**Student Name** : Kantipudi V V S S Raghu Ramakrishna

**Student id** :11803043

**Email Address** : [raghukantipudi12082001@gmail.com](mailto:raghukantipudi12082001@gmail.com)

**Problem :**

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.

**Solution:**

It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a foreground(interactive)process and background(batch)processes. These two classes have different scheduling needs. For this kind of situation Multilevel queue scheduling is used.

**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. One general classification of the processes is foreground processes and background processes. 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](https://en.wikipedia.org/wiki/Round-robin_scheduling) or [FCFS](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)). 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.

Consider the following table with the arrival time, execute time and type of the process (foreground or background - where foreground processes are given high priority) to understand non pre-emptive and pre-emptive multilevel scheduling in depth with FCFS algorithm for both the queues:

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Name** | **Arrival Time** | **Execute Time** | **Type** |
| P0 | 0 | 5 | Foreground |
| P1 | 1 | 8 | Background |
| P2 | 3 | 7 | Background |
| P3 | 4 | 3 | Foreground |
| P4 | 5 | 3 | Foreground |
| P5 | 8 | 11 | Background |
| P6 | 15 | 3 | Foreground |
| P7 | 25 | 4 | Foreground |

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;

}

