Student Name: Kishan Kumar
Student ID: 11705028
Email Address: kumarkishan2601@gmail.com
GitHub Link: https://github.com/kumarkishan26/CSE316-Assignment.git

QUESTION NO 09:

Design a scheduler that uses a preemptive priority scheduling algorithm based on dynamically changing priority. Larger number for priority indicates higher priority.
Assume that the following processes with arrival time and service time wants to execute (for reference):

Process ID	Arrival Time	Service Time
P1	0	4
P2	1	1
P3	2	2
P4	3	1

When the process starts execution (i.e. CPU assigned), priority for that process changes at the rate of m=1.When the process waits for CPU in the ready queue (but not yet started execution), its priority changes at a rate n=2. All the processes are initially assigned priority value of 0 when they enter ready queue for the first time .The time slice for each process is q = 1. When two processes want to join ready queue simultaneously, the process which has not executed recently is given priority. Calculate the average waiting time for each process. The program must be generic i.e. number of processes, their burst time and arrival time must be entered by user.

DESCRIPTION:

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

ALGORITHM:

1. Priority is assigned for each process.
2. Process with highest priority is executed first and so on.
3. Processes with same priority are executed in FCFS manner.
4. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Source Code:

```
#include<stdio.h>

    struct process
    {
        char p_name;
        int at, bt, ct, wt, tat, priority;
        int status;
    }p_queue[10];

    int limit;

    void at_Sorting()
    {
        struct process temp;
        int i, j;
        for(i = 0; i < limit - 1; i++)
        {
            for(j = i + 1; j < limit; j++)
            {
                if(p_queue[i].at > p_queue[j].at)
                {
                    temp = p_queue[i];
                    p_queue[i] = p_queue[j];
                    p_queue[j] = temp;
                }
            }
        }
    }

    void main()
    {
        int i, time = 0, bt = 0, largest;
        char c;
        float wait_time = 0, tat = 0, Avg_wt, Avg_tat;
        printf("\nEnter Total Number of Processes:\t");
        scanf("%d", &limit);
        for(i = 0, c = 'A'; i < limit; i++, c++)
        {
            p_queue[i].p_name = c;
            printf("\nEnter Details For Process[%C]:\n", p_queue[i].p_name);
            printf("Enter Arrival Time:\t");
            scanf("%d", &p_queue[i].at );
            printf("Enter Burst Time:\t");
```

```
            scanf("%d", &p_queue[i].bt);
            printf("Enter Priority:\t");
            scanf("%d", &p_queue[i].priority);
            p_queue[i].status = 0;
            bt = bt + p_queue[i].bt;
        }
        at_Sorting();
        p_queue[9].priority = -9999;
        printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");
        for(time = p_queue[0].at; time < bt;)
        {
            largest = 9;
            for(i = 0; i < limit; i++)
            {
                if(p_queue[i].at <= time && p_queue[i].status != 1 && p_queue[i].priority
> p_queue[largest].priority)
                {
                    largest = i;
                }
            }
            time = time + p_queue[largest].bt;
            p_queue[largest].ct = time;
            p_queue[largest].wt = p_queue[largest].ct - p_queue[largest].at -
p_queue[largest].bt;
            p_queue[largest].tat = p_queue[largest].ct - p_queue[largest].at;
            p_queue[largest].status = 1;
            wait_time = wait_time + p_queue[largest].wt;
            tat = tat + p_queue[largest].tat;
            printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", p_queue[largest].p_name,
p_queue[largest].at, p_queue[largest].bt, p_queue[largest].priority, p_queue[largest].wt);
        }
        Avg_wt = wait_time / limit;
        Avg_tat = tat / limit;
        printf("\n\nAvg waiting time:\t%f\n", Avg_wt);
        printf("Avg Turnaround Time:\t%f\n", Avg_tat);

    }
```

Complete Solution: -

1. **Completion Time:** Time at which process completes its execution.
2. **TurnAround Time:** Time Difference between completion time and arrival time.
   - Turn Around Time = Completion Time – Arrival Time
3. **Waiting Time:** Time Difference between turn around time and burst time.
   - Waiting Time = Turn Around Time – Burst Time

ANSWER:

Process	Arrival-Time	Service-Time
P1	0	4
P2	1	1
P3	2	2
P4	3	1

Enter no of Processes n

: 4

Arrival Time of P[1] ::

0

Service Time of P[1] ::

4

Enter priority ::1

Arrival Time of P[2] ::

1

Service Time of P[2] ::

1

Enter priority::2

Arrival Time of P[3] ::

2

service Time of P[3] ::

2

Enter priority::3

Arrival Time of P[4] ::

3

Service Time of P[4] ::

1

Enter priority::4

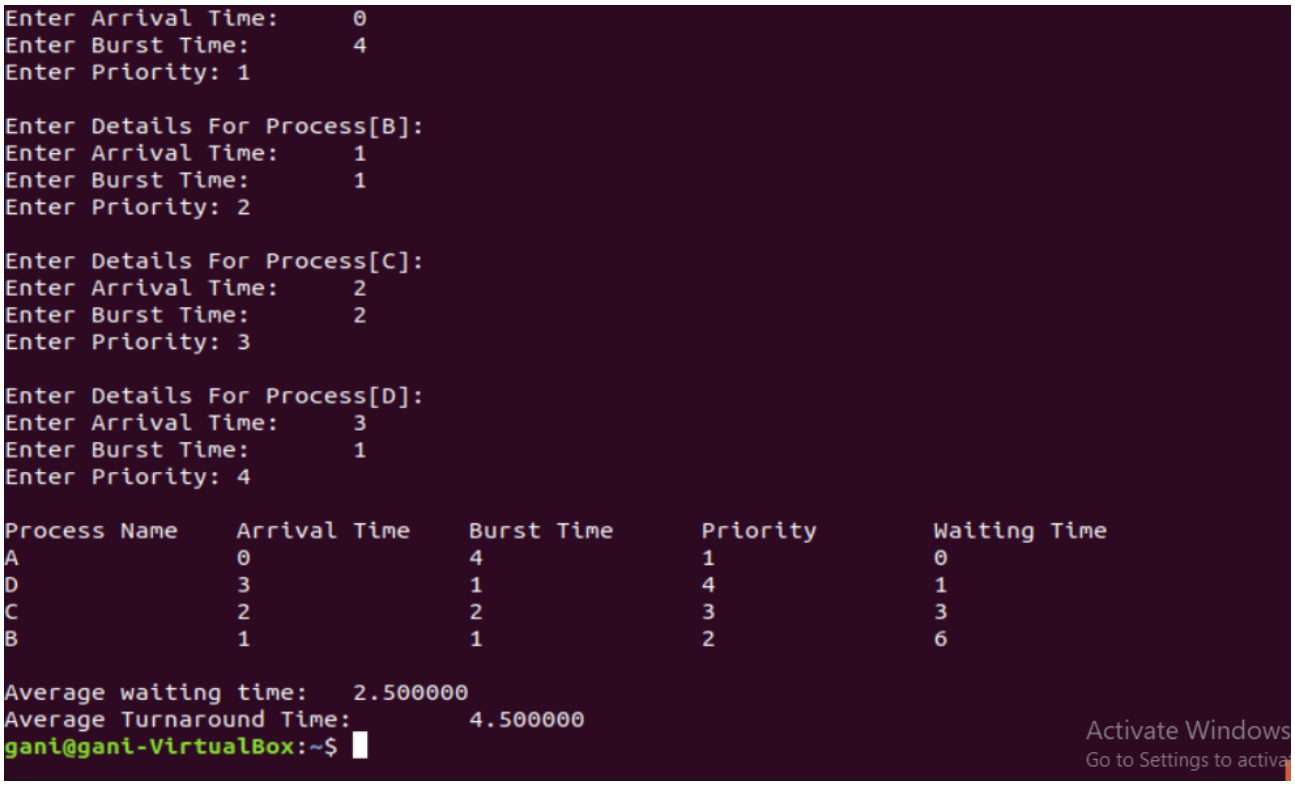
Process	**Turn Around Time**
P[1]	2
P[2]	3
P[3]	1
P[4]	4

The Average Turn Around Time is 4.5000000

Process	**Waiting Time**
P[1]	0
P[2]	6
P[3]	3
P[4]	1

The Average Waiting Time is 2.500000.

Output for Given Program:-



Test Cases: -

Test Case1: -

Process	**ArrivalTime**	**BurstTime**	**Waiting. T**
P1	0	6	0
P2	1	8	5
P3	9	5	5

Avg Turn Around Time :- 9.66666

Avg Waiting Time :- 3.33333

```
Enter Total Number of Processes:        3

Enter Details For Process[A]:
Enter Arrival Time:     0
Enter Burst Time:       6
Enter Priority: 1

Enter Details For Process[B]:
Enter Arrival Time:     1
Enter Burst Time:       8
Enter Priority: 2

Enter Details For Process[C]:
Enter Arrival Time:     9
Enter Burst Time:       5
Enter Priority: 3

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
A               0               6               1               0
B               1               8               2               5
C               9               5               3               5

Average waiting time:   3.333333
Average Turnaround Time:        9.666667
gani@gani-VirtualBox:~$
```

Test Case2: -

Process	**ArrivalTime**	**BurstTime**	**Waiting. T**
P1	0	4	**0**
P2	2	5	3
P3	3	9	7
P4	5	2	7
P5	4	1	11
P6	1	7	21

Average Turn Around Time :- 12.83333
Average Waiting Time :- 8.16666
Reference: -

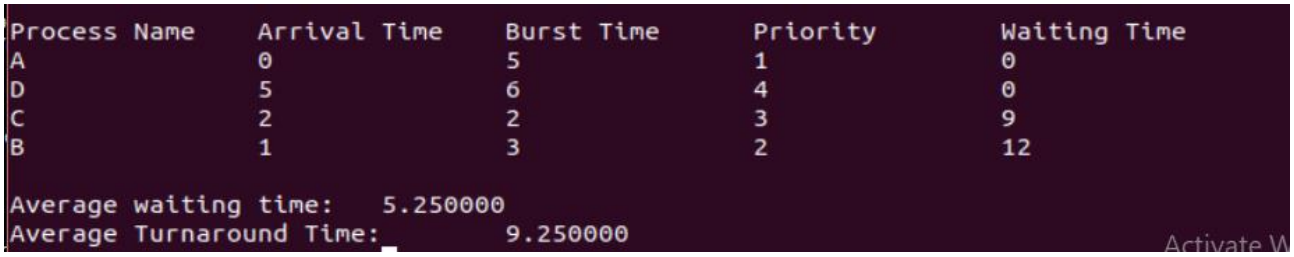
```
Process Name    Arrival Time    Burst Time      Priority        Waiting Time
A               0               4               1               0
F               1               7               6               3
E               4               1               5               7
D               5               2               4               7
C               3               9               3               11
B               2               5               2               21

Average waiting time:   8.166667
Average Turnaround Time:        12.833333
```

Test Case3: -

Process	ArrivalTime	BurstTime	Waiting. T
P1	0	5	**0**
P2	1	3	0
P3	2	2	9
P4	5	6	12

Average Turn Around Time :- 9.25000
Average Waiting Time :- 5.2500000
Reference: -



Constraints: -

Some Constraints used in my scheduling program are :

- Condition used for 2 units time to take by the CPU in the process

```
for(i=2;i<n;i++)
{
r=r+1;
readyQueue1[r]=i+1;
}
```

- To check whether all the processes are completed their execution or not

```
int IsAllExecuted()
{
int i;
for(i=0;i<2;i++)
{
```

```
if(process[i][2]>0)
break;
}
if(i<2)
return 1;
return 0;
}
```

- Here is the constraint of the given question to take the value of the no. of processes to execute and details are given by the user.

```
printf("\n Enter no of Processes n : ");
scanf("%d",&n);
```

- Also the compiler will take the values of arrival and burst time by the user as mentioned in question

```
for(i=0;i<n;i++)
{
printf("Enter ArrivalTime of P[%d] :: ",i+1);
scanf("%d",&arrivalTime[i]);
printf("Enter BurstTime of P[%d] :: ",i+1);
scanf("%d",&burst[i]);
process[i][1]=arrivalTime[i];
process[i][2]=burst[i];
}
```

Boundary Condition: -

Here the main boundary condition is to execute the process maximum of 2 units time and to then it holds the process after the 2 units. In this period of time the other process will execute which has less arrival time as compare to previous holding process and it will go on with this boundary condition. If the time limit exceeds for a single process among all processes then the program or the output will become incorrect.