NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

OPERATING SYSTEM (KCS-451)

Department of Computer Science and Engineering

SUBMITTED BY:

AKANSHA GUPTA
(1901330100026)

SUBMITTED TO:

DR.CHANDRA SHEKHAR YADAV
(HEAD OF DEPARTMENT)

INDEX

Aim: Implementation of the Multi level queue cpu scheduling algorithm

```
#include<stdio.h>
int main()
{
int p[20],bt[20],wt[20],su[20],tat[20],i,k,n,temp;
float wtavg,tatavg;
printf("Enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++){
p[i]=i;
printf("Enter the Burst Time of Process %d:",i);
scanf("%d",&bt[i]);
printf("System/User Process (0/1) ?");
scanf("%d",&su[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(su[i]>su[k])
{
temp=p[i];
p[i]=p[k];
p[k]=temp;
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=su[i];
su[i]=su[k];
su[k]=temp;
}

wtavg=wt[0]=0;
tatavg=tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=tat[i-1]+bt[i];
wtavg=wtavg+wt[i];
tatavg=tatavg+tat[i];
}

printf("PROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND
TIME\n");
for(i=0;i<n;i++)
printf("%d \t\t %d \t\t %d \t\t %d \t\t %d \n",p[i],su[i],bt[i],wt[i],tat[i]);
printf("Average Waiting Time is --- %f\n",wtavg/n);
printf("Average Turnaround Time is --- %f\n",tatavg/n);
return 0;
```

Test Case - 1

User Output

```
Enter the number of processes:2
Enter the Burst Time of Process 0:45
System/User Process (0/1) ?0
Enter the Burst Time of Process 1:67
System/User Process (0/1) ?1
PROCESS          SYSTEM/USER PROCESS    BURST TIME       WAITING TIME     TURNAROUND TIME
0                0                 45               0                45
1                1                 67               45               112
Average Waiting Time is --- 22.500000
Average Turnaround Time is --- 78.500000
```

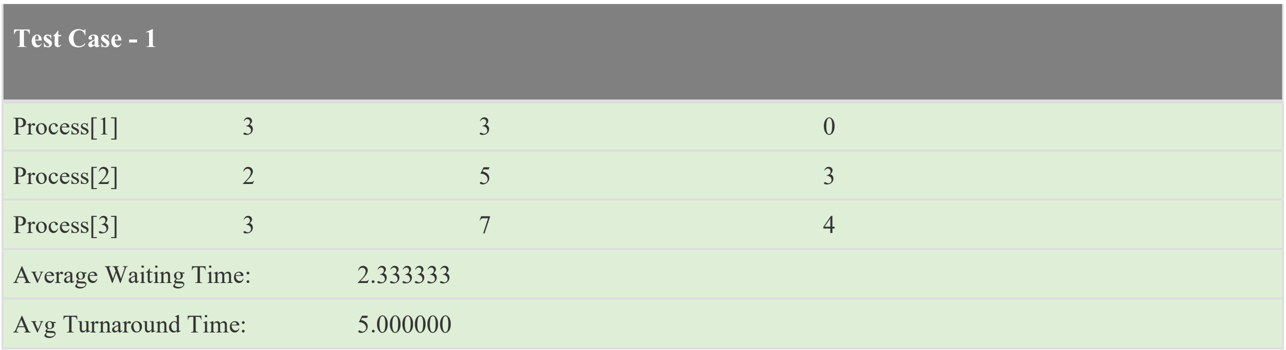
Aim: Implementation of the Round Robin cpu scheduling algorithm

```
#include<stdio.h>
#include<conio.h>
int main(){
 int i,NOP,sum=0,count=0,y,quant,wt=0,tat=0,at[10],bt[10],temp[10];
 float avg_wt,avg_tat;
 printf("Enter Total Number of Processes: ");
 scanf("%d",&NOP);
 y=NOP;
 for(i=0;i<NOP;i++){
  printf("Enter Details of Process[%d]: ",i+1);
  printf("Arrival Time:\t");
  scanf("%d",&at[i]);
  printf("Burst Time:\t");
  scanf("%d",&bt[i]);
  temp[i]=bt[i];
 }
 printf("Enter Time Quantum:\t");
 scanf("%d",&quant);
 printf("Process ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
 for(sum=0,i=0;y!=0;){
  if(temp[i]<=quant&&temp[i]>0){
   sum=sum+temp[i];
   temp[i]=0;
   count=1;
  }
  else if(temp[i]>0){
   temp[i]=temp[i]-quant;
   sum=sum+quant;
  }
  if(temp[i]==0&&count==1){
   y--;
   printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n",i+1,bt[i],sum-at[i],sum-at[i]-bt[i]);
   wt=wt+sum-at[i]-bt[i];
   tat=tat+sum-at[i];
   count=0;
  }
  if(i==NOP-1){
   i=0;
  }
  else if(at[i+1]<=sum){
   i++;
  }
  else
  {
   i=0;
  }
 }
   avg_wt=wt*1.0/NOP;
   avg_tat=tat*1.0/NOP;
   printf("Average Waiting Time:\t%f\n",avg_wt);
   printf("Avg Turnaround Time:\t%f\n",avg_tat );
   return 0;
}
```

Test Case - 1

User Output

```
Enter Total Number of Processes: 3
Enter Details of Process[1]: Arrival Time:      0
Burst Time:                3
Enter Details of Process[2]: Arrival Time:      0
Burst Time:                2
Enter Details of Process[3]: Arrival Time:      1
Burst Time:                3
Enter Time Quantum: 5
Process ID                 Burst Time Turnaround Time        Waiting Time
```



Test Case - 1			
Process[1]	3	3	0
Process[2]	2	5	3
Process[3]	3	7	4
Average Waiting Time:	2.333333		
Avg Turnaround Time:	5.000000		

Aim: Write a program to implement the PRIORITY based cpu scheduling algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
 int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
 int totwt=0,totta=0;
 float awt,ata;
 char pn[10][10],t[10];
 printf("Enter the number of process:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("Enter process name,arrivaltime,execution time & priority:");
  scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   if(p[i]<p[j])
   {
    temp=p[i];
    p[i]=p[j];
    p[j]=temp;
    temp=at[i];
    at[i]=at[j];
    at[j]=temp;
    strcpy(t,pn[i]);
    strcpy(pn[i],pn[j]);
    strcpy(pn[j],t);
   }
  }
 }
 for (i=0;i<n;i++){

  if(i==0)
```

```
   st[i]=at[i];
  else
    st[i]=ft[i-1];
  wt[i]=st[i]-at[i];
  ft[i]=st[i]+et[i];
  ta[i]=ft[i]-at[i];
  totwt+=wt[i];
  totta+=ta[i];
 }
 awt=(float)totwt/n;
 ata=(float)totta/n;
 printf("Pname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime\n");
 for(i=0;i<n;i++)
 {
  if(i==0)
  printf("%s\t %d\t\t %d\t\t %d\t\t %d\t\t %d\n",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
  else
  printf("%s\t %d\t\t %d\t\t %d\t\t %d\t\t %d\n",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
 }
 printf("Average waiting time is:%f\n",awt);
 printf("Average turnaroundtime is:%f\n",ata);
 return 0;
}
```

Test Case - 1

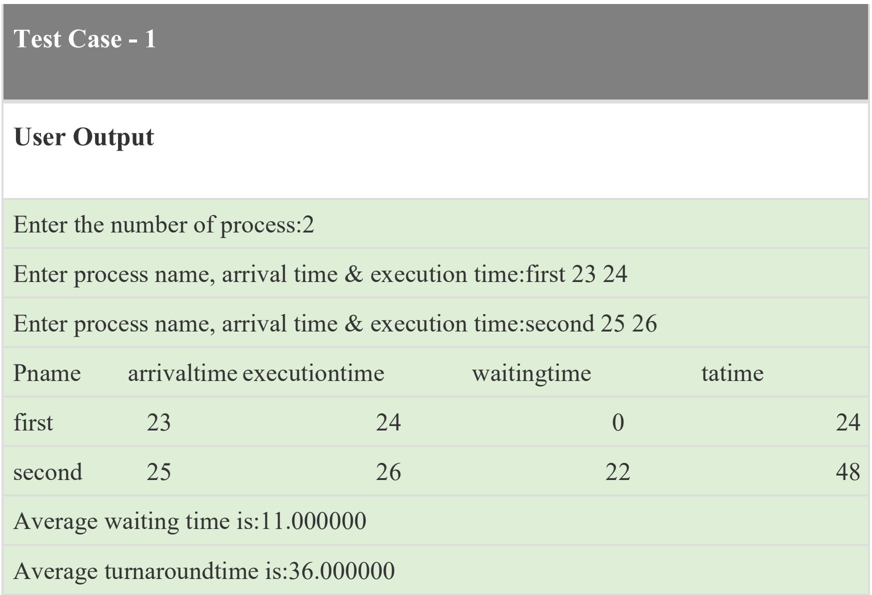
User Output

```
Enter the number of process:2
Enter process name,arrivaltime,execution time & priority:first 4 6 7
Enter process name,arrivaltime,execution time & priority:second 5 7 8
Pname       arrivaltime executiontime          priority      waitingtime            tatime
first          4                    6                     7                    0                    6
second         5                    7                     8                    5                   12
Average waiting time is:2.500000
Average turnaroundtime is:9.000000
```

Aim: Write a program to implement the SJF Scheduling Algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
 int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
 int totwt=0,totta=0;
 float awt,ata;
 char pn[10][10],t[10];
 printf("Enter the number of process:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("Enter process name, arrival time & execution time:");
  scanf("%s%d%d",pn[i],&at[i],&et[i]);
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   if(et[i]<et[j])
   {
    temp=at[i];
    at[i]=at[j];
    at[j]=temp;
    temp=et[i];
    et[i]=et[j];
    et[j]=temp;
    strcpy(t,pn[i]);
    strcpy(pn[i],pn[j]);
    strcpy(pn[j],t);

   }
  }
 }
 for(i=0;i<n;i++)
 {
  if(i==0)
  st[i]=at[i];
  else
    st[i]=ft[i-1];
   wt[i]=st[i]-at[i];
   ft[i]=st[i]+et[i];
   ta[i]=ft[i]-at[i];
   totwt+=wt[i];
   totta+=ta[i];
 }
 awt=(float)totwt/n;
 ata=(float)totta/n;
 printf("Pname\tarrivaltime\texecutiontime\twaitingtime\ttatime\n");
 for(i=0;i<1;i++)
 printf("%s\t %d\t\t %d\t\t %d\t\t %d\n",pn[i],at[i],et[i],wt[i],ta[i]);
 for(i=1;i<n;i++)
 printf("%s\t %d\t\t %d\t\t %d\t\t %d\n",pn[i],at[i],et[i],wt[i],ta[i]);
 printf("Average waiting time is:%f",awt);
 printf("\nAverage turnaroundtime is:%f",ata);
}
```



Test Case - 1

User Output

```
Enter the number of process:2
Enter process name, arrival time & execution time:first 23 24
Enter process name, arrival time & execution time:second 25 26
Pname      arrivaltime executiontime          waitingtime          tatime
first        23                 24                    0                 24
second       25                 26                   22                 48
Average waiting time is:11.000000
Average turnaroundtime is:36.000000
```

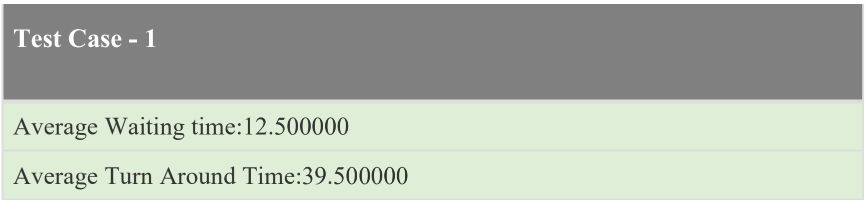
Aim: Write a program to implement the FCFS process scheduling algorithm.

```
#include<stdio.h>
#include<conio.h>
#define max 30
int main()
{
 int n,i,pn[max],at[max],bt[max],wt[max],tat[max],start[max],finish[max];
 float awt=0,atat=0;
 printf("Enter the number of processes: ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("Enter the Process Name, Arrival Time & Burst Time:");
  scanf("%d%d%d",&pn[i],&at[i],&bt[i]);
 }
 printf("Process Name\tArrival Time\tBurst Time\n");
 for(i=0;i<n;i++){
  printf(" %d\t %d\t %d\n",pn[i],at[i],bt[i]);
  }
 printf("PName Arrtime Bursttime Start WT\t TAT Finish\n");
 start[0]=at[0];
 finish[0]=start[0]+bt[0];
 for(i=0;i<n;i++)
 {
  if(i>0){
   start[i]=finish[i-1];
  }
  finish[i]=start[i]+bt[i];
  wt[i]=start[i]-at[i];
  tat[i]=bt[i]+wt[i];
 }
 for(i=0;i<1;i++)
 {
  printf("%d\t %d\t\t %d\t %d\t %d\t %d\t %d\n",pn[i],at[i],bt[i],start[i],wt[i],tat[i],finish[i]);
 }
 for(i=1;i<n;i++)
 {
  printf("%d\t %d\t\t %d\t %d\t %d\t %d\t %d\n",pn[i],at[i],bt[i],start[i],wt[i],tat[i],finish[i]);
 }
 for(i=0;i<n;i++)
 {
  awt+=wt[i];
  atat+=tat[i];
 }
 awt=awt/n;
 atat=atat/n;
 printf("Average Waiting time:%f",awt);
 printf("\nAverage Turn Around Time:%f",atat);
 return 0;
}
```

Test Case - 1

User Output

```
Enter the number of processes: 2
Enter the Process Name, Arrival Time & Burst Time:1 24 27
Enter the Process Name, Arrival Time & Burst Time:1 26 27
Process Name          Arrival Time          Burst Time
   1            24          27
   1            26          27
PName     Arrtime     Bursttime    Start    WT     TAT Finish
1           24                       27        24          0         27        51
1           26                       27        51         25         52        78
```



Test Case - 1

Average Waiting time:12.500000

Average Turn Around Time:39.500000

Aim: Write a program to implement the Banker’s algorithm.

```
#include<stdio.h>
void main(){
 int n,r,i,j,k,p,u=0,s=0,m;
 int block[10],run[10],active[10],newreq[10];
 int max[10][10],resalloc[10][10],resreq[10][10];
 int totalloc[10],totext[10],simalloc[10];
 printf("Enter the no of processes: ");
 scanf("%d",&n);
 printf("Enter the no of resource classes: ");
 scanf("%d",&r);
 printf("Enter the total existed resource in each class: ");
 for(k=1; k<=r; k++)
 scanf("%d",&totext[k]);
 printf("Enter the allocated resources: ");
 for(i=1; i<=n; i++)
 for(k=1; k<=r; k++)
 scanf("%d",&resalloc);
 printf("Enter the process making the new request: ");
 scanf("%d",&p);
 printf("Enter the requested resource: ");
 for(k=1; k<=r; k++)
 scanf("%d",&newreq[k]);
 printf("Enter the process which are n blocked or running\n");
 for(i=1; i<=n; i++) {
  if(i!=p) {
   printf("process %d: \n",i+1);
   scanf("%d%d",&block[i],&run[i]);
   }
   }
   block[p]=0;
   run[p]=0;
   for(k=1; k<=r; k++)
   {
    j=0;
    for(i=1; i<=n; i++)
     {
     totalloc[k]=j+resalloc[i][k];
     j=totalloc[k];
      }

   }
   for(i=1; i<=n; i++)
   {
```

```
if(block[i]==1||run[i]==1)
active[i]=1;
else
active[i]=0;
}
for(k=1; k<=r; k++)
{
 resalloc[p][k]+=newreq[k];
 totalloc[k]+=newreq[k];
 }
 for(k=1; k<=r; k++)
 {
  if(totext[k]-totalloc[k]<0)
  {
   u=1;
   break;
   }
   }
   if(u==0) {
    for(k=1; k<=r; k++)
    simalloc[k]=totalloc[k];
    for(s=1; s<=n; s++)
    for(i=1; i<=n; i++)
     {
     if(active[i]==1)
      {
      j=0;
      for(k=1; k<=r; k++)
       {
       if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
        {
        j=1;
        break;

        }
        }

      }
     if(j==0)
      {
      active[i]=0;
      for(k=1; k<=r; k++)
      simalloc[k]=resalloc[i][k];
      }
      }
      m=0;
      for(k=1; k<=r; k++)
      resreq[p][k]=newreq[k];
      printf("Deadlock willn't occur\n");
      }
      else
       {
       for(k=1; k<=r; k++)
        {
            resalloc[p][k]=newreq[k];
            totalloc[k]=newreq[k];

           }
           printf("Deadlock will occur\n");
           }
}
```

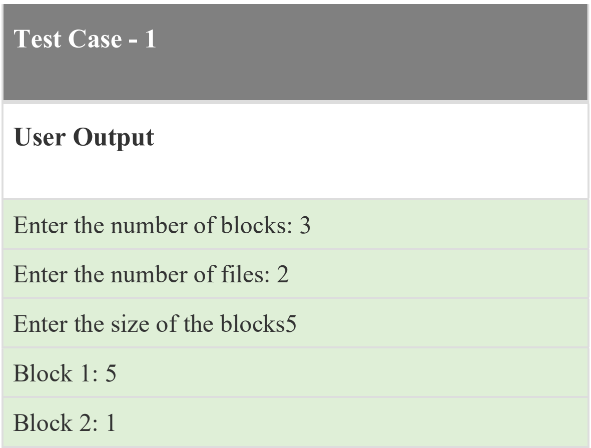
Test Case - 1
User Output
Enter the no of processes: 2
Enter the no of resource classes: 2
Enter the total existed resource in each class: 2 4 3 7
Enter the allocated resources: 5 9
Enter the process making the new request: 2 6
Enter the requested resource: 5 3
Enter the process which are n blocked or running2 6
process 2: 2 6
Deadlock will occur

Test Case - 2
User Output
Enter the no of processes: 1
Enter the no of resource classes: 1
Enter the total existed resource in each class: 1
Enter the allocated resources: 1
Enter the process making the new request: 1
Enter the requested resource: 1
Enter the process which are n blocked or running
Deadlock willn't occur

Aim: Write a C program to implement the Contiguous allocation technique: - First-Fit

```
#include<stdio.h>
#define max 25
void main(){
  int frag[max],b[max],f[max],i,j,nb,nf,temp;
   static int bf[max],ff[max];
    printf("Enter the number of blocks: ");
     scanf("%d",&nb);
      printf("Enter the number of files: ");
       scanf("%d",&nf);
        printf("Enter the size of the blocks\n");
         for(i=1;i<=nb;i++){
            printf("Block %d: ",i);
              scanf("%d",&b[i]);

         }
          printf("Enter the size of the files\n");
           for(i=1;i<=nf;i++){
              printf("File %d: ",i);
                scanf("%d",&f[i]);
           } 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;break;
                               }
                    }
               }
                    frag[i]=temp;
                          bf[ff[i]]=1;
           }
                printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
                      for(i=1;i<=nf;i++)
                            printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]); }
```



Test Case - 1

User Output

```
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks5
Block 1: 5
Block 2: 1
```

Test Case - 1

```
Block 3: 4
Enter the size of the files2
File 1: 2
File 2: 4
File_no     File_size Block_no Block_size Fragement
1           2         1        5          3
2           4         3        4          0
```

Test Case - 2

User Output

```
Enter the number of blocks: 4
Enter the number of files: 6
Enter the size of the blocks2
Block 1: 2
Block 2: 6
Block 3: 1
Block 4: 8
Enter the size of the files6
File 1: 6
File 2: 8
File 3: 1
File 4: 3
File 5: 5
File 6: 9
File_no     File_size Block_no Block_size Fragement
1           6         2        6          0
2           8         4        8          0
3           1         1        2          1
4           3         0        144        -2
5           5         0        144        -4
6           9         0        144        -8
```

Aim: Write a program to Implementation of Contiguous allocation technique: - Best-Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
int main()
{
  int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
   static int bf[max],ff[max];
    printf("Memory Management Scheme for contigus memeory allocation - Best Fit\n");
     printf("Enter the number of blocks:");
      scanf("%d",&nb);
       printf("Enter the number of files:");
        scanf("%d",&nf);
         printf("Enter the size of the blocks:-\n");
          for(i=1;i<=nb;i++)
           {
              printf("Block %d:",i);
                scanf("%d",&b[i]);
           }
            printf("Enter the size of the files :-\n");
             for(i=1;i<=nf;i++)
              {
                 printf("File %d:",i);
                   scanf("%d",&f[i]);
              }
               for(i=1;i<=nf;i++)
                {
                   for(j=1;j<=nb;j++)
                     {
                         if(bf[j]!=1)
                            {
                           temp=b[j]-f[i];
                         if(temp>=0)
                            if(lowest>temp)
                                {
                     ff[i]=j;

                          lowest=temp;
                     }
                }
}
  frag[i]=lowest;
    bf[ff[i]]=1;
    lowest=10000;
       }
        printf("File No\tFile Size \tBlock No\tBlock Size\tFragment");
         for(i=1;i<=nf && ff[i]!=0;i++)
         printf("%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
         return 0;
           }
```

Test Case - 1

User Output

```
Memory Management Scheme for contigus memeory allocation - Best Fit3
Enter the number of blocks:3
Enter the number of files:2
Enter the size of the blocks:-5
Block 1:5
Block 2:1
Block 3:4
Enter the size of the files :-3
File 1:3
File 2:4
File No   File Size   Block No   Block Size Fragment1        3            3            4
          12                     4                    1            5            1
```

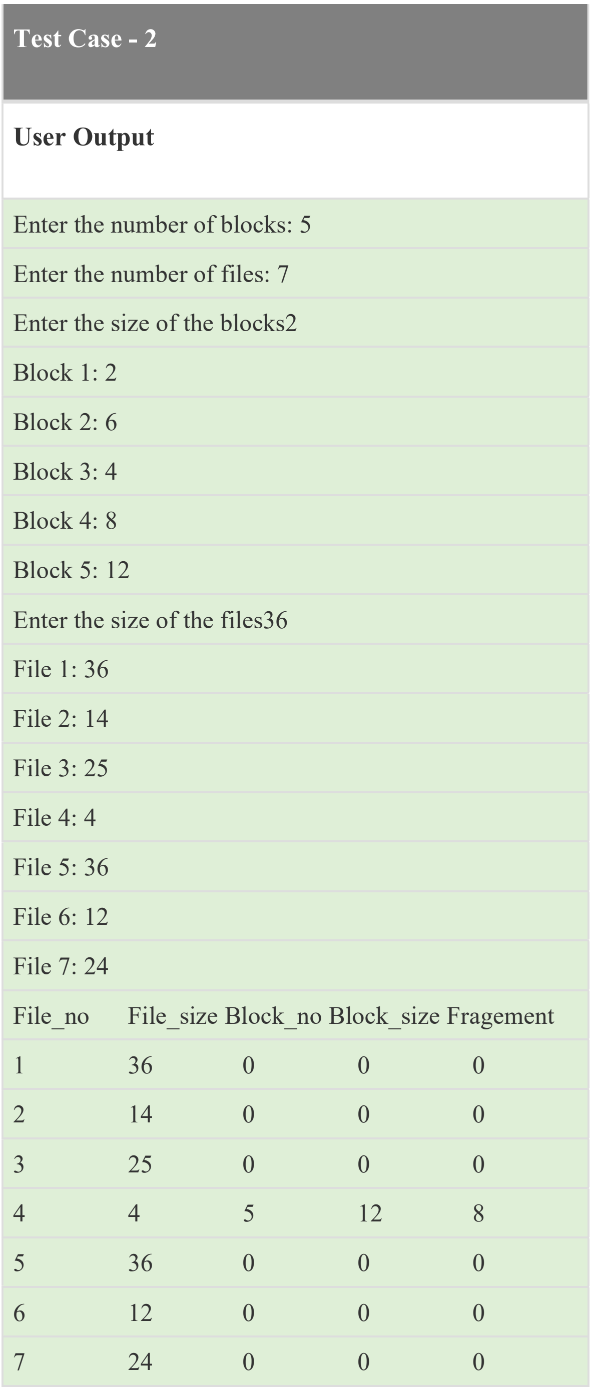
Aim: Write a program to Implementation of Contiguous allocation technique :- Worst-Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
  int frag[max],b[7],f[max],i,j,nb,nf,temp,highest=0;
   static int bf[max],ff[max];
    printf("Enter the number of blocks: ");
     scanf("%d",&nb);
      printf("Enter the number of files: ");
       scanf("%d",&nf);
        printf("Enter the size of the blocks\n");
         for(i=1;i<=nb;i++)
          {
           printf("Block %d: ",i);
            scanf("%d",&b[i]);
          }
           printf("Enter the size of the files\n");
            for(i=1;i<=nf;i++)
             {
              printf("File %d: ",i);
               scanf("%d",&f[i]);
            }
             for(i=1;i<=nf;i++)
              {
                for(j=1;j<=nb;j++)
                 {
                  if(bf[j]!=1)
                   {
                     temp=b[j]-f[i];
                      if(temp>=0)
                      if(highest<temp)
                        {
                    ff[i]=j;
                    highest=temp;
                   }
                 }
              }
               frag[i]=highest;
               bf[ff[i]]=1;
               highest=0;
}
 printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
  for(i=1;i<=nf;i++)
  printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Test Case - 1

User Output

```
Enter the number of blocks: 4
Enter the number of files: 3
Enter the size of the blocks5
Block 1: 5
Block 2: 4
Block 3: 3
Block 4: 5
Enter the size of the files2
File 1: 2
File 2: 9
File 3: 4
File_no     File_size Block_no Block_size Fragement
1           2           1           5           3
2           9           0           0           0
3           4           4           5           1
```



Test Case - 2

User Output

```
Enter the number of blocks: 5
Enter the number of files: 7
Enter the size of the blocks2
Block 1: 2
Block 2: 6
Block 3: 4
Block 4: 8
Block 5: 12
Enter the size of the files36
File 1: 36
File 2: 14
File 3: 25
File 4: 4
File 5: 36
File 6: 12
File 7: 24
File_no     File_size Block_no Block_size Fragement
1           36        0        0          0
2           14        0        0          0
3           25        0        0          0
4           4         5        12         8
5           36        0        0          0
6           12        0        0          0
7           24        0        0          0
```

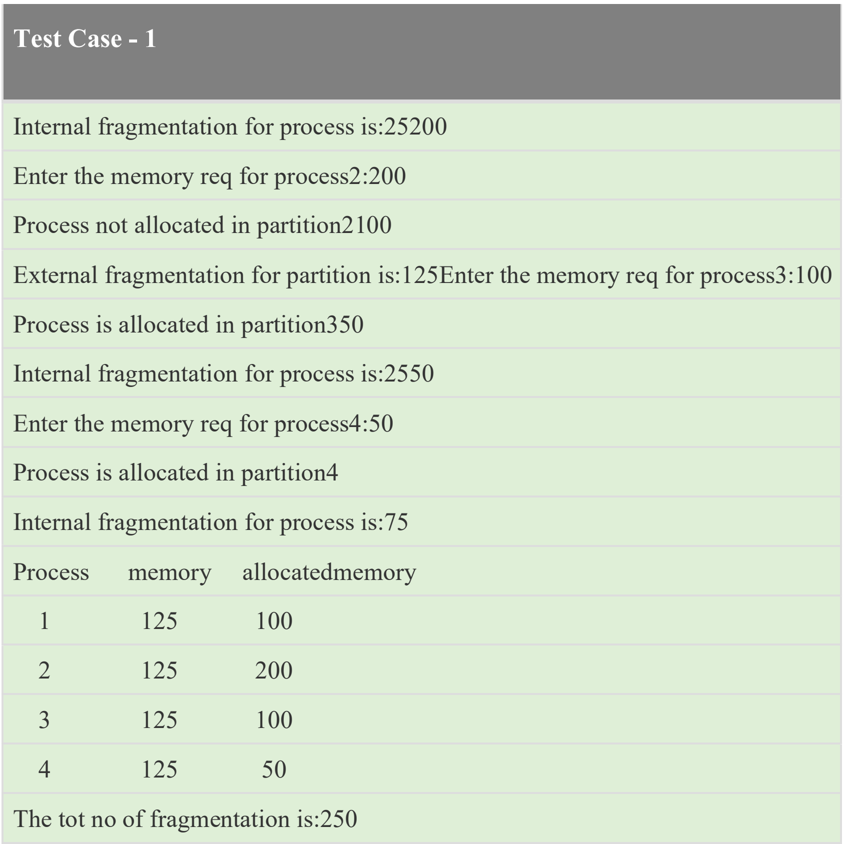
Aim: Write a program to Implementation of contiguous memory fixed partition technique(MFT)

```
#include<stdio.h>
#include<conio.h>
int main()
{
  int m,p,s,p1;
   int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,pos;
    printf("Enter the memory size:");
    scanf("%d",&m);
      printf("Enter the no of partitions:");
      scanf("%d",&p);
       s=m/p;
        printf("Each partn size is:%d",s);
         printf("Enter the no of processes:");
         scanf("%d",&p1);
           pos=m;
            for(i=0;i<p1;i++)
             {
               printf("Enter the memory req for process%d:",i+1);
                scanf("%d",&m1[i]);
                 if(m1[i]<=s)
                  {
                    printf("Process is allocated in partition%d\n",i+1);
                     fra1=s-m1[i];
                      printf("Internal fragmentation for process is:%d\n",fra1);
                       f1=f1+fra1;
                        pos=pos-s;
                  }
                   else
                    {
                      printf("Process not allocated in partition%d\n",i+1);
                       s1=m1[i];
                        while(s1>s)
                         {
                     s1=s1-s;
                     pos=pos-s;
                    }
                     pos=pos-s;
                      fra2=s;
                      f2=f2+fra2;
                        printf("External fragmentation for partition is:%d",fra2);
             }
}
 printf("Process\tmemory\tallocatedmemory");
  for(i=0;i<p1;i++)
  printf("\n%5d\t%5d\t%5d",i+1,s,m1[i]);
  f=f1+f2;
     printf("\nThe tot no of fragmentation is:%d",f);
}
```

Test Case - 1

User Output

```
Enter the memory size:500
Enter the no of partitions:4
Each partn size is:125Enter the no of processes:4
Enter the memory req for process1:100
Process is allocated in partition1200
```



Test Case - 1

```
Internal fragmentation for process is:25200
Enter the memory req for process2:200
Process not allocated in partition2100
External fragmentation for partition is:125Enter the memory req for process3:100
Process is allocated in partition350
Internal fragmentation for process is:2550
Enter the memory req for process4:50
Process is allocated in partition4
Internal fragmentation for process is:75
Process    memory    allocatedmemory
   1         125        100
   2         125        200
   3         125        100
   4         125         50
The tot no of fragmentation is:250
```

Aim: Write a program to Implementation of contiguous memory Variable partition technique (MVT)

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int m=0,m1=0,m2=0,p,count=0,i;
 printf("enter the memory capacity:");
 scanf("%d",&m);
 printf("enter the no of processes:");
 scanf("%d",&p);
 for(i=0;i<p;i++){
  printf("enter memory req for process%d:",i+1);
  scanf("%d",&m1);
  count=count+m1;
  if(count==m)
  printf("there is no further memory remaining:\n");
  else if(m1<m){
   printf("the memory allocated for process%d is: %d ",i+1,m);
   m2=m-m1;
   printf("\nremaining memory is: %d\n",m2);
   m=m2;
  }
 else
 {
  printf("memory is not allocated for process%d",i+1);
 }
 printf("external fragmentation for this process is:%d\n",m2);
}
return 0;
}
```

Test Case - 1

User Output

```
enter the memory capacity:500
enter the no of processes:2
enter memory req for process1:250
the memory allocated for process1 is: 500 50
remaining memory is: 25050
external fragmentation for this process is:25050
enter memory req for process2:50
the memory allocated for process2 is: 250
remaining memory is: 200
external fragmentation for this process is:200
```

Test Case - 2

User Output

```
enter the memory capacity:250
enter the no of processes:2
enter memory req for process1:250
there is no further memory remaining:120
external fragmentation for this process is:0120
enter memory req for process2:120
the memory allocated for process2 is: 250
remaining memory is: 130
external fragmentation for this process is:130
```