# SCS1214 Operating Systems I

Assignment 8 - Multilevel Queue Scheduling

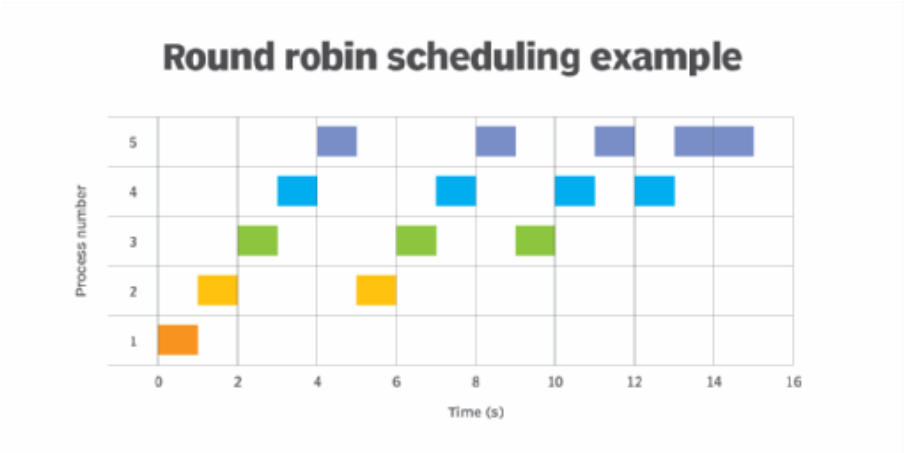
*Index Number – 21000557*

## 1. What is a Multilevel Queue?

When different processes needs different priorities for example if we consider a system process and a user process where the system process needs to be executed first we need a system to sort out the processes in a certain order. This could be implemented in a multilevel queue. A multilevel queue contains few separate queues with a different priority number where queue with the highest priority always gets executed first. In this scenario we have a multilevel queue with 4 queues.

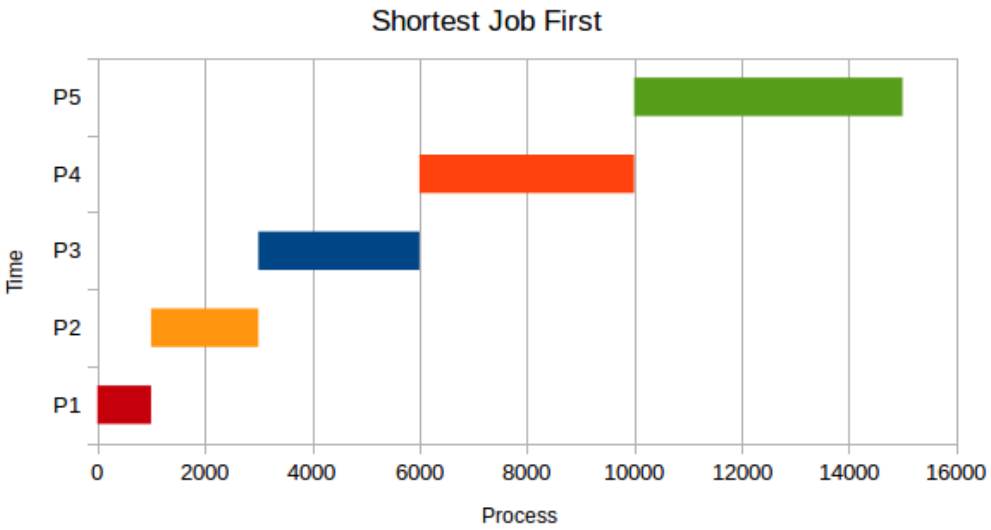
- ***q0 – Round Robin***
- ***q1 – Shortest Job First***
- ***q2 – Shortest Job First***
- ***q3 – First-In-First-Out***

### Round Robin Scheduling




In this scheduling algorithm each process is given a pre-defined time slice and after the time slice the process is sent to the back of the queue and the CPU is given to the next process in the queue. To put it simply each process takes turns running and this way no process is left untouched for a long time. The time each process is given is called *time quantum* and it is pre-defined. In this case it is 20. Typically turnaround time is higher in this scheduling algorithm because every process is given a time frame to run. An example for a round robin in computer operations is round robin DNS, a method that allows multiple DNS records to be created for the same IP address. This ensures that no matter how many users are accessing the same domain name, each user has equal access to information and services through round robin.

**Shortest Job First**



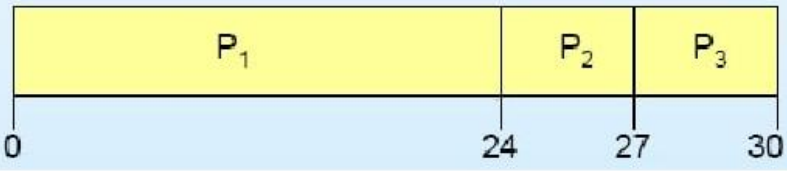

In this algorithm the process with the smallest burst time in the running queue is executed first. Then the CPU is given to the next smallest process and so on. Processes are sorted based on their burst time and then added to the running queue. Turnaround time for smaller processes can be very quick and larger processes may take a while to complete. Copying data from a place to another place is done using Shortest Job First algorithm.

**First-In-First-Out**

This is a simple queue where processes are ordered based on their arrival time and only after the execution of the current process is over, the CPU is given over to the next process. If the processor is busy with a very large process, the processes at the back of the queue will take a while to be completed and the turnaround time for each will be large. If the larger processes are at the back of the queue, the turnaround time would be decent depending on the order of the processes. The FIFO algorithm is used in the paging method for memory management in an operating system that decides which existing page needs to be replaced in the queue.

| Process | Burst time |
|---|---|
| P1 | 24 |
| P2 | 3 |
| P2 | 3 |

Suppose that the processes arrive in the order: $P_1$ , $P_2$ , $P_3$
The Gantt Chart for the schedule is:



## 2. Implementation

```python
class Process:
    def __init__(self, pid, priority, burst_time):
        self.pid = pid
        self.priority = priority
        self.burst_time = burst_time
        self.arrival_time = 0
        self.waiting_time = 0
        self.turnaround_time = 0

with open("processes.txt", "r") as f:
    lines = f.readlines()

processes = []
for i, line in enumerate(lines):
    pid, priority, burst_time = line.split()
    processes.append(Process(i, int(priority), int(burst_time)))

q0 = []
q1 = []
q2 = []
q3 = []

for process in processes:
    if process.priority == 0:
        q0.append(process)
    elif process.priority == 1:
        q1.append(process)
    elif process.priority == 2:
        q2.append(process)
    else:
        q3.append(process)
```

The implementation is done using python language. Each algorithm is represented by a list and a class object is created for each process. Each object stores *process_ID, priority, burst_time, arrival_time, waiting_time* and *turnaround_time*. The *waiting_time* and the *turnaround_time* is set to 0 at the creation of the object. The list of processes are imported to the program using a .txt file called *processes.txt*. Each process is given a line number using *enumerate ()* function and sorted into the different queues based on the *priority* of each process.

A lambda function is used when sorting q1 and q2 and the key used for sorting is the burst time of each process.

The *calculate_times()* function calculates the total *waiting_time* and *turnaround_time* of each queue. And also *waiting_time* and *turnaround_time* for each process as well. However it must be noted that *total_waiting_time* and *total_turnaround_time* are calculated separately for each queue. So each value of the high priority queues are added to the low priority queues to get the accurate values for the queues below. This is done because in Multilevel Queues only after the higher priority queues are empty the CPU is given to the lower priority queues.

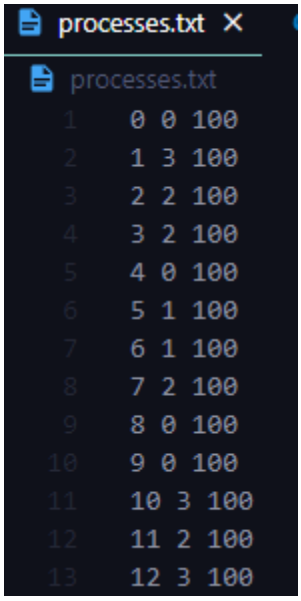
```python
def calculate_times(processes):
    current_time = 0
    total_waiting_time = 0
    total_turnaround_time = 0
    current_queue = processes

    for process in processes:
        process.waiting_time = current_time

        if current_queue == q0:
            time_slice = min(time_quantum, process.remaining_time)
        else:
            time_slice = process.remaining_time

        current_time += time_slice
        process.remaining_time -= time_slice

        if process.remaining_time <= 0:
            process.turnaround_time = current_time
            if current_queue == q0:
                process.waiting_time = process.turnaround_time - process.burst_time
            total_waiting_time += process.waiting_time
            total_turnaround_time += process.turnaround_time
        else:
            current_queue.append(process)

    return (total_waiting_time, total_turnaround_time)
```

After calculating the all the values formatted string outputs are used to print out the values for each process and average values for each queues.

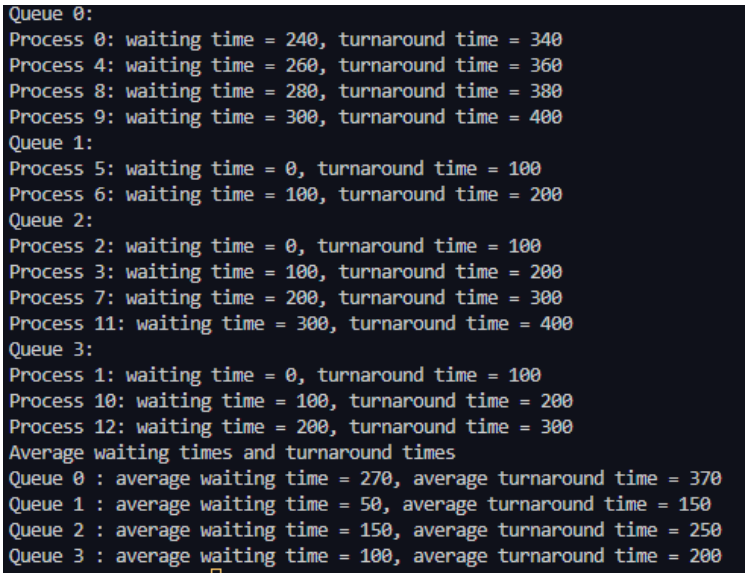
## 3. Results and Analysis

***CASE 01:***

We will use several test case sets to get an idea of these algorithms. In the first case 12 processes with 100 burst time each is used. This is a simple case just to test the functionality of the program. Expected results are 100 as turnaround time for first process in q1, q2 and q3. And incremented by 100 for each process after the first. And for the q0, expected values are *burst_time* + *waiting_time* for rest of the processes which is 340 for the first process incremented by 20 for each after that. As for the waiting times, q1, q2 and q3 are expected to have 0 for the first process and incremented by 100 for each process after first. For q0, 60 for each round and 4 rounds, so 240 for the first process and incremented by 20 for each process after first.




*Text file for CASE 01*




*Output for CASE 01*

**** It should be noted that since we are using local variables inside the calculate_times() function we can only calculate average(total values) using the waiting times from the queues before. Inside each queue waiting_time and turnaround_time are calculated from zero. If we want to get the correct value regarding each process, turnaround_time or waiting_time of the last process in each queue before should be added to the first of the queue below*

*i.e.: waiting time for the 1st process in q1 should be 400 since q0 is executed completely before moving to q1.*

***CASE 02:***

In the input file the first column of each line represent the *process_ID*. Second column for the *priority* of the process (0-highest, 3-lowest) and last column for the *burst_time* of each process.

**** For the cases from this point onward average time (only) will be calculated with the times it takes to completes the queues before*

*i.e.: total waiting time of q2 will be the sum of total waiting time of q2, q1 and q0.*

processes.txt

```
0 0 25
1 3 70
2 1 56
3 2 60
4 0 40
5 1 87
6 1 35
7 2 30
8 0 90
9 0 38
10 3 11
11 2 45
12 3 150
```

*Text file for CASE 02*

```
Queue 0:
Process 0: waiting time = 60, turnaround time = 85
Process 4: waiting time = 65, turnaround time = 105
Process 8: waiting time = 103, turnaround time = 193
Process 9: waiting time = 105, turnaround time = 143
Queue 1:
Process 6: waiting time = 0, turnaround time = 35
Process 2: waiting time = 35, turnaround time = 91
Process 5: waiting time = 91, turnaround time = 178
Queue 2:
Process 7: waiting time = 0, turnaround time = 30
Process 11: waiting time = 30, turnaround time = 75
Process 3: waiting time = 75, turnaround time = 135
Queue 3:
Process 1: waiting time = 0, turnaround time = 70
Process 10: waiting time = 70, turnaround time = 81
Process 12: waiting time = 81, turnaround time = 231
Average waiting times and turnaround times
Queue 0 : average waiting time = 83, average turnaround time = 131
Queue 1 : average waiting time = 217, average turnaround time = 276
Queue 2 : average waiting time = 311, average turnaround time = 356
Queue 3 : average waiting time = 407, average turnaround time = 484
```

*Output of CASE 02*

***q0:*** process order is 0 -> 4 -> 8 -> 9

process 0 –> burst time = 25

process 4 –> burst time = 40

process 8 –> burst time = 98

process 9 –> burst time = 38

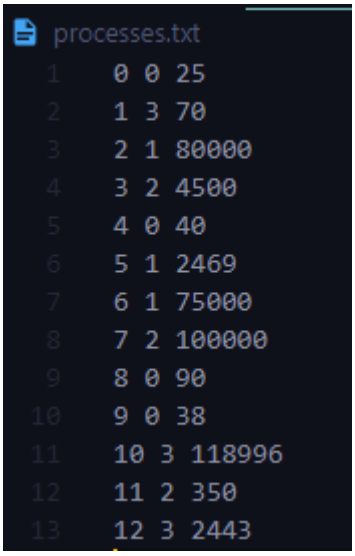
First round each process gets 20 time quantum. After the first round p0 has 5 *time_remaining*, p4 with 20, p8 with 78 and p9 with 18. After the first round each process has a *waiting_time* of 60. In 2nd round 1st process is allocated 20 time quantum but only needs 5. So the value of the *time_slice* is 5 and the process is done with turnaround time of 60 + 20 + 5 = 85. Like that all the processes gets executed until all the processes are done executing. Here we can see even though p8 has the highest turnaround time of 193 it doesn't have the highest waiting time of the 4 processes. ***This queue has the highest priority which means until all the processes in this queue are executed, we will not move to the next queue.***

***q1: q2:*** Both of these queues are **Shortest Job First** queues. Which mean in q1 even though process 6 is the last job for q1, it is executed first since it has the lowest *burst_time* of 35.

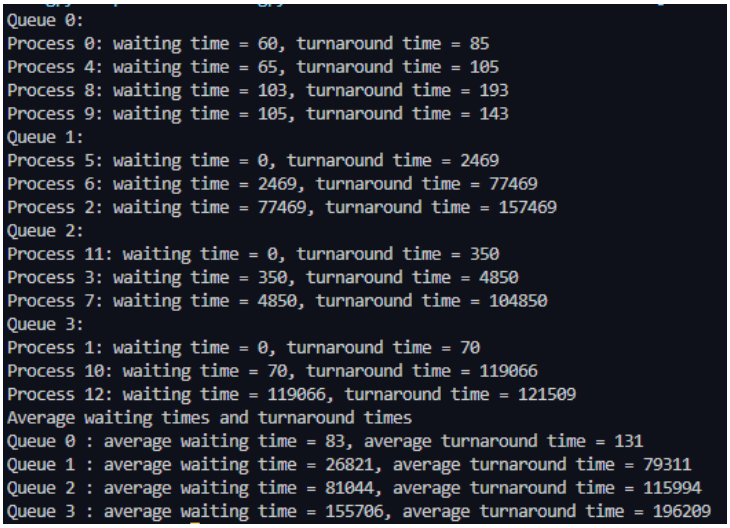
***q3:*** This is a **First-come-first-serve** queue. The processes are executed in the order in the list.

***CASE 03:***

Since there not much else to test in Round Robin queue we will test how q1, q2 and q3 will handle extremely large processes and what happens to the outputs. I will keep the q0 values intact.




*Text file for CASE 03*




*Output of CASE 03*

Here in **q1** because the *burst_time* of process 2 is very large, it is sorted into the back of the queue to give space for the smaller processes to execute first. The order doesn't matter in the **Shortest Job First** scheduling. But in **q3** because process 10 is in the list first it is executed before process 12 which has a much smaller *burst_time*. Due to this reason process 12 has to wait 118996 just to execute his 2443 burst. So the order matter in **First-come-first-serve** scheduling.

**** Since all the points are covered I will show some tests cases with different priorities and different number of processes to show the functionalities of the algorithms.*

## *Case 04:*

processes.txt

```
0 0 25
1 3 70
2 1 80
3 2 45
4 0 40
5 1 24
6 1 75
7 2 10
8 0 90
9 0 38
10 3 11
11 2 35
12 3 244
13 0 113
14 2 47
15 1 22
16 2 80
17 1 24
18 0 700
19 3 19
20 1 250
```

*Text file for CASE 04*

```
Queue 0:
Process 0: waiting time = 100, turnaround time = 125
Process 4: waiting time = 105, turnaround time = 145
Process 8: waiting time = 263, turnaround time = 353
Process 9: waiting time = 145, turnaround time = 183
Process 13: waiting time = 293, turnaround time = 406
Process 18: waiting time = 306, turnaround time = 1006
Queue 1:
Process 15: waiting time = 0, turnaround time = 22
Process 5: waiting time = 22, turnaround time = 46
Process 17: waiting time = 46, turnaround time = 70
Process 6: waiting time = 70, turnaround time = 145
Process 2: waiting time = 145, turnaround time = 225
Process 20: waiting time = 225, turnaround time = 475
Queue 2:
Process 7: waiting time = 0, turnaround time = 10
Process 11: waiting time = 10, turnaround time = 45
Process 3: waiting time = 45, turnaround time = 90
Process 14: waiting time = 90, turnaround time = 137
Process 16: waiting time = 137, turnaround time = 217
Queue 3:
Process 1: waiting time = 0, turnaround time = 70
Process 10: waiting time = 70, turnaround time = 81
Process 12: waiting time = 81, turnaround time = 325
Process 19: waiting time = 325, turnaround time = 344
Average waiting times and turnaround times
Queue 0 : average waiting time = 202, average turnaround time = 369
Queue 1 : average waiting time = 454, average turnaround time = 533
Queue 2 : average waiting time = 696, average turnaround time = 740
Queue 3 : average waiting time = 1044, average turnaround time = 1130
```

*Output of CASE 04*

## *Case 05:*

processes.txt

```
0 3 455
1 0 12
2 1 266
3 0 678
4 0 344
5 1 66
6 3 75
7 2 14
```

*Text file for CASE 05*

```
Queue 0:
Process 1: waiting time = 0, turnaround time = 12
Process 3: waiting time = 356, turnaround time = 1034
Process 4: waiting time = 372, turnaround time = 716
Queue 1:
Process 5: waiting time = 0, turnaround time = 66
Process 2: waiting time = 66, turnaround time = 332
Queue 2:
Process 7: waiting time = 0, turnaround time = 14
Queue 3:
Process 0: waiting time = 0, turnaround time = 455
Process 6: waiting time = 455, turnaround time = 530
Average waiting times and turnaround times
Queue 0 : average waiting time = 242, average turnaround time = 587
Queue 1 : average waiting time = 914, average turnaround time = 1080
Queue 2 : average waiting time = 2160, average turnaround time = 2174
Queue 3 : average waiting time = 1314, average turnaround time = 1579
```

*Output of CASE 05*

## 4. Pros and Cons

*Round Robin Scheduling*

***Pros:***

- Every process gets equal share of CPU time.
- Since every process takes turns using the CPU there is no issue of starvation. This mean no process is left untouched for a long time.

***Cons:***

- Average waiting time could be very large due to the cyclic nature. A process with a slightly larger *burst_time* than the time quantum would have to wait for a whole cycle to finish his process.
- If the time quantum is very low the overhead increases and the CPU efficiency becomes lower. And if time quantum is very high **Round Robin** changes into **First-come-first-serve.**

*Shortest Job First Scheduling*

***Pros:***

- Usually gives the minimum average time waiting for a set of processes.
- Shorter jobs gets executed quickly.

***Cons:***

- If shorter processes keeps on coming one after another, longer processes may never get the CPU time to execute. This is knows and *starvation*.

*First-Come-First-Serve Scheduling*

***Pros:***

- Easy and simple to understand

***Cons:***

- If a process with a lower *burst_time* is at the back of the queue with a process with a very large *burst_time* in front of it, the process at the back may have to wait long time in the running queue for the first process to be over.
- Depending on the order of processes average waiting time can be very large.
- Troublesome for multiprogramming systems due to the processes having to wait for another to complete before.
- Could result to lower CPU utilization.

**** We can check the above qualities by giving exactly the same number of processes with same burst time to each queue. For this I will disable the summation when checking averages so that we can get values for each queue separately.*

processes.txt

```
0 0 100
1 0 70
2 0 33
3 0 255
4 1 100
5 1 70
6 1 33
7 1 255
8 2 100
9 2 70
10 2 33
11 2 255
12 3 100
13 3 70
14 3 33
15 3 255
```

*Text file*

```
Queue 0:
Process 0: waiting time = 183, turnaround time = 283
Process 1: waiting time = 173, turnaround time = 243
Process 2: waiting time = 100, turnaround time = 133
Process 3: waiting time = 203, turnaround time = 458
Queue 1:
Process 6: waiting time = 0, turnaround time = 33
Process 5: waiting time = 33, turnaround time = 103
Process 4: waiting time = 103, turnaround time = 203
Process 7: waiting time = 203, turnaround time = 458
Queue 2:
Process 10: waiting time = 0, turnaround time = 33
Process 9: waiting time = 33, turnaround time = 103
Process 8: waiting time = 103, turnaround time = 203
Process 11: waiting time = 203, turnaround time = 458
Queue 3:
Process 12: waiting time = 0, turnaround time = 100
Process 13: waiting time = 100, turnaround time = 170
Process 14: waiting time = 170, turnaround time = 203
Process 15: waiting time = 203, turnaround time = 458
Average waiting times and turnaround times
Queue 0 : average waiting time = 164, average turnaround time = 279
Queue 1 : average waiting time = 84, average turnaround time = 199
Queue 2 : average waiting time = 84, average turnaround time = 199
Queue 3 : average waiting time = 118, average turnaround time = 232
```

*Output*

In the code, averages takes the total turnaround time from the queue with the higher priorities because in multilevel queues, the queues with higher priorities always gets executed first.

## 5. Conclusion and Summarization

When it comes to scheduling algorithms it is clear that by using a **Multilevel Queue** we can order the processes in a way that can maximize user experience and CPU utilization. For example we can use **Round Robin Queue** for system processes that needs immediate attention and user processes like displaying contents. Whereas we can put processes like copying a movie into a hard drive on a **Shortest Job First Queue**. Since it does not need immediate attention but no need to wait for a longer time if the file isn't that big. And for last something like a software update that isn't much important can be put into a **First-come-first-serve Queue** where the update will happen when no one is using the computer.

Even though **Round Robin Queue** has the highest priority it is not great when it comes to waiting time. The processes get CPU time but it is limited. Average turnaround time and waiting time is most of the time better in **Shortest Job First Queue**. But the queue suffers from starvation. However we can address that issue by aging processes where we can increase the priority of a process when it is left untouched for a while.

From the examples before **Round Robin Queue** is not good even when it comes to turnaround time. There is always a lot of unnecessary waiting time in the algorithm. **Shortest job first Queue** seems to be the best when it comes to turnaround time as well.

## 6. Limitations of the Program

In real scenarios all programs doesn't start at the same time. The scheduling algorithms has to manage processes as they come at different paces. If we have a process running in q2 and both q1 and q0 is empty and we get a process with priority 0 (q0) the CPU should always be given to the process at q0 and stop whatever the process running at the moment. This program is not built to handle that. In this program is it assumed that the arrival time of each process is always 0.

Also as said before q1 and q2 suffer from *starvation*. Which is fixed by *aging* processes. But this program doesn't allow processes to switch the queues so it is not possible to age processes in this program.

Also we cannot add processes as on run time. All the processes has to be pre-defined.