Final Report on

OPERATING SYSTEM

CA ASSINGMENT

LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

Subject : OPERATING SYSTEMS (CSE 316)

Submitted to :SHIVALI CHOPRA

Section : K18TS

S.No	Roll No.	REG.NO	NAME
	12	11805680	NISHANT TIWARI

GITHUB-

12. Implement the multi-level feedback queue scheduling algorithm by considering the following diagram: You can use the code of others to implement Roud-Robin, and FCFS implement aging by your own self.

SOL-

```
#include<stdio.h>
int main()
{
/////~~ My Multilevel Feedback Queue Scheduling Algo ~~/////

    int n,i,c=0,ts=0,ws=0;
    int at[100],bt[100],p,m[100],wt[100],t[100],temp[100];
    int cm[100]={0};
    double v,vs;

    printf("*WELCOME TO MY MULTILEVEL FEEDBACK QUEUE
SCHEDULING PROGRAM*\n\n");
    printf("Enter number of Process you want : ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
       printf("FOR  PROCESS NO. %d ENTER \n",i+1);
       A:
       printf(" Burst Time : ");
       scanf("%d",&bt[i]);
       if(bt[i]<0)
       {
               printf("!!CAUTION BURST CANNOT BE NEGATIVE!!\n\n");
               goto A;
                }
      at[i]=0;
       printf("\n");
        }



        for(i=0;i<n;i++)
        {
               temp[i]=bt[i];
        }

/////~~ FIRST QUEUE HIGHEST PRIOITY USING ROUND ROBIN WITH
TIME QUANTUM 8 ~~/////

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



    if(bt[i]<=8)
    {
       c=c+8;
       cm[i]=c;
```

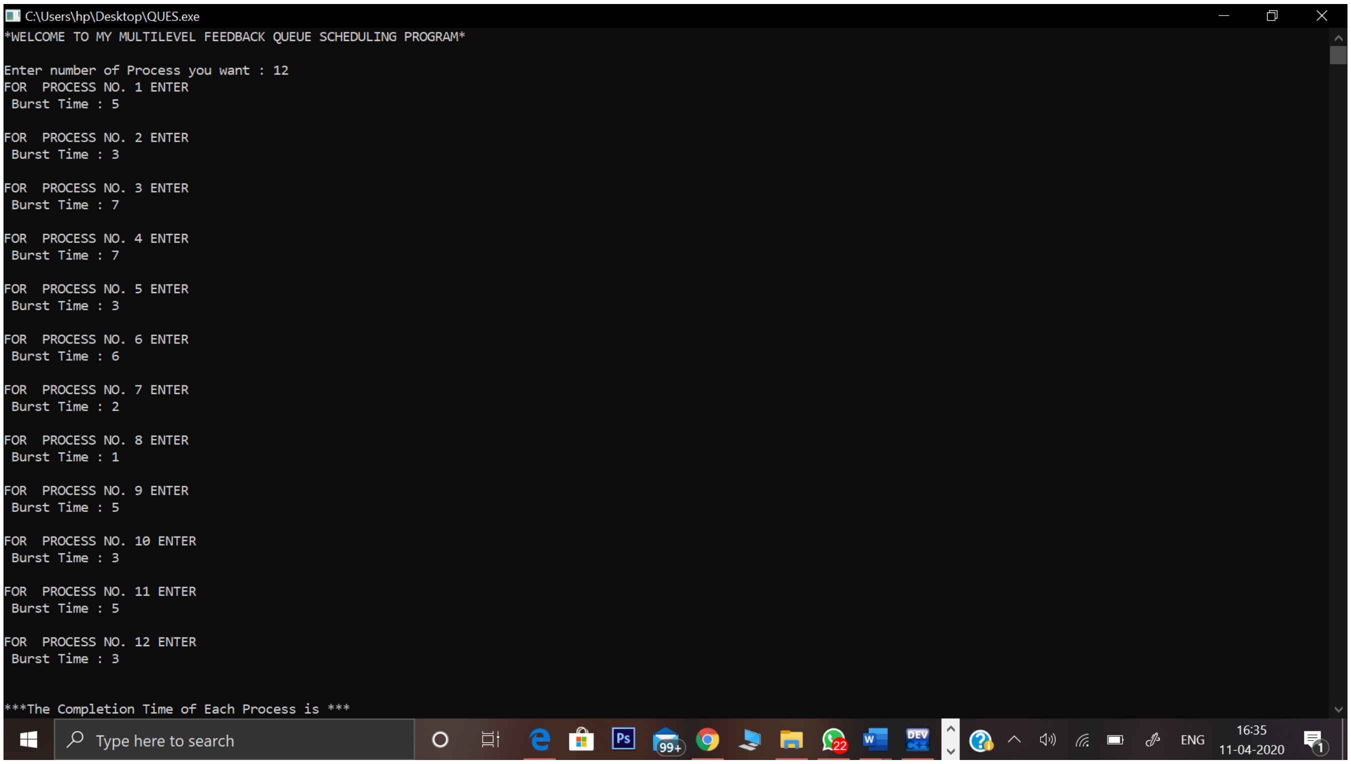
```
        }


        else
        {
              bt[i]=bt[i]-8;
              c=c+8;
        }


        }




/////~~ SECOND QUEUE SECOND HIGHEST PRIOITY USING ROUND
ROBIN WITH TIME QUANTUM 16  ~~/////


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


    if(cm[i]==0)
        {
```

```
     if(bt[i]<=16)
     {
        c=c+bt[i];
        cm[i]=c;
         }


         else
         {
                bt[i]=bt[i]-16;
                c=c+16;
         }


         }

    }


/////~~ THIRD QUEUE WITH LEAST PRIORITY USING FCFS ~~/////

        for(i=0;i<n;i++)
        {
                if(cm[i]==0)
                {
                        c=c+bt[i];
                        cm[i]=c;
                }
```

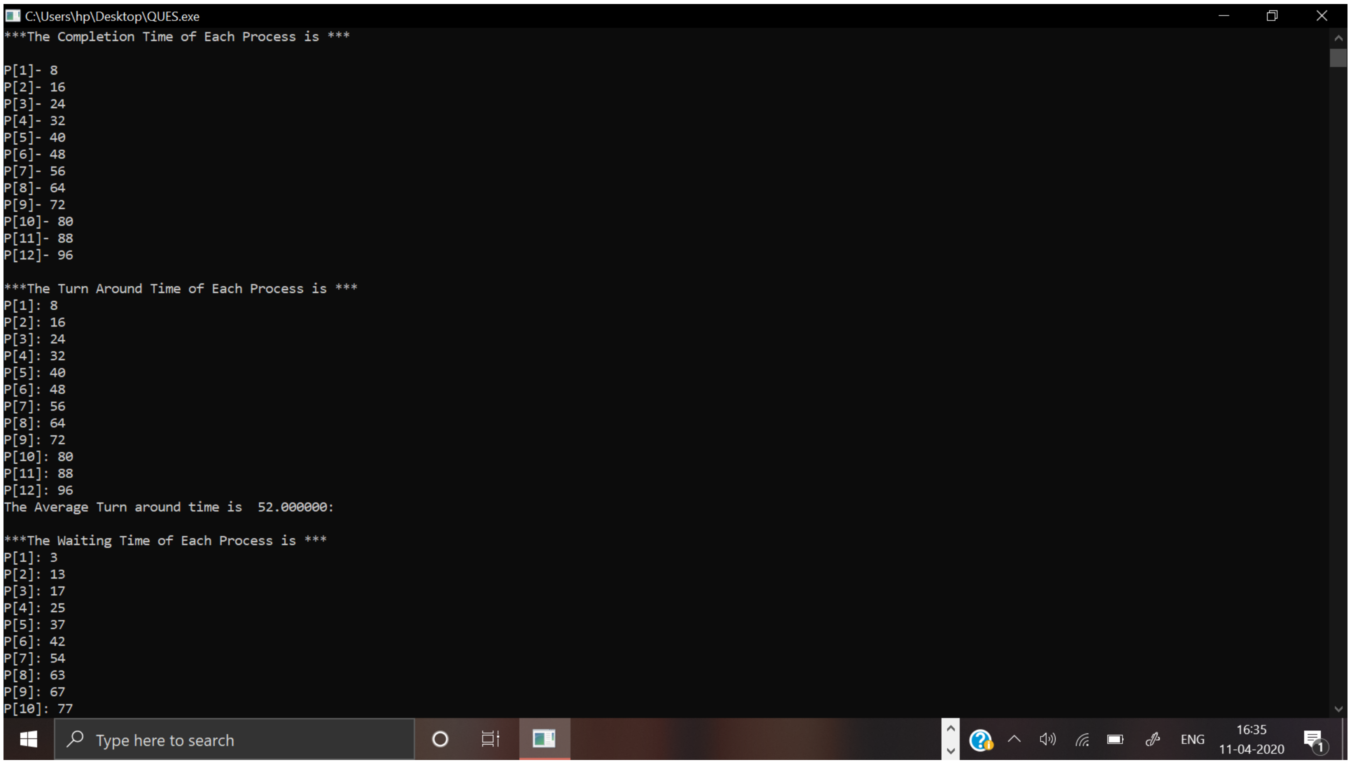
```
   }


printf("\n***The Completion Time of Each Process is *** \n\n");
    for(i=0;i<n;i++)
{
   printf("P[%d]- %d",i+1,cm[i]);
  printf("\n");
    }

    printf("\n***The Turn Around Time of Each Process is ***\n");
   for(i=0;i<n;i++)
{
   t[i]=cm[i]-at[i];
   printf("P[%d]: %d\n",i+1,t[i]);
   ts=ts+t[i];
    }

      v=ts/(float)n;
   printf("The Average Turn around time is  %f: ",v);
   printf("\n");



     printf("\n***The Waiting Time of Each Process is ***\n");
   for(i=0;i<n;i++)
{
   wt[i]=t[i]-temp[i];
   printf("P[%d]: %d\n",i+1,wt[i]);
```

```
    ws=ws+wt[i];
     }

       vs=ws/(float)n;
    printf("The Average Waiting time is %f: ",vs);
    printf("\n");



}
```

Output-



```
*WELCOME TO MY MULTILEVEL FEEDBACK QUEUE SCHEDULING PROGRAM*

Enter number of Process you want : 12
FOR  PROCESS NO. 1 ENTER
 Burst Time : 5

FOR  PROCESS NO. 2 ENTER
 Burst Time : 3

FOR  PROCESS NO. 3 ENTER
 Burst Time : 7

FOR  PROCESS NO. 4 ENTER
 Burst Time : 7

FOR  PROCESS NO. 5 ENTER
 Burst Time : 3

FOR  PROCESS NO. 6 ENTER
 Burst Time : 6

FOR  PROCESS NO. 7 ENTER
 Burst Time : 2

FOR  PROCESS NO. 8 ENTER
 Burst Time : 1

FOR  PROCESS NO. 9 ENTER
 Burst Time : 5

FOR  PROCESS NO. 10 ENTER
 Burst Time : 3

FOR  PROCESS NO. 11 ENTER
 Burst Time : 5

FOR  PROCESS NO. 12 ENTER
 Burst Time : 3


***The Completion Time of Each Process is ***
```



```
***The Completion Time of Each Process is ***

P[1]- 8
P[2]- 16
P[3]- 24
P[4]- 32
P[5]- 40
P[6]- 48
P[7]- 56
P[8]- 64
P[9]- 72
P[10]- 80
P[11]- 88
P[12]- 96

***The Turn Around Time of Each Process is ***
P[1]: 8
P[2]: 16
P[3]: 24
P[4]: 32
P[5]: 40
P[6]: 48
P[7]: 56
P[8]: 64
P[9]: 72
P[10]: 80
P[11]: 88
P[12]: 96
The Average Turn around time is  52.000000:

***The Waiting Time of Each Process is ***
P[1]: 3
P[2]: 13
P[3]: 17
P[4]: 25
P[5]: 37
P[6]: 42
P[7]: 54
P[8]: 63
P[9]: 67
P[10]: 77
```

15. CPU schedules N processes which arrive at different time intervals and each process is allocated the CPU for a specific user input time unit, processes are scheduled using a preemptive round robin scheduling algorithm. Each process must be assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes one task has priority 0. The length of a time quantum is

T units, where T is the custom time considered as time quantum for processing. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue. Design a scheduler so that the task with priority 0 does not starve for resources and gets the CPU at some time unit to execute. Also compute waiting time, turn around.

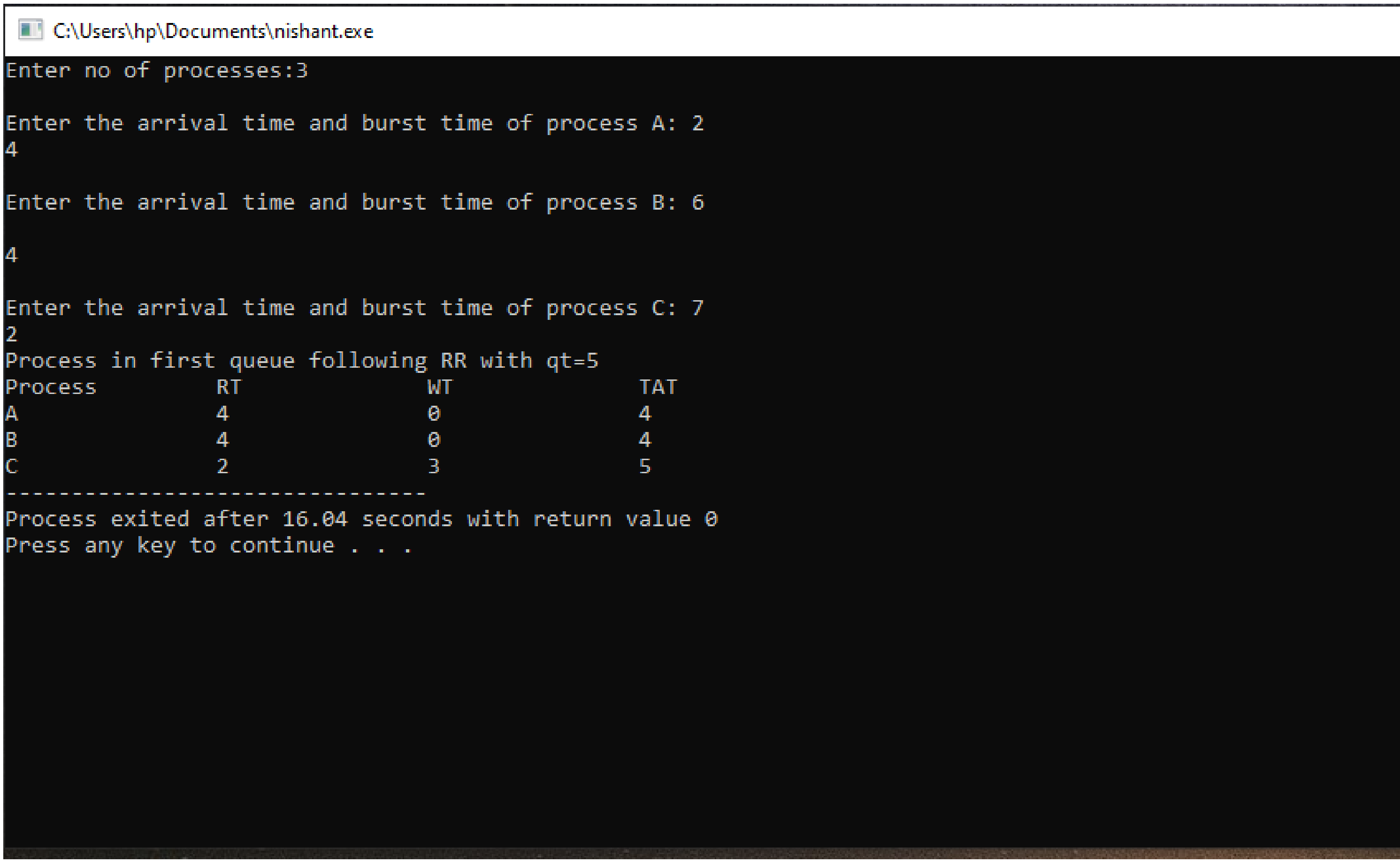
Sol-

```
#include<stdio.h>
struct process
{
   char name;
   int AT,BT,WT,TAT,RT,CT;
}Q1[10],Q2[10],Q3[10];//Three queues//
int n;
void sortByArrival()
```

```
{
struct process temp;
int i,j;
for(i=0;i<n;i++)
   {
       for(j=i+1;j<n;j++)
           {
               if(Q1[i].AT>Q1[j].AT)
                   {
                        temp=Q1[i];
                        Q1[i]=Q1[j];
                        Q1[j]=temp;
                   }
           }
   }
}

int main()
{
    int i,j,k=0,r=0,time=0,tq1=8,tq2=16,flag=0;
    char c;
    printf("Enter no of processes:");
    scanf("%d",&n);
    for(i=0,c='A';i<n;i++,c++)
    {
        Q1[i].name=c;
        printf("\nEnter the arrival time and burst time of process %c:
",Q1[i].name);
        scanf("%d%d",&Q1[i].AT,&Q1[i].BT);
        Q1[i].RT=Q1[i].BT;//save burst time in remaining time for each
process//


   }
sortByArrival();
time=Q1[0].AT;
printf("Process in first queue following RR with qt=5");
printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
for(i=0;i<n;i++)
{
if(Q1[i].RT<=tq1)
  {
      time+=Q1[i].RT;//from arrival time of first process to completion
of this process//
      Q1[i].RT=0;
      Q1[i].WT=time-Q1[i].AT-Q1[i].BT;//amount of time process has
been waiting in the first queue//
      Q1[i].TAT=time-Q1[i].AT;//amount of time to execute the
process//

printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i
].TAT);


  }
  else//process moves to queue 2 with qt=8//
  {
     Q2[k].WT=time;
     time+=tq1;
     Q1[i].RT-=tq1;
     Q2[k].BT=Q1[i].RT;
     Q2[k].RT=Q2[k].BT;
     Q2[k].name=Q1[i].name;
     k=k+1;
```

```
      flag=1;
   }
}
if(flag==1)
 {printf("\nProcess in second queue following RR with qt=8");
  printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");
}for(i=0;i<k;i++)
   {
    if(Q2[i].RT<=tq2)
     {
       time+=Q2[i].RT;//from arrival time of first process +BT of this process//
       Q2[i].RT=0;
       Q2[i].WT=time-tq1-Q2[i].BT;//amount of time process has been waiting in the ready queue//
       Q2[i].TAT=time-Q2[i].AT;//amount of time to execute the process//

printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);


    }
  else//process moves to queue 3 with FCFS//
  {
   Q3[r].AT=time;
   time+=tq2;
   Q2[i].RT-=tq2;
   Q3[r].BT=Q2[i].RT;
   Q3[r].RT=Q3[r].BT;
   Q3[r].name=Q2[i].name;
   r=r+1;
   flag=2;
  }
 }

{if(flag==2)
printf("\nProcess in third queue following FCFS ");
}
for(i=0;i<r;i++)
{
  if(i==0)
      Q3[i].CT=Q3[i].BT+time-tq1-tq2;
      else
         Q3[i].CT=Q3[i-1].CT+Q3[i].BT;


}


for(i=0;i<r;i++)
   {
      Q3[i].TAT=Q3[i].CT;
      Q3[i].WT=Q3[i].TAT-Q3[i].BT;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,
Q3[i].TAT);
}
}
```

Output-



```
Enter no of processes:3

Enter the arrival time and burst time of process A: 2
4

Enter the arrival time and burst time of process B: 6

4

Enter the arrival time and burst time of process C: 7
2
Process in first queue following RR with qt=5
Process         RT              WT              TAT
A               4               0               4
B               4               0               4
C               2               3               5
--------------------------------
Process exited after 16.04 seconds with return value 0
Press any key to continue . . .
```