For the second OSP programming assignment, we were told to implement round robin CPU scheduling. We were to do this by changing the methods of the *ThreadCB* and *TimerInterrupt* classes. To begin, I created a GenericList that would contain the ready queue for the threads. I initiated readyQueue in init() of ThreadCB.

From there, I implemented the do\_create() method of *ThreadCB*. There was nothing particularly complicated about this implementation. We simply make sure that if the task is null or if the task cannot handle anymore threads, we do not make a new thread and instead dispatch a thread. If it didn’t fulfill those two conditions, then we made a new thread that copied the properties of the task it is passed and set the status of the thread to ThreadReady. We append this new thread to the readyQueue and we make sure to dispatch the thread, no matter what conditions occur.

do\_kill() was equally straight forward to implement. Based on the status of the thread, we had to do some setup work. If the status was ThreadReady, then we had to remove it from the readyQueue. If the status was ThreadRunning, then we had to determine if this thread we were trying to kill was the current thread by comparing it to the current thread of the current task of the page table base register (PTBR). We made sure to surround this in a try/catch to ensure the prevention of null pointer exceptions. If the current thread matched this thread we were trying to kill, we set the PTBR to null and removed the thread from the current task. Once we have done the setup work, we set the status to ThreadKill, cancelled all pending IO operations, and gave up the resources of this thread in the ResourceCB class. Then, we dispatched a thread. As a final step, we checked if this task had any threads after all of this, and if it didn’t, we killed the task.

The project requirement gives an explicit outline of how to do each method, and the first two methods followed the outline to the dot. With the way I wrote do\_suspend(), I needed to alter to ordering a bit so I could avoid a thread from meeting both the conditions for ThreadRunning, where code changes the status to ThreadWaiting, and then the thread meeting the condition for the ThreadWaiting block. If the thread is already waiting when we call do\_suspend(), we increment the waiting state. If the thread is running when we call do\_suspend(), we check if the thread we are calling the method on is the same as the current thread of the current task based on the PTBR, just like we did in do\_kill(). From there, we make sure that the thread is not in the readyQueue, add it to the event queue, and dispatch.

The do\_resume() is copied from page 41 of the OSP2 text.

The do\_dispatch() simply changes the ordering of step 4 and 3 of the outline. This is because it doesn’t make sense to remove the head of a list that is empty. Other than that, we do the normal check if this thread we are dispatching is the current thread. If it is, we release it by setting the appropriate values to null. If the readyQueue is empty, we set PTBR to null and return failure. Otherwise, we take the head from the readyQueue, set the page table of the task of this thread to the PTBR, ensure the current thread of the task is the thread, and set the status to ThreadRunning. And to simply the code in the TimerInterruptHandler class, we simply set the quantum to 50 here.

The TimerInterruptHandler simply calls ThreadCB.dispatch(). This is implemented this way because now the dispatcher has complete control of the quantum, rather than having control in different places.