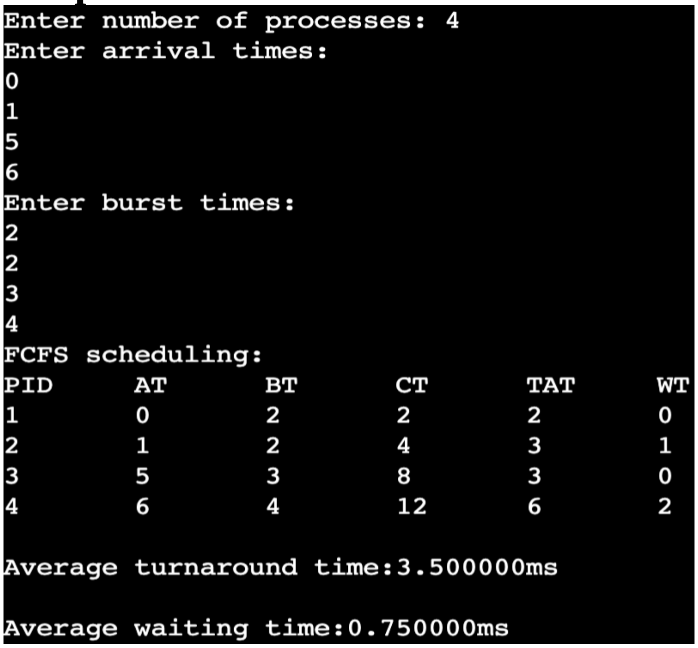
Program: 1
Question:
Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.
FCFS
Code:

```
#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:



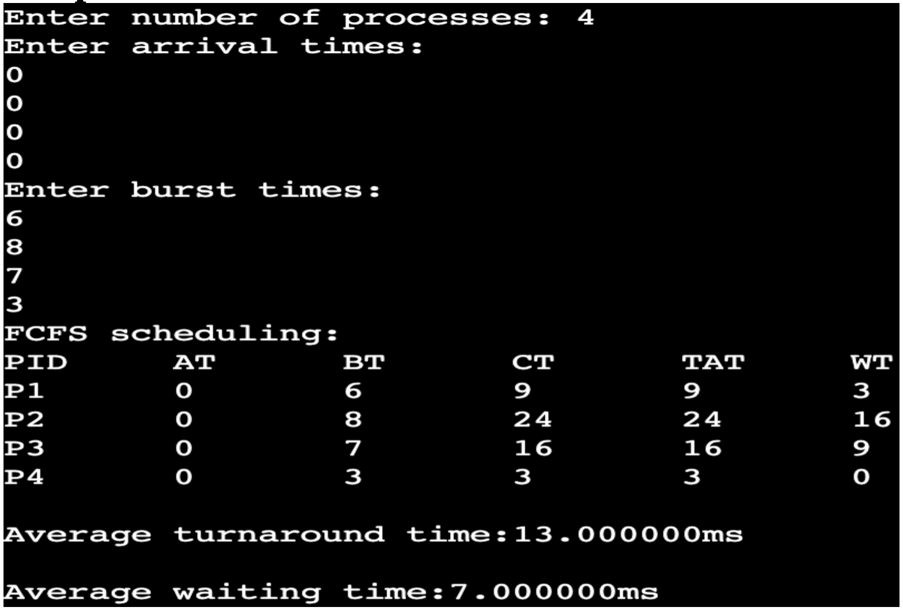
```
Enter number of processes: 4
Enter arrival times:
0
1
5
6
Enter burst times:
2
2
3
4
FCFS scheduling:
PID     AT      BT      CT      TAT     WT
1       0       2       2       2       0
2       1       2       4       3       1
3       5       3       8       3       0
4       6       4       12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

Average turnaround time:3.500000ms

Average waiting time:0.750000ms
```

SJF-Non Preemptive

Code:

```
#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;
        }
    }
    /*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("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:



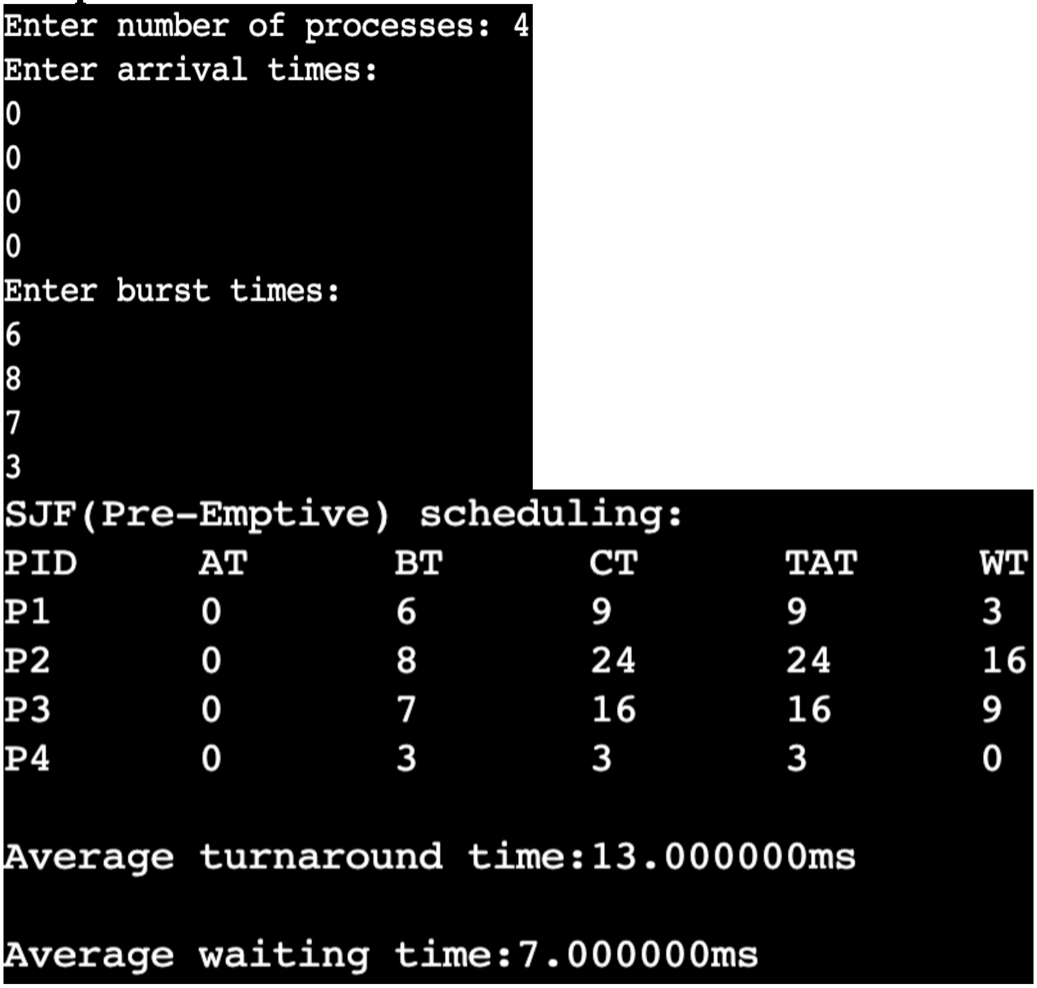
```
Enter number of processes: 4
Enter arrival times:
0
0
0
0
Enter burst times:
6
8
7
3
FCFS scheduling:
PID     AT      BT      CT      TAT     WT
P1      0       6       9       9       3
P2      0       8       24      24      16
P3      0       7       16      16      9
P4      0       3       3       3       0

Average turnaround time:13.000000ms

Average waiting time:7.000000ms
```

SJF Preemptive:

Code:

```
#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: 4
Enter arrival times:
0
0
0
0
Enter burst times:
6
8
7
3
SJF(Pre-Emptive) scheduling:
PID     AT      BT      CT      TAT     WT
P1      0       6       9       9       3
P2      0       8       24      24      16
P3      0       7       16      16      9
P4      0       3       3       3       0

Average turnaround time:13.000000ms

Average waiting time:7.000000ms
```

Program: 2

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

a) Priority (pre-emptive & Non-preemptive)

b) Round Robin

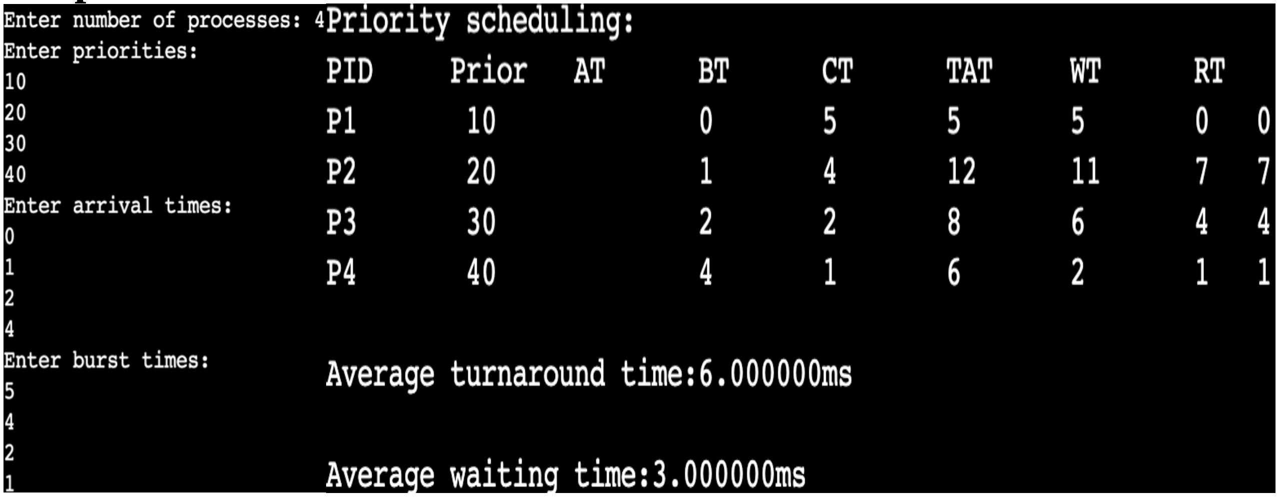
a) Priority Non-Preemptive:

CODE:

```
#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:



```
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
```

a)Priority (Preemptive):

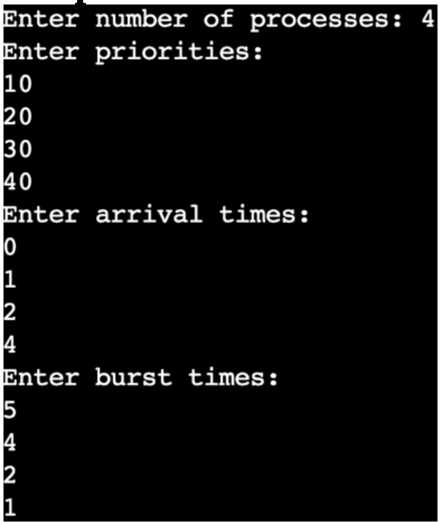
Code:

```
#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");
```

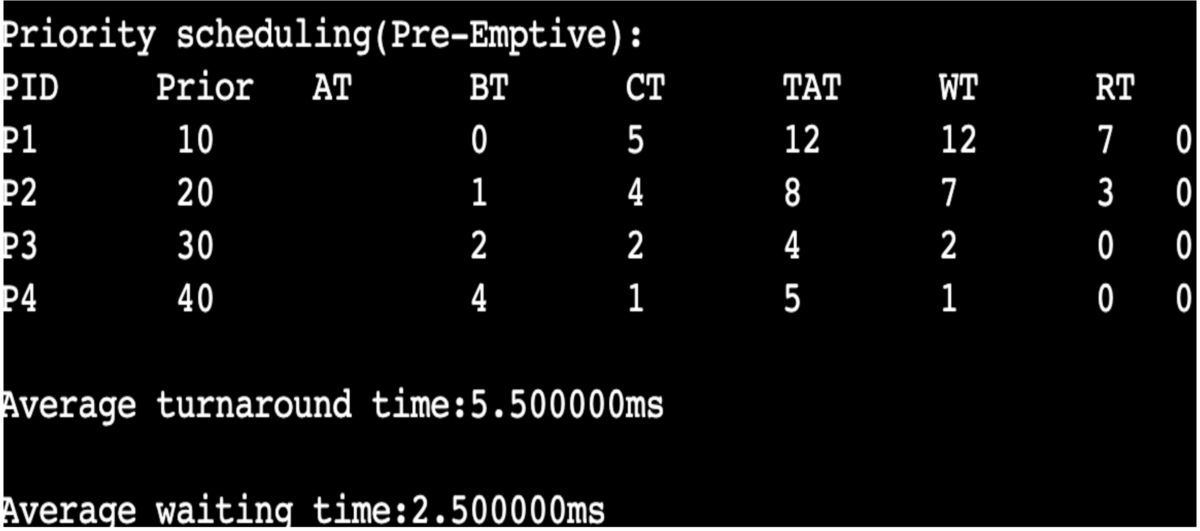
```
for (int i = 0; i < n; i++)
          scanf ("%d", &at[i]);
printf ("Enter burst times:\n");
for (int i = 0; i < n; i++)
          {
            scanf ("%d", &bt[i]);
            b[i] = bt[i];
            m[i] = -1;
            rt[i] = -1;
          }

sort (proc_id, p, at, bt, b, n);
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:



```
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
```

b) RoundRobin:

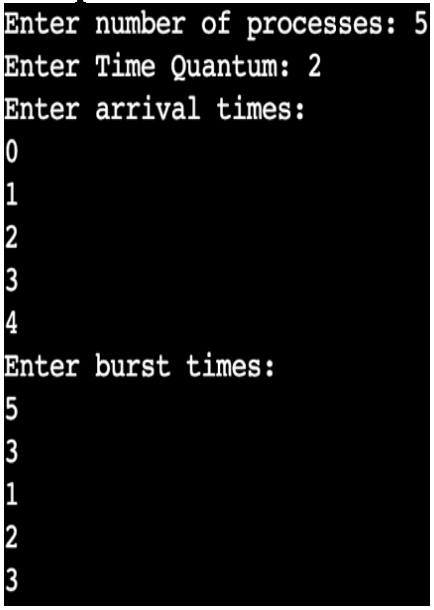
Code:

```
//RRS
#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++)
```

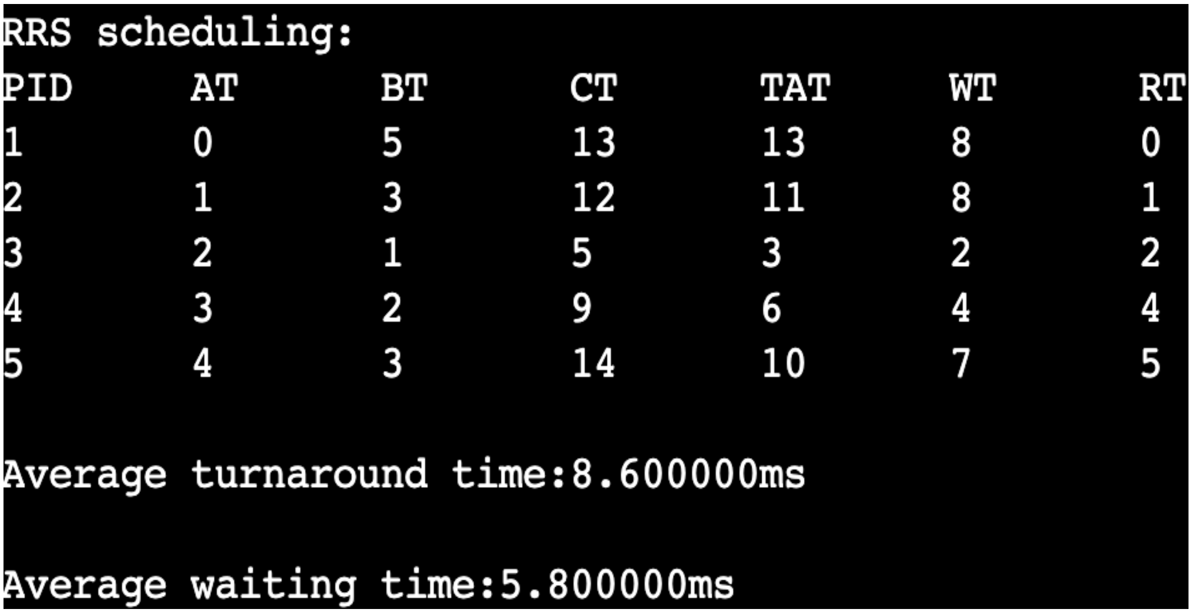
```
        scanf ("%d", &at[i]);
printf ("Enter burst times:\n");
for (int i = 0; i < n; i++)
        {
          scanf ("%d", &bt[i]);
          b[i] = bt[i];
          m[i] = 0;
          rt[i] = -1;
        }

sort (proc_id, at, bt, b, n);
f = r = 0;
q[0] = proc_id[0];
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;
             }
  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:



```
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
```

Program:3

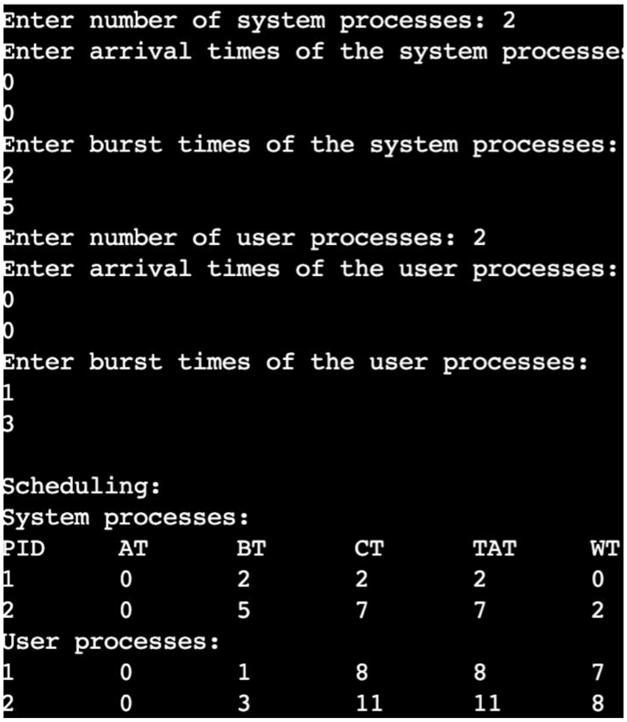
Question:

Write a C program to simulate 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.

Code:

```
#include<stdio.h>

void sort(int proc_id[],int at[],int bt[],int n)
{
    int temp=0;
    for(int i=0;i<n;i++)
    {
        for(int j=i;j<n;j++)
        {
            if(at[j]<at[i])
            {
                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 fcfs(int at[],int bt[],int ct[],int tat[],int wt[],int n,int *c)
{
    double ttat=0.0,twt=0.0;
    //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];
}

void main()
{
    int sn,un,c=0;int n=0;
    printf("Enter number of system processes: ");
    scanf("%d",&sn);n=sn;
    int sproc_id[n],sat[n],sbt[n],sct[n],stat[n],swt[n];
    for(int i=0;i<sn;i++)
        sproc_id[i]=i+1;
    printf("Enter arrival times of the system processes:\n");
    for(int i=0;i<sn;i++)
        scanf("%d",&sat[i]);
    printf("Enter burst times of the system processes:\n");
    for(int i=0;i<sn;i++)
        scanf("%d",&sbt[i]);

    printf("Enter number of user processes: ");
    scanf("%d",&un);n=un;
    int uproc_id[n],uat[n],ubt[n],uct[n],utat[n],uwt[n];
    for(int i=0;i<un;i++)
        uproc_id[i]=i+1;
    printf("Enter arrival times of the user processes:\n");
    for(int i=0;i<un;i++)
        scanf("%d",&uat[i]);
    printf("Enter burst times of the user processes:\n");
    for(int i=0;i<un;i++)
        scanf("%d",&ubt[i]);

    sort(sproc_id,sat,sbt,sn);
    sort(uproc_id,uat,ubt,un);

    fcfs(sat,sbt,sct,stat,swt,sn,&c);
    fcfs(uat,ubt,uct,utat,uwt,un,&c);

    printf("\nScheduling:\n");
    printf("System processes:\n");
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<sn;i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",sproc_id[i],sat[i],sbt[i],sct[i],stat[i],swt[i]);
    printf("User processes:\n");
    for(int i=0;i<un;i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",uproc_id[i],uat[i],ubt[i],uct[i],utat[i],uwt[i]);
}
```

Output:



```
Enter number of system processes: 2
Enter arrival times of the system processe
0
0
Enter burst times of the system processes:
2
5
Enter number of user processes: 2
Enter arrival times of the user processes:
0
0
Enter burst times of the user processes:
1
3

Scheduling:
System processes:
PID     AT      BT      CT      TAT     WT
1       0       2       2       2       0
2       0       5       7       7       2
User processes:
1       0       1       8       8       7
2       0       3       11      11      8
```

Program:4
Question:
Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First
c) Proportional scheduling

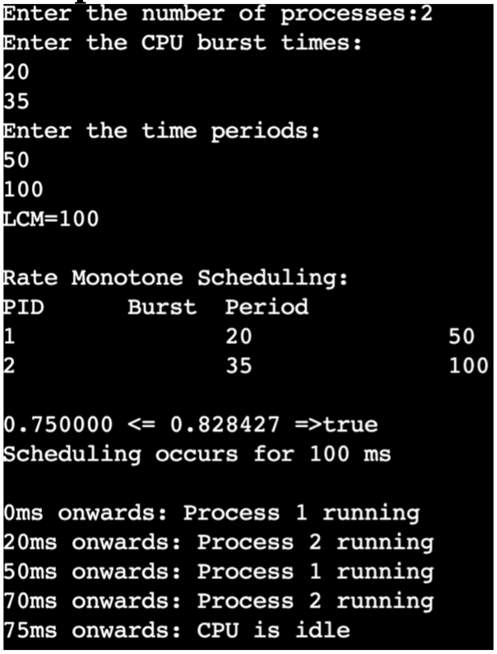
a) Rate-Monotonic:

Code:

```
#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



```
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:

Code:

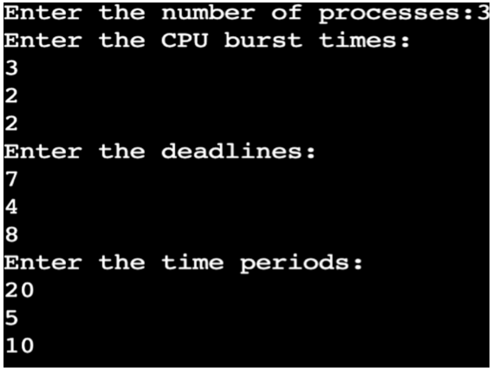
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void
sort (int proc[], int d[], int b[], int pt[], int n)
{
  int temp = 0;
  for (int i = 0; i < n; i++)
      {
        for (int j = i; j < n; j++)
                {
                  if (d[j] < d[i])
                          {
                            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++)
```

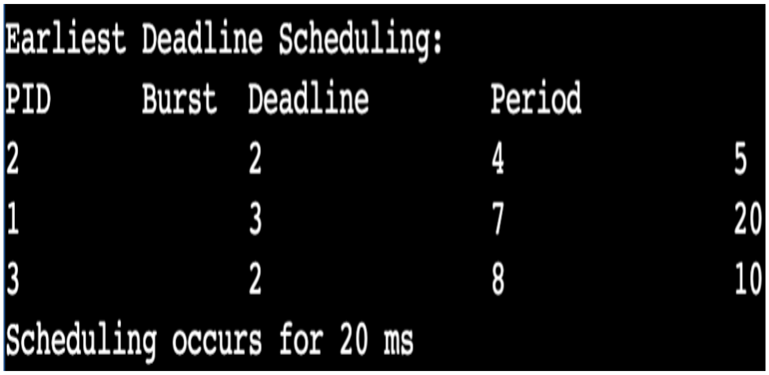
```
    scanf ("%d", &pt[i]);
for (int i = 0; i < n; i++)
    proc[i] = i + 1;

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++;
          }
    }
```

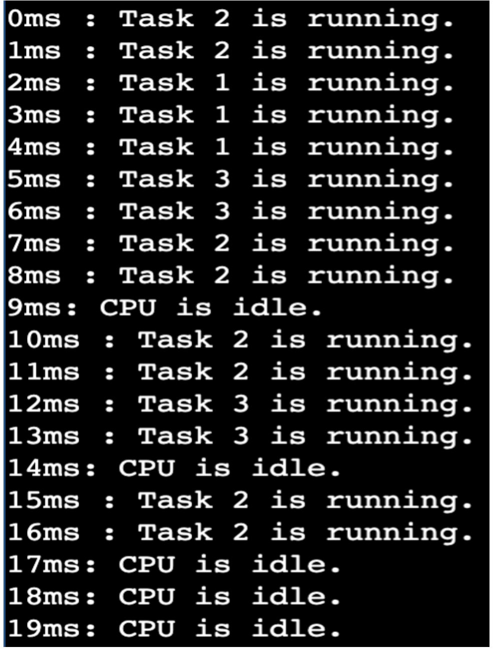
Output:



```
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
```



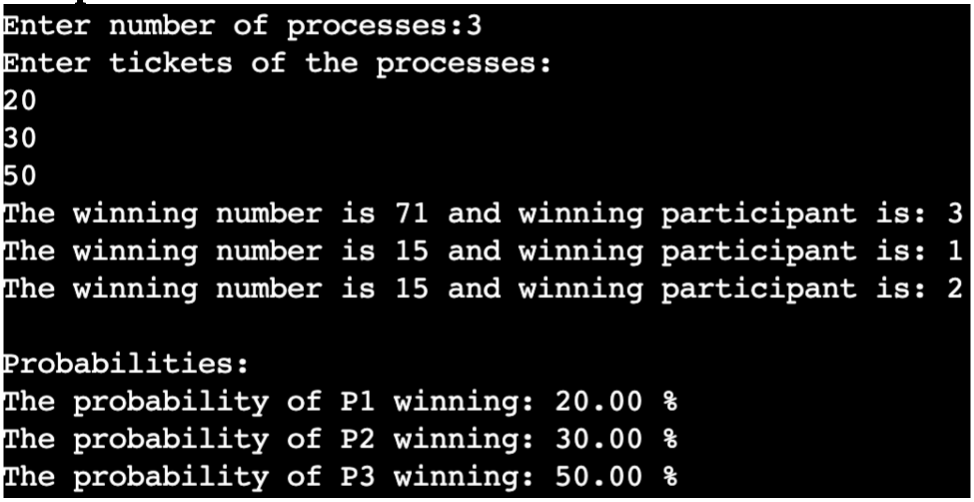
```
0ms  : Task 2 is running.
1ms  : Task 2 is running.
2ms  : Task 1 is running.
3ms  : Task 1 is running.
4ms  : Task 1 is running.
5ms  : Task 3 is running.
6ms  : Task 3 is running.
7ms  : Task 2 is running.
8ms  : Task 2 is running.
9ms: CPU is idle.
10ms  : Task 2 is running.
11ms  : Task 2 is 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

Code:

```
#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:

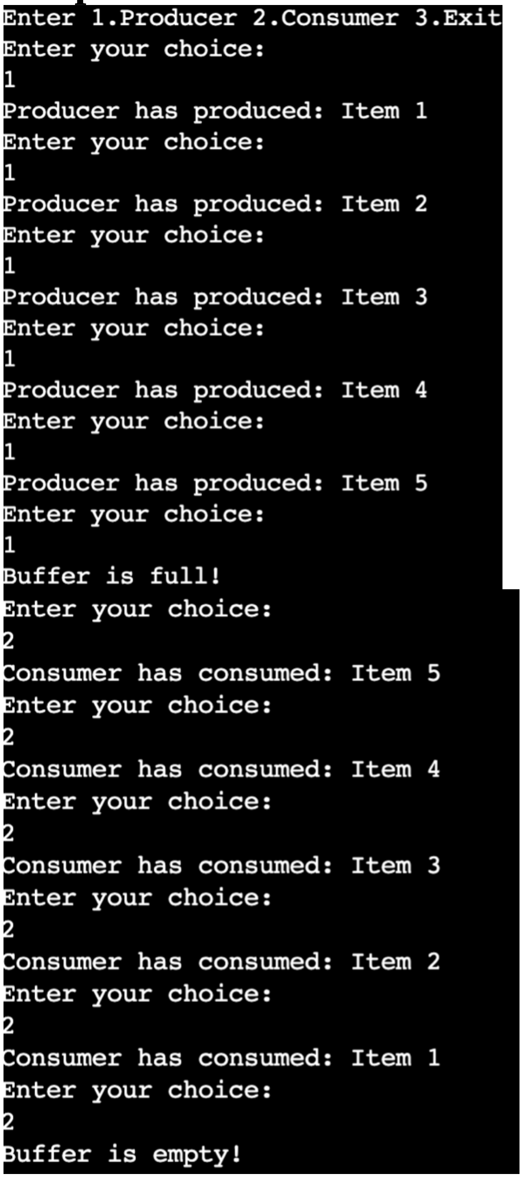


```
Enter number of processes:3
Enter tickets of the processes:
20
30
50
The winning number is 71 and winning participant is: 3
The winning number is 15 and winning participant is: 1
The winning number is 15 and winning participant is: 2

Probabilities:
The probability of P1 winning: 20.00 %
The probability of P2 winning: 30.00 %
The probability of P3 winning: 50.00 %
```

Program:5
Question:
Write a C program to simulate producer-consumer problem using semaphores.
Code:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=5,x=0;
void wait()
{
    --mutex;
}
void signal()
{
    ++mutex;
}
void producer()
{
    wait();++full;--empty;x++;
    printf("Producer has produced: Item %d\n",x);
    signal();
}
void consumer()
{
    wait();--full;++empty;
    printf("Consumer has consumed: Item %d\n",x);
    x--;signal();
}
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:



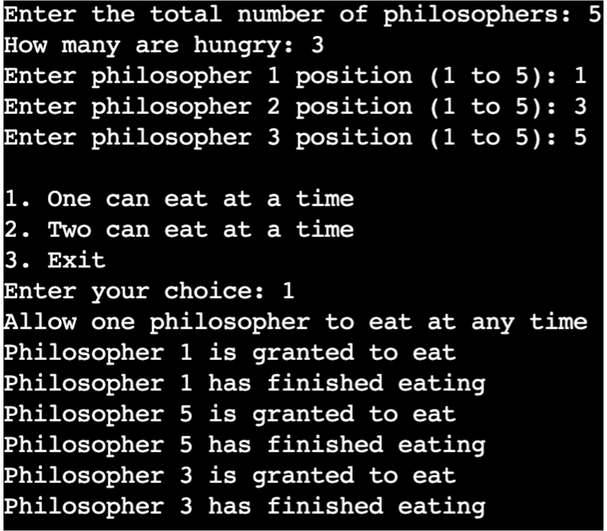
```
Enter 1.Producer 2.Consumer 3.Exit
Enter your choice:
1
Producer has produced: Item 1
Enter your choice:
1
Producer has produced: Item 2
Enter your choice:
1
Producer has produced: Item 3
Enter your choice:
1
Producer has produced: Item 4
Enter your choice:
1
Producer has produced: Item 5
Enter your choice:
1
Buffer is full!
Enter your choice:
2
Consumer has consumed: Item 5
Enter your choice:
2
Consumer has consumed: Item 4
Enter your choice:
2
Consumer has consumed: Item 3
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!
```

Program:6
Question:
Write a C program to simulate the concept of Dining-Philosophers problem.

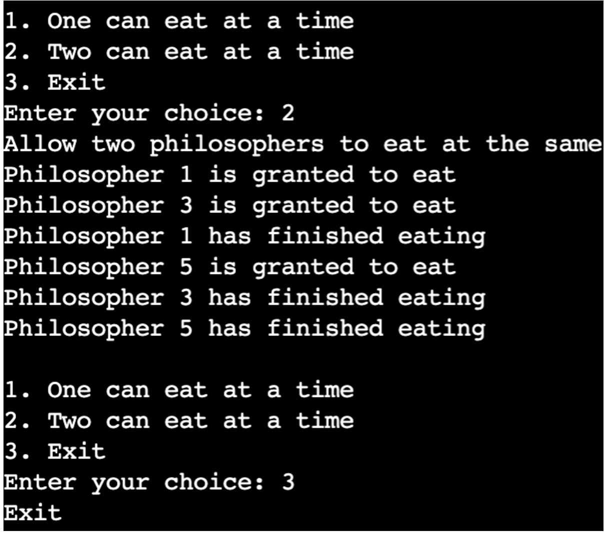
Code:

```
#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 5 is granted to eat
Philosopher 5 has finished eating
Philosopher 3 is granted to eat
Philosopher 3 has finished eating
```

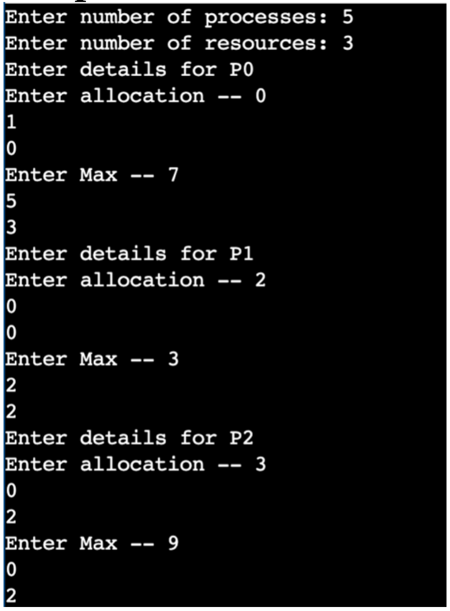


```
1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
Allow two philosophers to eat at the same
Philosopher 1 is granted to eat
Philosopher 3 is granted to eat
Philosopher 1 has finished eating
Philosopher 5 is granted to eat
Philosopher 3 has finished eating
Philosopher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

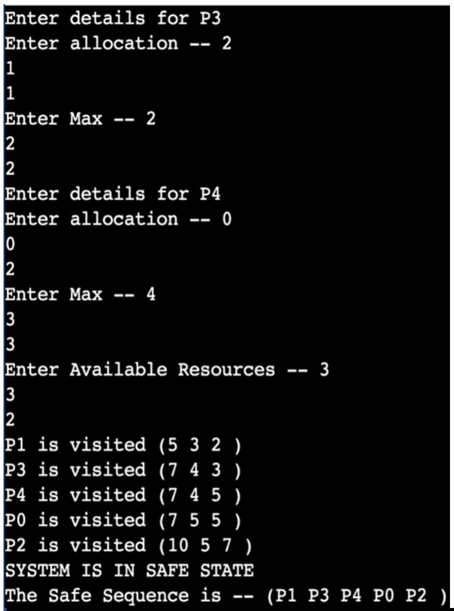
Program:7
Question:
Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.
Code:

```
#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;
}
```

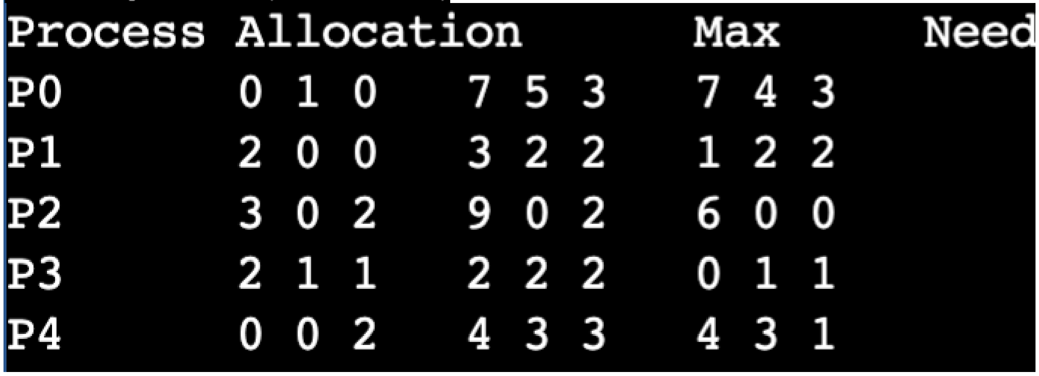
Output:



```
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 )
```

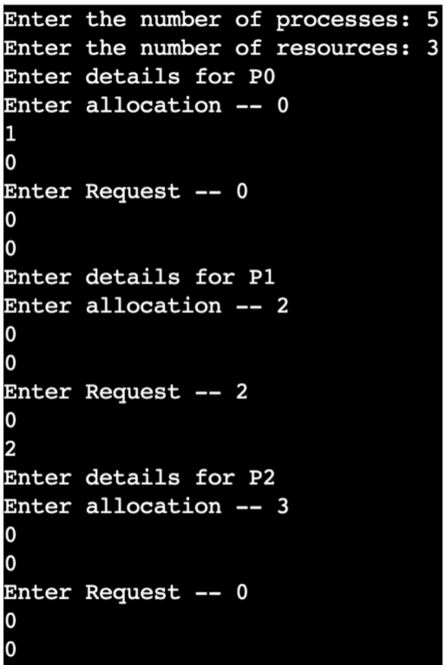


Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

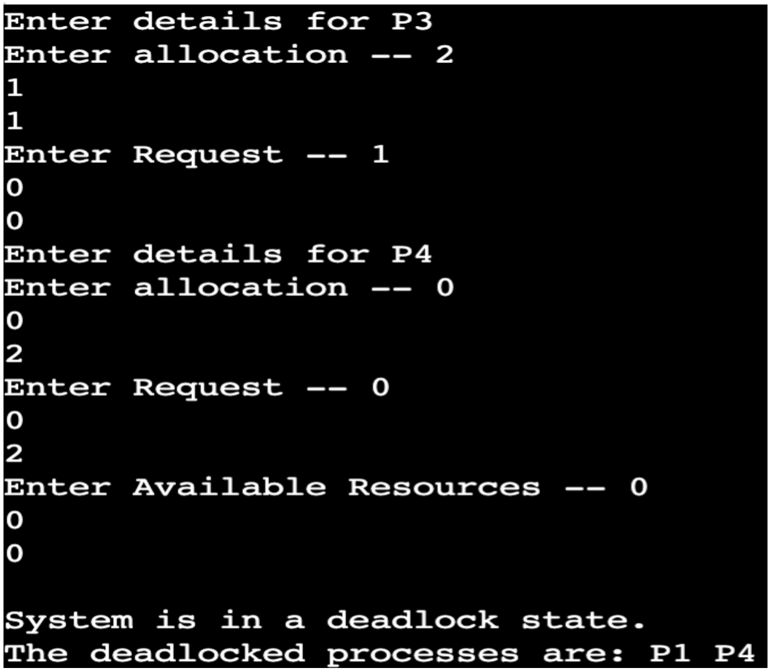
Program:8
Question:
Write a C program to simulate deadlock detection.
Code:

```
#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 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
0
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 in a deadlock state.
The deadlocked processes are: P1 P4
```

Program:9

Question:

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

a) Worst-fit

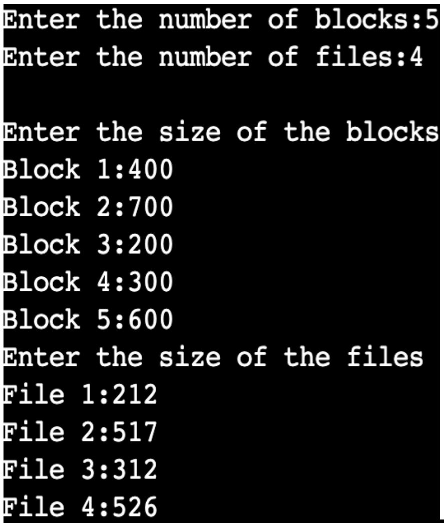
b) Best-fit

c) First-fit

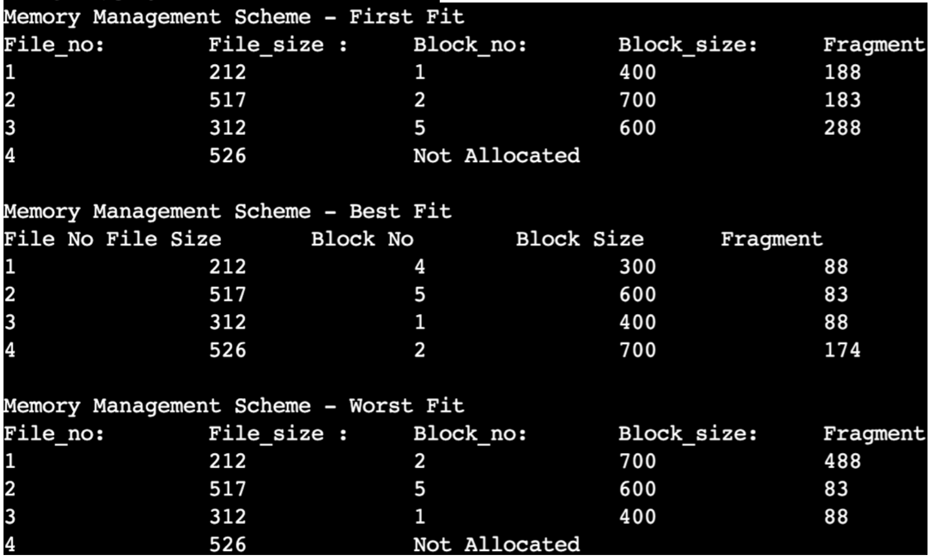
Code:

```
#include <stdio.h>

#define MAX 25

void firstFit(int nb, int nf, int b[], int f[]) {
    int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
    int i, j, temp;

    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0) {
                    ff[i] = j;
                    frag[i] = temp;
                    bf[j] = 1;
                    break;
                }
            }
        }
    }

    printf("\nMemory Management Scheme - First Fit\n");
    printf("File_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment\n");
    for (i = 1; i <= nf; i++) {
        printf("%d\t\t%d\t\t", i, f[i]);
        if (ff[i] != 0) {
            printf("%d\t\t%d\t\t%d\n", ff[i], b[ff[i]], frag[i]);
        } else {
            printf("Not Allocated\n");
        }
    }
}

void bestFit(int nb, int nf, int b[], int f[]) {
    int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
    int i, j, temp, lowest = 10000;

    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0 && lowest > temp) {
                    ff[i] = j;
                    lowest = temp;
                }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }

    printf("\nMemory Management Scheme - Best Fit\n");
    printf("File No\tFile Size \tBlock No\tBlock Size\tFragment\n");
    for (i = 1; i <= nf; i++) {
        printf("%d\t\t%d\t\t", i, f[i]);
        if (ff[i] != 0) {
            printf("%d\t\t%d\t\t%d\n", ff[i], b[ff[i]], frag[i]);
        } else {
            printf("Not Allocated\n");
        }
    }
}

void worstFit(int nb, int nf, int b[], int f[]) {
    int frag[MAX], bf[MAX] = {0}, ff[MAX] = {0};
    int i, j, temp, highest = 0;

    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0 && highest < temp) {
                    ff[i] = j;
                    highest = temp;
                }
            }
        }
        frag[i] = highest;
        bf[ff[i]] = 1;
        highest = 0;
    }
    printf("\nMemory Management Scheme - Worst Fit\n");
    printf("File_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragment\n");
    for (i = 1; i <= nf; i++) {
        printf("%d\t\t%d\t\t", i, f[i]);
        if (ff[i] != 0) {
            printf("%d\t\t%d\t\t%d\n", ff[i], b[ff[i]], frag[i]);
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main() {
    int b[MAX], f[MAX], nb, nf;

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

    int b1[MAX], b2[MAX], b3[MAX];
    for (int i = 1; i <= nb; i++) {
        b1[i] = b[i];
        b2[i] = b[i];
        b3[i] = b[i];
    }

    firstFit(nb, nf, b1, f);
    bestFit(nb, nf, b2, f);
    worstFit(nb, nf, b3, f);

    return 0;
}
```

Output:



```
Enter the number of blocks:5
Enter the number of files:4

Enter the size of the blocks
Block 1:400
Block 2:700
Block 3:200
Block 4:300
Block 5:600
Enter the size of the files
File 1:212
File 2:517
File 3:312
File 4:526
```



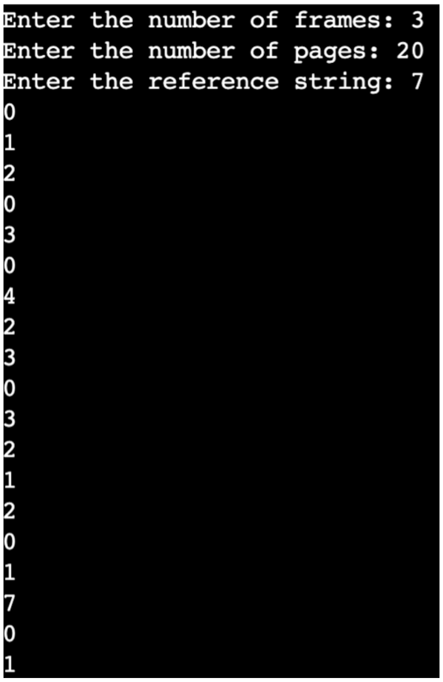
```
Memory Management Scheme - First Fit
File_no:        File_size :     Block_no:       Block_size:     Fragment
1               212             1               400             188
2               517             2               700             183
3               312             5               600             288
4               526             Not Allocated

Memory Management Scheme - Best Fit
File No File Size       Block No        Block Size      Fragment
1               212             4               300             88
2               517             5               600             83
3               312             1               400             88
4               526             2               700             174

Memory Management Scheme - Worst Fit
File_no:        File_size :     Block_no:       Block_size:     Fragment
1               212             2               700             488
2               517             5               600             83
3               312             1               400             88
4               526             Not Allocated
```

Program:10
Question:
Write a C program to simulate page replacement algorithms
a) FIFO
b) LRU
c) Optimal

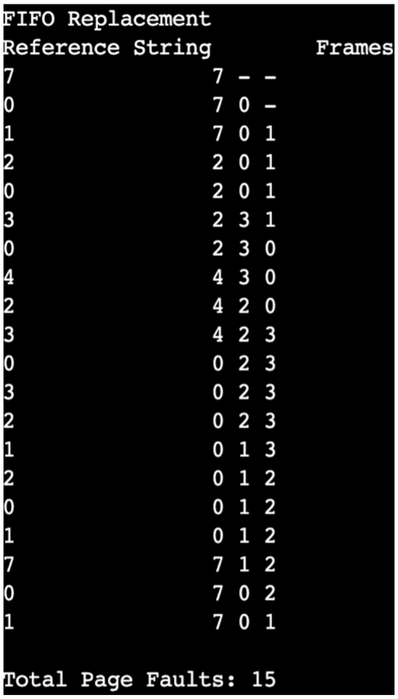
Code:

```
#include <stdio.h>

// Function to check if the page is present in the frames
int isPagePresent(int frames[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frames[i] == page) {
            return 1;
        }
    }
    return 0;
}

// Function to print the frames
void printFrames(int frames[], int n) {
    for (int i = 0; i < n; i++) {
        if (frames[i] != -1) {
            printf("%d ", frames[i]);
        } else {
            printf("- ");
        }
    }
    printf("\n");
}

// Function to implement FIFO page replacement
void fifoPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int front = 0, pageFaults = 0;

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

    printf("FIFO Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);
        if (!isPagePresent(frames, numFrames, pages[i])) {
            frames[front] = pages[i];
            front = (front + 1) % numFrames;
            pageFaults++;
        }
        printFrames(frames, numFrames);
    }

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

// Function to find the page to replace using the Optimal page replacement algorithm
int findOptimalReplacementIndex(int pages[], int numPages, int frames[], int numFrames, int
currentIndex) {
    int farthest = currentIndex;
    int index = -1;

    for (int i = 0; i < numFrames; i++) {
        int j;
        for (j = currentIndex; j < numPages; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    index = i;
                }
                break;
            }
        }
        // If the page is not found in future, return this index
        if (j == numPages) {
            return i;
        }
    }

    // If all pages are found in future, return the one with farthest future use
    return (index == -1) ? 0 : index;
}

// Function to implement Optimal page replacement
void optPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int pageFaults = 0;

    // Initialize frames
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
    }
    printf("Optimal Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);

        if (!isPagePresent(frames, numFrames, pages[i])) {
            if (isPagePresent(frames, numFrames, -1)) {
                for (int j = 0; j < numFrames; j++) {
                    if (frames[j] == -1) {
                        frames[j] = pages[i];
                        break;
                    }
                }
            } else {
                int index = findOptimalReplacementIndex(pages, numPages, frames, numFrames, i + 1);
                frames[index] = pages[i];
            }
            pageFaults++;
        }
        printFrames(frames, numFrames);
    }

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

// Function to implement LRU page replacement
void lruPageReplacement(int pages[], int numPages, int numFrames) {
    int frames[numFrames];
    int pageFaults = 0;
    int timestamps[numFrames];

    // Initialize frames and timestamps
    for (int i = 0; i < numFrames; i++) {
        frames[i] = -1;
        timestamps[i] = -1;
    }

    printf("LRU Replacement\n");
    printf("Reference String\tFrames\n");
    for (int i = 0; i < numPages; i++) {
        printf("%d\t\t", pages[i]);

        if (!isPagePresent(frames, numFrames, pages[i])) {
            int lruIndex = 0;
            for (int j = 1; j < numFrames; j++) {
                if (timestamps[j] < timestamps[lruIndex]) {
                    lruIndex = j;
                }
            }
            frames[lruIndex] = pages[i];
            timestamps[lruIndex] = i;
            pageFaults++;
        } else {
            for (int j = 0; j < numFrames; j++) {
                if (frames[j] == pages[i]) {
                    timestamps[j] = i;
                    break;
                }
            }
        }
        printFrames(frames, numFrames);
    }

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

int main() {
    int numFrames, numPages;

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

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

    int pages[numPages];

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

    fifoPageReplacement(pages, numPages, numFrames);
    optPageReplacement(pages, numPages, numFrames);
    lruPageReplacement(pages, numPages, numFrames);

    return 0;
}
```

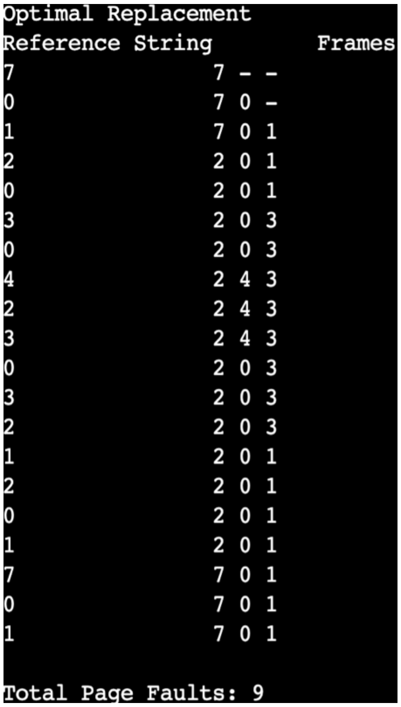
Output:



```
Enter the number of frames: 3
Enter the number of pages: 20
Enter the reference string: 7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
7
0
1
```



```
FIFO Replacement
Reference String        Frames
7               7 - -
0               7 0 -
1               7 0 1
2               2 0 1
0               2 0 1
3               2 3 1
0               2 3 0
4               4 3 0
2               4 2 0
3               4 2 3
0               0 2 3
3               0 2 3
2               0 2 3
1               0 1 3
2               0 1 2
0               0 1 2
1               0 1 2
7               7 1 2
0               7 0 2
1               7 0 1

Total Page Faults: 15
```



```
Optimal Replacement
Reference String        Frames
7               7 - -
0               7 0 -
1               7 0 1
2               2 0 1
0               2 0 1
3               2 0 3
0               2 0 3
4               2 4 3
2               2 4 3
3               2 4 3
0               2 0 3
3               2 0 3
2               2 0 3
1               2 0 1
2               2 0 1
0               2 0 1
1               2 0 1
7               7 0 1
0               7 0 1
1               7 0 1

Total Page Faults: 9
```