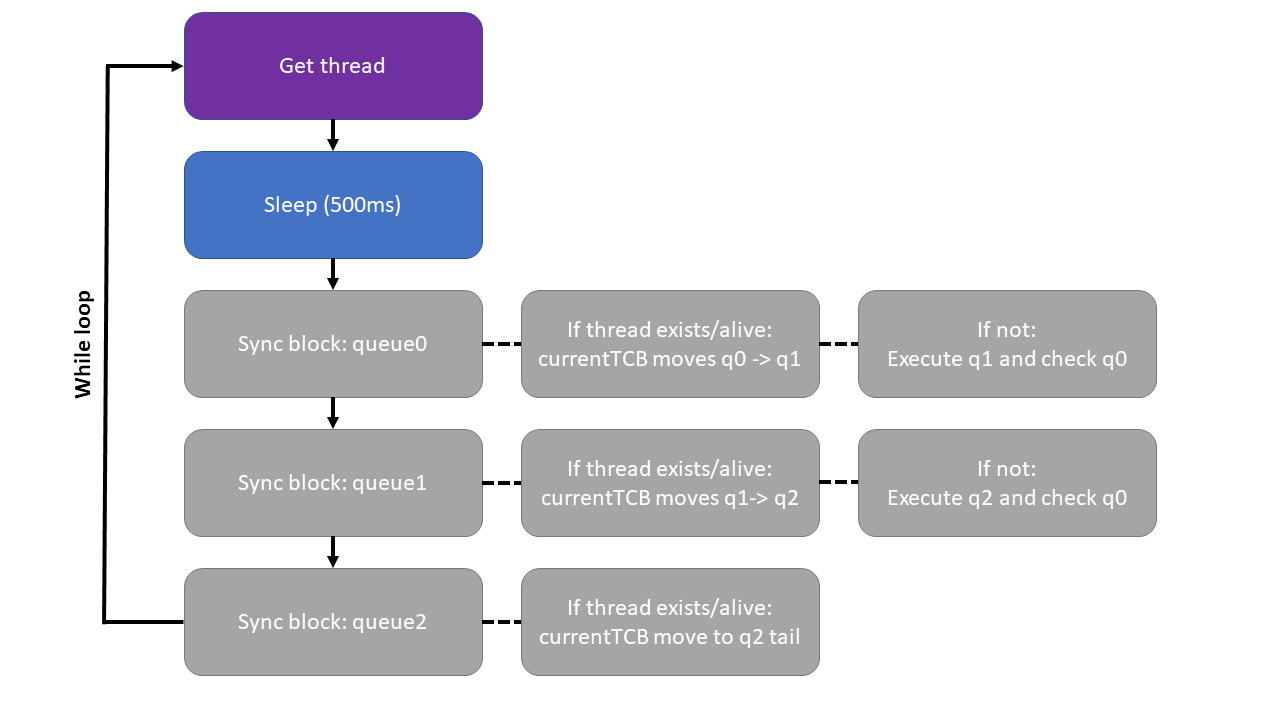
**Chandler Ford | February 10th, 2019**  
**CSS 430 Operating Systems | Program 2 Report**

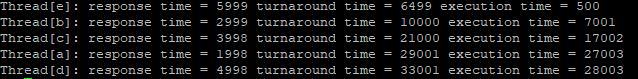
**Design and Algorithm for Part 2**For Part 2, the goal was to convert the modified Round Robin Scheduler.java into a Multi-level Feedback Queue scheduler. To accomplish this, I implemented three queues numbered from 0 to 2 along with an additional master queue. This required the three Scheduler constructors to be modified to account for creating the new vectors. The presence of more than one queue required a change to the getMyTcb() method. The number of sync blocks and for loops were increased to three to allow all three queues to be searched to return the correct tcb. A small change was needed in the addThread() to specify the queue being added to was queue0.

The run() method needed to be altered as well. This is where queueMaster is utilized. The first time the method is called, the queueMaster initialized as a shallow copy of queue0. This allows much of the existing logic before the schedulerSleep() call to be kept with the exception of changing the previous queue over to queueMaster. After the program gets a thread it starts it and runs it again with queueMaster set to queue0 once more. The current thread resumes until it reaches the schedulerSleep() call and sleep for 500ms (the value being changed from 1000ms previously).

The majority of the rest of the run() method is divided into three synchronized blocks, one for each of the three queues. In the sync block for queue0, the program checks if the current thread exists and is alive after the 500ms schedulerSleep() call. If so, the current thread is suspended and the currentTCB is removed from queue0 to be added to queue1. The queueMaster is set as a shallow copy of queue1. Otherwise the current thread is suspended if getMyTcb() isn’t null as this means that there is something to execute in queue0. The sync block for queue1 works in a similar fashion to the sync block for queue0 except that it adds to queue2. The sync block for queue2 is different in that it removes and adds to queue2 if the current thread exists and is alive – this puts the thread at the tail of queue2.

Execution enters an infinite loop after creating a thread, so I don’t get output. I believe this is because I’m not suspending and resuming threads properly. There also may be issues with sleeping threads, I was trying to use only the 500ms schedulerSleep() call but other calls may have been needed. I don’t think that the algorithms in the sync blocks for queue0 and queue1 are suspending and going back to lower queues are entirely right, though I think some of the basic logic is there.

**Compare Test Results**



Above are the test results for the Round Robin scheduler. I couldn’t get test results to output for the Multi-level Feedback Queue scheduler. Based off research (see sources below), I believe that the Multi-level Feedback Queue scheduler would reduce the response time because priority is given to shorter running threads. The Multi-level Feedback Queue scheduler would also reduce the turnaround time because threads that are shorter run first plus the algorithm learns from and can account for thread behavior. However, the execution time might be worse in queue1 and queue2 because of starvation (which doesn’t happen in the Round Robin scheduler). Sources for above answers listed are listed below:  
<https://www.geeksforgeeks.org/multilevel-feedback-queue-scheduling/>  
<http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf>  
<https://en.wikipedia.org/wiki/Scheduling_(computing)#Round-robin_scheduling>

**FCFS Implementation**If Part 2 were implemented using a FCFS queue2 without preemption (OS doesn’t interrupt thread) rather than a Round Robin, delays could occur if larger threads arrive first. For example, a large thread that takes 5000ms arriving before two small 50ms threads is going to massively increase the average waiting time as opposed to the two small threads arriving first. I have created two simplified Gantt chart examples that illustrate this point based on the CSS430\_L7\_L8\_Scheduling lecture slides (as I don’t have Test2.java execution data for Part 2).

|  |  |  |
| --- | --- | --- |
| **5000ms thread** | **50ms thread** | **50ms thread** |

Average waiting time: (0ms + 5000ms + 5050ms)/3 = 3350ms

|  |  |  |
| --- | --- | --- |
| **50ms thread** | **50ms thread** | **5000ms thread** |

Average waiting time: (0ms + 50ms + 100ms)/3 = 50ms