Report

School of Computer Science and Engineering

Lovely Professional University.

Phagwara, Punjab (India).

CSE316 – OPERATING SYSTEMS

Submitted to:

Dr. Baljit Singh Saini Sir.

Submitted By:

Name: Lucky JANGRA

Reg no: 11804238

Roll no: 55

Github link:

Question:

Write a program for multilevel queue scheduling algorithm. There must be three queues generated. There must be specific range of priority associated with every queue. Now prompt the user to enter number of processes along with their priority and burst time. Each process must occupy the respective queue with specific priority range according to its priority. Apply Round Robin algorithm with quantum time 4 on queue with highest priority range. Apply priority scheduling algorithm on the queue with medium range of priority and First come first serve algorithm on the queue with lowest range of priority. Each and every queue should get a quantum time of 10 seconds. CPU will keep on shifting between queues after every 10 seconds.

Description:

For this program of Multilevel queue algorithm I have used different algorithm
Round robin algorithm
First come first serve algorithm
Priority algorithm

Multilevel Queue scheduling:
Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.
For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.
A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.
For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

Round robin algorithm:

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching

ALGO:

1- Create an array rem_bt[] to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array wt[] to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem_bt[i] > quantum

(i) t = t + quantum

(ii) bt_rem[i] -= quantum;

c- Else

(i) t = t + bt_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt_rem[i] = 0;

First come first serve algorithm:-

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.
In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Algo:

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find waiting time for all other processes i.e. for

process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find turnaround time = waiting_time + burst_time

for all processes.

6- Find average waiting time =

total_waiting_time / no_of_processes.

7- Similarly, find average turnaround time =

total_turn_around_time / no_of_processes.

Priority algorithm:-

```
#include <bits/stdc++.h>

using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process
{
int at,bt,pr,pno;
};

process proc[50];

/*
Writing comparator function to sort according to priority if
arrival time is same
*/

bool comp(process a,process b)
{
if(a.at == b.at)
{
return a.pr<b.pr;
}
else
```

```
{
        return a.at<b.at;
}
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(int wt[])
{
// declaring service array that stores cumulative burst time
int service[50];

// Initilising initial elements of the arrays
service[0]=0;
wt[0]=0;


for(int i=1;i<totalprocess;i++)
{
service[i]=proc[i-1].bt+service[i-1];

wt[i]=service[i]-proc[i].at+1;

// If waiting time is negative, change it into zero

        if(wt[i]<0)
        {
        wt[i]=0;
        }
}
```

```
}

void get_tat_time(int tat[],int wt[])
{
// Filling turnaroundtime array

for(int i=0;i<totalprocess;i++)
{
        tat[i]=proc[i].bt+wt[i];
}

}

void findgc()
{
//Declare waiting time and turnaround time array
int wt[50],tat[50];

double wavg=0,tavg=0;

// Function call to find waiting time array
get_wt_time(wt);
//Function call to find turnaround time
get_tat_time(tat,wt);

int stime[50],ctime[50];
stime[0]=1;
ctime[0]=stime[0]+tat[0];
// calculating starting and ending time
for(int i=1;i<totalprocess;i++)
```

```
        {
                stime[i]=ctime[i-1];
                ctime[i]=stime[i]+tat[i]-wt[i];
        }

cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time"<<endl;

        // display the process details

for(int i=0;i<totalprocess;i++)
        {
                wavg += wt[i];
                tavg += tat[i];

                cout<<proc[i].pno<<"\t\t"<<
                        stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<
                        tat[i]<<"\t\t\t"<<wt[i]<<endl;
        }

                // display the average waiting time
                //and average turn around time

        cout<<"Average waiting time is : ";
        cout<<wavg/(float)totalprocess<<endl;
        cout<<"average turnaround time : ";
        cout<<tavg/(float)totalprocess<<endl;

}

int main()
```

```
{
int arrivaltime[] = { 1, 2, 3, 4, 5 };
int bursttime[] = { 3, 5, 1, 7, 4 };
int priority[] = { 3, 4, 1, 7, 8 };

for(int i=0;i<totalprocess;i++)
{
        proc[i].at=arrivaltime[i];
        proc[i].bt=bursttime[i];
        proc[i].pr=priority[i];
        proc[i].pno=i+1;
        }

        //Using inbuilt sort function

        sort(proc,proc+totalprocess,comp);

        //Calling function findgc for finding Gantt Chart

        findgc();

        return 0;
}
```

Program

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

int currTime=0,queueTimer=10,total;

struct MutilevelQ
{
	int p_id;int ct;int arr;int start;int prio;
	int bt;int store;
};

void roundRobinAlgorithm(struct MutilevelQ p[],int queueNo)
{		printf("Round Robin is Running at %d sec\n",currTime);
		int timeQuantum=10;
		int i;
		for( i=0;i<queueNo;i++)
		{
			if(timeQuantum<=queueTimer&&p[i].arr<=currTime)
			{
			if(p[i].start==-1)
			{
			p[i].start=currTime;
			}
			if(p[i].bt>=timeQuantum)
```

```
                {
                p[i].bt=p[i].bt-timeQuantum;
                currTime=currTime+timeQuantum;
                queueTimer-=timeQuantum;
                }
                if(p[i].bt<timeQuantum&&p[i].bt>0)
                {
                currTime=currTime+p[i].bt;
                queueTimer-=p[i].bt;
                p[i].bt=0;
                }
                if(p[i].bt==0&&p[i].ct==-1)
                {
                        p[i].ct=currTime;
                        total--;
                }
                }
        }
        queueTimer=10;
}

void firstComeFirstServe(struct MutilevelQ p[],int r)
{
        printf("First Come First Serve is Running at %d sec\n",currTime);
        int i;
```

```
for(i=0;i<r;i++)
{
    if(p[i].bt<=queueTimer && p[i].arr<=currTime && p[i].bt>0)
    {
    if(p[i].start==-1)
    p[i].start=currTime;
    p[i].bt=0;
    currTime=currTime+p[i].bt;
    queueTimer-=p[i].bt;
    if(p[i].bt==0&&p[i].ct==-1)
    {
        p[i].ct=currTime;
        total--;
    }
  }
    else
    {
        if((p[i].bt-
queueTimer>0)&&p[i].arr<=currTime&&p[i].bt>0)
        {
        if(p[i].start=-1)
        p[i].start==currTime;
        p[i].bt=p[i].bt-queueTimer;
        currTime=currTime+queueTimer;
        queueTimer=0;
        if(p[i].bt==0&&p[i].ct==-1)
```

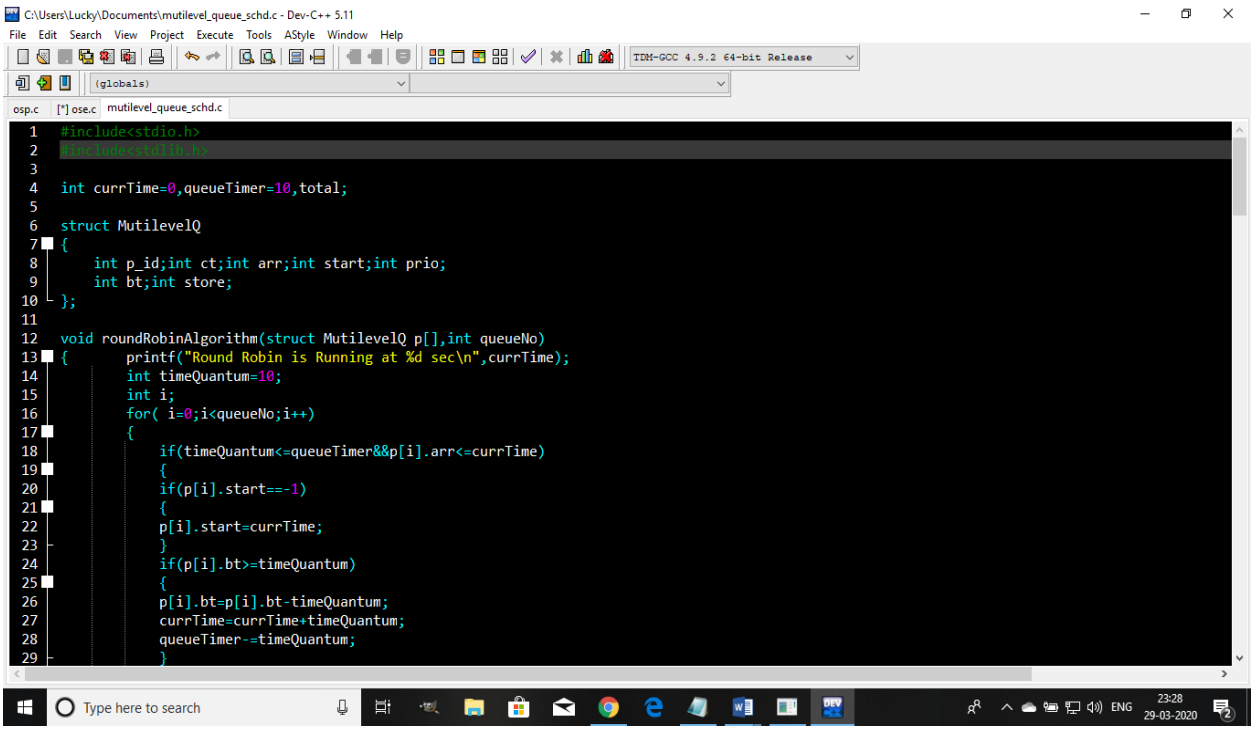
```
                        {
                                p[i].ct=currTime;
                                total--;
                        }
                        }
                }
        }
        queueTimer=10;
}

int priorityAlgo(struct MutilevelQ p[],int r,int remain)
{
    printf("Priority Queue is Running at %d sec \n",currTime);
    int smallest;
    int i;
    while(remain!=0&&queueTimer>0)
    {
       smallest=r;
       for(i=0;i<r;i++)
       {
          if(p[i].arr<=currTime && p[i].prio<p[smallest].prio && p[i].bt>0)
          {
             smallest=i;
          }
       }
```

```
            if(p[smallest].start==-1)
            {
               p[smallest].start=currTime;
            }
            queueTimer-=1;
            p[smallest].bt-=1;
            if(p[smallest].bt==0&&p[smallest].ct==-1)
            {
                 p[smallest].ct=currTime;
               total--;
               remain--;
            }
            currTime+=1;
         }
         queueTimer=10;
}

int main()
{
         int n,i,rr=0,pr=0,fcfs=0;
         printf("Enter the number of processes: ");
         scanf("%d",&n);
         total=n;
         struct MutilevelQ ps[n];
    for(i=0;i<n;i++)
```

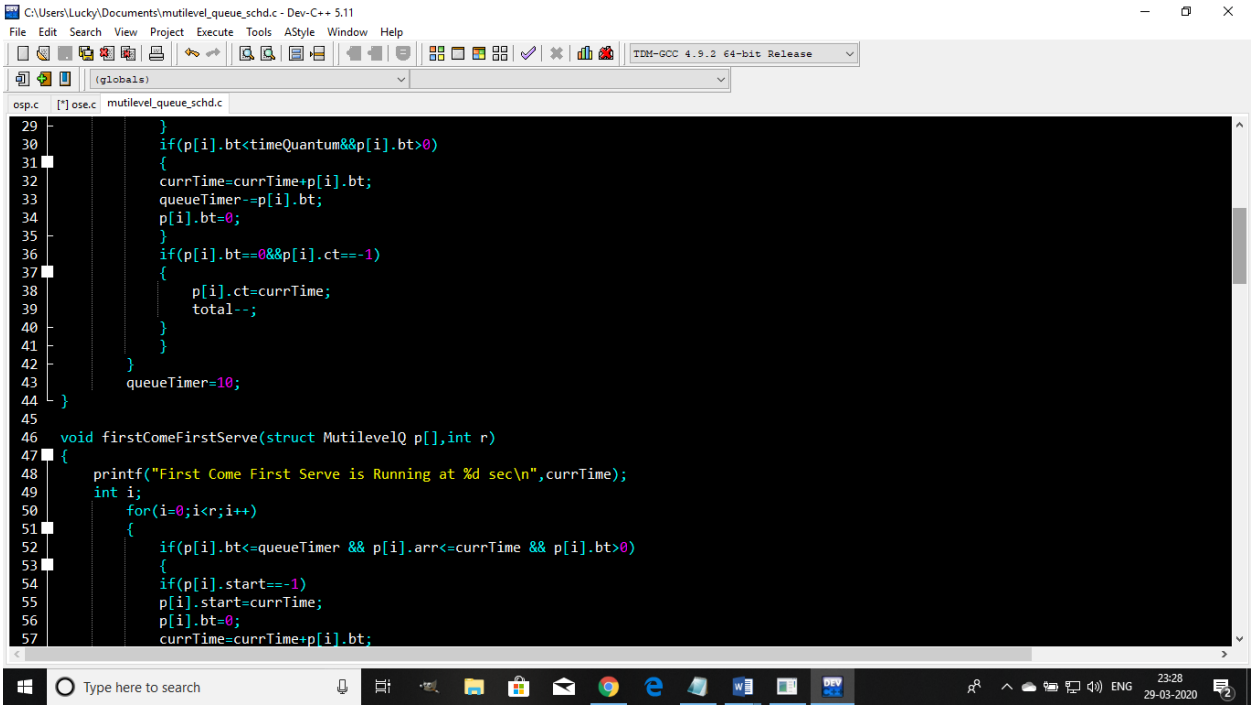
```
{
    printf("PROCESS %d :\n",i+1);
        ps[i].p_id=i+1;
        printf("Enter the Arrival Time: ");
    scanf("%d",&ps[i].arr);
    printf("Enter the priority 1 for Round Robin, 2 for First Come First Serve
and 3 for priority: ");
    scanf("%d",&ps[i].prio);
    if(ps[i].prio==1)
        rr++;
        else if(ps[i].prio==2)
        fcfs++;
        else if(ps[i].prio==3)
        pr++;
        printf("Enter burst time: ");
    scanf("%d",&ps[i].bt);
        ps[i].store=ps[i].bt;
        ps[i].start=-1;
    ps[i].ct=-1;
    }
    int rrid=0,prid=0,fcfsid=0;
    struct MutilevelQ roundqueue[rr],priorqueue[pr+1],fcfsqueue[fcfs];
    for(i=0;i<n;i++)
    {
        if(ps[i].prio==1)
        {
```

```
                roundqueue[rrid]=ps[i];
                rrid++;
            }
            else if(ps[i].prio==2)
            {
                fcfsqueue[fcfsid]=ps[i];
                fcfsid++;
            }
            else if(ps[i].prio==3)
            {
                priorqueue[prid]=ps[i];
                prid++;
            }
        }
        int remain=pr;
        while(total>0)
        {
            roundRobinAlgorithm(roundqueue,rr);
            remain=priorityAlgo(priorqueue,pr,remain);
            firstComeFirstServe(fcfsqueue,fcfs);
        }
        printf("Process Id\tArrival Time\tBrust Time\tStarting Time\tTurn Around Time\tPriority\n");
        for(i=0;i<rr;i++)
        {
```

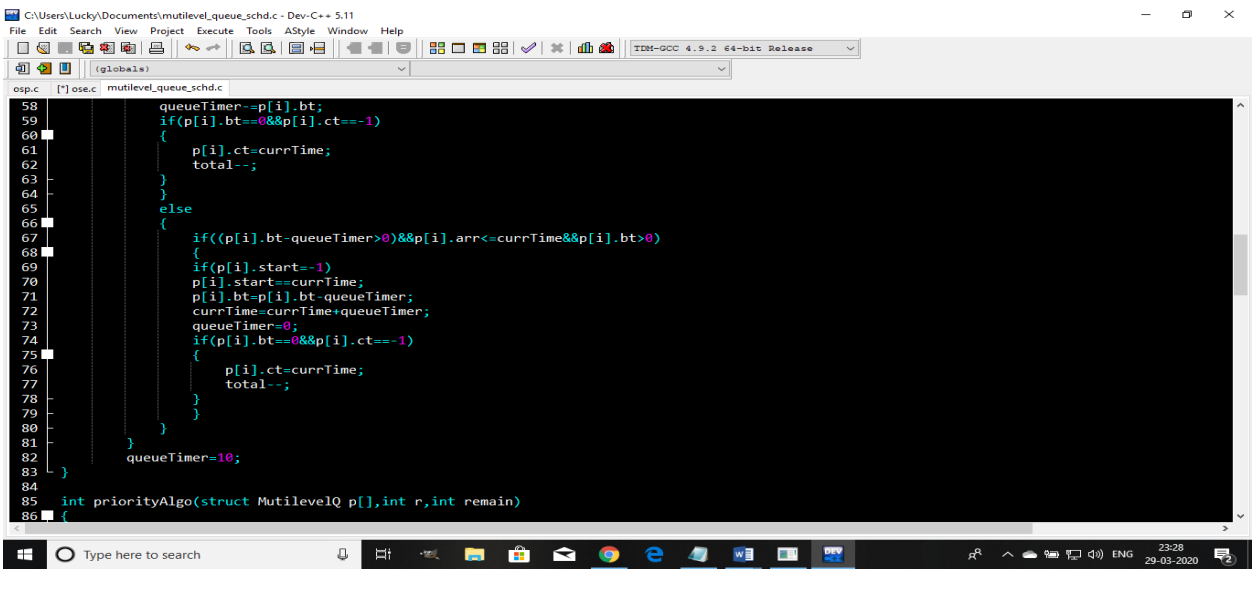
```
	printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",roundqueue[i].p_id,roundqueue[i].arr,roundqueue[i].store,roundqueue[i].start,roundqueue[i].ct,roundqueue[i].prio);

	}

	for(i=0;i<fcfs;i++)

	{


	printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",fcfsqueue[i].p_id,fcfsqueue[i].arr,fcfsqueue[i].store,fcfsqueue[i].start,fcfsqueue[i].ct,fcfsqueue[i].prio);

	}

	for(i=0;i<pr;i++)

	{


	printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",priorqueue[i].p_id,priorqueue[i].arr,priorqueue[i].store,priorqueue[i].start,priorqueue[i].ct,priorqueue[i].prio);

	}

}
```



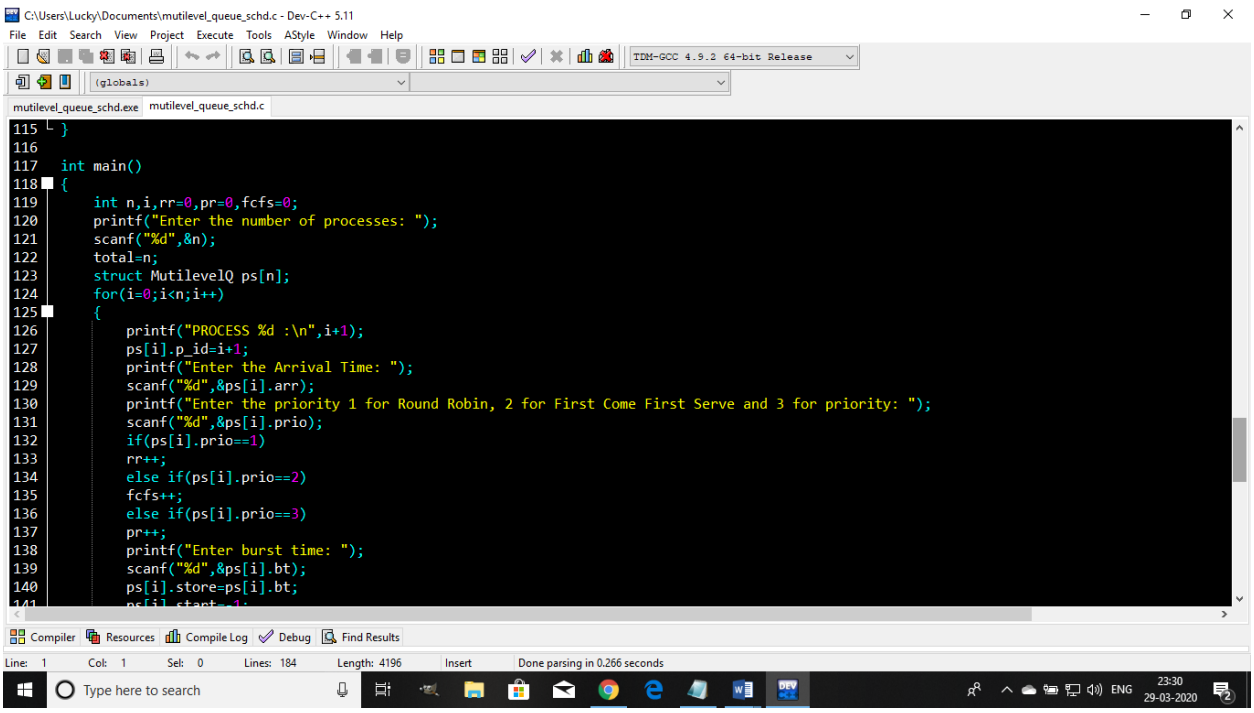
```c
#include<stdio.h>
#include<stdlib.h>

int currTime=0,queueTimer=10,total;

struct MutilevelQ
{
    int p_id;int ct;int arr;int start;int prio;
    int bt;int store;
};

void roundRobinAlgorithm(struct MutilevelQ p[],int queueNo)
{       printf("Round Robin is Running at %d sec\n",currTime);
        int timeQuantum=10;
        int i;
        for( i=0;i<queueNo;i++)
        {
            if(timeQuantum<=queueTimer&&p[i].arr<=currTime)
            {
            if(p[i].start==-1)
            {
            p[i].start=currTime;
            }
            if(p[i].bt>=timeQuantum)
            {
            p[i].bt=p[i].bt-timeQuantum;
            currTime=currTime+timeQuantum;
            queueTimer-=timeQuantum;
            }
```



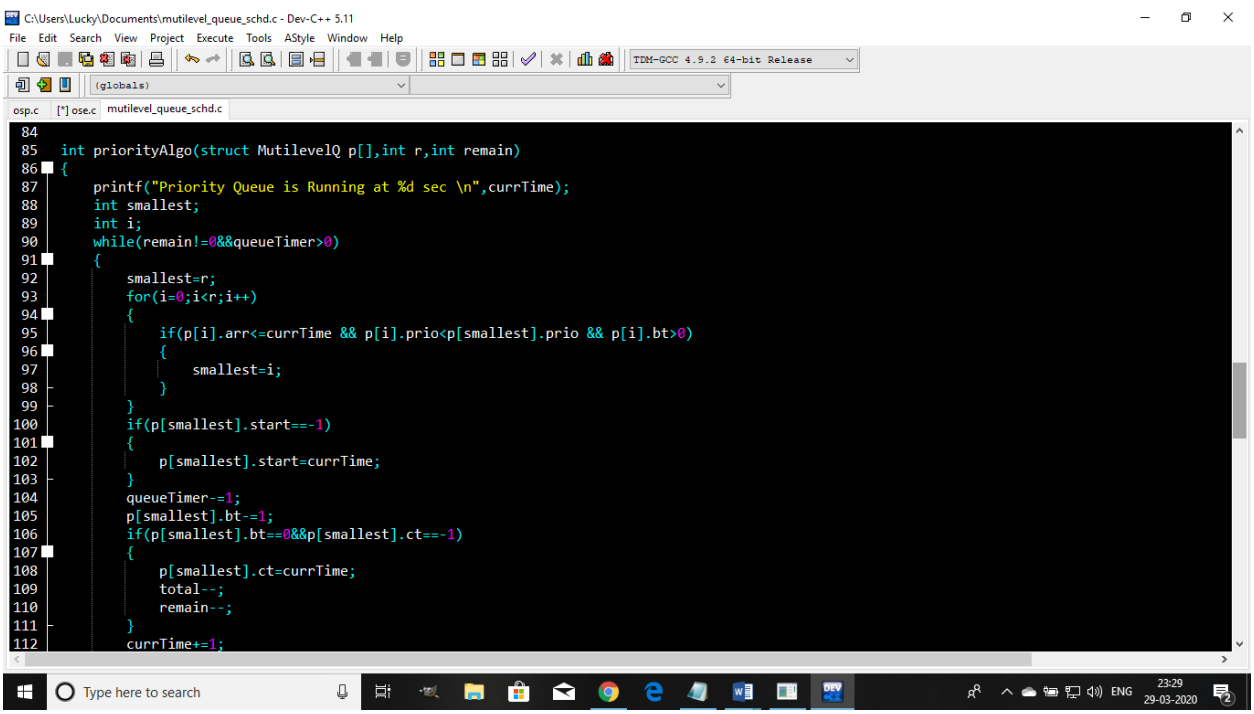
```c
            }
            if(p[i].bt<timeQuantum&&p[i].bt>0)
            {
            currTime=currTime+p[i].bt;
            queueTimer-=p[i].bt;
            p[i].bt=0;
            }
            if(p[i].bt==0&&p[i].ct==-1)
            {
                p[i].ct=currTime;
                total--;
            }
            }
        }
        queueTimer=10;
}

void firstComeFirstServe(struct MutilevelQ p[],int r)
{
    printf("First Come First Serve is Running at %d sec\n",currTime);
    int i;
        for(i=0;i<r;i++)
        {
            if(p[i].bt<=queueTimer && p[i].arr<=currTime && p[i].bt>0)
            {
            if(p[i].start==-1)
            p[i].start=currTime;
            p[i].bt=0;
            currTime=currTime+p[i].bt;
```



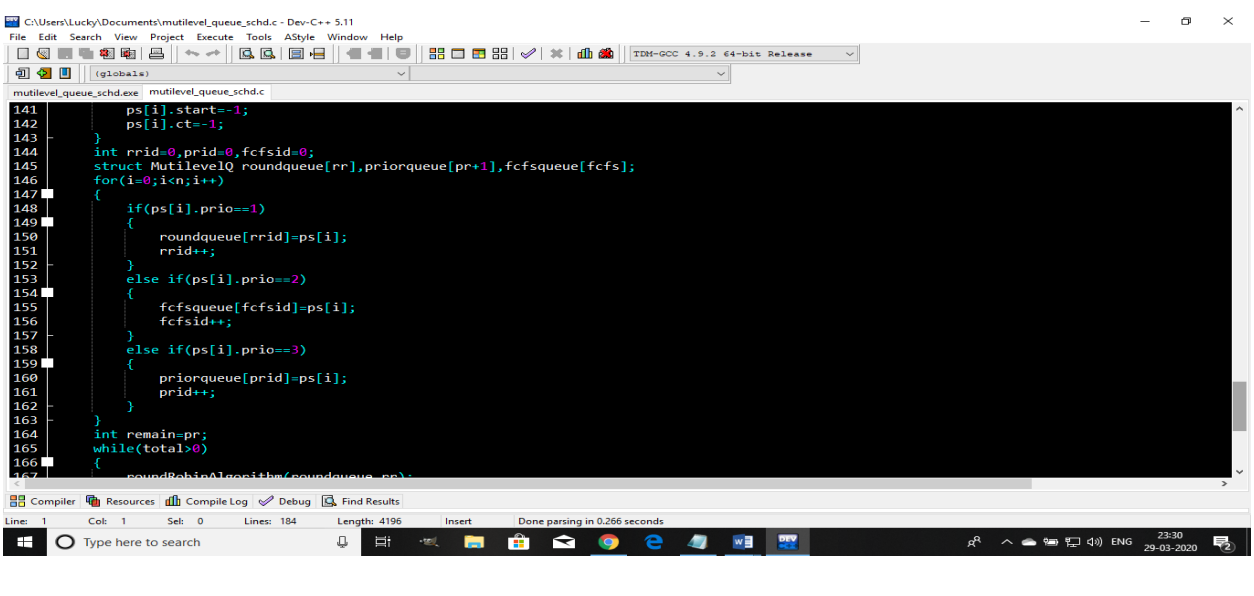
```c
            queueTimer-=p[i].bt;
            if(p[i].bt==0&&p[i].ct==-1)
            {
                p[i].ct=currTime;
                total--;
            }
            }
            else
            {
                if((p[i].bt-queueTimer>0)&&p[i].arr<=currTime&&p[i].bt>0)
                {
                if(p[i].start==-1)
                p[i].start==currTime;
                p[i].bt=p[i].bt-queueTimer;
                currTime=currTime+queueTimer;
                queueTimer=0;
                if(p[i].bt==0&&p[i].ct==-1)
                {
                    p[i].ct=currTime;
                    total--;
                }
                }
            }
        }
        queueTimer=10;
}

int priorityAlgo(struct MutilevelQ p[],int r,int remain)
{
```



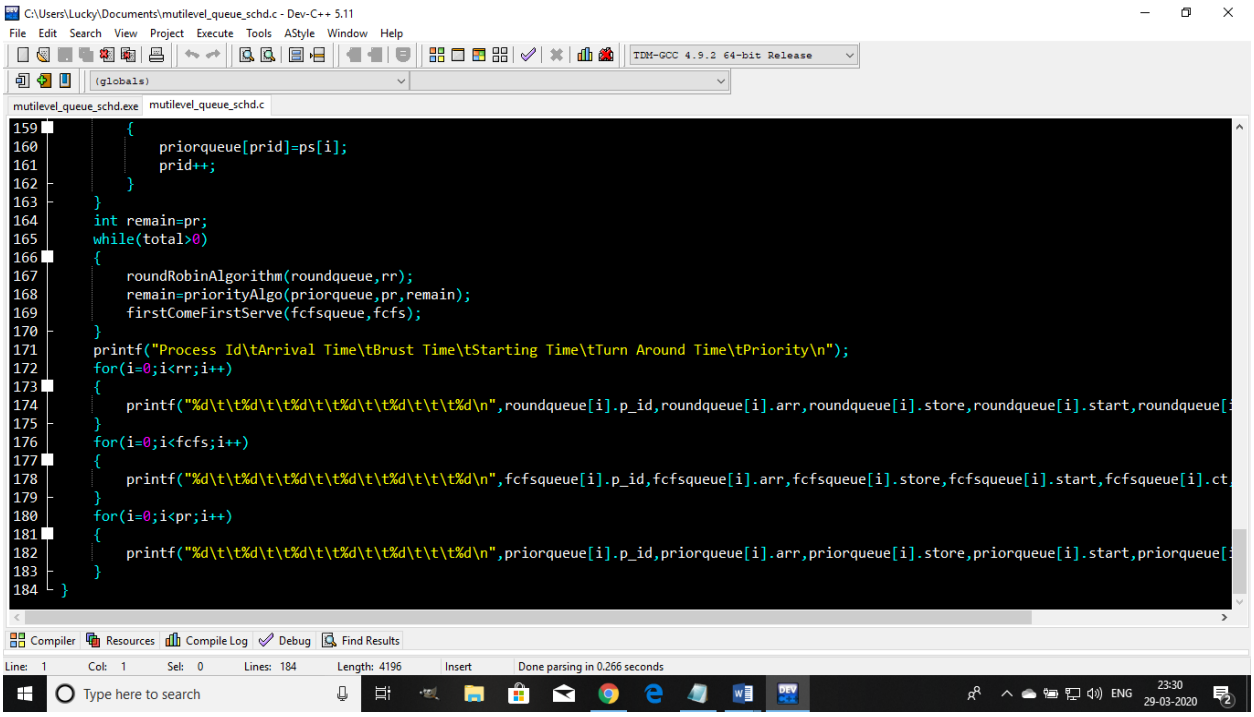
```
115 }
116
117 int main()
118 {
119     int n,i,rr=0,pr=0,fcfs=0;
120     printf("Enter the number of processes: ");
121     scanf("%d",&n);
122     total=n;
123     struct MutilevelQ ps[n];
124     for(i=0;i<n;i++)
125     {
126         printf("PROCESS %d :\n",i+1);
127         ps[i].p_id=i+1;
128         printf("Enter the Arrival Time: ");
129         scanf("%d",&ps[i].arr);
130         printf("Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: ");
131         scanf("%d",&ps[i].prio);
132         if(ps[i].prio==1)
133         rr++;
134         else if(ps[i].prio==2)
135         fcfs++;
136         else if(ps[i].prio==3)
137         pr++;
138         printf("Enter burst time: ");
139         scanf("%d",&ps[i].bt);
140         ps[i].store=ps[i].bt;
```



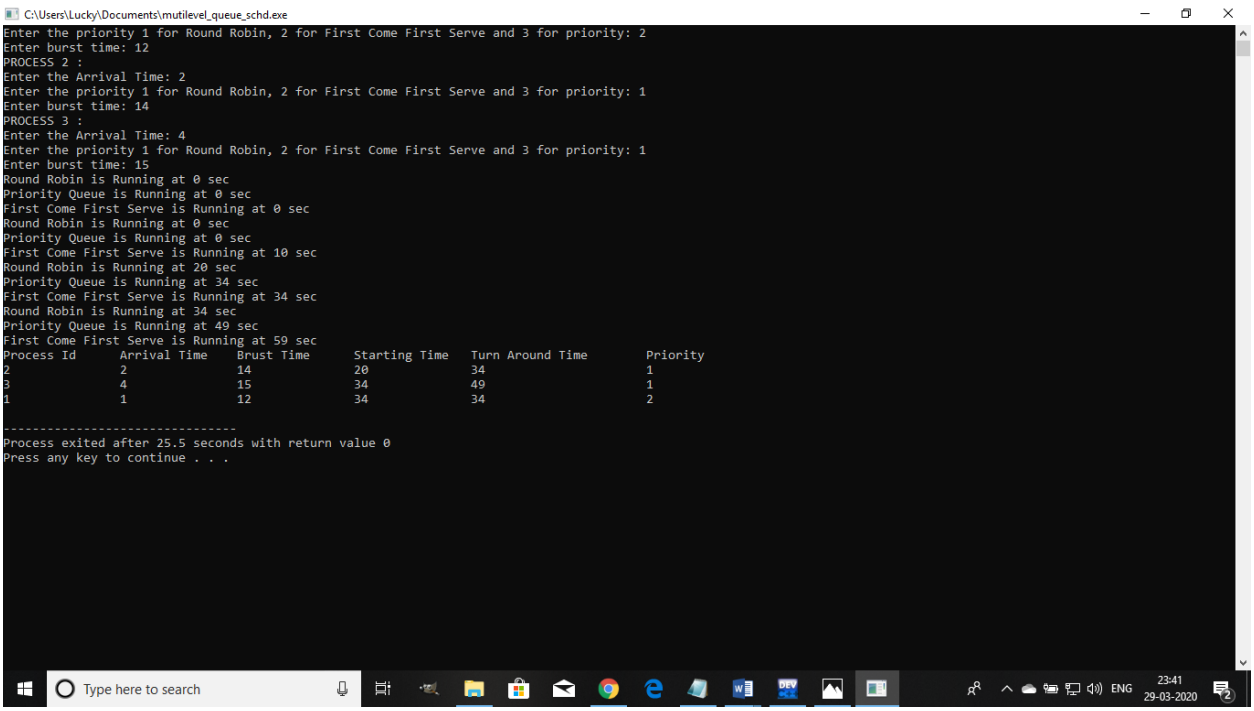
```
 84
 85 int priorityAlgo(struct MutilevelQ p[],int r,int remain)
 86 {
 87     printf("Priority Queue is Running at %d sec \n",currTime);
 88     int smallest;
 89     int i;
 90     while(remain!=0&&queueTimer>0)
 91     {
 92         smallest=r;
 93         for(i=0;i<r;i++)
 94         {
 95             if(p[i].arr<=currTime && p[i].prio<p[smallest].prio && p[i].bt>0)
 96             {
 97                 smallest=i;
 98             }
 99         }
100         if(p[smallest].start==-1)
101         {
102             p[smallest].start=currTime;
103         }
104         queueTimer-=1;
105         p[smallest].bt-=1;
106         if(p[smallest].bt==0&&p[smallest].ct==-1)
107         {
108             p[smallest].ct=currTime;
109             total--;
110             remain--;
111         }
112         currTime+=1;
```

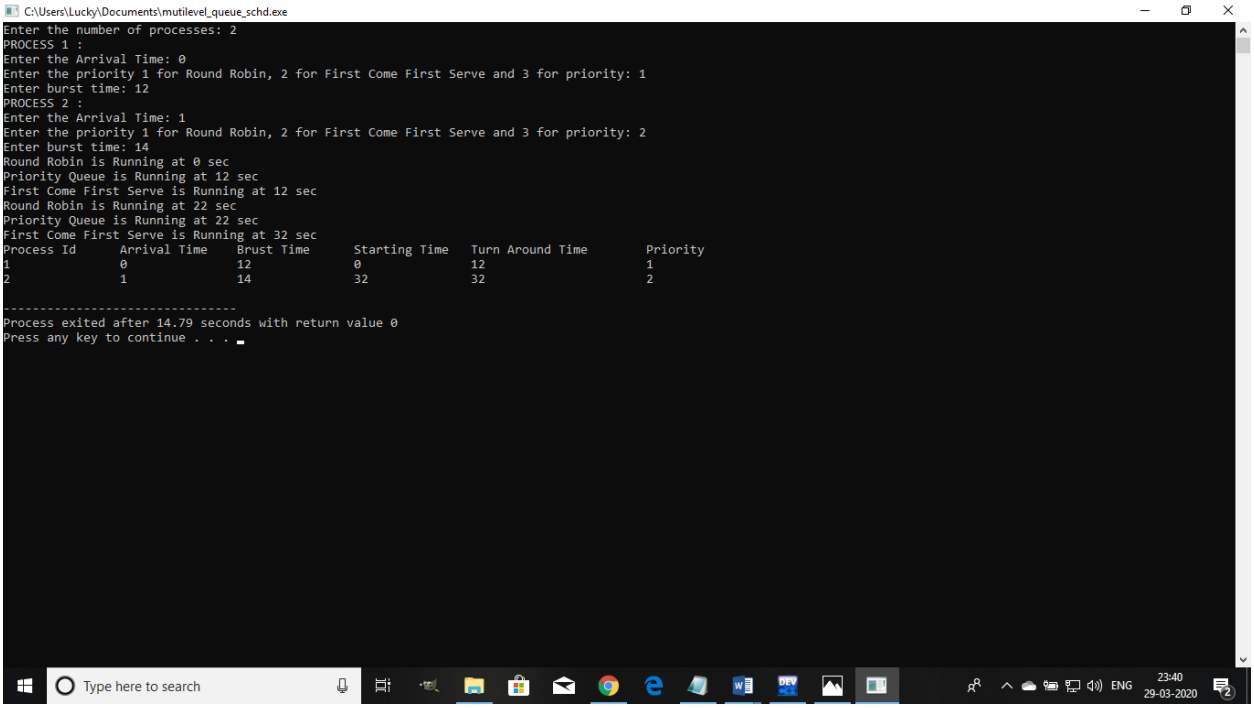


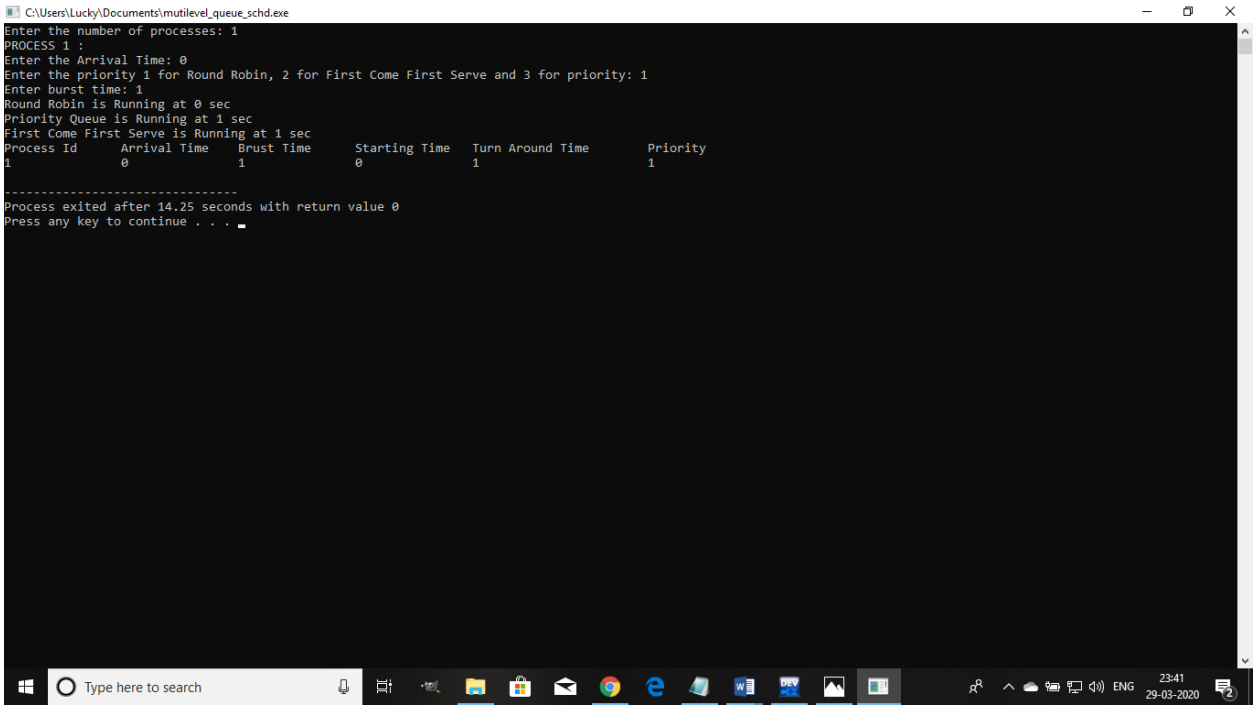
```
141         ps[i].start=-1;
142         ps[i].ct=-1;
143     }
144     int rrid=0,prid=0,fcfsid=0;
145     struct MutilevelQ roundqueue[rr],priorqueue[pr+1],fcfsqueue[fcfs];
146     for(i=0;i<n;i++)
147     {
148         if(ps[i].prio==1)
149         {
150             roundqueue[rrid]=ps[i];
151             rrid++;
152         }
153         else if(ps[i].prio==2)
154         {
155             fcfsqueue[fcfsid]=ps[i];
156             fcfsid++;
157         }
158         else if(ps[i].prio==3)
159         {
160             priorqueue[prid]=ps[i];
161             prid++;
162         }
163     }
164     int remain=pr;
165     while(total>0)
166     {
```


C:\Users\Lucky\Documents\mutilevel_queue_schd.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
mutilevel_queue_schd.exe
mutilevel_queue_schd.c
{
priorqueue[prid]=ps[i];
prid++;
}
}
int remain=pr;
while(total>0)
{
roundRobinAlgorithm(roundqueue,rr);
remain=priorityAlgo(priorqueue,pr,remain);
firstComeFirstServe(fcfsqueue,fcfs);
}
printf("Process Id\tArrival Time\tBrust Time\tStarting Time\tTurn Around Time\tPriority\n");
for(i=0;i<rr;i++)
{
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",roundqueue[i].p_id,roundqueue[i].arr,roundqueue[i].store,roundqueue[i].start,roundqueue[
}
for(i=0;i<fcfs;i++)
{
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",fcfsqueue[i].p_id,fcfsqueue[i].arr,fcfsqueue[i].store,fcfsqueue[i].start,fcfsqueue[i].ct,
}
for(i=0;i<pr;i++)
{
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",priorqueue[i].p_id,priorqueue[i].arr,priorqueue[i].store,priorqueue[i].start,priorqueue[
}
}
Compiler Resources Compile Log Debug Find Results
Line: 1 Col: 1 Sel: 0 Lines: 184 Length: 4196 Insert Done parsing in 0.266 seconds
Type here to search
ENG
23:30
29-03-2020

Test cases:


C:\Users\Lucky\Documents\mutilevel_queue_schd.exe
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 2
Enter burst time: 12
PROCESS 2 :
Enter the Arrival Time: 2
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 1
Enter burst time: 14
PROCESS 3 :
Enter the Arrival Time: 4
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 1
Enter burst time: 15
Round Robin is Running at 0 sec
Priority Queue is Running at 0 sec
First Come First Serve is Running at 0 sec
Round Robin is Running at 0 sec
Priority Queue is Running at 0 sec
First Come First Serve is Running at 10 sec
Round Robin is Running at 20 sec
Priority Queue is Running at 34 sec
First Come First Serve is Running at 34 sec
Round Robin is Running at 34 sec
Priority Queue is Running at 49 sec
First Come First Serve is Running at 59 sec
Process Id Arrival Time Brust Time Starting Time Turn Around Time Priority
2 2 14 20 34 1
3 4 15 34 49 1
1 1 12 34 34 2
Process exited after 25.5 seconds with return value 0
Press any key to continue . . .
Type here to search
ENG
23:41
29-03-2020


C:\Users\Lucky\Documents\mutilevel_queue_schd.exe
Enter the number of processes: 2
PROCESS 1 :
Enter the Arrival Time: 0
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 1
Enter burst time: 12
PROCESS 2 :
Enter the Arrival Time: 1
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 2
Enter burst time: 14
Round Robin is Running at 0 sec
Priority Queue is Running at 12 sec
First Come First Serve is Running at 12 sec
Round Robin is Running at 22 sec
Priority Queue is Running at 22 sec
First Come First Serve is Running at 32 sec
Process Id Arrival Time Brust Time Starting Time Turn Around Time Priority
1 0 12 0 12 1
2 1 14 32 32 2
Process exited after 14.79 seconds with return value 0
Press any key to continue . . .
Type here to search
ENG
23:40
29-03-2020


C:\Users\Lucky\Documents\mutilevel_queue_schd.exe
Enter the number of processes: 1
PROCESS 1 :
Enter the Arrival Time: 0
Enter the priority 1 for Round Robin, 2 for First Come First Serve and 3 for priority: 1
Enter burst time: 1
Round Robin is Running at 0 sec
Priority Queue is Running at 1 sec
First Come First Serve is Running at 1 sec
Process Id Arrival Time Brust Time Starting Time Turn Around Time Priority
1 0 1 0 1 1
Process exited after 14.25 seconds with return value 0
Press any key to continue . . .
Type here to search
ENG
23:41
29-03-2020