**Student Name: Deeksha Tripathi**

**Student ID: 11710413**

**Email Address:** [**deeksha18799@gmail.com**](mailto:deeksha18799@gmail.com)

**GitHub Link:** [**https://github.com/DeekshaTripathi29**](https://github.com/DeekshaTripathi29)

**Code: will find on the github repositories name CSE\_316-Project.**

**Problem:**

In operating system cpu scheduling, there are many algorithms available. But every algorithm has some sort of drawback. First we are going to have a look at FCFS algorithm. This algorithm works on the first come and first serve basis. If the process come at the first, going to be execute at first. Drawback of this is that if a process come at first but have large burst time so process which come at last can meet the problem of starvation. Same type of thing happens with SJF, processes which have large burst time can go to starvation. To overcome this robin round come in light. In round robin we have a fixed time period called time quantum for which every process is going to be execute whether the process has burst time high or low. By doing so we allocate cpu to every process which avoids the processes going to starvation

But in our solution to the problems is a multilevel queue algorithm. In which we use two queues. The multilevel queue algorithm uses existing algorithm to maintain the process with common features. The purpose of multilevel scheduling algorithm is to work in the situation the where processes are needed to categorized in different groups.

**Algorithm:**

Step 1. Enter process ID, Arrival time, Burst Time and Priority.

Step 2. Store the data in a queue.

Step 3. Crate two queue for higher priorities and lower priorities respectively.

Step 4. Do step 5 to 11 until remaining burst time of both queues become zero.

Step5. Select any process from any queue on alternate basis.

Step 6. If the remain burst time of the selected process is greater or equal to the time quantum then do step 7 otherwise do step 8.

Step 7. Execute process for the duration of time quantum.

Step 8. Execute the process until its burst time become zero.

Step 9. Update the burst time of the process in the queue.

Step 10. Calculate the completion and waiting time by their formula

Step 11. Print all the details at screen.

**Complexity:**

Complexity of this algorithm is based on the number of process that is given in input by user.

Because the loops run for the time equals to the number of process so that the complexity is going to be O(n).

Overall complexity of the program is also O(n).

**Boundary condition:**

Our algorithm can take maximum 1000 input processes.

Waiting time can’t be negative number and boundary condition for time quantum is that it can’t be less than 2 seconds. If we do this then there will be so much context switching which will lead to high complexity cause of accessing the stored data of process by context switching. Time quantum should be normal not too less nor too high.

**Test Cases:**

**Test - case 1 :**

**Input :** we assume that the user inputs 5 process as follow-

**pid BT AT priority**

1 5 0 2

2 3 3 3

3 9 5 1

4 8 2 4

5 11 1 5

**Output :** Program gives output as follow -

**pid response time CT TAT WT**

1 0 18 18 13

2 1 20 17 14

3 0 33 28 19

4 7 28 26 18

5 12 36 35 24

**average waiting time** **:** 17.600

**average TAT time :** 24.79

**Test Case 2:**

**INPUT :**

**pid AT BT PRIORITY**

1 0 4 3

2 2 5 2

3 5 4 6

4 7 3 4

5 4 6 8

6 17 8 9

**Outputs :**

**PID RT CT TAT WT**

1 0 20 0 16

2 0 21 19 14

3 1 24 19 15

4 0 10 3 0

5 6 26 22 16

6 7 30 23 15

**AVG WT**: 12.66

**AVG TAT**: 17.66

**Test Case 3 :**

**Input :**

**PID AT BT Priority**

1 0 3 1

2 1 2 3

3 1 4 6

4 2 5 5

5 3 6 4

**Outputs :**

**PID RT CT TAT WT**

1 0 3 3 0

2 2 5 4 2

3 12 17 16 12

4 7 20 18 13

5 2 19 16 10

**AVG WT:** 7.400

**AVG TAT:** 11.40

**GitHub Link:** [**https://github.com/DeekshaTripathi29**](https://github.com/DeekshaTripathi29)**/CSE\_316-Project**