**Student Name: Mohare Pavan Kumar**

**Student ID: 11705249**

**Email Address: pavankumar0366@gmail.com**

**GitHub Link:**

# QUESTION NO 13:

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  i.e. to apply round robin algorithm OF 10 seconds on over all structure.

Calculate Waiting time and turnaround time for every process. The input for number of processes  should be given by the user.

**DESCRIPTION:**

Multi-level queue scheduling algorithm is used in scenarios where the processes can be classified into groups

based on property like process type, CPU time, IO access, memory size, etc. In a multi-level queue scheduling

algorithm, there will be 'n' number of queues, where 'n' is the number of groups the processes are classified

into. Each queue will be assigned a priority and will have its own scheduling algorithm like round-robin

scheduling or FCFS. For the process in a queue to execute, all the queues of priority higher than it should be

empty, meaning the process in those high priority queues should have completed its execution. In this

scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

**ALGORITHM:**

1. Take three queue’s.
2. Here Q2 is “First come First serve”.
3. Roundrobin with quantum 4 seconds it is consider as Q0.
4. Roundrobin with quantum 10 seconds it is consider as Q1.
5. A new job enters queue *Q0* which is servedFCFS. When it gains CPU, job receives 4 seconds. If it does not finish in 4 seconds, job is moved to queue *Q*1.
6. At *Q*1 job is again served FCFS and receives 10 additional seconds. If it still does not complete, it is preempted and moved to queue *Q*2

**Entire Code:**

|  |
| --- |
| #include<stdio.h> |
|  |  |
|  | int main() |
|  | { |
|  | int i, limit, total = 0, x, counter = 0, time\_quantum,j; |
|  |  |
|  | int wait\_time = 0, turnaround\_time = 0,pos,z,p[10],prio[10], a\_time[10], b\_time[10], temp[10],b; |
|  |  |
|  | float average\_wait\_time, average\_turnaround\_time; |
|  |  |
|  | printf("\nEnter Total Number of Processes:"); |
|  |  |
|  | scanf("%d", &limit); |
|  |  |
|  | x = limit; |
|  | for(i = 0; i < limit; i++) |
|  | { |
|  | p[i]=i+1; |
|  |  |
|  | prio[i]=0; |
|  | printf("\nEnter total Details of Process[%d]\n", i + 1); |
|  | printf("Arrival Time:\t"); |
|  | scanf("%d", &a\_time[i]); |
|  | printf("Burst Time:\t"); |
|  | scanf("%d", &b\_time[i]); |
|  | temp[i] = b\_time[i]; |
|  | } |
|  |  |
|  | printf("\nEnter the Time Quantum:"); |
|  | scanf("%d", &time\_quantum); |
|  | printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\t Priority\n"); |
|  | for(total = 0, i = 0; x != 0;) |
|  | { |
|  |  |
|  | for(z=0;z<limit;z++) |
|  | { |
|  | int temp1; |
|  | pos=z; |
|  | for(j=z+1;j<limit;j++) |
|  | { |
|  | if(prio[j]<prio[pos]) |
|  | pos=j; |
|  | } |
|  |  |
|  | temp1=prio[z]; |
|  |  |
|  | prio[z]=prio[pos]; |
|  |  |
|  | prio[pos]=temp1; |
|  |  |
|  | temp1=b\_time[z]; |
|  | b\_time[z]=b\_time[pos]; |
|  | b\_time[pos]=temp1; |
|  | temp1=a\_time[z]; |
|  | a\_time[z]=a\_time[pos]; |
|  | a\_time[pos]=temp1; |
|  |  |
|  | temp1=p[z]; |
|  | p[z]=p[pos]; |
|  | p[pos]=temp1; |
|  |  |
|  | temp1=temp[z]; |
|  | temp[z]=temp[pos]; |
|  | temp[pos]=temp1; |
|  | } |
|  | { |
|  | } |
|  |  |
|  | if(temp[i] <= time\_quantum && temp[i] > 0) |
|  | { |
|  | total = total + temp[i]; |
|  | temp[i] = 0; |
|  | counter = 1; |
|  | } |
|  |  |
|  | else if(temp[i] > 0) |
|  | { |
|  | temp[i] = temp[i] - time\_quantum; |
|  | total = total + time\_quantum; |
|  | } |
|  |  |
|  | for(b=0;b<limit;b++) |
|  | { |
|  | if(b==i) |
|  | prio[b]+=1; |
|  | else |
|  | prio[b]+=2; |
|  | } |
|  |  |
|  | if(temp[i] == 0 && counter == 1) |
|  | { |
|  | x--; |
|  | printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\t\t%d", p[i], b\_time[i], total - a\_time[i], total - a\_time[i] - b\_time[i],prio[i]); |
|  | wait\_time = wait\_time + total - a\_time[i] - b\_time[i]; |
|  | turnaround\_time = turnaround\_time + total - a\_time[i]; |
|  | counter = 0; |
|  | } |
|  | if(i == limit - 1) |
|  | { |
|  | i = 0; |
|  |  |
|  | } |
|  | else if(a\_time[i + 1] <= total) |
|  | { |
|  | i++; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | i = 0; |
|  |  |
|  | } |
|  | } |
|  | return 0; |
|  | } |

**Complete Solution: -**

1. **Turn Around Time (T.A.T):** Time Difference between completion time and arrival time.

* Turn Around Time = Completion Time – Arrival Time

2. **Waiting Time (W.T):** Time Difference between turn around time and burst time.

* Waiting Time = Turn Around Time – Burst Time

**ANSWER:**

**Process** **Arrival-Time** **Burst-Time Turnaround time waiting time priority**

-----------------------------------------------

|  |  |  |  |
| --- | --- | --- | --- |
| P1 | 0 | 2 2 0 1 | |
| P2 | 4 | 5 7 2 -2147 |
| P3 | 3 | 7 11 4 - 2147 |
|  |  |  |

----------------------------------------------

Enter no of Processes n : 3

Arrival Time of P[1] :: 0

Burst Time of P[1] :: 2

Arrival Time of P[2] :: 4

Burst Time of P[2] :: 5

Arrival Time of P[3] :: 3

Burst Time of P[3] :: 7

Enter the time quantum::4

**Process** **Turn Around Time**

------------------------------------

|  |  |
| --- | --- |
| P[1] | 2 |
| P[2] | 7 |
| P[3] | 11 |
|  |  |

The Turn Around Time of process p1=2.

The Turn Around Time of process p2=9.

The Turn Around Time of process p3=20.

**Process** **Waiting Time**

--------------------------------

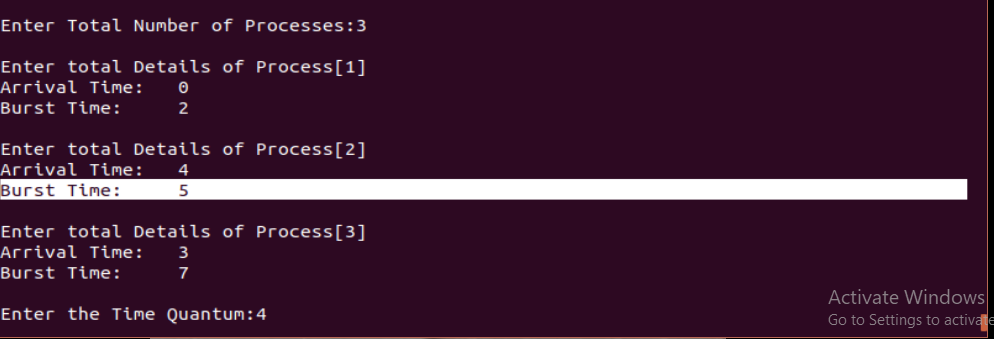
|  |  |
| --- | --- |
| P[1] | 0 |
| P[2] | 2 |
| P[3] | 4 |
|  |  |

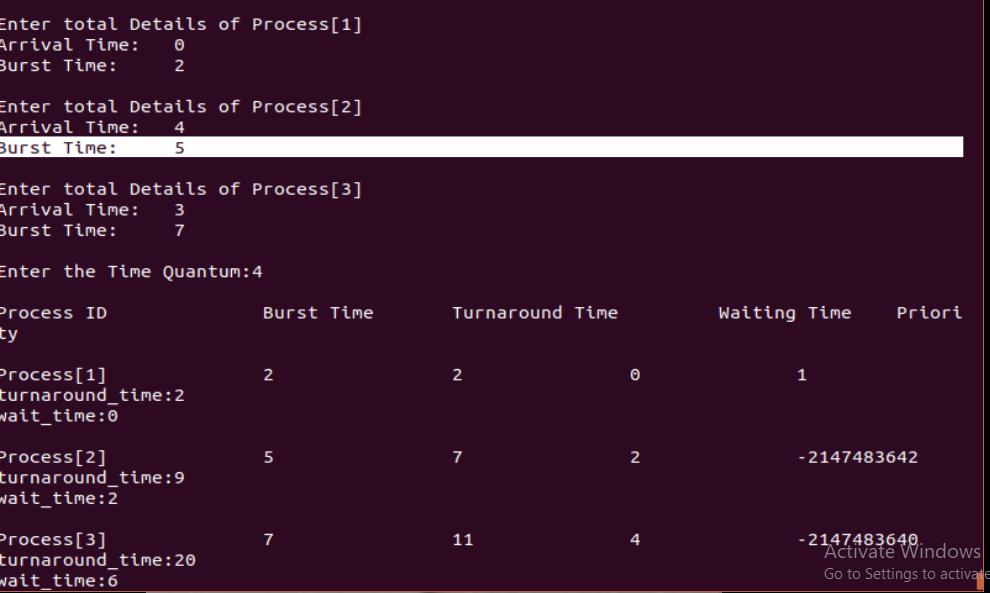
The Waiting Time of process p1= 0.

The Waiting Time of process p2= 2

The Waiting Time of process p3= 6

**Output for Given Program:-**





**Test Cases: -**

Test Case1: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 610 4

P2 1 8 13 5

P3 9 5 15 14

P4 4 1 11 6

Turn Around Time of p1 :- 10

Turn Around Time of p2 :- 23

Turn Around Time of p3 :- 38

Turn Around Time of p4 :- 49

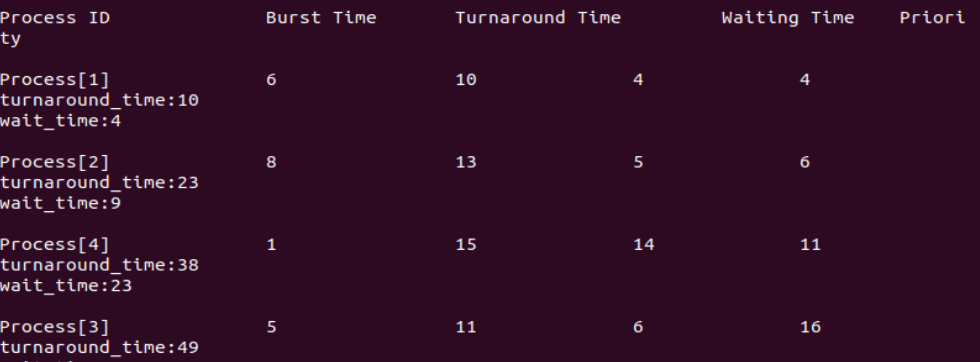
Waiting Time of p1 :- 4

Waiting Time of p2 :- 9

Waiting Time of p3 :- 23

Waiting Time of p4 :- 29

Reference: -



Test Case2: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 48 4

P2 2 2 5 3

P3 3 1 15 14

P4 5 5 18 13

P5 4 7 26 19

P6 1 9 25 16

Turn Around Time of p1 :- 46

Turn Around Time of p2 :- 8

Turn Around Time of p3 :- 13

Turn Around Time of p4 :- 97

Turn Around Time of p5 :- 72

Turn Around Time of p6 :- 28

Waiting Time of p1 :- 34

Waiting Time of p2 :- 4

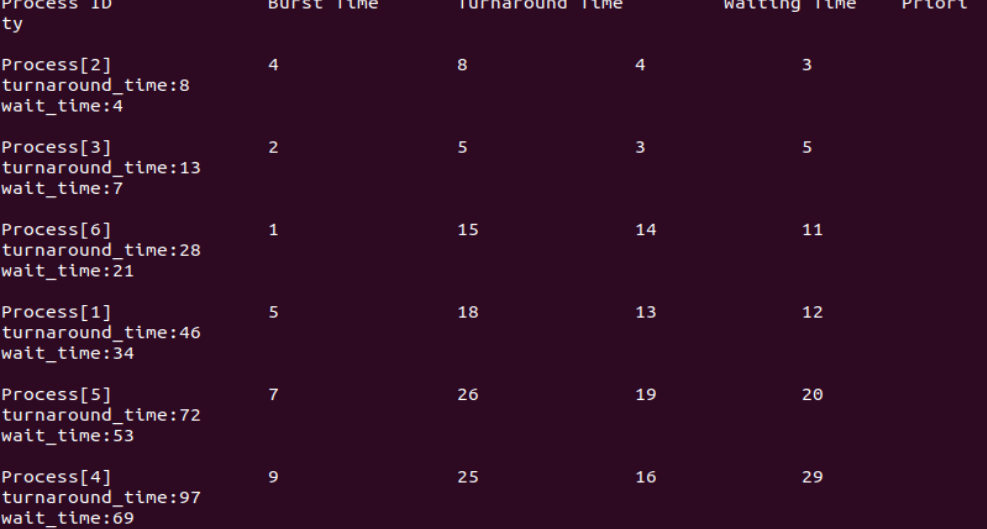
Waiting Time of p3 :- 7

Waiting Time of p4 :- 69

Waiting Time of p5 :- 53

Waiting Time of p6 :- 21

Reference: -



Test Case3: -

**Process ArrivalTime BurstTime Turn. A. T Waiting. T**

P1 0 518 13

P2 1 3 6 3

P3 2 2 7 5

P4 5 6 15 9

P5 4 8 20 12

Turn Around Time of p1 :- 31

Turn Around Time of p2 :- 6

Turn Around Time of p3 :- 13

Turn Around Time of p4 :- 46

Turn Around Time of p5 :- 66

Waiting Time of p1 :- 21

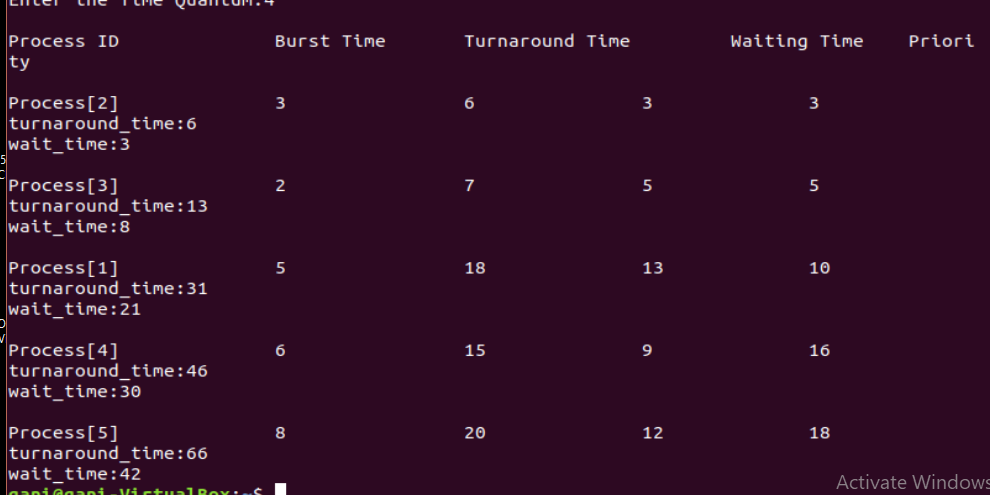
Waiting Time of p2 :- 3

Waiting Time of p3 :- 8

Waiting Time of p4 :- 30

Waiting Time of p5 :- 42

Reference: -



**Constraints: -**

Some Constraints used in my scheduling program are :

* Under Condition used to implement the how many number of processes are taken by user.

|  |
| --- |
| for(i = 0; i < limit; i++) |
| { |
| p[i]=i+1; |
|  |
| prio[i]=0; |
| printf("\nEnter total Details of Process[%d]\n", i + 1); |
| printf("Arrival Time:\t"); |
| scanf("%d", &a\_time[i]); |
| printf("Burst Time:\t"); |
| scanf("%d", &b\_time[i]); |
| temp[i] = b\_time[i]; |
| } |

* To check the priority.

|  |
| --- |
| for(z=0;z<limit;z++) |
| { |
| int temp1; |
| pos=z; |
| for(j=z+1;j<limit;j++) |
| { |
| if(prio[j]<prio[pos]) |
| pos=j; |
| } |

* Here is the constraint that is used get the output of the above program.

.

|  |
| --- |
| if(temp[i] == 0 && counter == 1) |
| { |
| x--; |
| printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\t\t%d", p[i], b\_time[i], total - a\_time[i], total - a\_time[i] - b\_time[i],prio[i]); |
| wait\_time = wait\_time + total - a\_time[i] - b\_time[i]; |
| turnaround\_time = turnaround\_time + total - a\_time[i]; |
| counter = 0; |
| } |
| if(i == limit - 1) |
| { |
| i = 0; |
|  |
| } |
| else if(a\_time[i + 1] <= total) |
| { |
| i++; |
|  |
| } |
| else |
| { |
| i = 0; |
|  |
| } |
| } |
| return 0; |
| } |

**Boundary Condition: -**

Here the main boundary condition is to execute the process maximum of n 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(i.e. 2 units will be as 3 or 4 units ) then the program or the output will become error or incorrect.