**Part 2 Algorithm:**

The normal constructor of this Scheduler.java sets the quantum of each thread to 500ms. It also instantiates an array of 3 vectors named queues, which represent the three queues of the MFQS algorithm. This array of vectors has decreasing priorities, meaning that element 0 or the first queue of this array has the highest priority. The priorities signify which queue of threads will be executed first. The scheduler also initializes an array of booleans whose indices keep track of which thread IDs are in use.

The main function of this class is the run() method which overrides the built-in run() method of the Thread class. This run() method is executed by each thread. Basically, each thread in execution is constantly in a while loop that calls on 3 methods which handle the execution of the threads in each queue. Every while loop executes the first queue first ( runQueueZero() ), then the second ( runQueueOne() ), then the third ( runQueueTwo() ).

These runQueue() methods as mentioned above are the same for the most part.

runQueueZero() is executed first. This ultimately runs the threads in queue one if they exist, until there are no more left. It does this using a while loop which is only broken out of when the queue is empty of threads. If not, we check the front thread of the queue and check if it has terminated its execution. If it has, it's removed from the queue, and the while loop is resetted. This means that this method will check again if the queue is empty; if not, we check the front again until the current front thread hasn't terminated its execution. If it hasn't, we check if its started its execution before, in which case, we simply resume the thread. If not, we start the thread. We then let the thread execute for the time quantum allotted to queue one which is 500 ms. This is done using schedulerSleep() which paused the scheduler for 500 ms so that the thread may execute at this time. runQueueOne() and runQueueTwo() do the same exact thing on their respected queues from the invocation of runQueueZero() up to this point. runQueueZero() then locks the array of queues so that only one thread may access it at a time. If the current thread wasn't able to finish execution within the quantum, we suspend it and move it the back of queue 2. we then re-loop and check the next thread of the queue.

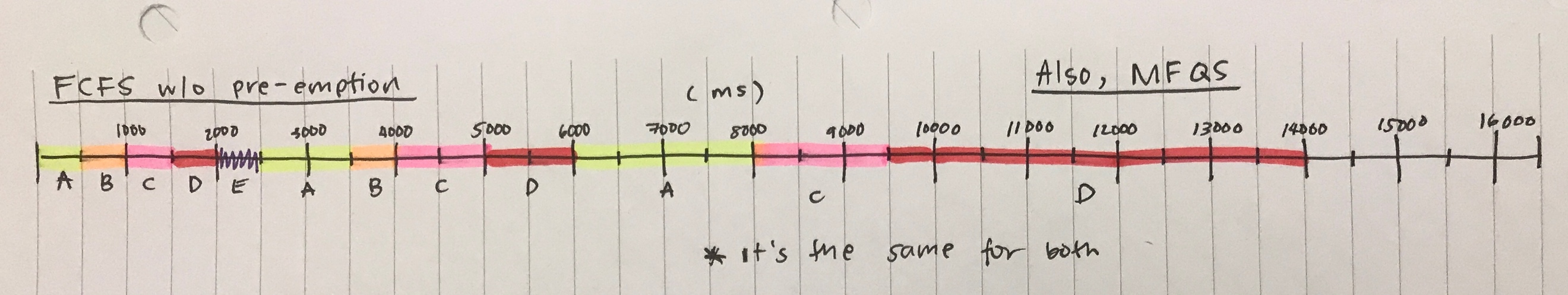
As I mentioned, runQueueOne() does the same exact thing with the second queue up until the first call of schedulerSleep() in the method. runQueueOne() is allotted a 1000ms quantum. However, we split this quantum into 2 500ms increments. If the current thread hasn't finished execution after the first 500ms increment, we must suspend it, then call on runQueueZero(). This is so we can check if there are new threads added to the first queue because those have the highest priority and must thus be executed first. After runQueueZero(), we go back to the current thread and call on schedulerSleep() so that thread may execute for the remaining 500ms of its quantum. runQueueOne() then puts a lock on the array of queues. If the current thread hasn't finished executing after its total quantum of 1000ms, we suspend it and move it to the back of the third and final queue.

runQueueTwo(), the final one, does the same exact thing again with the third queue up until the first call of schedulerSleep() in the method. runQueueTwo() is allotted a 2000ms quantum. However, we split this quantum into 4 500ms increments, so that in between increments, we may call on runQueueZero() first then runQueueOne() to check if there are new threads added to those queues since those need to be executed first. After every increment, if the thread still has not finished its execution, we suspend it then call on runQueueZero() and runQueueOne(). Afterwards, we resume the current thread for the next increment until we've reached a total of 2000ms execution time. At this point, if the current thread still hasn't finished executing, we suspend and move it to the back of the same queue.

**MFQS vs. RR:**

My MFQS scheduler performed better than my round-robin scheduler. Even though execution time for both is about the same, my MFQS scheduler responded much faster than my RR scheduler. Moreover, my MFQS scheduler had a lower turnaround time than my RR scheduler which means the MFQS scheduler threads are completed quicker from the time of submission than threads handled by the RR scheduler.

The response time of MFQS is much faster than RR because MFQS checks the the first queue, the only place where a new thread is added/submitted to, for a new thread every 500ms while RR goes through the queue of existing threads first before checking the front again for new threads. Moreover, MFQS' turnaround time is also lower due to its constant checking for new threads being added. As soon as it checks for new threads, it executes them, allowing threads with short CPU burst times to complete right away, hence the lower turnaround time.

**Part 2 implemented as FCFS w/o pre-emption vs. RR:**

If part 2 was implemented based on FCFS without pre-emption, the output would be the same. The main difference between a FCFS-without-pre-emption-based part 2 and a MFQS-based part 2 is the execution flow of Queue 2. In FCFS with no pre-emption, threads in queue 2 are executed to completion with no interrupts (pre-emption). In MFQS, threads in queue 2 are executed with interrupts every 500ms. However, for both cases, by the time you get to queue 2, no more new threads are being added, so there are no pre-emptions for either scheduler algorithm, hence, the same output.

Based on the 5 threads being added as part of the ThreadOS package, there will be no performance differences. The threads will be executed in the same flow for both cases, with the only difference that MFQS checks every 500ms for new threads which there are none. This checking barely makes a difference so the performance will be the same. However, in general, the response time will be slower for FCFS w/o pre-emption, and the turnaround time increases for each thread increases as you go down the indices in the queue. FCFS' response time is slower because in queue 1 and queue 2, threads are executed to the full completion of each respective queue's allotted quantum with no pre-emptions. This means that unlike MFQS, it completes the whole span of 1000 ms (queue 1) and 2000 ms (queue 2) before checking for new threads. MFQS, on the other hand, checks every 500 ms for new threads and starts them right away, thus decreasing response time.

As for turnaround time, in FCFS w/o pre-emption, threads will take longer the farther their positions are in the queue because they have to wait longer due to threads before executing for the full queue's time slice without being pre-empted. In MFQS, turnaround time is low due to threads with short CPU bursts finishing pretty fast since this scheduler checks for new threads every 500 ms.