ACADEMIC TASK 3 SIMULATION BASED

CSE316

STUDENT NAME :- Sanjay Das

STUDENT ID :-11802671

EMAIL ADDRESS :-sanjayfires11@gmail.com

GITHUB LINK : https://github.com/sd9989/RoundRobinPriorityScheduling/tree/master

Question:- . 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.

DESCRIPTION :-

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time. Waiting Time = Turn Around Time – Burst Time

Round Robin Scheduling: Each process is assigned a fixed time(Time Quantum/Time Slice) in cyclic way.It is designed especially for the time-sharing system. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum. To implement Round Robin scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than 1-time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1-time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

ALGORITHM :-

Time complexity: O(n)

Consider the following five processes each having its own unique burst time, priority and arrival time.

Process Queue	Burst time	Arrival time	Priority
P1	6	2	0
P2	2	5	1
P3	8	1	4
P4	3	0	3
P5	4	4	5

CODE:

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

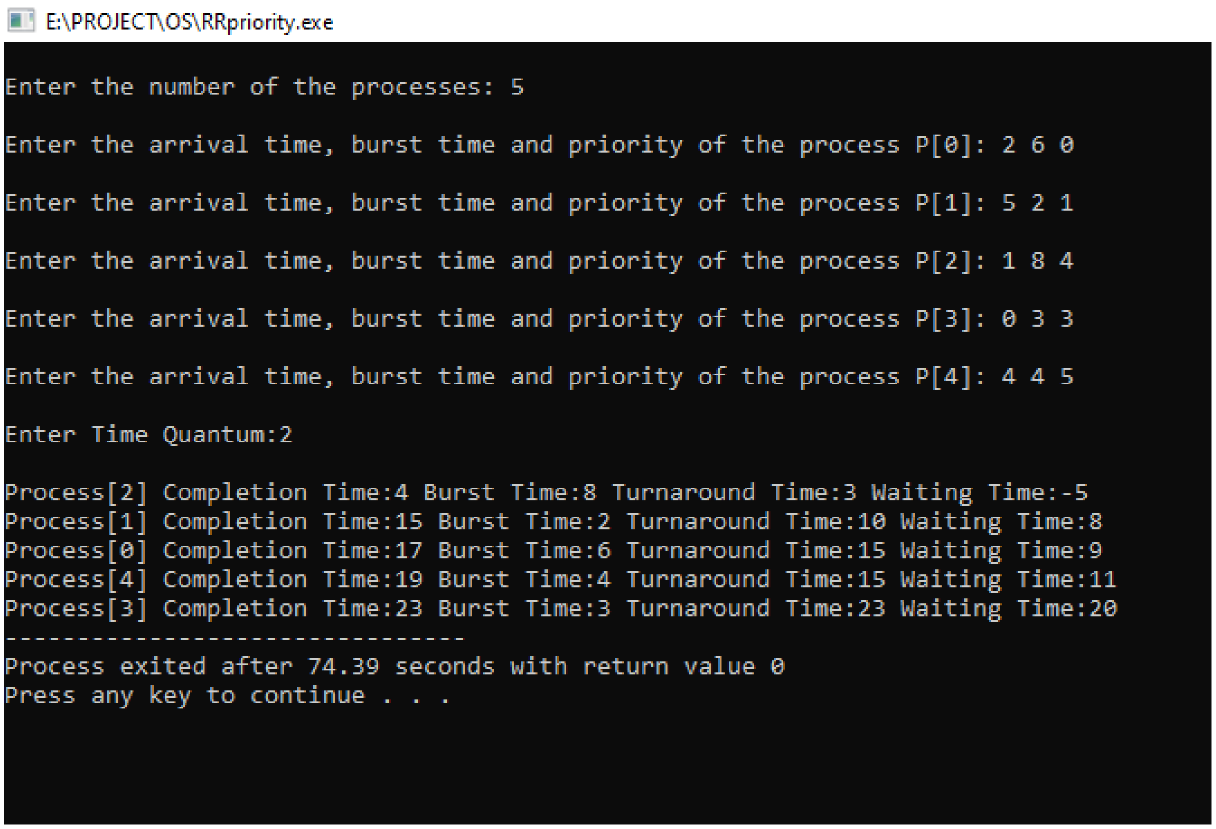
```
int main()
{



            int i,j, limit, total = 0, x, counter = 0, time_quantum,key,temp;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10],
temp2[10],priority[20],p[20];


    printf("\nEnter the number of the processes: ");

    scanf("%d",&limit);

    x=limit;

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

    {

      printf("\nEnter the arrival time, burst time and priority of the process P[%d]: ",i);

      scanf("%d",&arrival_time[i]);
                  scanf("%d",&burst_time[i]);
      temp2[i]=burst_time[i];
      scanf("%d",&priority[i]);
      p[i]=i;

    }
```

```
for(i=0;i<limit;i++)
{
  key=i;
  for(j=i+1;j<limit;j++)
  {
    if(priority[j]>priority[key])
    {
      key=j;
    }
  }
  temp=burst_time[i];
  burst_time[i]=burst_time[key];
  burst_time[key]=temp;

  temp=priority[i];
  priority[i]=priority[key];
  priority[key]=temp;

  temp=p[i];
  p[i]=p[key];
  p[key]=temp;

  temp=arrival_time[i];
  arrival_time[i]=arrival_time[key];
  arrival_time[key]=temp;
}
```

```
         printf("\nEnter Time Quantum:");
      scanf("%d", &time_quantum);



      for(total = 0, i = 0; x != 0;)
      {
            if(temp2[i] <= time_quantum && temp2[i] > 0)
            {
                  total = total + temp2[i];
                  temp2[i] = 0;
                  counter = 1;
            }
            else if(temp2[i] > 0)
            {
                  temp2[i] = temp2[i] - time_quantum;
                  total = total + time_quantum;
            }
            if(temp2[i] == 0 && counter == 1)
            {
                  x--;
                  printf("\nProcess[%d] Completion Time:%d Burst Time:%d Turnaround Time:%d
Waiting Time:%d", p[i], total, burst_time[i], total - arrival_time[i], total - arrival_time[i] -
burst_time[i]);
                  wait_time = wait_time + total - arrival_time[i] - burst_time[i];
                  turnaround_time = turnaround_time + total - arrival_time[i];
                  counter = 0;
            }
            if(i == limit - 1)
```

```
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= total)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }

    return 0;

}
```

OUTPUT:



```
E:\PROJECT\OS\RRpriority.exe

Enter the number of the processes: 5

Enter the arrival time, burst time and priority of the process P[0]: 2 6 0

Enter the arrival time, burst time and priority of the process P[1]: 5 2 1

Enter the arrival time, burst time and priority of the process P[2]: 1 8 4

Enter the arrival time, burst time and priority of the process P[3]: 0 3 3

Enter the arrival time, burst time and priority of the process P[4]: 4 4 5

Enter Time Quantum:2

Process[2] Completion Time:4 Burst Time:8 Turnaround Time:3 Waiting Time:-5
Process[1] Completion Time:15 Burst Time:2 Turnaround Time:10 Waiting Time:8
Process[0] Completion Time:17 Burst Time:6 Turnaround Time:15 Waiting Time:9
Process[4] Completion Time:19 Burst Time:4 Turnaround Time:15 Waiting Time:11
Process[3] Completion Time:23 Burst Time:3 Turnaround Time:23 Waiting Time:20
--------------------------------
Process exited after 74.39 seconds with return value 0
Press any key to continue . . .
```

BOUNDARY CONDITION:-

Processes can’t be greater than 20.

GITHUB LINK:- https://github.com/sd9989/RoundRobinPriorityScheduling/tree/master