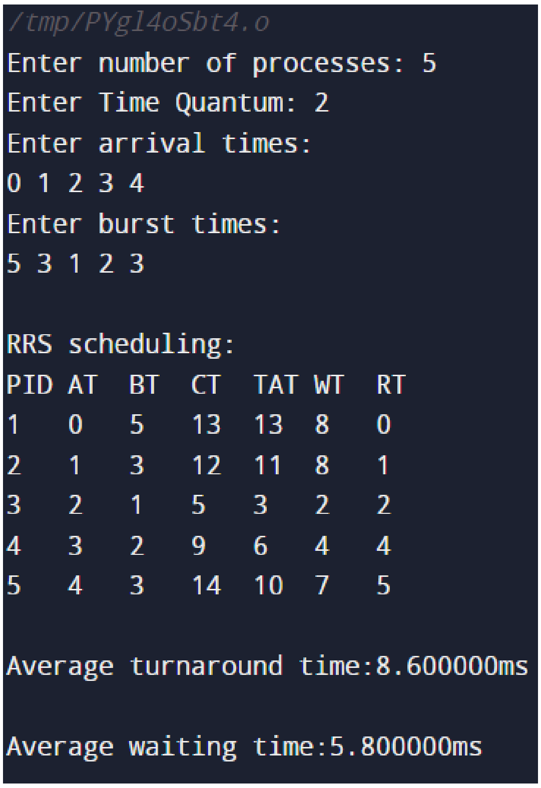
Round Robin (Experiment with different quantum sizes for RR algorithm)

```
#include<stdio.h>
void sort (int proc_id[], int at[], int bt[], int b[], int n)
{
  int min = at[0], temp = 0;
  for (int i = 0; i < n; i++)
      {
        min = at[i];
        for (int j = i; j < n; j++)
            {
              if (at[j] < min)
                  {
                    temp = at[i];
                    at[i] = at[j];
                    at[j] = temp;
                    temp = bt[j];
                    bt[j] = bt[i];
                    bt[i] = temp;
                    temp = b[j];
                    b[j] = b[i];
                    b[i] = temp;
                    temp = proc_id[i];
                    proc_id[i] = proc_id[j];
                    proc_id[j] = temp;
                  }
            }
      }
}
void main ()
{
  int n, c = 0, t = 0;
  printf ("Enter number of processes: ");
  scanf ("%d", &n);
  printf ("Enter Time Quantum: ");
  scanf ("%d", &t);
  int proc_id[n], at[n], bt[n], ct[n], tat[n], wt[n], b[n], rt[n], m[n];
  int f = -1, r = -1;
  int q[100];
  int count = 0;
  double avg_tat = 0.0, ttat = 0.0, avg_wt = 0.0, twt = 0.0;
  for (int i = 0; i < n; i++)
      proc_id[i] = i + 1;
  printf ("Enter arrival times:\n");
  for (int i = 0; i < n; i++)
```

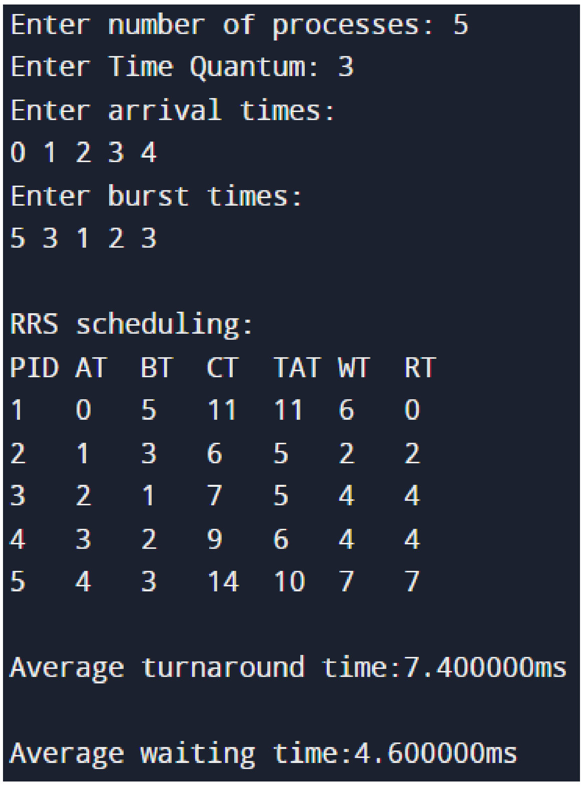
```
   scanf ("%d", &at[i]);
 printf ("Enter burst times:\n");
 for (int i = 0; i < n; i++)
   {
    scanf ("%d", &bt[i]);
    b[i] = bt[i];
    m[i] = 0;
    rt[i] = -1;
   }
sort (proc_id, at, bt, b, n);
f = r = 0;
 q[0] = proc_id[0];
//completion time
 int p = 0, i = 0;
 while (f >= 0)
   {
    p = q[f++];
    i = 0;
    while (p != proc_id[i])
      i++;
    if (b[i] >= t)
      {
       if (rt[i] == -1)
         rt[i] = c;
       b[i] -= t;
       c += t;
       m[i] = 1;
      }
    else
      {
       if (rt[i] == -1)
         rt[i] = c;
       c += b[i];
       b[i] = 0;
       m[i] = 1;
      }
    m[0] = 1;
    for (int j = 0; j < n; j++)
      {
       if (at[j] <= c && proc_id[j] != p && m[j] != 1)
         {
          q[++r] = proc_id[j];
          m[j] = 1;
         }
```

```
            }
        if (b[i] == 0)
            {
              count++;
              ct[i] = c;
            }
        else
            q[++r] = proc_id[i];

        if (f > r)
            f = -1;
      }
 //turnaround time and RT
  for (int i = 0; i < n; i++)
      {
        tat[i] = ct[i] - at[i];
        rt[i] = rt[i] - at[i];
      }
  //waiting time
  for (int i = 0; i < n; i++)
      wt[i] = tat[i] - bt[i];
 printf ("\nRRS scheduling:\n");
  printf ("PID\tAT\tBT\tCT\tTAT\tWT\tRT\n");
  for (int i = 0; i < n; i++)
      printf  ("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",  proc_id[i],  at[i],  bt[i],
ct[i],
                  tat[i], wt[i], rt[i]);
for (int i = 0; i < n; i++)
      {
        ttat += tat[i];
        twt += wt[i];
      }
  avg_tat = ttat / (double) n;
  avg_wt = twt / (double) n;
  printf ("\nAverage turnaround time:%lfms\n", avg_tat);
  printf ("\nAverage waiting time:%lfms\n", avg_wt);
}
```

Output:



```
/tmp/PYgl4oSbt4.o
Enter number of processes: 5
Enter Time Quantum: 2
Enter arrival times:
0 1 2 3 4
Enter burst times:
5 3 1 2 3

RRS scheduling:
PID AT  BT  CT  TAT WT  RT
1   0   5   13  13  8   0
2   1   3   12  11  8   1
3   2   1   5   3   2   2
4   3   2   9   6   4   4
5   4   3   14  10  7   5

Average turnaround time:8.600000ms

Average waiting time:5.800000ms
```



```
Enter number of processes: 5
Enter Time Quantum: 3
Enter arrival times:
0 1 2 3 4
Enter burst times:
5 3 1 2 3

RRS scheduling:
PID AT  BT  CT  TAT WT  RT
1   0   5   11  11  6   0
2   1   3   6   5   2   2
3   2   1   7   5   4   4
4   3   2   9   6   4   4
5   4   3   14  10  7   7

Average turnaround time:7.400000ms

Average waiting time:4.600000ms
```

Program 3:

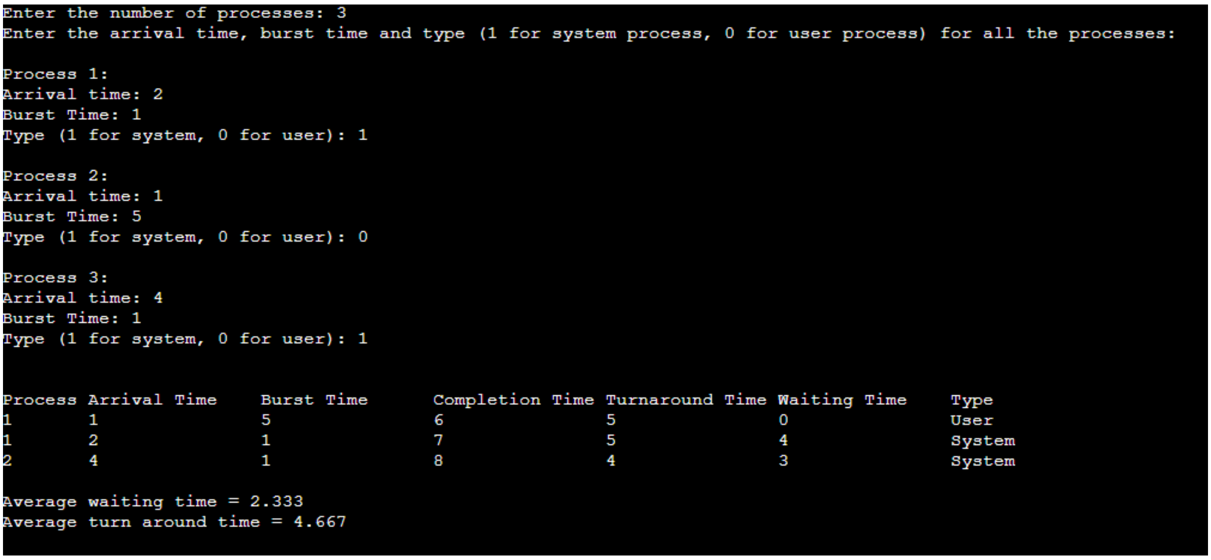
Write a C program to simulate a multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

void FCFS(int AT[], int BT[], int CT[], int TAT[], int WT[], int n, int *current_time) {
    for (int i = 0; i < n; i++) {
        if (*current_time < AT[i])
            *current_time = AT[i];

        CT[i] = *current_time + BT[i];
        TAT[i] = CT[i] - AT[i];
        WT[i] = TAT[i] - BT[i];

        *current_time = CT[i];
    }
}

void sort_by_arrival(int AT[], int BT[], int isSystem[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (AT[j] > AT[j + 1]) {
                int temp = AT[j];
                AT[j] = AT[j + 1];
                AT[j + 1] = temp;

                temp = BT[j];
                BT[j] = BT[j + 1];
                BT[j + 1] = temp;

                temp = isSystem[j];
                isSystem[j] = isSystem[j + 1];
                isSystem[j + 1] = temp;
            }
        }
    }
}
```

```
int main() {
    int n, sys_count = 0, user_count = 0;
    int AT[MAX], BT[MAX], CT[MAX], TAT[MAX], WT[MAX], isSystem[MAX];
    int sys_AT[MAX], sys_BT[MAX], user_AT[MAX], user_BT[MAX];

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the arrival time, burst time and type (1 for system
process, 0 for user process) for all the processes:\n");

    for (int i = 0; i < n; i++) {
        printf("\nProcess %d:\n", i + 1);
        printf("Arrival time: ");
        scanf("%d", &AT[i]);
        printf("Burst Time: ");
        scanf("%d", &BT[i]);
        printf("Type (1 for system, 0 for user): ");
        scanf("%d", &isSystem[i]);

        if (isSystem[i]) {
            sys_AT[sys_count] = AT[i];
            sys_BT[sys_count] = BT[i];
            sys_count++;
        } else {
            user_AT[user_count] = AT[i];
            user_BT[user_count] = BT[i];
            user_count++;
        }
    }

    sort_by_arrival(sys_AT, sys_BT, isSystem, sys_count);
    sort_by_arrival(user_AT, user_BT, isSystem, user_count);

    int current_time = 0;
    int total_wt = 0, total_tat = 0;

    printf("\n\nProcess\tArrival Time\tBurst Time\tCompletion
Time\tTurnaround Time\tWaiting Time\tType\n");

    int i = 0, j = 0;
    while (i < sys_count || j < user_count) {
        if (i < sys_count && (j >= user_count || sys_AT[i] <=
current_time)) {
```

```
            if (current_time < sys_AT[i])
                current_time = sys_AT[i];

            CT[i] = current_time + sys_BT[i];
            TAT[i] = CT[i] - sys_AT[i];
            WT[i] = TAT[i] - sys_BT[i];

            current_time = CT[i];

               printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\tSystem\n", i + 1,
sys_AT[i], sys_BT[i], CT[i], TAT[i], WT[i]);
            total_wt += WT[i];
            total_tat += TAT[i];

            i++;
        } else if (j < user_count) {
            if (current_time < user_AT[j])
                current_time = user_AT[j];

            CT[i + j] = current_time + user_BT[j];
            TAT[i + j] = CT[i + j] - user_AT[j];
            WT[i + j] = TAT[i + j] - user_BT[j];

            current_time = CT[i + j];

             printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\tUser\n", i + 1 + j,
user_AT[j], user_BT[j], CT[i + j], TAT[i + j], WT[i + j]);
            total_wt += WT[i + j];
            total_tat += TAT[i + j];

            j++;
        }
    }

    float avg_wt = (float)total_wt / n;
    float avg_tat = (float)total_tat / n;

    printf("\nAverage waiting time = %0.3f", avg_wt);
    printf("\nAverage turn around time = %0.3f\n", avg_tat);

    return 0;
}
```

Output:



```
Enter the number of processes: 3
Enter the arrival time, burst time and type (1 for system process, 0 for user process) for all the processes:

Process 1:
Arrival time: 2
Burst Time: 1
Type (1 for system, 0 for user): 1

Process 2:
Arrival time: 1
Burst Time: 5
Type (1 for system, 0 for user): 0

Process 3:
Arrival time: 4
Burst Time: 1
Type (1 for system, 0 for user): 1


Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time    Type
1       1               5               6               5               0               User
1       2               1               7               5               4               System
2       4               1               8               4               3               System

Average waiting time = 2.333
Average turn around time = 4.667
```

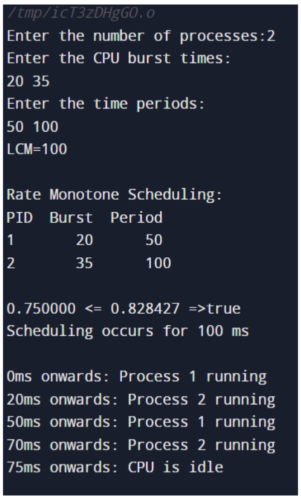
Program 4:

Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) **Rate- Monotonic**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort (int proc[], int b[], int pt[], int n)
{
  int temp = 0;
  for (int i = 0; i < n; i++)
      {
        for (int j = i; j < n; j++)
            {
              if (pt[j] < pt[i])
                  {
                    temp = pt[i];
                    pt[i] = pt[j];
                    pt[j] = temp;
                    temp = b[j];
                    b[j] = b[i];
                    b[i] = temp;
                    temp = proc[i];
                    proc[i] = proc[j];
                    proc[j] = temp;
                  }
            }
      }
}
int gcd (int a, int b)
{
  int r;
  while (b > 0)
      {
        r = a % b;
        a = b;
        b = r;
      }
  return a;
}
int lcmul (int p[], int n)
{
  int lcm = p[0];
  for (int i = 1; i < n; i++)
      {
        lcm = (lcm * p[i]) / gcd (lcm, p[i]);
      }
  return lcm;
}
void main ()
{
  int n;
  printf ("Enter the number of processes:");
  scanf ("%d", &n);
  int proc[n], b[n], pt[n], rem[n];
  printf ("Enter the CPU burst times:\n");
  for (int i = 0; i < n; i++)
      {
        scanf ("%d", &b[i]);
        rem[i] = b[i];
      }
  printf ("Enter the time periods:\n");
  for (int i = 0; i < n; i++)
      scanf ("%d", &pt[i]);
  for (int i = 0; i < n; i++)
      proc[i] = i + 1;

  sort (proc, b, pt, n);
  //LCM
  int l = lcmul (pt, n);
  printf ("LCM=%d\n", l);
  printf ("\nRate Monotone Scheduling:\n");
  printf ("PID\t Burst\tPeriod\n");
  for (int i = 0; i < n; i++)
      printf ("%d\t\t%d\t\t%d\n", proc[i], b[i], pt[i]);
  //feasibility
double sum = 0.0;
 for (int i = 0; i < n; i++)
      {
        sum += (double) b[i] / pt[i];
      }
  double rhs = n * (pow (2.0, (1.0 / n)) - 1.0);
   printf ("\n%lf  <=  %lf  =>%s\n", sum, rhs, (sum  <=  rhs) ? "true" :
"false");
  if (sum > rhs)
      exit (0);

  printf ("Scheduling occurs for %d ms\n\n", l);
//RMS
  int time = 0, prev = 0, x = 0;
  while (time < l)
      {
        int f = 0;
        for (int i = 0; i < n; i++)
            {
              if (time % pt[i] == 0)
                  rem[i] = b[i];
              if (rem[i] > 0)
                  {
                    if (prev != proc[i])
                        {
                              printf   ("%dms   onwards:   Process   %d
running\n", time,
                                      proc[i]);
                          prev = proc[i];
                        }
                    rem[i]--;
                    f = 1;
                    break;
                    x = 0;
                  }
            }
        if (!f)
            {
              if (x != 1)
                  {
                    printf ("%dms onwards: CPU is idle\n", time);
                    x = 1;
                  }
            }
        time++;
      }
}
```

Output:



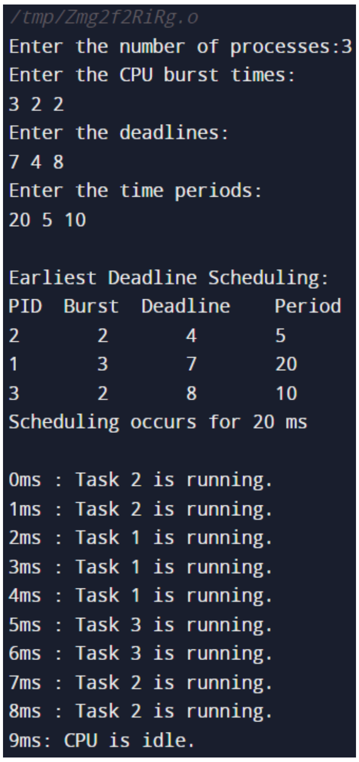
```
/tmp/icT3zDHgGO.o
Enter the number of processes:2
Enter the CPU burst times:
20 35
Enter the time periods:
50 100
LCM=100

Rate Monotone Scheduling:
PID  Burst  Period
1       20      50
2       35      100

0.750000 <= 0.828427 =>true
Scheduling occurs for 100 ms

0ms onwards: Process 1 running
20ms onwards: Process 2 running
50ms onwards: Process 1 running
70ms onwards: Process 2 running
75ms onwards: CPU is idle
```

b) Earliest-deadline First

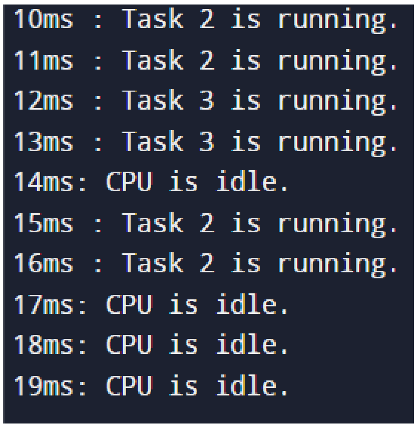
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void sort (int proc[], int d[], int b[], int pt[], int n)
{
  int temp = 0;
  for (int i = 0; i < n; i++)
      {
        for (int j = i; j < n; j++)
            {
              if (d[j] < d[i])
                  {
                    temp = d[j];d[j] = d[i];d[i] = temp;
                    temp = pt[i];pt[i] = pt[j];pt[j] = temp;
                    temp = b[j];b[j] = b[i];b[i] = temp;
                    temp = proc[i];proc[i] = proc[j];
                    proc[j] = temp;
                  }
            }
      }
}
int gcd (int a, int b)
{
  int r;
  while (b > 0)
      {
        r = a % b;
        a = b;
        b = r;
      }
  return a;
}
int lcmul (int p[], int n)
{
  int lcm = p[0];
  for (int i = 1; i < n; i++)
      {
        lcm = (lcm * p[i]) / gcd (lcm, p[i]);
      }
  return lcm;
}
void main ()
{
  int n;
  printf ("Enter the number of processes:");
  scanf ("%d", &n);
  int proc[n], b[n], pt[n], d[n], rem[n];
  printf ("Enter the CPU burst times:\n");
  for (int i = 0; i < n; i++)
      {
        scanf ("%d", &b[i]);
        rem[i] = b[i];
      }
  printf ("Enter the deadlines:\n");
  for (int i = 0; i < n; i++)
      scanf ("%d", &d[i]);
  printf ("Enter the time periods:\n");
  for (int i = 0; i < n; i++)
      scanf ("%d", &pt[i]);
  for (int i = 0; i < n; i++)
      proc[i] = i + 1;

  sort (proc, d, b, pt, n);
  //LCM
  int l = lcmul (pt, n);

  printf ("\nEarliest Deadline Scheduling:\n");
  printf ("PID\t Burst\tDeadline\tPeriod\n");
  for (int i = 0; i < n; i++)
      printf ("%d\t\t%d\t\t%d\t\t%d\n", proc[i], b[i], d[i], pt[i]);

  printf ("Scheduling occurs for %d ms\n\n", l);
 //EDF
  int time = 0, prev = 0, x = 0;
```

```
    int nextDeadlines[n];
    for (int i = 0; i < n; i++)
        {
          nextDeadlines[i] = d[i];
          rem[i] = b[i];
        }
    while (time < l)
        {
          for (int i = 0; i < n; i++)
            {
              if (time % pt[i] == 0 && time != 0)
                {
                  nextDeadlines[i] = time + d[i];
                  rem[i] = b[i];
                }
            }
          int minDeadline = l + 1;
          int taskToExecute = -1;
          for (int i = 0; i < n; i++)
            {
              if (rem[i] > 0 && nextDeadlines[i] < minDeadline)
                {
                  minDeadline = nextDeadlines[i];
                  taskToExecute = i;
                }
            }
          if (taskToExecute != -1)
            {
                printf  ("%dms  :  Task  %d  is  running.\n",  time,
proc[taskToExecute]);
              rem[taskToExecute]--;
            }
          else
            {
              printf ("%dms: CPU is idle.\n", time);
            }

          time++;
        }
}
```

Output:



```
/tmp/Zmg2f2RiRg.o
Enter the number of processes:3
Enter the CPU burst times:
3 2 2
Enter the deadlines:
7 4 8
Enter the time periods:
20 5 10

Earliest Deadline Scheduling:
PID  Burst  Deadline   Period
2      2       4        5
1      3       7        20
3      2       8        10
Scheduling occurs for 20 ms

0ms : Task 2 is running.
1ms : Task 2 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 3 is running.
6ms : Task 3 is running.
7ms : Task 2 is running.
8ms : Task 2 is running.
9ms: CPU is idle.
```

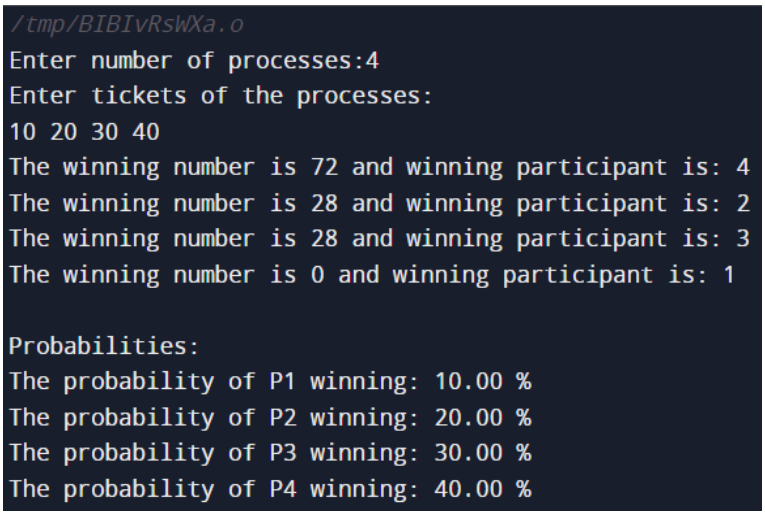


```
10ms : Task 2 is running.
11ms : Task 2 is running.
12ms : Task 3 is running.
13ms : Task 3 is running.
14ms: CPU is idle.
15ms : Task 2 is running.
16ms : Task 2 is running.
17ms: CPU is idle.
18ms: CPU is idle.
19ms: CPU is idle.
```

c) Proportional scheduling

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL));
    int n;
    printf("Enter number of processes:");
    scanf("%d",&n);
    int p[n],t[n],cum[n],m[n];int c=0;int total = 0,count=0;
    printf("Enter tickets of the processes:\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&t[i]);
        c+=t[i];
        cum[i]=c;
        p[i]=i+1;
        m[i]=0;
        total+= t[i];
    }
    while(count<n)
    {
        int wt=rand()%total;
        for (int i=0;i<n;i++)
        {
            if (wt<cum[i] && m[i]==0)
            {
                       printf("The winning number is %d and winning
participant is: %d\n",wt,p[i]);
                m[i]=1;count++;
            }
        }
    }
    printf("\nProbabilities:\n");
    for (int i = 0; i < n; i++)
    {
                    printf("The probability of P%d winning: %.2f
%\n",p[i],((double)t[i]/total*100));
    }
}
```

Output:



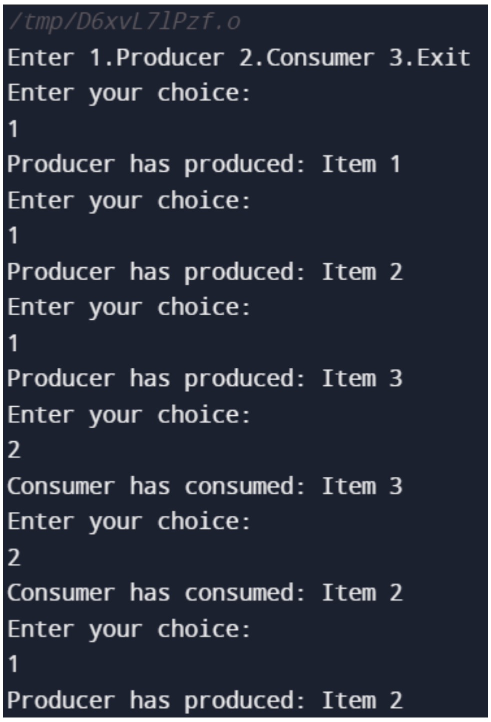
```
/tmp/BIBIvRsWXa.o
Enter number of processes:4
Enter tickets of the processes:
10 20 30 40
The winning number is 72 and winning participant is: 4
The winning number is 28 and winning participant is: 2
The winning number is 28 and winning participant is: 3
The winning number is 0 and winning participant is: 1

Probabilities:
The probability of P1 winning: 10.00 %
The probability of P2 winning: 20.00 %
The probability of P3 winning: 30.00 %
The probability of P4 winning: 40.00 %
```

Program 5:

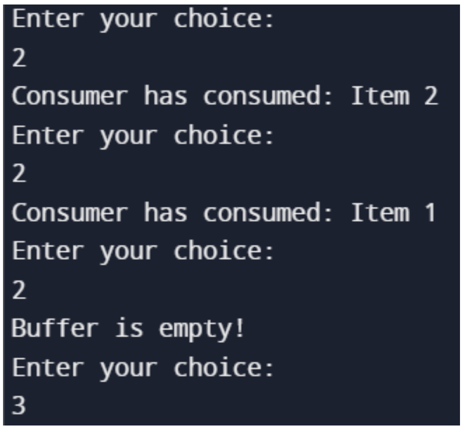
Write a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=5,x=0;
void wait(int *s) {
    while(*s <= 0);
    (*s)--;
}

void signal(int *s) {
    (*s)++;
}
void producer()
{
    wait(&empty);
    wait(&mutex);
    x++;
    printf("Producer has produced: Item %d\n",x);
    signal(&mutex);
    signal(&full);
}
void consumer()
{
    wait(&full);
    wait(&mutex);
    printf("Consumer has consumed: Item %d\n",x);
    x--;
    signal(&mutex);
    signal(&empty);
}
void main()
{
    int ch;
    printf("Enter 1.Producer 2.Consumer 3.Exit\n");
    while(1)
    {
        printf("Enter your choice:\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                if(mutex==1 && empty!=0)
                producer();
                else
                    printf("Buffer is full!\n");
                break;
            case 2:
                if(mutex==1 && full!=0)
                consumer();
                else
                    printf("Buffer is empty!\n");
                break;
            case 3:exit(0);
            default:printf("Invalid choice!\n");
        }
    }
}
```

Output:



```
/tmp/D6xvL71Pzf.o
Enter 1.Producer 2.Consumer 3.Exit
Enter your choice:
1
Producer has produced: Item 1
Enter your choice:
1
Producer has produced: Item 2
Enter your choice:
1
Producer has produced: Item 3
Enter your choice:
2
Consumer has consumed: Item 3
Enter your choice:
2
Consumer has consumed: Item 2
Enter your choice:
1
Producer has produced: Item 2
```



```
Enter your choice:
2
Consumer has consumed: Item 2
Enter your choice:
2
Consumer has consumed: Item 1
Enter your choice:
2
Buffer is empty!
Enter your choice:
3
```

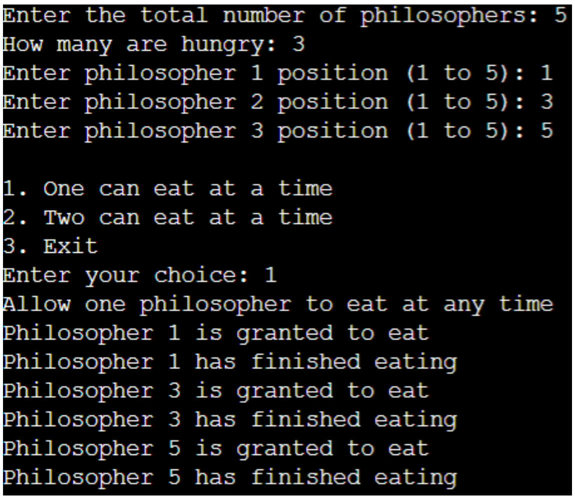
Program 6:

Write a C program to simulate the concept of Dining-Philosophers problem.

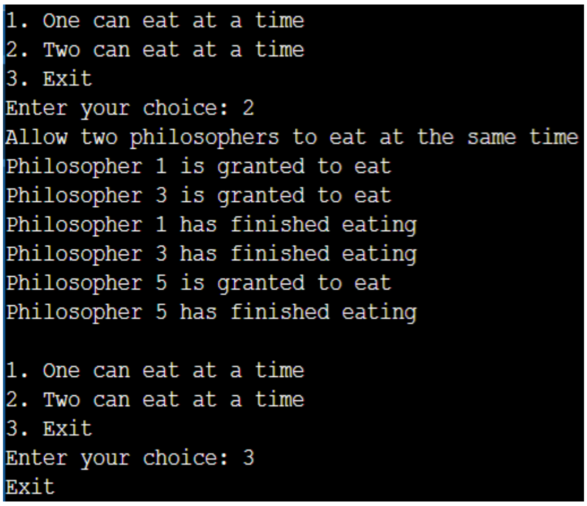
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#define MAX_PHILOSOPHERS 100
int mutex = 1;
int mutex2 = 2;
int philosophers[MAX_PHILOSOPHERS];
void wait(int *sem) {
    while (*sem <= 0);
    (*sem)--;
}
void signal(int *sem) {
    (*sem)++;
}void* one_eat_at_a_time(void* arg) {
    int philosopher = *((int*) arg);
   wait(&mutex);
    printf("Philosopher %d is granted to eat\n", philosopher + 1);
    sleep(1);
    printf("Philosopher %d has finished eating\n", philosopher + 1);
    signal(&mutex);
    return NULL;
}
void* two_eat_at_a_time(void* arg) {
    int philosopher = *((int*) arg);
   wait(&mutex2);
    printf("Philosopher %d is granted to eat\n", philosopher + 1);
    sleep(1);
    printf("Philosopher %d has finished eating\n", philosopher + 1);
    signal(&mutex2);

    return NULL;
}
int main() {
    int N;
    printf("Enter the total number of philosophers: ");
    scanf("%d", &N);
   int hungry_count;
    printf("How many are hungry: ");
    scanf("%d", &hungry_count);
   int hungry_philosophers[hungry_count];
```

```
  for (int i = 0; i < hungry_count; i++) {
    printf("Enter philosopher %d position (1 to %d): ", i + 1, N);
    scanf("%d", &hungry_philosophers[i]);
    hungry_philosophers[i]--;
  }
 pthread_t thread[hungry_count];
 int choice;
 do {
     printf("\n1. One can eat at a time\n2. Two can eat at a time\n3.
Exit\nEnter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
      case 1:
        printf("Allow one philosopher to eat at any time\n");
        for (int i = 0; i < hungry_count; i++) {
          philosophers[i] = hungry_philosophers[i];
            pthread_create(&thread[i], NULL, one_eat_at_a_time,
&philosophers[i]);
        }
        for (int i = 0; i < hungry_count; i++) {
          pthread_join(thread[i], NULL);
        }
        break;
   case 2:
        printf("Allow two philosophers to eat at the same time\n");
        for (int i = 0; i < hungry_count; i++) {
          philosophers[i] = hungry_philosophers[i];
            pthread_create(&thread[i], NULL, two_eat_at_a_time,
&philosophers[i]);
        }
        for (int i = 0; i < hungry_count; i++) {
          pthread_join(thread[i], NULL);
        }
        break;
      case 3:
        printf("Exit\n");
        break;
      default:
        printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 3);
 return 0;
}
```

Output:



```
Enter the total number of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position (1 to 5): 1
Enter philosopher 2 position (1 to 5): 3
Enter philosopher 3 position (1 to 5): 5

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosopher to eat at any time
Philosopher 1 is granted to eat
Philosopher 1 has finished eating
Philosopher 3 is granted to eat
Philosopher 3 has finished eating
Philosopher 5 is granted to eat
Philosopher 5 has finished eating
```



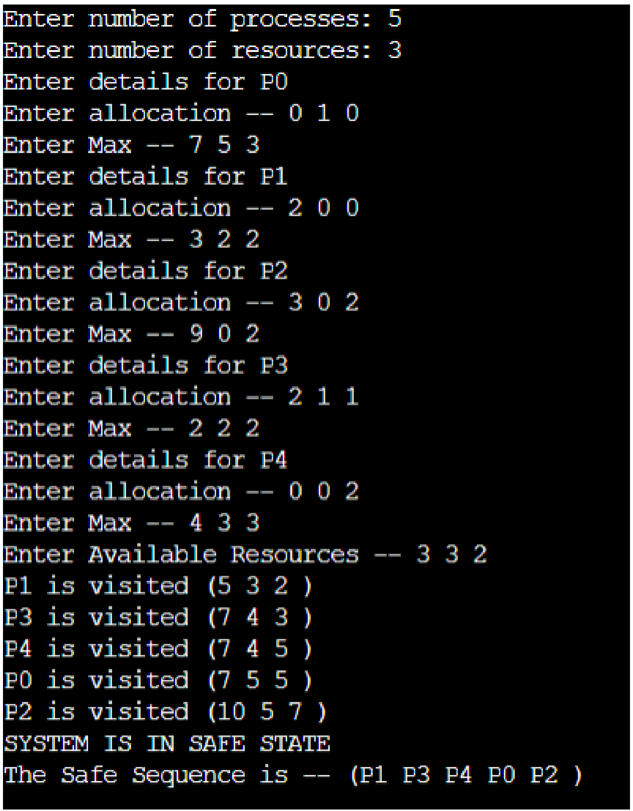
```
1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
Allow two philosophers to eat at the same time
Philosopher 1 is granted to eat
Philosopher 3 is granted to eat
Philosopher 1 has finished eating
Philosopher 3 has finished eating
Philosopher 5 is granted to eat
Philosopher 5 has finished eating

1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 3
Exit
```

Program 7:

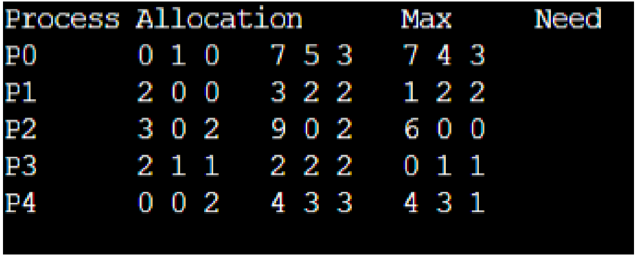
Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
#include <stdbool.h>

void calculateNeed(int P, int R, int need[P][R], int max[P][R], int
allot[P][R]) {
    for (int i = 0; i < P; i++)
        for (int j = 0; j < R; j++)
            need[i][j] = max[i][j] - allot[i][j];
}

bool isSafe(int P, int R, int processes[], int avail[], int max[][R], int
allot[][R]) {
    int need[P][R];
    calculateNeed(P, R, need, max, allot);

    bool finish[P];
    for (int i = 0; i < P; i++) {
        finish[i] = 0;
    }

    int safeSeq[P];
    int work[R];
    for (int i = 0; i < R; i++) {
        work[i] = avail[i];
    }

    int count = 0;
    while (count < P) {
        bool found = false;
        for (int p = 0; p < P; p++) {
            if (finish[p] == 0) {
                int j;
                for (j = 0; j < R; j++)
                    if (need[p][j] > work[j])
                        break;

                if (j == R) {
                    printf("P%d is visited (", p);
                    for (int k = 0; k < R; k++) {
                        work[k] += allot[p][k];
                        printf("%d ", work[k]);
                    }
                    printf(")\n");
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
                }
            }
        }

        if (found == false) {
            printf("System is not in safe state\n");
            return false;
        }
    }

    printf("SYSTEM IS IN SAFE STATE\nThe Safe Sequence is -- (");
    for (int i = 0; i < P; i++) {
        printf("P%d ", safeSeq[i]);
    }
    printf(")\n");

    return true;
}

int main() {
    int P, R;
    printf("Enter number of processes: ");
    scanf("%d", &P);
    printf("Enter number of resources: ");
    scanf("%d", &R);

    int processes[P];
    int avail[R];
    int max[P][R];
    int allot[P][R];

    for (int i = 0; i < P; i++) {
        processes[i] = i;
    }
    for (int i = 0; i < P; i++) {
        printf("Enter details for P%d\n", i);
        printf("Enter allocation -- ");
        for (int j = 0; j < R; j++) {
            scanf("%d", &allot[i][j]);
        }
        printf("Enter Max -- ");
        for (int j = 0; j < R; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter Available Resources -- ");
    for (int i = 0; i < R; i++) {
        scanf("%d", &avail[i]);
    }

    isSafe(P, R, processes, avail, max, allot);

    printf("\nProcess\tAllocation\tMax\tNeed\n");
    for (int i = 0; i < P; i++) {
        printf("P%d\t", i);
        for (int j = 0; j < R; j++) {
            printf("%d ", allot[i][j]);
        }
        printf("\t");
        for (int j = 0; j < R; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t");
        for (int j = 0; j < R; j++) {
            printf("%d ", max[i][j] - allot[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:



```
Enter number of processes: 5
Enter number of resources: 3
Enter details for P0
Enter allocation -- 0 1 0
Enter Max -- 7 5 3
Enter details for P1
Enter allocation -- 2 0 0
Enter Max -- 3 2 2
Enter details for P2
Enter allocation -- 3 0 2
Enter Max -- 9 0 2
Enter details for P3
Enter allocation -- 2 1 1
Enter Max -- 2 2 2
Enter details for P4
Enter allocation -- 0 0 2
Enter Max -- 4 3 3
Enter Available Resources -- 3 3 2
P1 is visited (5 3 2 )
P3 is visited (7 4 3 )
P4 is visited (7 4 5 )
P0 is visited (7 5 5 )
P2 is visited (10 5 7 )
SYSTEM IS IN SAFE STATE
The Safe Sequence is -- (P1 P3 P4 P0 P2 )
```



```
Process Allocation       Max     Need
P0       0 1 0    7 5 3   7 4 3
P1       2 0 0    3 2 2   1 2 2
P2       3 0 2    9 0 2   6 0 0
P3       2 1 1    2 2 2   0 1 1
P4       0 0 2    4 3 3   4 3 1
```

Program 8:

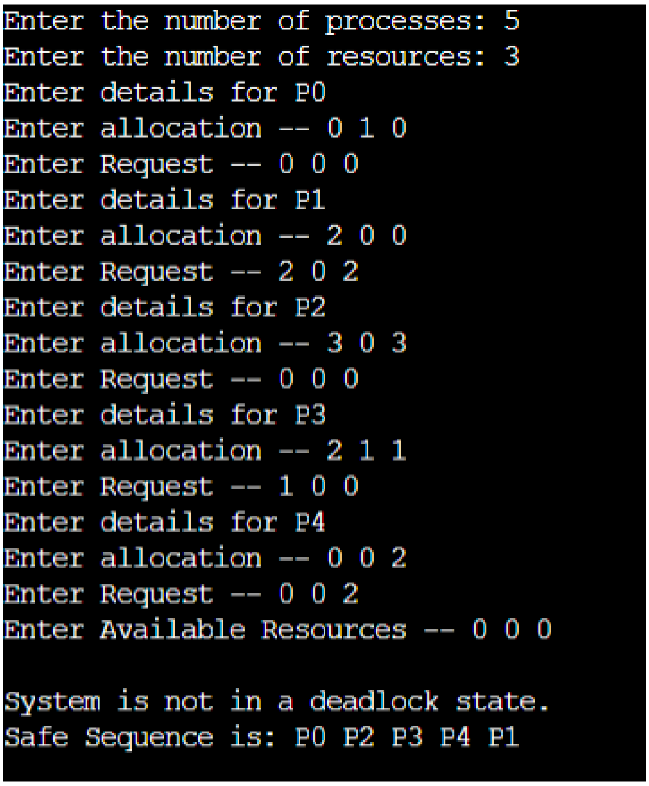
Write a C program to simulate deadlock detection.

```
#include <stdio.h>

int main() {
    int n, m, i, j, k;

    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], request[n][m], avail[m];

    for (int i = 0; i < n; i++) {
        printf("Enter details for P%d\n", i);
        printf("Enter allocation -- ");
        for (int j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
        printf("Enter Request -- ");
        for (int j = 0; j < m; j++) {
            scanf("%d", &request[i][j]);
        }
    }

    printf("Enter Available Resources -- ");
    for (int i = 0; i < m; i++) {
        scanf("%d", &avail[i]);
    }

    int finish[n], safeSeq[n], work[m], flag,f=0;
    for (i = 0; i < n; i++) {
        finish[i] = 0;
    }

    for (j = 0; j < m; j++) {
        work[j] = avail[j];
    }

    int count = 0;
    while (count < n) {
```

```
    flag = 0;f=0;
    for (i = 0; i < n; i++) {
        if (finish[i] == 0)
        {
            for(int j=0;j<m;j++)
                if(alloc[i][j]!=0)
                    f=1;
            if(f)
            {
                int canProceed = 1;
                for (j = 0; j < m; j++) {
                    if (request[i][j] > work[j]) {
                        canProceed = 0;
                        break;
                    }
                }
                if (canProceed) {
                    for (k = 0; k < m; k++) {
                        work[k] += alloc[i][k];
                    }
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    flag = 1;
                }
            }
            else
            {
                safeSeq[count++] = i;
                finish[i] = 1;
                flag = 1;
            }
        }
    }
    if (flag == 0) {
        break;
    }
}

int deadlock = 0;
for (i = 0; i < n; i++) {
    if (finish[i] == 0) {
        deadlock = 1;
        printf("\nSystem is in a deadlock state.\n");
        printf("The deadlocked processes are: ");
```

```
            for (j = 0; j < n; j++) {
                if (finish[j] == 0) {
                    printf("P%d ", j);
                }
            }
            printf("\n");
            break;
        }
    }

    if (deadlock == 0) {
        printf("\nSystem is not in a deadlock state.\n");
        printf("Safe Sequence is: ");
        for (i = 0; i < n; i++) {
            printf("P%d ", safeSeq[i]);
        }
        printf("\n");
    }

    return 0;
}
```

Output:



```
Enter the number of processes: 5
Enter the number of resources: 3
Enter details for P0
Enter allocation -- 0 1 0
Enter Request -- 0 0 0
Enter details for P1
Enter allocation -- 2 0 0
Enter Request -- 2 0 2
Enter details for P2
Enter allocation -- 3 0 3
Enter Request -- 0 0 0
Enter details for P3
Enter allocation -- 2 1 1
Enter Request -- 1 0 0
Enter details for P4
Enter allocation -- 0 0 2
Enter Request -- 0 0 2
Enter Available Resources -- 0 0 0

System is not in a deadlock state.
Safe Sequence is: P0 P2 P3 P4 P1
```

Program 9:

Write a C program to simulate the following contiguous memory allocation techniques

a) Worst-fit

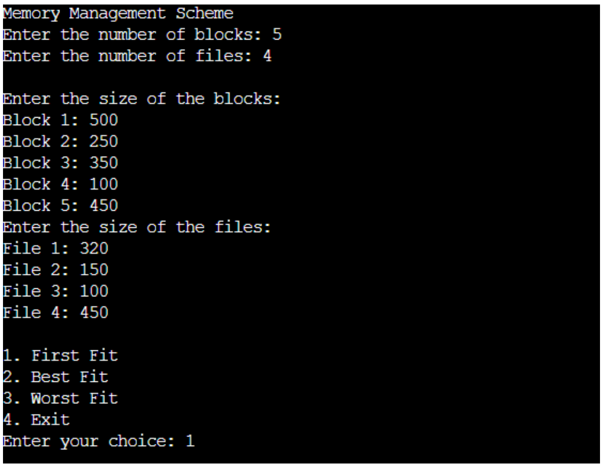
b) Best-fit

c) First-fit

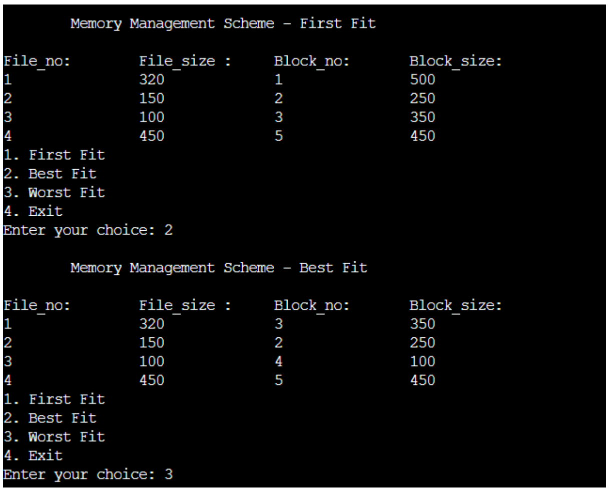
```
#include <stdio.h>
#include <stdlib.h>

#define MAX 25

void firstFit(int nb, int nf, int b[], int f[]) {
    int ff[MAX] = {0};
    int allocated[MAX] = {0};
    for (int i = 0; i < nf; i++) {
        ff[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (allocated[j] == 0 && b[j] >= f[i]) {
                ff[i] = j;
                allocated[j] = 1;
                break;
            }
        }
    }

    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:");
    for (int i = 0; i < nf; i++) {
        if (ff[i] != -1)
                printf("\n%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]]);
        else
            printf("\n%d\t\t%d\t\t-\t\t-", i + 1, f[i]);
    }
}

void bestFit(int nb, int nf, int b[], int f[]) {
    int ff[MAX] = {0};
    int allocated[MAX] = {0};
    for (int i = 0; i < nf; i++) {
        int best = -1;
        ff[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (allocated[j] == 0 && b[j] >= f[i]) {
                if (best == -1 || b[j] < b[best])
                    best = j;
            }
        }
        if (best != -1) {
            ff[i] = best;
            allocated[best] = 1;
        }
    }

    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:");
    for (int i = 0; i < nf; i++) {
        if (ff[i] != -1)
                printf("\n%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]]);
        else
            printf("\n%d\t\t%d\t\t-\t\t-", i + 1, f[i]);
    }
}

void worstFit(int nb, int nf, int b[], int f[]) {
    int ff[MAX] = {0};
    int allocated[MAX] = {0};

    for (int i = 0; i < nf; i++) {
        int worst = -1;
        ff[i] = -1;
        for (int j = 0; j < nb; j++) {
            if (allocated[j] == 0 && b[j] >= f[i]) {
                if (worst == -1 || b[j] > b[worst])
                    worst = j;
            }
        }
        if (worst != -1) {
            ff[i] = worst;
            allocated[worst] = 1;
        }
    }

    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:");
    for (int i = 0; i < nf; i++) {
        if (ff[i] != -1)
                printf("\n%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1,
b[ff[i]]);
        else
```

```
            printf("\n%d\t\t%d\t\t-\t\t-", i + 1, f[i]);
    }
}

int main() {
    int nb, nf, choice;

    printf("Memory Management Scheme");
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);
    int b[nb], f[nf];
    printf("\nEnter the size of the blocks:\n");
    for (int i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files:\n");
    for (int i = 0; i < nf; i++) {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }

    while (1) {
        printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("\n\tMemory Management Scheme - First Fit\n");
                firstFit(nb, nf, b, f);
                break;
            case 2:
                printf("\n\tMemory Management Scheme - Best Fit\n");
                bestFit(nb, nf, b, f);
                break;
            case 3:
                printf("\n\tMemory Management Scheme - Worst Fit\n");
                worstFit(nb, nf, b, f);
                break;
            case 4:
                printf("\nExiting...\n");
                exit(0);
                break;
            default:
                printf("\nInvalid choice.\n");
                break;
        }
    }

    return 0;
}
```

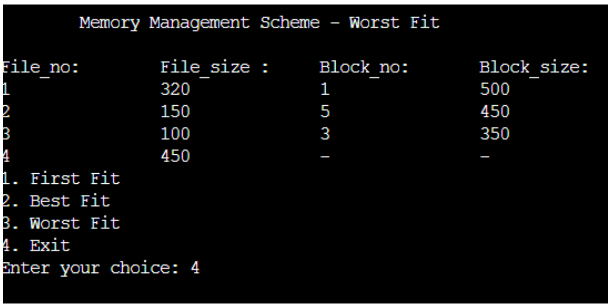
Output:



```
Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 500
Block 2: 250
Block 3: 350
Block 4: 100
Block 5: 450
Enter the size of the files:
File 1: 320
File 2: 150
File 3: 100
File 4: 450

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1
```



```
        Memory Management Scheme - First Fit

File_no:        File_size :     Block_no:       Block_size:
1               320             1               500
2               150             2               250
3               100             3               350
4               450             5               450
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

        Memory Management Scheme - Best Fit

File_no:        File_size :     Block_no:       Block_size:
1               320             3               350
2               150             2               250
3               100             4               100
4               450             5               450
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3
```



```
        Memory Management Scheme - Worst Fit

File_no:        File_size :     Block_no:       Block_size:
1               320             1               500
2               150             5               450
3               100             3               350
4               450             -               -
1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 4
```

Program 10:

Write a C program to simulate page replacement algorithms

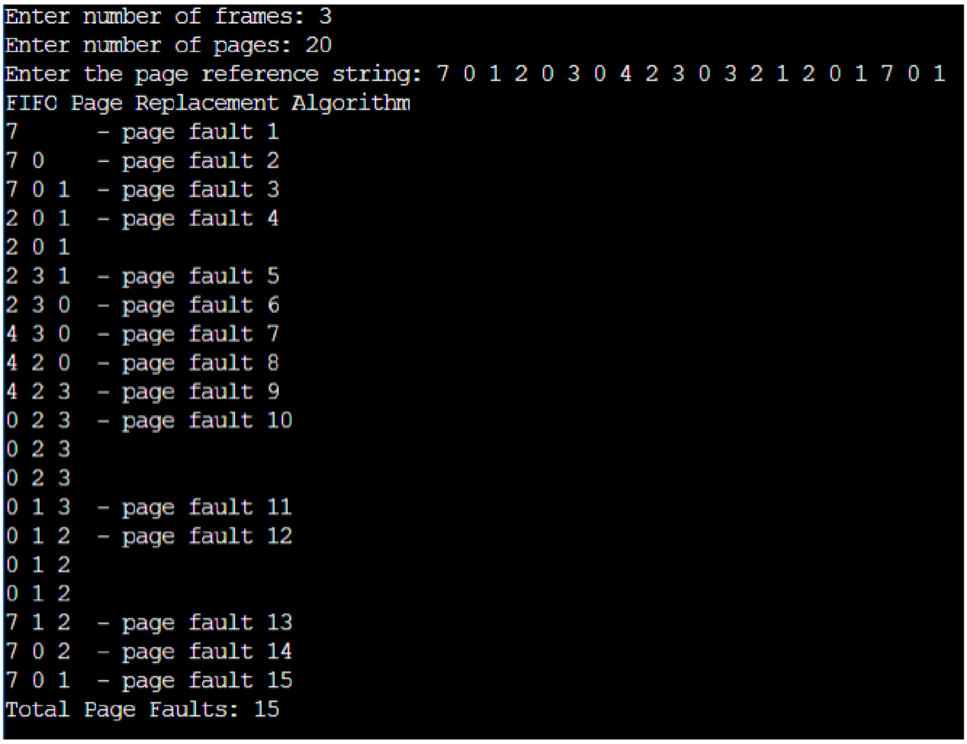
a) FIFO

b) LRU

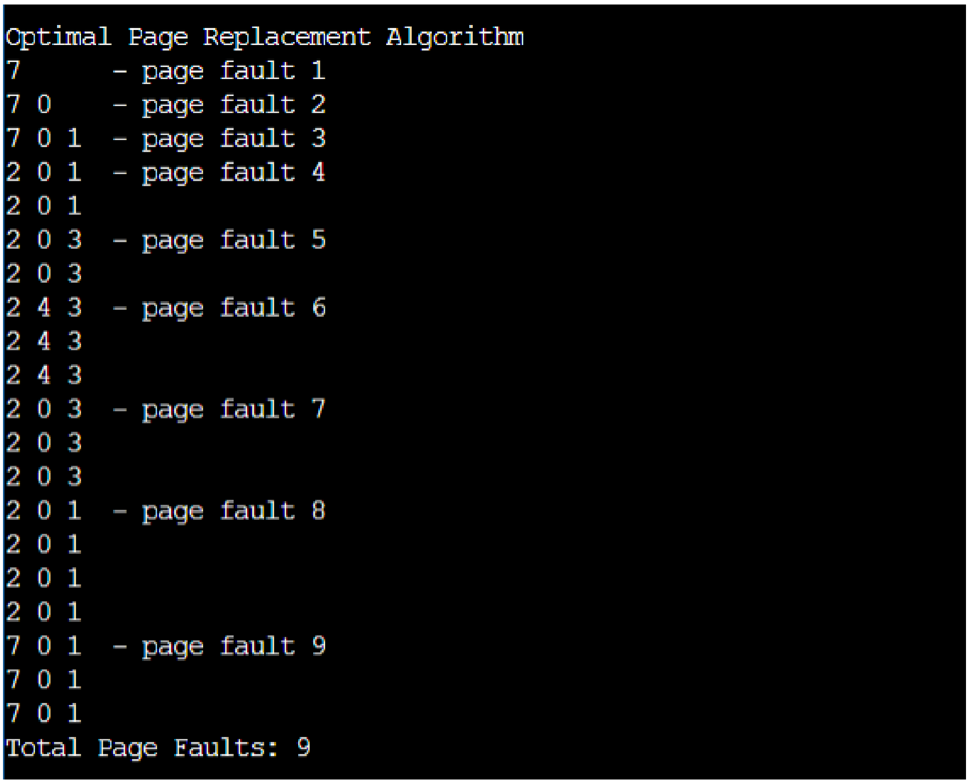
c) Optimal

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_FRAMES 10
#define MAX_PAGES 100
int x=0;
void printFrames(int frames[], int framesCount, bool fault) {

    for (int i = 0; i < framesCount; i++) {
        if (frames[i] == -1)
            printf("  ");
        else{
            printf("%d ", frames[i]);
            }
    }
    if (fault) printf(" - page fault %d",++x);
    else printf(" ");
    printf("\n");
}

int isPageInFrames(int page, int frames[], int framesCount) {
    for (int i = 0; i < framesCount; i++) {
        if (frames[i] == page) {
            return 1;
        }
    }
    return 0;
}

int getOptimalReplacementIndex(int pages[], int currentIndex, int frames[],
int framesCount, int pagesCount) {
    int farthest = currentIndex;
    int index = -1;

    for (int i = 0; i < framesCount; i++) {
        int j;
        for (j = currentIndex; j < pagesCount; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    index = i;
                }
                break;
            }
        }
        if (j == pagesCount) {
            return i;
        }
    }

    return index == -1 ? 0 : index;
}

void fifo(int pages[], int pagesCount, int framesCount) {
    x=0;
    printf("FIFO Page Replacement Algorithm\n");

    int frames[MAX_FRAMES];
    int currentFrame = 0;
    int pageFaults = 0;

    for (int i = 0; i < framesCount; i++) {
        frames[i] = -1;
    }

    for (int i = 0; i < pagesCount; i++) {
        bool fault = false;
        if (!isPageInFrames(pages[i], frames, framesCount)) {
            frames[currentFrame] = pages[i];
            currentFrame = (currentFrame + 1) % framesCount;
            fault = true;
            pageFaults++;
        }
        printFrames(frames, framesCount, fault);
    }

    printf("Total Page Faults: %d\n\n", pageFaults);
}

void optimal(int pages[], int pagesCount, int framesCount) {
    x=0;
    printf("Optimal Page Replacement Algorithm\n");

    int frames[MAX_FRAMES];
    int pageFaults = 0;

    for (int i = 0; i < framesCount; i++) {
        frames[i] = -1;
    }

    for (int i = 0; i < pagesCount; i++) {
        bool fault = false;
        if (!isPageInFrames(pages[i], frames, framesCount)) {
            if (frames[i % framesCount] == -1) {
                frames[i % framesCount] = pages[i];
            } else {
                      int index = getOptimalReplacementIndex(pages, i + 1,
frames, framesCount, pagesCount);
                frames[index] = pages[i];
            }
            fault = true;
            pageFaults++;
        }
        printFrames(frames, framesCount, fault);
    }

    printf("Total Page Faults: %d\n\n", pageFaults);
}

void lru(int pages[], int pagesCount, int framesCount) {
    x=0;
    printf("LRU Page Replacement Algorithm\n");

    int frames[MAX_FRAMES];
    int pageFaults = 0;
    int recent[MAX_FRAMES];

    for (int i = 0; i < framesCount; i++) {
        frames[i] = -1;
        recent[i] = -1;
    }

    for (int i = 0; i < pagesCount; i++) {
        bool fault = false;
        if (!isPageInFrames(pages[i], frames, framesCount)) {
            int lruIndex = 0;
            for (int j = 1; j < framesCount; j++) {
                if (recent[j] < recent[lruIndex]) {
                    lruIndex = j;
                }
            }
            frames[lruIndex] = pages[i];
            fault = true;
            pageFaults++;
        }
        for (int j = 0; j < framesCount; j++) {
            if (frames[j] == pages[i]) {
                recent[j] = i;
            }
        }
        printFrames(frames, framesCount, fault);
    }

    printf("Total Page Faults: %d\n\n", pageFaults);
}

int main() {
    int pages[MAX_PAGES];
    int pagesCount;
    int framesCount;

    printf("Enter number of frames: ");
    scanf("%d", &framesCount);

    printf("Enter number of pages: ");
    scanf("%d", &pagesCount);

    printf("Enter the page reference string: ");
    for (int i = 0; i < pagesCount; i++) {
        scanf("%d", &pages[i]);
    }

    fifo(pages, pagesCount, framesCount);
    optimal(pages, pagesCount, framesCount);
    lru(pages, pagesCount, framesCount);

    return 0;
}
```

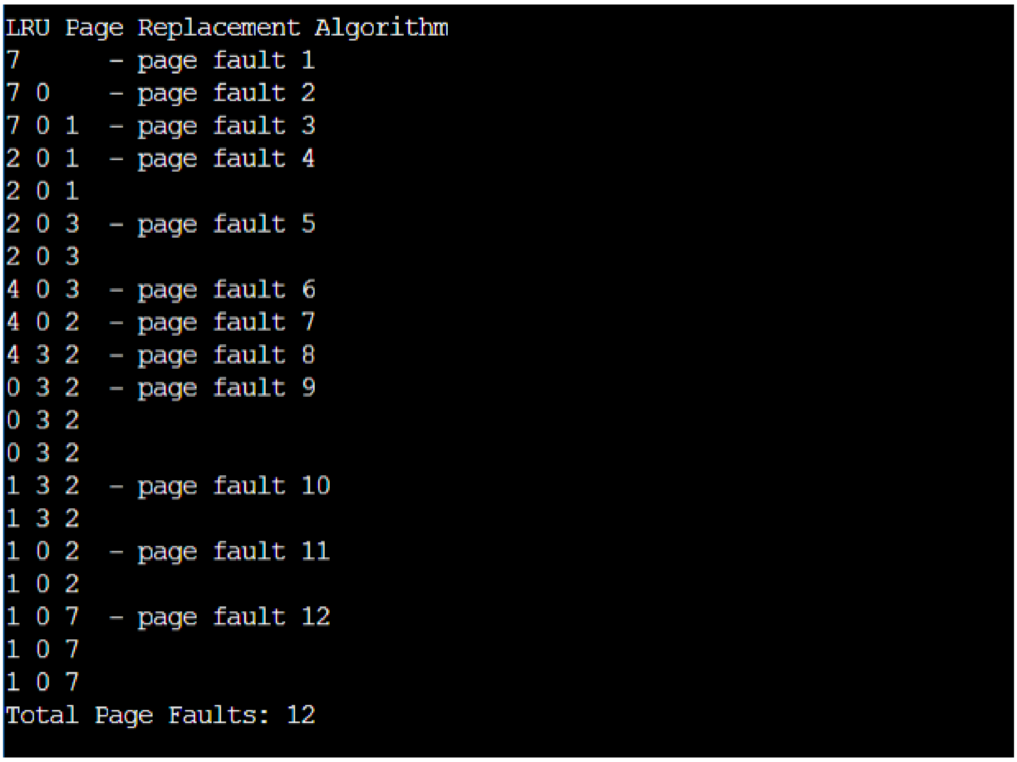
Output:



```
Enter number of frames: 3
Enter number of pages: 20
Enter the page reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
FIFO Page Replacement Algorithm
7       - page fault 1
7 0     - page fault 2
7 0 1   - page fault 3
2 0 1   - page fault 4
2 0 1
2 3 1   - page fault 5
2 3 0   - page fault 6
4 3 0   - page fault 7
4 2 0   - page fault 8
4 2 3   - page fault 9
0 2 3   - page fault 10
0 2 3
0 2 3
0 1 3   - page fault 11
0 1 2   - page fault 12
0 1 2
0 1 2
7 1 2   - page fault 13
7 0 2   - page fault 14
7 0 1   - page fault 15
Total Page Faults: 15
```



```
Optimal Page Replacement Algorithm
7       - page fault 1
7 0     - page fault 2
7 0 1   - page fault 3
2 0 1   - page fault 4
2 0 1
2 0 3   - page fault 5
2 0 3
2 4 3   - page fault 6
2 4 3
2 4 3
2 0 3   - page fault 7
2 0 3
2 0 3
2 0 1   - page fault 8
2 0 1
2 0 1
2 0 1
7 0 1   - page fault 9
7 0 1
7 0 1
Total Page Faults: 9
```



```
LRU Page Replacement Algorithm
7       - page fault 1
7 0     - page fault 2
7 0 1   - page fault 3
2 0 1   - page fault 4
2 0 1
2 0 3   - page fault 5
2 0 3
4 0 3   - page fault 6
4 0 2   - page fault 7
4 3 2   - page fault 8
0 3 2   - page fault 9
0 3 2
0 3 2
1 3 2   - page fault 10
1 3 2
1 0 2   - page fault 11
1 0 2
1 0 7   - page fault 12
1 0 7
1 0 7
Total Page Faults: 12
```