VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Modified Round Robin Algorithm

Slot:- L43+44

Reg No.:- 18BCI0113

Name:- PEDAMALLU LAKSHMI PURNA NIKHITA

Abstract:-

One of the most crucial problems in operating systems concepts is Scheduling the CPU to different processes and to design a particular system that will attain accurate results in scheduling along with high accuracy. In case of regular round robin algorithm, the time quantum is kept static, and requires all processes to wait for an entire cycle irrespective of remaining time. Due to this, processes with a very minimal remaining time also would have to wait for all other processes to execute until one time quantum or less is assigned to each process in queue. This leads to the increase in average waiting time of the sequence of processes. This issue would be more evident when there are many processes in the queue with a combination of large and small burst times. Along with that if the time quantum is higher, then the response time of the system will also be higher else If the time quantum is lower, then there is higher context switching overhead.

In order to increase the efficiency of the round robin algorithm, it is essential to decrease the average waiting time and average turnaround time for the set of processes being executed by the CPU. Along with that it is also required to consider the response time and context switches.

Hence, to overcome the above issues, a dynamic time quantum for the round robin scheduling algorithm was assigned such that the waiting time and turnaround time of processes reduce leading to a simultaneous decrease in average waiting time and average turnaround time.

Problem Statement:-

TO MINIMIZE THE WAITING TIME AND TURNAROUND TIME OF EACH PROCESS WHICH IN TURN MINIMIZES THE AVERAGE WAITING TIME AND AVERAGE TURNAROUND TIME OF SET OF PROCESSES.TO ENSURE REDUCED RESPONSE TIME MINIMIZED NUMBER OF CONTEXT SWITCHES TO AVOID LARGE OVERHEAD.

Proposed Model:-

This algorithm begins when first process request CPU cycle (i.e. Enters the ready queue). When a new process is added to the ready queue or a process finishes its given CPU cycle a cache called **sum_r** is assigned to store the values of sum of remaining burst times. Also, when a process enters the ready queue and finishes execution, a cache called **count** (initialized to 0) is incremented and decremented respectively. The value of time quantum changes in two cases: (i) when a new process enters the ready queue, (ii) when every process in the ready queue is executed once, twice and so on. The value of the time quantum everytime is changed to the concatenated value of **(sum_r/count)**.

Algorithm: Modified Round Robin Algorithm

Begin

I/P: sum_r1, sum_r2, sum_r3, count1, count2, count3, readyqueue1, readyqueue2, readyqueue3, quantum1, quantum2, quantum3, priority

New process P arrives

P enters ready queue[i] according to priority

Update sum_r[i] and count[i] according to the priority

quantum[i]=sum_r[i]/count[i] for next process

If (only one process arrives at t=0)

    quantum[i]=k

End if

Process P is loaded to CPU to get executed

While (all readyqueues are empty)

    While (readyqueue[i]!=null) **do**

        Update sum_r[i]

    If (P terminated)

        Update count[i]

        Load next process

    Else

        Move P to readyqueue[i+1];

    End else

    i++;

    End while

End while

Calculate waiting time and turnaround time for each process

Calculate average turnaround time and average waiting time

Code: Modified Round Robin Algorithm

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

```
#define MAX 200
int quantum1,quantum2,quantum3;



struct process{
    int pno,bt,rbt,ft,at,z,oq,priority,queue;
    int state; //state 0: executed atleast once   state 1: new process   state 2:first process in the cycle
};



int main(){
    register int count1=0,sum_r1=0,count2=0,sum_r2=0,count3=0,sum_r3=0;
    int flag1=0,flag2=0;
    int n,t=0;
    printf("Enter no.of processes (max 30): ");
    scanf("%d",&n);
    struct process input[30];
    struct process inpfinal[30];
    struct process readyqueue[30];
    //Input-----------------------------------
    for(int i=0;i<n;i++){
        input[i].pno=i;
        printf("Enter the arrival time for P%d: ",i);
```

```
        scanf("%d",&input[i].at);
        printf("Enter the burst time for P%d: ",i);
        scanf("%d",&input[i].bt);
        printf("Enter the priority for P%d  (lowest:1 -> highest:3): ",i);
        scanf("%d",&input[i].priority);
        input[i].rbt=input[i].bt;
        input[i].state=1;
        input[i].z=0;
        input[i].ft=0;
        input[i].oq=0;
        input[i].queue=0;
    }
//Input--------------------------------------


//soritng according to arrival time----------------


    int k=0,cq=1;
    while(k<=n){
        for(int i=0;i<n;i++){
            if(input[i].at==t){
                inpfinal[k]=input[i];
                k++;
            }
        }
        t++;
```

```
    }
    t=0;


    int top=0,i=0;


    int flag=0;
    while(flag<n){
        printf("At time %d:\n",t);
        for(int i=0;i<n;i++){    //finding the quantum for at=0 and also adding at=0 processes to
ready queue
        if(inpfinal[i].at==t){
            //-----------------------------------------------
            if(count1>=8&&count1<10){
                if(inpfinal[i].priority==3){// high priority to queue1
                    count1++;
                    inpfinal[i].queue=1;
                    sum_r1+=inpfinal[i].bt;
                    printf("P%d enters the ready queue 1 at t=%d\n",inpfinal[i].pno,t);
                    quantum1=sum_r1/count1;
                    printf("The quantum for queue 1 is %d\n", quantum1);
                    if(count1==8)
                        flag1=1;
                }
                else{
```

```
        if(count2<10 && flag2==0){//queue1 full
            count2++;
            inpfinal[i].queue=2;
            sum_r2+=inpfinal[i].bt;
            printf("P%d enters the ready queue 2 at t=%d\n",inpfinal[i].pno,t);
            quantum2=sum_r2/count2;
            printf("The quantum for queue 2 is %d\n", quantum2);
            if(count2==10)
                flag2=1;
        }
        else{//queue1 and queue2 full

            count3++;
            inpfinal[i].queue=3;
            sum_r3+=inpfinal[i].bt;
            printf("P%d enters the ready queue 3 at t=%d\n",inpfinal[i].pno,t);
            quantum3=sum_r3/count3;
            printf("The quantum for queue 3 is %d\n", quantum3);
        }
    }
}
else{
    if(flag1==0){//queue1
    count1++;
    inpfinal[i].queue=1;
```

```
            sum_r1+=inpfinal[i].bt;
            printf("P%d enters the ready queue 1 at t=%d\n",inpfinal[i].pno,t);
            printf("The quantum for queue 1 is %d\n", quantum1);
            quantum1=sum_r1/count1;
            if(count1==8)
                flag1=1;
            }
        }

        //-----------------------------------------------



        readyqueue[top]=inpfinal[i];
        top++;
    }
    else if(inpfinal[i].at==t+1)
        break;
  }
  if(count1==1&&t==0)
    quantum1=2;

  if(t==0)
    printf("Queue 1 has begun execution!\n");
// end-----------------------------------
```

```
if(readyqueue[i].z==0){//assigning oq the value of current quantum
        if(cq==1)
            readyqueue[i].oq=quantum1-1;
        else if(cq==2)
            readyqueue[i].oq=quantum2-1;
        else if(cq==3)
            readyqueue[i].oq=quantum3-1;
        printf("P%d gets CPU cycle\n",readyqueue[i].pno);
    }
readyqueue[i].rbt-=1;
if(readyqueue[i].rbt==0){// if process finishes exec.
        flag++;
        //printf("flag is %d\n",flag);
        readyqueue[i].ft=t+1;
        printf("Process%d finishes execution!\n",readyqueue[i].pno);
        readyqueue[i].state=3;

        if(cq==1)//removing z from quantum
            sum_r1-=readyqueue[i].z;
        else if(cq==2)
            sum_r2-=readyqueue[i].z;
        else if(cq==3)
            sum_r3-=readyqueue[i].z;

        if(cq==1)//decreasing count
```

```
    count1--;
else if(cq==2)
    count2--;
else if(cq==3)
    count3--;

if(cq==1){//   move to next queue
    if(count1==0&&flag1==1){
        cq=2;
        printf("Queue 2 has begun execution\n");
    }
}
else if(cq==2){
    if(count2==0){
        cq=3;
        printf("Queue 3 has begun execution\n");
    }
}
if(flag<n){
    while(1){//move to next live process of the current queue
        if(i==top)
            i=0;
        else
            i++;
        if(readyqueue[i].state!=3&&readyqueue[i].queue==cq)
```

```
                break;
            }
        }
    }
    else{
        if (readyqueue[i].z!=readyqueue[i].oq)//inc z till it becomes oq
            readyqueue[i].z++;
        else{
            if(cq==1){// moving process to next queue
                sum_r1-=readyqueue[i].z;
                if(count2!=10 && flag2!=1){
                    readyqueue[i].queue=2;
                    count1--;
                    count2++;
                }
                else{
                    readyqueue[i].queue=3;
                    count1--;
                    count3++;
                }
            }
            else if(cq==2){
                sum_r2-=readyqueue[i].z;
                readyqueue[i].queue=3;
                count2--;
```

```
    count3++;
  }
  else if(cq==3){
    sum_r3-=readyqueue[i].z;
  }
  readyqueue[i].z=0;



  printf("Process%d has been moved to queue
%d\n",readyqueue[i].pno,readyqueue[i].queue);


  if(cq==1){//   move to next queue
        if(count1==0&&flag1==1){
            cq=2;
            printf("Queue 2 has begun execution\n");


        }
    }
  else if(cq==2){
    if(count2==0){
        cq=3;
        printf("Queue 3 has begun execution\n");


    }
  }
```

```
            if(flag<n){
                while(1){//move to next live process of the current queue
                    if(i==top)
                        i=0;
                    else
                        i++;
                    if(readyqueue[i].state!=3&&readyqueue[i].queue==cq)
                        break;

                }
            }



        }


    }


    t++;
}// end of while



int wt[n],tat[n];
float awt=0,atat=0;
for(int q=0;q<n;q++){
    tat[q]=readyqueue[q].ft-readyqueue[q].at;
```

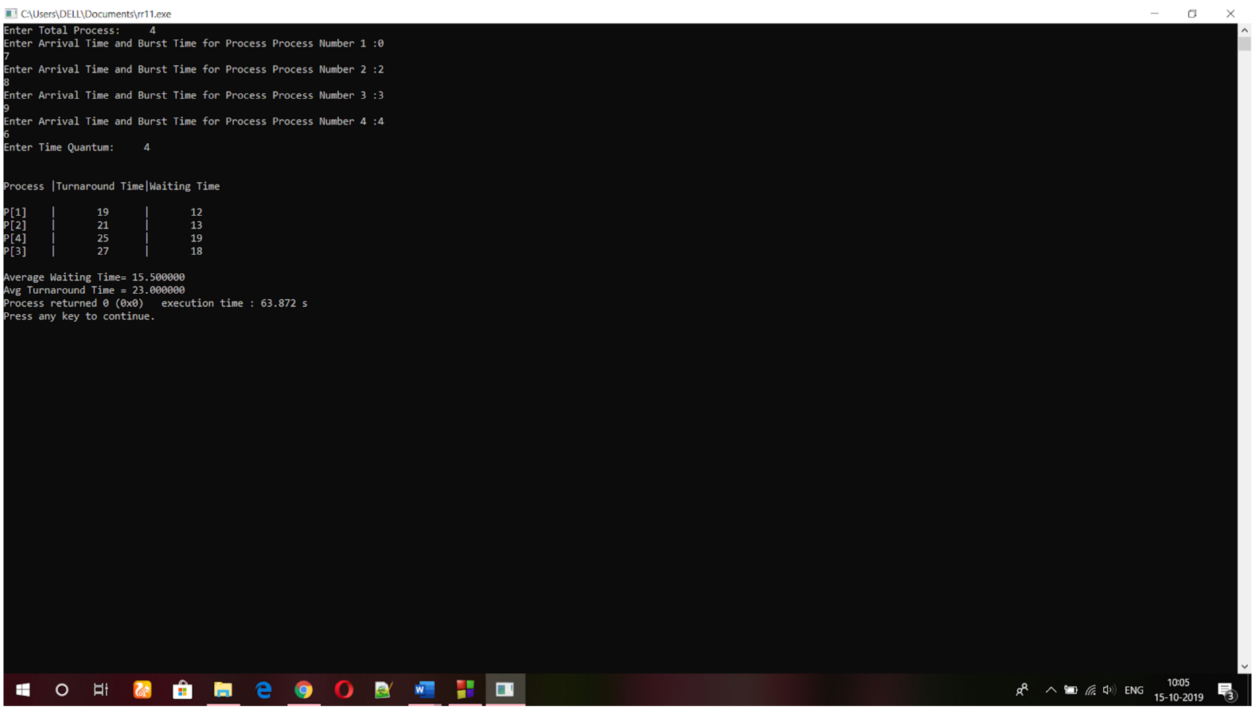
```
        atat+=tat[q];
        wt[q]=tat[q]-readyqueue[q].bt;
        awt+=wt[q];
    }

    atat=atat/n;
    awt=awt/n;
    printf("Process No. Arrival Time Finish time Waiting time Turnaround time\n");
    for(int l=0;l<n;l++){
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",readyqueue[l].pno,readyqueue[l].at,readyqueue[l].ft,wt[l],
tat[l]);
    }
    printf("Average Waiting time=%f\nAverage Turnaround time=%f\n",awt,atat);
    return 0;
}
```

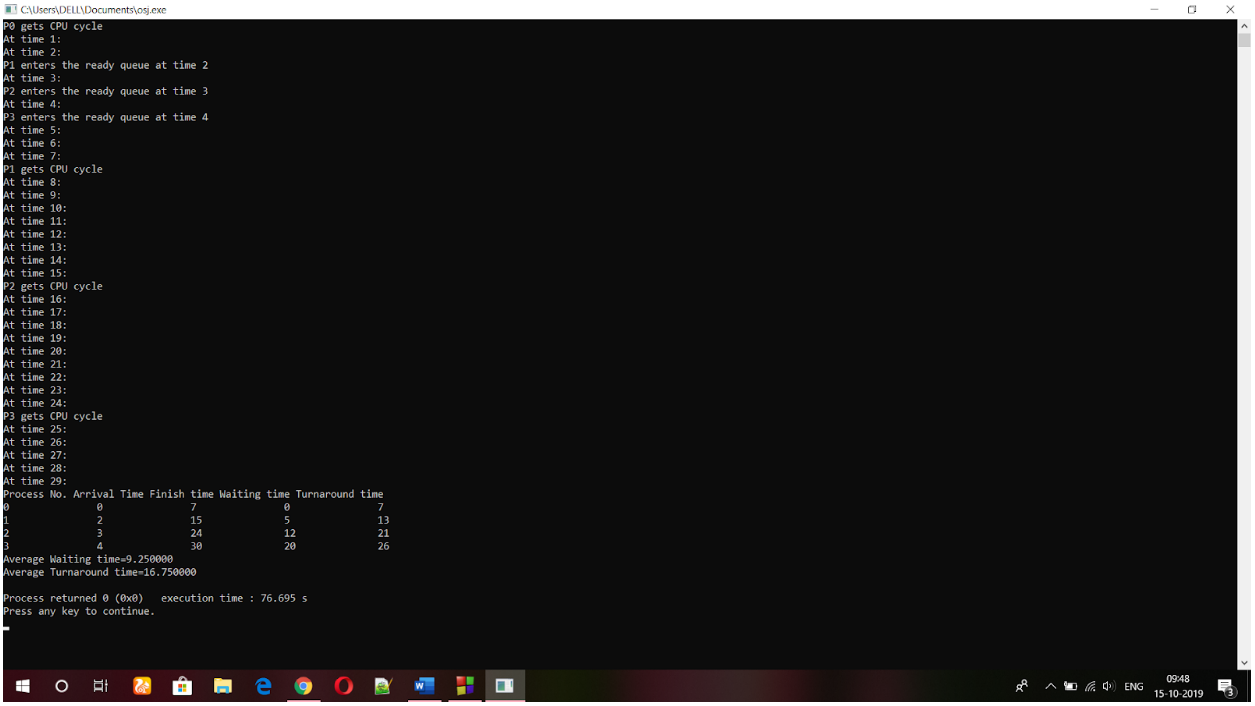
Comparative Analysis:-

Average waiting time and average turn around time are important factors which determine the efficiency of round robin algorithm. So we are comparing our algorithm with remaining round robin algorithms based on average waiting time and average turnaround time.The traditional Round Robin algorithm proves to be inefficient when compared to the modified Round Robin algorithm. A total of 4 processes with respective Arrival Time and Burst Time have been taken for the experiment as the data set. The same data set has been supplied to all 4 algorithms Traditional Round Robin , Modified Round Robin , Round Robin with Time quantum equals to 2 times of time quantum and Round Robin with Time quantum equals 0.3 times AT + 0.2 times BT. The traditional round robin gives 15.5 ms as the average Waiting Time and 23 ms as the average TAT. Whereas, the modified round robin gives 9.25 ms as the average waiting time and 16.75 ms as the average TAT. The other variation of Round robin with Time quantum equals to 2 times of time quantum gives 11ms as the average waiting time and 18.5 ms as the average TAT .The other variation of Round robin with Time quantum equals 0.3 times AT + 0.2 times BT gives 23 ms as the average waiting time and 17.75 ms as the average TAT .This depicts that the traditional Round Robin is not a good alternative when compared to the modified Round Robin.

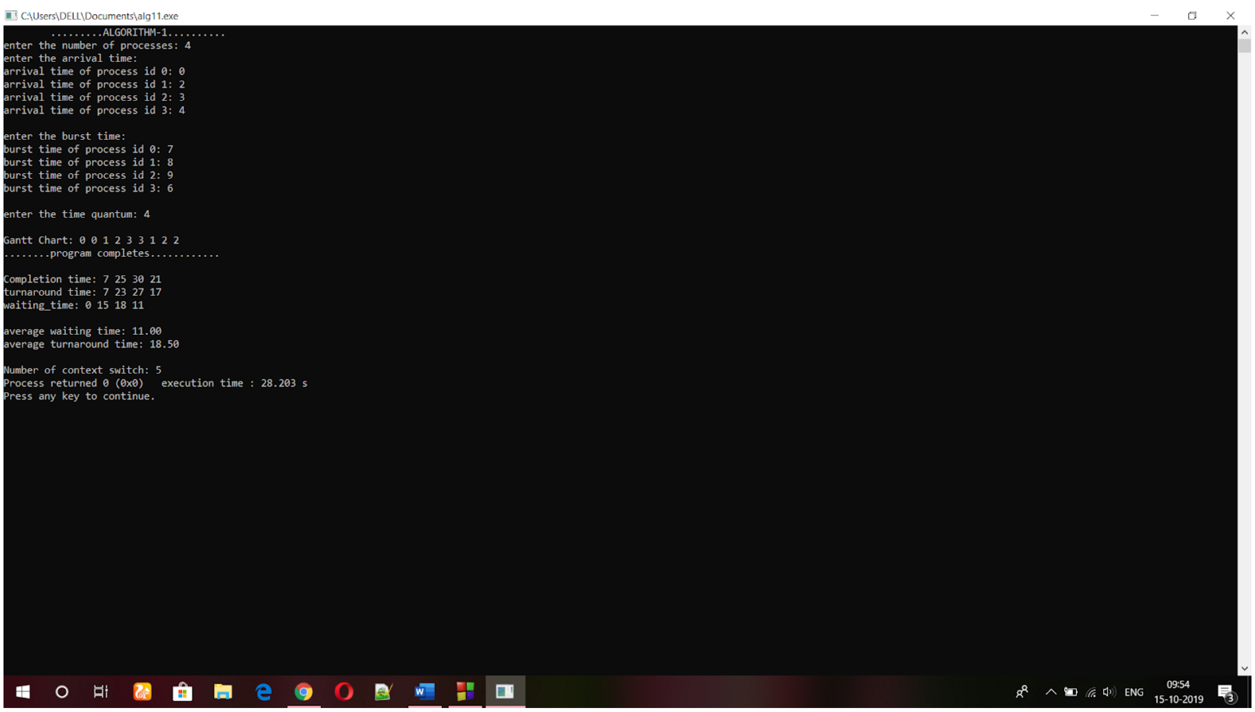
OUTPUT FOR SIMPLE ROUND ROBIN ALGORITHM:



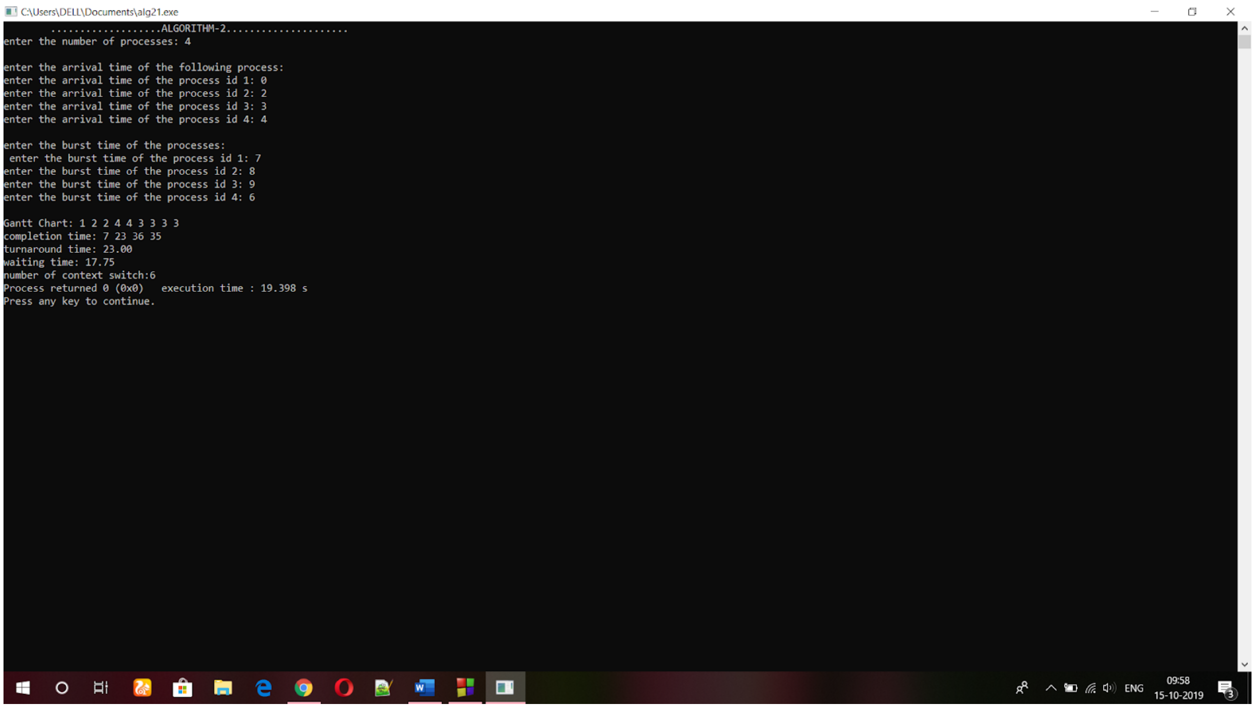
Modified round robin:



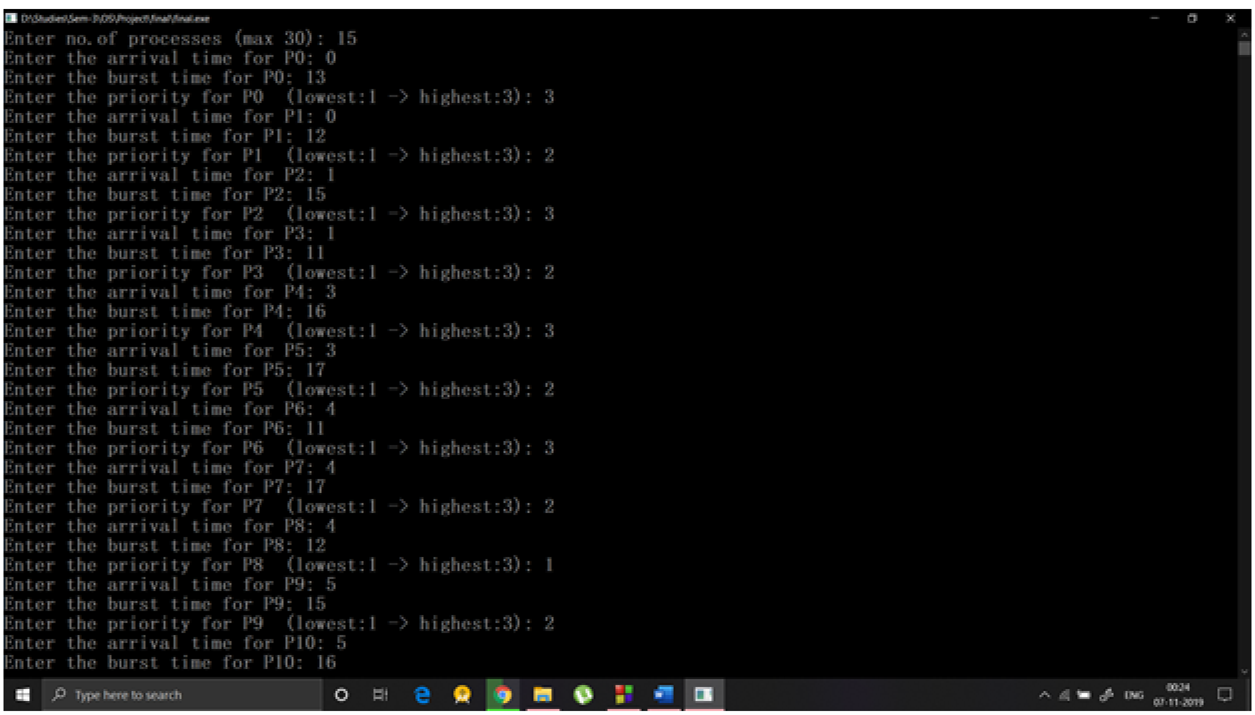
WHEN TIME QUANTUM IS 2*tq



WHEN TIME QUANTUM IS 0.3*at+0.2*bt

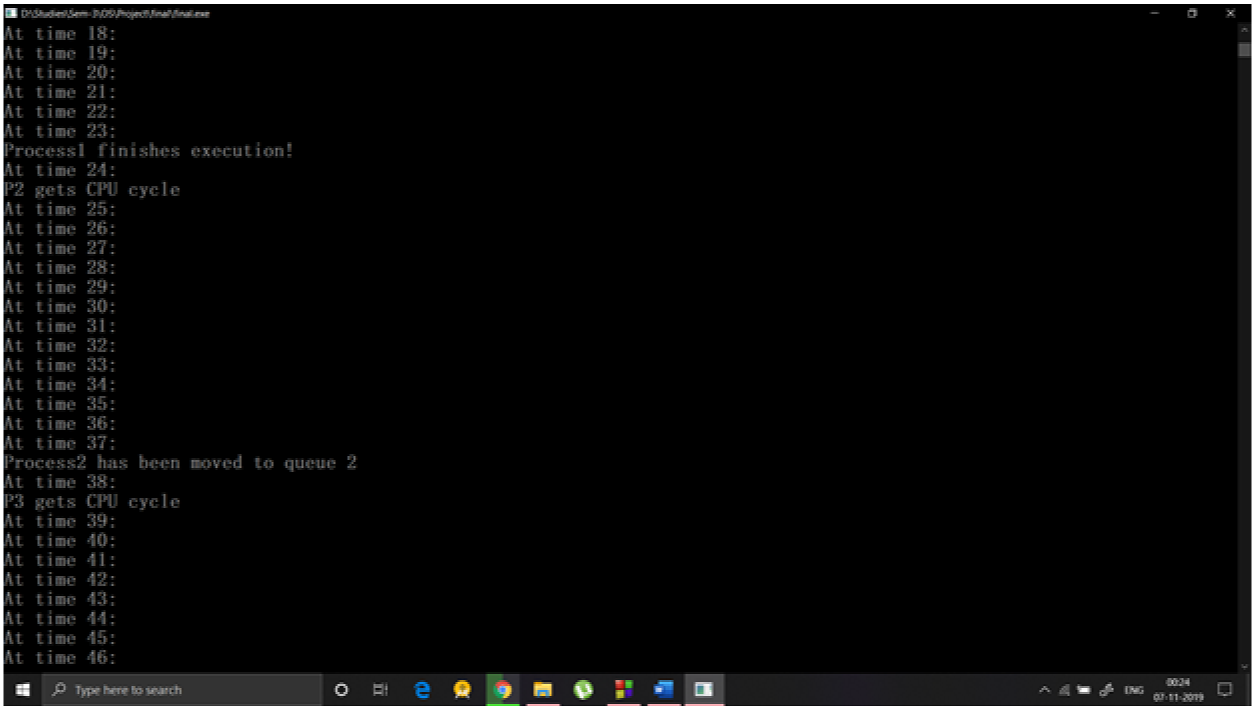


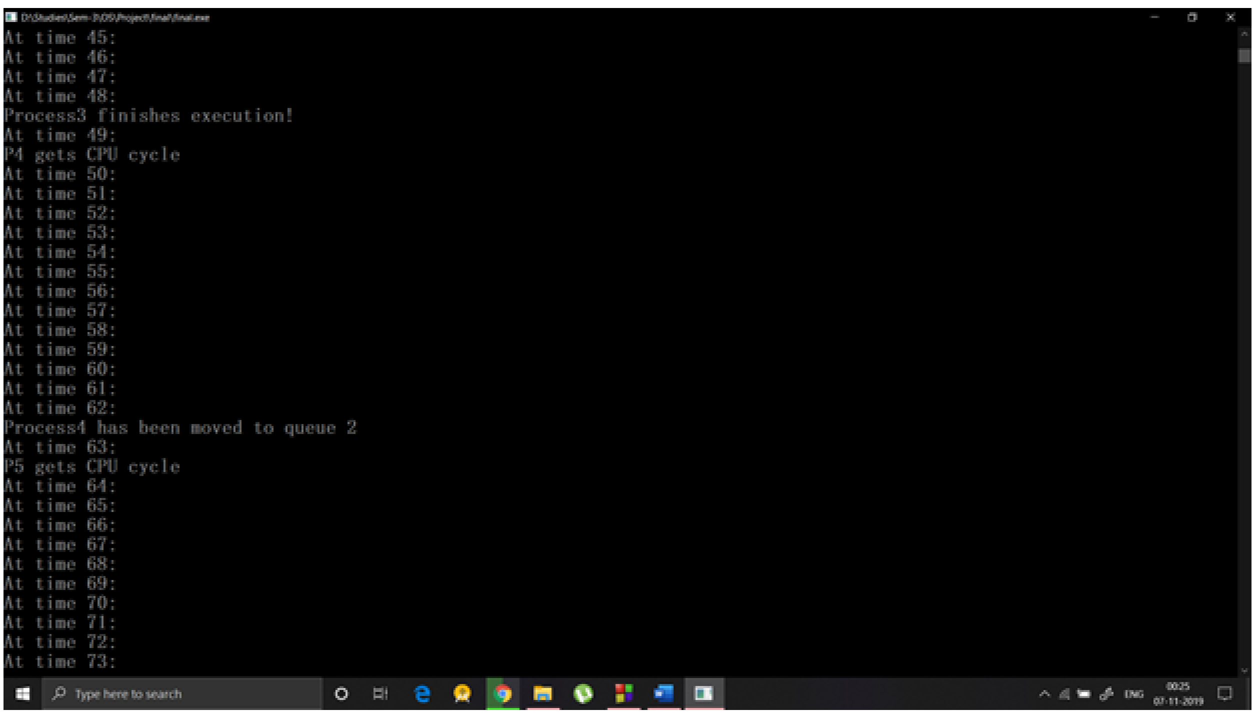
Review 3 modified code output:

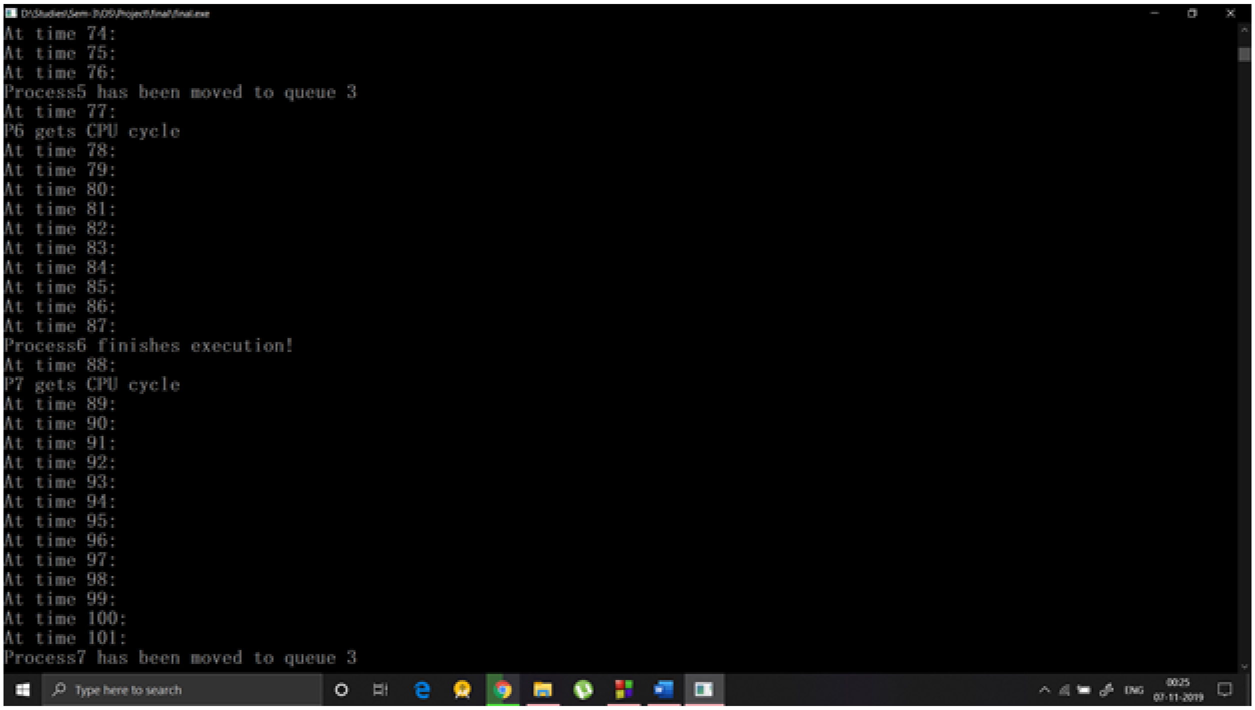


```
Enter the burst time for P10: 16
Enter the priority for P10  (lowest:1 -> highest:3): 1
Enter the arrival time for P11: 7
Enter the burst time for P11: 13
Enter the priority for P11  (lowest:1 -> highest:3): 2
Enter the arrival time for P12: 8
Enter the burst time for P12: 19
Enter the priority for P12  (lowest:1 -> highest:3): 2
Enter the arrival time for P13: 8
Enter the burst time for P13: 11
Enter the priority for P13  (lowest:1 -> highest:3): 1
Enter the arrival time for P14: 10
Enter the burst time for P14: 10
Enter the priority for P14  (lowest:1 -> highest:3): 2
At time 0:
P0 enters the ready queue 1 at t=0
The quantum for queue 1 is 0
P1 enters the ready queue 1 at t=0
The quantum for queue 1 is 13
Queue 1 has begun execution!
P0 gets CPU cycle
At time 1:
P2 enters the ready queue 1 at t=1
The quantum for queue 1 is 12
P3 enters the ready queue 1 at t=1
The quantum for queue 1 is 13
At time 2:
At time 3:
P4 enters the ready queue 1 at t=3
The quantum for queue 1 is 12
P5 enters the ready queue 1 at t=3
The quantum for queue 1 is 13
At time 4:
```

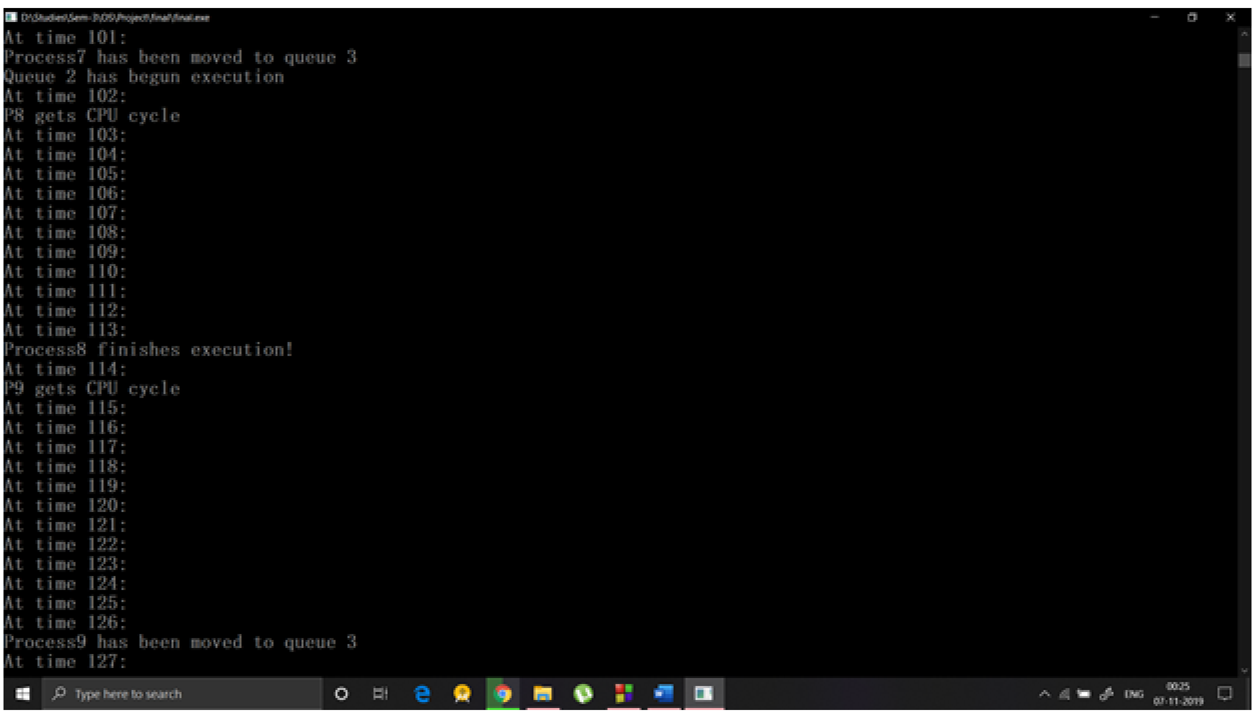
```
P6 enters the ready queue 1 at t=4
The quantum for queue 1 is 14
P7 enters the ready queue 1 at t=4
The quantum for queue 1 is 13
P8 enters the ready queue 2 at t=4
The quantum for queue 2 is 12
At time 5:
P9 enters the ready queue 2 at t=5
The quantum for queue 2 is 13
P10 enters the ready queue 2 at t=5
The quantum for queue 2 is 14
At time 6:
At time 7:
P11 enters the ready queue 2 at t=7
The quantum for queue 2 is 14
At time 8:
P12 enters the ready queue 2 at t=8
The quantum for queue 2 is 15
P13 enters the ready queue 2 at t=8
The quantum for queue 2 is 14
At time 9:
At time 10:
P14 enters the ready queue 2 at t=10
The quantum for queue 2 is 13
At time 11:
Process0 has been moved to queue 2
At time 12:
P1 gets CPU cycle
At time 13:
At time 14:
At time 15:
At time 16:
At time 17:
```


At time 18:
At time 19:
At time 20:
At time 21:
At time 22:
At time 23:
Process1 finishes execution!
At time 24:
P2 gets CPU cycle
At time 25:
At time 26:
At time 27:
At time 28:
At time 29:
At time 30:
At time 31:
At time 32:
At time 33:
At time 34:
At time 35:
At time 36:
At time 37:
Process2 has been moved to queue 2
At time 38:
P3 gets CPU cycle
At time 39:
At time 40:
At time 41:
At time 42:
At time 43:
At time 44:
At time 45:
At time 46:
Type here to search

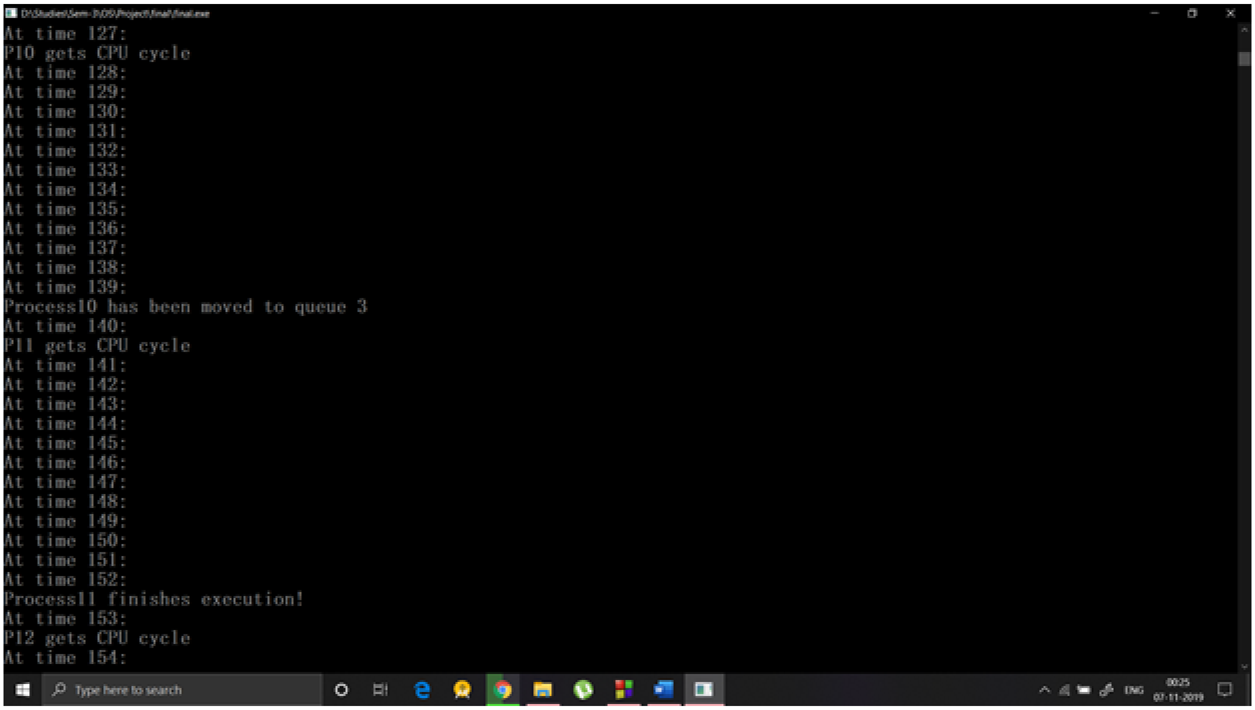

At time 45:
At time 46:
At time 47:
At time 48:
Process3 finishes execution!
At time 49:
P4 gets CPU cycle
At time 50:
At time 51:
At time 52:
At time 53:
At time 54:
At time 55:
At time 56:
At time 57:
At time 58:
At time 59:
At time 60:
At time 61:
At time 62:
Process4 has been moved to queue 2
At time 63:
P5 gets CPU cycle
At time 64:
At time 65:
At time 66:
At time 67:
At time 68:
At time 69:
At time 70:
At time 71:
At time 72:
At time 73:
Type here to search

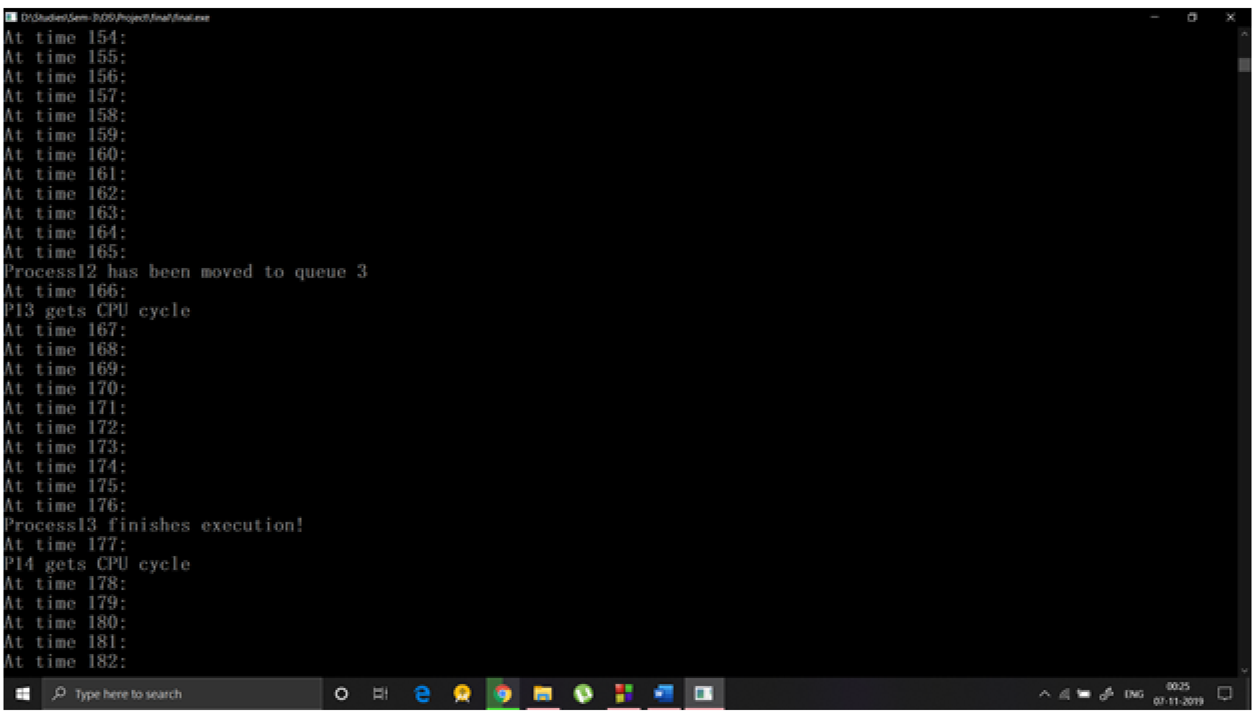


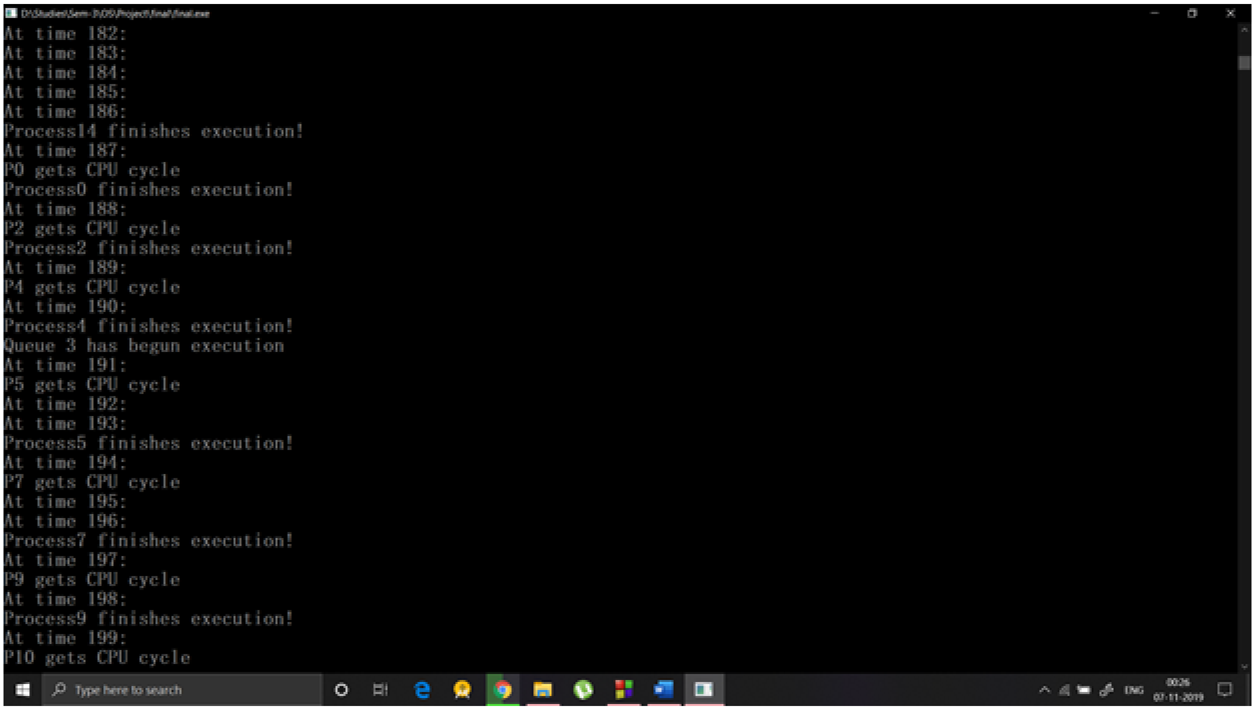
```
At time 74:
At time 75:
At time 76:
Process5 has been moved to queue 3
At time 77:
P6 gets CPU cycle
At time 78:
At time 79:
At time 80:
At time 81:
At time 82:
At time 83:
At time 84:
At time 85:
At time 86:
At time 87:
Process6 finishes execution!
At time 88:
P7 gets CPU cycle
At time 89:
At time 90:
At time 91:
At time 92:
At time 93:
At time 94:
At time 95:
At time 96:
At time 97:
At time 98:
At time 99:
At time 100:
At time 101:
Process7 has been moved to queue 3
```



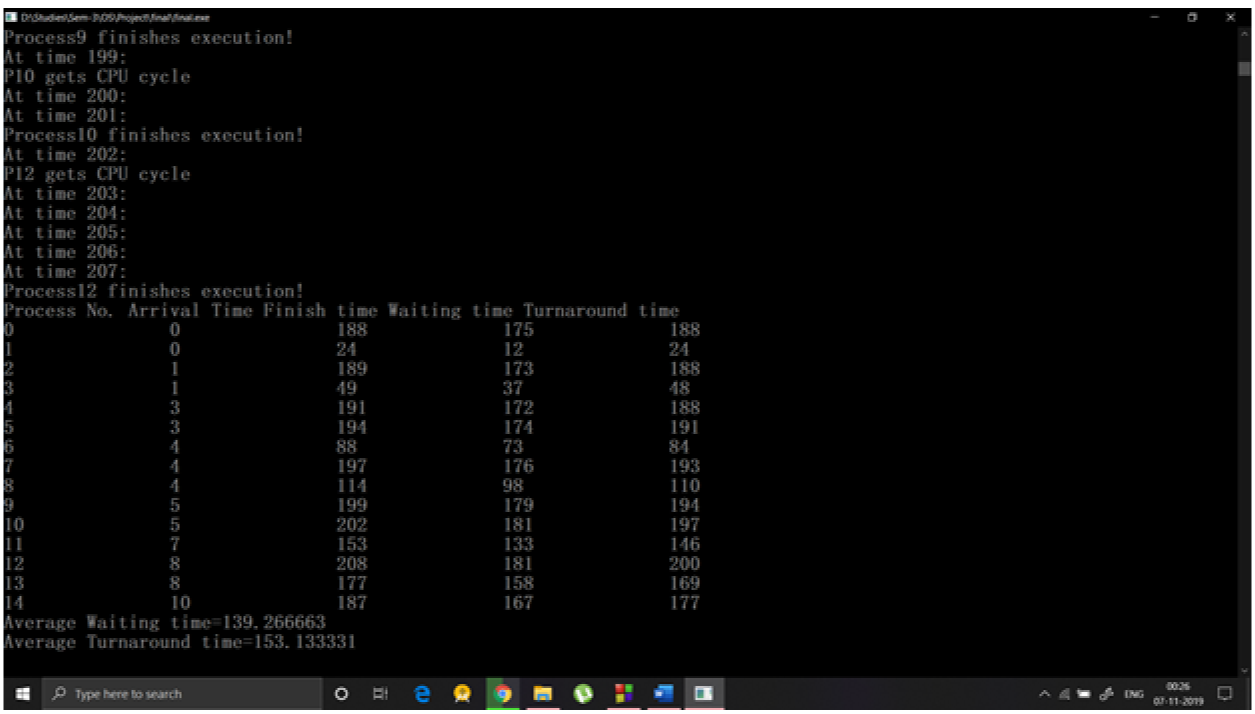
```
At time 101:
Process7 has been moved to queue 3
Queue 2 has begun execution
At time 102:
P8 gets CPU cycle
At time 103:
At time 104:
At time 105:
At time 106:
At time 107:
At time 108:
At time 109:
At time 110:
At time 111:
At time 112:
At time 113:
Process8 finishes execution!
At time 114:
P9 gets CPU cycle
At time 115:
At time 116:
At time 117:
At time 118:
At time 119:
At time 120:
At time 121:
At time 122:
At time 123:
At time 124:
At time 125:
At time 126:
Process9 has been moved to queue 3
At time 127:
```


At time 127:
P10 gets CPU cycle
At time 128:
At time 129:
At time 130:
At time 131:
At time 132:
At time 133:
At time 134:
At time 135:
At time 136:
At time 137:
At time 138:
At time 139:
Process10 has been moved to queue 3
At time 140:
P11 gets CPU cycle
At time 141:
At time 142:
At time 143:
At time 144:
At time 145:
At time 146:
At time 147:
At time 148:
At time 149:
At time 150:
At time 151:
At time 152:
Process11 finishes execution!
At time 153:
P12 gets CPU cycle
At time 154:
Type here to search


At time 154:
At time 155:
At time 156:
At time 157:
At time 158:
At time 159:
At time 160:
At time 161:
At time 162:
At time 163:
At time 164:
At time 165:
Process12 has been moved to queue 3
At time 166:
P13 gets CPU cycle
At time 167:
At time 168:
At time 169:
At time 170:
At time 171:
At time 172:
At time 173:
At time 174:
At time 175:
At time 176:
Process13 finishes execution!
At time 177:
P14 gets CPU cycle
At time 178:
At time 179:
At time 180:
At time 181:
At time 182:
Type here to search



```
At time 182:
At time 183:
At time 184:
At time 185:
At time 186:
Process14 finishes execution!
At time 187:
P0 gets CPU cycle
Process0 finishes execution!
At time 188:
P2 gets CPU cycle
Process2 finishes execution!
At time 189:
P4 gets CPU cycle
At time 190:
Process4 finishes execution!
Queue 3 has begun execution
At time 191:
P5 gets CPU cycle
At time 192:
At time 193:
Process5 finishes execution!
At time 194:
P7 gets CPU cycle
At time 195:
At time 196:
Process7 finishes execution!
At time 197:
P9 gets CPU cycle
At time 198:
Process9 finishes execution!
At time 199:
P10 gets CPU cycle
```



```
Process9 finishes execution!
At time 199:
P10 gets CPU cycle
At time 200:
At time 201:
Process10 finishes execution!
At time 202:
P12 gets CPU cycle
At time 203:
At time 204:
At time 205:
At time 206:
At time 207:
Process12 finishes execution!
Process No.  Arrival Time Finish time Waiting time Turnaround time
0               0             188          175          188
1               0             24           12           24
2               1             189          173          188
3               1             49           37           48
4               3             191          172          188
5               3             194          174          191
6               4             88           73           84
7               4             197          176          193
8               4             114          98           110
9               5             199          179          194
10              5             202          181          197
11              7             153          133          146
12              8             208          181          200
13              8             177          158          169
14              10            187          167          177
Average Waiting time=139.266663
Average Turnaround time=153.133331
```

Matrix :

Modified round robin algorithm which was proposed in this paper is compared withTraditional Round Robin , Modified Round Robin , Round Robin with Time quantum equals to 2 times of time quantum and Round Robin with Time quantum equals 0.3 times AT + 0.2 times . All these algorithms are compared with the proposed algorithm based on , average turn around time and average waiting time. Results show context average turn around time, average waiting time are less for proposed algorithm compared to remaining.

Conclusion and Future Work:-

In this paper, we presented an efficient round robin algorithm that yields best efficiency performance such as average waiting time and average turnaround time.This research work has provided a better solution to the classical Round Robin CPU scheduling algorithm.

In this algorithm, the base idea is to change the value of the time quantum(dynamic time quantum) as per the respective remaining time of process.(depends on the state of the process).This improves the average waiting and turn around time.

The algorithm can be operated within an algorithm of selecting the best scheduling algorithm dynamically i.e. based on the type of usage, different algorithms can be utilized together to give a better and more efficient process scheduling.

In future,we try to improve the execution time ,resource allocation and memory management.