**Student Name: Syed Mohd Omar**

**Student ID:** 11805045 (38) K18JF

**Email Address:** [syedomar512@gmail.com](mailto:syedomar512@gmail.com)

**GitHub Link:**

Problem: 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**

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

**For example:** separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.

### Algorithm

Multiple [FIFO](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) queues are used and the operation is as follows:

1. A new process is inserted at the end (tail) of the top-level [FIFO](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) queue.
2. At some stage the process reaches the head of the queue and is assigned the [CPU](https://en.wikipedia.org/wiki/Central_processing_unit).
3. If the process is completed within the [time quantum](https://en.wikipedia.org/wiki/Preemption_(computing)#Time_slice) of the given queue, it leaves the system.
4. If the process voluntarily relinquishes control of the CPU, it leaves the queuing network, and when the process becomes ready again it is inserted at the tail of the same queue which it relinquished earlier.
5. If the process uses all the quantum time, it is [pre-empted](https://en.wikipedia.org/wiki/Preemption_(computing)) and inserted at the end of the next lower level queue. This next lower level queue will have a time quantum which is more than that of the previous higher level queue.
6. This scheme will continue until the process completes or it reaches the base level queue.

* At the base level queue the processes circulate in [round robin](https://en.wikipedia.org/wiki/Round-robin_scheduling) fashion until they complete and leave the system. Processes in the base level queue can also be scheduled on a [first come first served](https://en.wikipedia.org/wiki/First-come,_first-served) basis.[[4]](https://en.wikipedia.org/wiki/Multilevel_feedback_queue#cite_note-OS_Concepts,_8th_edition-4)
* Optionally, if a process blocks for I/O, it is 'promoted' one level, and placed at the end of the next-higher queue. This allows I/O bound processes to be favored by the scheduler and allows processes to 'escape' the base level queue.

For scheduling, the scheduler always starts picking up processes from the head of the highest level queue. Only if the highest level queue has become empty will the scheduler take up a process from the next lower level queue. The same policy is implemented for picking up in the subsequent lower level queues. Meanwhile, if a process comes into any of the higher level queues, it will preempt a process in the lower level queue.

Also, a new process is always inserted at the tail of the top level queue with the assumption that it will complete in a short amount of time. Long processes will automatically sink to lower level queues based on their time consumption and interactivity level. In the multilevel feedback queue a process is given just one chance to complete at a given queue level before it is forced down to a lower level queue.

**Purpose**

Multi level queue scheduling was created for situation in which processes are easily classified into different groups.

Multilevel queue scheduling has the following characteristics:

Processes are divided into different queue based on their type. Process are permanently assigned to one queue, generally based on some property of process i.e. system process, interactive, batch system, end user process, memory size, process priority and process type.

Each queue has its own scheduling algorithm. For example interactive process may use round robin scheduling method, while batch job use the FCFS method.

In addition, there must be scheduling among the queue and is generally implemented as fixed priority preemptive scheduling. Foreground process may have higher priority over the background process

**Procedure**

Round Robin Scheduling

Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called ***time quantum***.

When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.

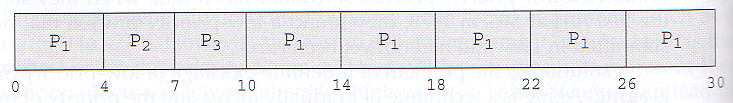
If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.

If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.

The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.

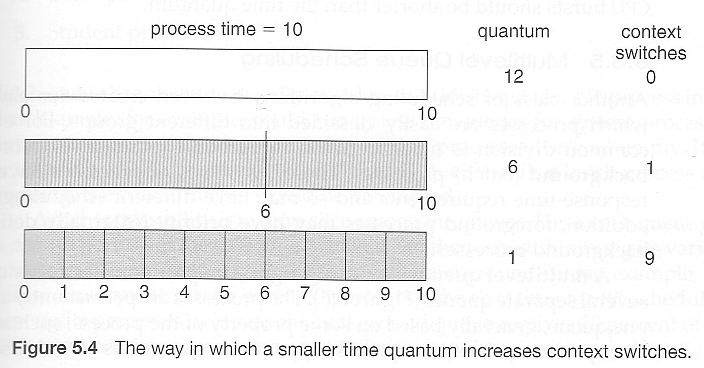
RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms. In the following example the average wait time is 5.66 ms.

|  |  |
| --- | --- |
| **Process** | **Burst Time** |
| P1 | 24 |
| P2 | 3 |
| P3 | 3 |

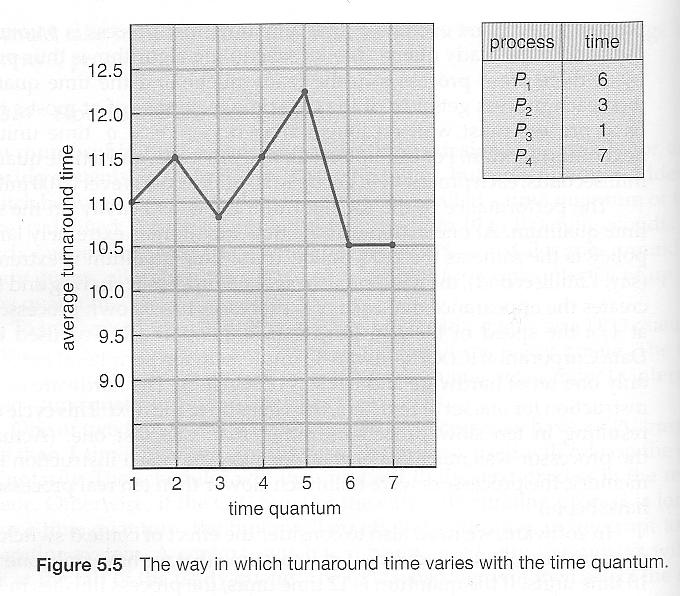


The performance of RR is sensitive to the time quantum selected. If the quantum is large enough, then RR reduces to the FCFS algorithm; If it is very small, then each process gets 1/nth of the processor time and share the CPU equally.

**BUT**, a real system invokes overhead for every context switch, and the smaller the time quantum the more context switches there are. ( See Figure 5.4 below. ) Most modern systems use time quantum between 10 and 100 milliseconds, and context switch times on the order of 10 microseconds, so the overhead is small relative to the time quantum.



* Turn around time also varies with quantum time, in a non-apparent manner. Consider, for example the processes shown in Figure 5.5:



**Code By Syed Omar:**

**#include<stdio.h>**

**int arrival\_time1[30],arrival\_time2[30],priority2[30],process2[30],arrival\_time3[30];**

**int burst\_time1[30],burst\_time2[30],burst\_time3[30];**

**int Total=0,t1=0,t2=0,t3=0;**

**int n,i,at[30],bt[30],pr[30],j=0,k=0,l=0;**

**int total,x,temp[30],counter=0;**

**float avg\_waiting\_time1=0.0,avg\_turnaround\_time1=0.0;**

**int p,waiting\_time3[30],turnaround\_time3[30];**

**float avg\_waiting\_time3=0.0,avg\_turnaround\_time3=0.0;**

**int position,q,temp1,sum=0,waiting\_time2[30],turnaround\_time2[30];**

**float avg\_waiting\_time2,avg\_turnaround\_time2;**

**void round\_robin()**

**{**

**printf("Time Quantum for Queue1 is 4\n");**

**for(i=0;i<j;i++)**

**{**

**temp[i]=burst\_time1[i];**

**}**

**printf("\nProcess ID\tBurst Time\t Turnaround Time\t Waiting Time\n");**

**x=j;**

**for(i=0,total=0;x!=0;)**

**{**

**if(temp[i]<=4&&temp[i]>0)**

**{**

**printf("\nProcess[%d] of Queue1 is running for %d units",i+1,temp[i]);**

**total=total+temp[i];**

**temp[i]=0;**

**counter=1;**

**}**

**else if(temp[i]>0)**

**{**

**printf("\nProcess[%d] of Queue1 is running for 4 units",i+1);**

**temp[i]=temp[i]-4;**

**total=total+4;**

**}**

**if(temp[i]==0&&counter==1)**

**{**

**x--;**

**printf("\nProcess[%d]\t%d\t%d\t%d",i+1,burst\_time1[i],total-arrival\_time1[i],total-arrival\_time1[i]-burst\_time1[i]);**

**avg\_waiting\_time1=avg\_waiting\_time1+total-arrival\_time1[i]-burst\_time1[i];**

**avg\_turnaround\_time1=avg\_turnaround\_time1+total-arrival\_time1[i];**

**counter = 0;**

**}**

**if(i==j-1)**

**{**

**i=0;**

**}**

**else if(arrival\_time1[i+1]<=total)**

**{**

**i++;**

**}**

**else**

**{**

**i=0;**

**}**

**}**

**avg\_waiting\_time1=avg\_waiting\_time1/j;**

**avg\_turnaround\_time1=avg\_turnaround\_time1/j;**

**printf("\nAverage Waiting Time:%f",avg\_waiting\_time1);**

**printf("\nAverage Turnaround Time:%f\n",avg\_turnaround\_time1);**

**}**

**void priority()**

**{**

**for(i=0;i<k;i++)**

**{**

**position=i;**

**for(q=i+1;q<k;q++)**

**{**

**if(priority2[q]<priority2[position])**

**{**

**position=q;**

**}**

**}**

**temp1=priority2[i];**

**priority2[i]=priority2[position];**

**priority2[position]=temp1;**

**temp1=burst\_time2[i];**

**burst\_time2[i]=burst\_time2[position];**

**burst\_time2[position]=temp1;**

**temp1=process2[i];**

**process2[i]=process2[position];**

**process2[position]=temp1;**

**}**

**waiting\_time2[0]=0;**

**for(i=1;i<k;i++)**

**{**

**waiting\_time2[i]=0;**

**for(q=0;q<i;q++)**

**{**

**waiting\_time2[i]=waiting\_time2[i]+burst\_time2[j];**

**}**

**sum=sum+waiting\_time2[i];**

**}**

**avg\_waiting\_time2=sum/k;**

**sum=0;**

**printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");**

**for(i=0;i<k;i++)**

**{**

**turnaround\_time2[i]=burst\_time2[i]+waiting\_time2[i];**

**sum=sum+turnaround\_time2[i];**

**printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n",process2[i],burst\_time2[i],waiting\_time2[i],turnaround\_time2[i]);**

**}**

**avg\_turnaround\_time2=sum/k;**

**printf("\nAverage Waiting Time:\t%f",avg\_waiting\_time2);**

**printf("\nAverage Turnaround Time:\t%f\n",avg\_turnaround\_time2);**

**for(i=0;i<k;i++)**

**{**

**while(burst\_time2[i]!=0)**

**{**

**if(burst\_time2[i]>10)**

**{**

**printf("\nProcess[%d] of Queue2 is running for 10 units",i+1);**

**burst\_time2[i]=burst\_time2[i]-10;**

**}**

**else if(burst\_time2[i]<=10)**

**{**

**printf("\nProcess[%d] of Queue2 is running for %d units",i+1,burst\_time2[i]);**

**burst\_time2[i]=0;**

**}**

**}**

**}**

**}**

**void fcfs()**

**{**

**waiting\_time3[0] = 0;**

**for(i=1;i<l;i++)**

**{**

**waiting\_time3[i] = 0;**

**for(p=0;p<l;p++)**

**{**

**waiting\_time3[i]=waiting\_time3[i]+burst\_time3[p];**

**}**

**}**

**printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time\n");**

**for(i=0;i<l;i++)**

**{**

**turnaround\_time3[i]=burst\_time3[i]+waiting\_time3[i];**

**avg\_waiting\_time3=avg\_waiting\_time3+waiting\_time3[i];**

**avg\_turnaround\_time3=avg\_turnaround\_time3+turnaround\_time3[i];**

**printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d\n",i+1,burst\_time3[i],waiting\_time3[i],turnaround\_time3[i]);**

**}**

**avg\_waiting\_time3=avg\_waiting\_time3/l;**

**avg\_turnaround\_time3=avg\_turnaround\_time3/l;**

**printf("\nAverage Waiting Time=%f",avg\_waiting\_time3);**

**printf("\nAverage Turnaround Time=%f",avg\_turnaround\_time3);**

**for(i=0;i<l;i++)**

**{**

**while(burst\_time3[i]!=0)**

**{**

**if(burst\_time3[i]>10)**

**{**

**printf("\nProcess[%d] of Queue3 is running for 10 units",i+1);**

**burst\_time3[i]=burst\_time3[i]-10;**

**}**

**else if(burst\_time3[i]<=10)**

**{**

**printf("\nProcess[%d] of Queue2 is running for %d units",i+1,burst\_time3[i]);**

**burst\_time3[i]=0;**

**}**

**}**

**}**

**}**

**void round\_robin1()**

**{**

**printf("Time Quantum between the 3 queues is 10\n");**

**for(i=1;i<Total;i=i+10)**

**{**

**if(t1>10)**

**{**

**printf("Queue1 is running for 10 units\n");**

**t1=t1-10;**

**}**

**else if(t1<=10&&t1!=0)**

**{**

**printf("Queue1 is running for %d units\n",t1);**

**t1=0;**

**}**

**if(t2>10)**

**{**

**printf("Queue2 is running for 10 units\n");**

**t2=t2-10;**

**}**

**else if(t2<=10&&t2!=0)**

**{**

**printf("Queue2 is running for %d units\n",t2);**

**t2=0;**

**}**

**if(t3>10)**

**{**

**printf("Queue3 is running for 10 units\n");**

**t3=t3-10;**

**}**

**else if(t3<=10&&t3!=0)**

**{**

**printf("Queue3 is running for %d units\n",t3);**

**t3=0;**

**}**

**}**

**}**

**int main()**

**{**

**printf("Enter the no. of process you want to enter\n");**

**scanf("%d",&n);**

**for(i=0;i<n;i++)**

**{**

**printf("Enter details of process[%d]\n",i+1);**

**printf("Arrival Time:");**

**scanf("%d",&at[i]);**

**printf("Burst Time:");**

**scanf("%d",&bt[i]);**

**printf("Priority(1 to 15):");**

**scanf("%d",&pr[i]);**

**Total=Total+bt[i];**

**}**

**for(i=0;i<n;i++)**

**{**

**if(pr[i]>=1&&pr[i]<=5)**

**{**

**printf("\n\nProcess[%d] belongs to Queue 1\n",i+1);**

**arrival\_time1[j]=at[i];**

**burst\_time1[j]=bt[i];**

**j++;**

**t1=t1+bt[i];**

**}**

**else if(pr[i]>=6&&pr[i]<=10)**

**{**

**printf("Process[%d] belongs to Queue 2\n",i+1);**

**arrival\_time2[k]=at[i];**

**burst\_time2[k]=bt[i];**

**priority2[k]=pr[i];**

**process2[k]=k+1;**

**k++;**

**t2=t2+bt[i];**

**}**

**else if(pr[i]>=11&&pr[i]<=15)**

**{**

**printf("Process[%d] belongs to Queue 3\n\n\n\n",i+1);**

**arrival\_time3[l]=at[i];**

**burst\_time3[l]=bt[i];**

**l++;**

**t3=t3+bt[i];**

**}**

**}**

**round\_robin1();**

**round\_robin();**

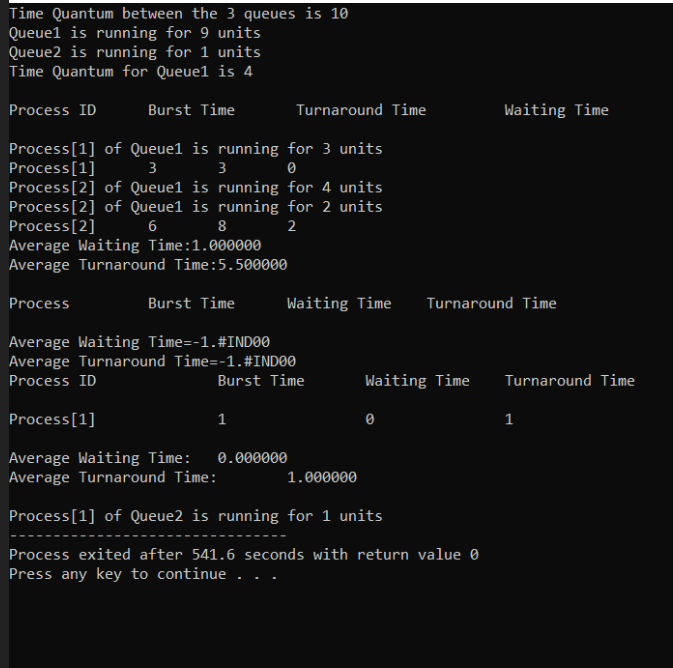
**fcfs();**

**priority();**

**return 0;**

**}**

**Output:**



**GitHub Link**

[**https://github.com/AnukulClasher/CSE-316-**](https://github.com/AnukulClasher/CSE-316-)