**CIS 657: Principles of Operating Systems**

**Lab1 TasksCB**

- Bharath Darapu

- Avinash Peddi

In this lab project we are assigned to finish the following files:  
1. ThreadCB.java

2. TimerInterruptHandler.java

And finish the following method implementations:

1. ThreadCB constructor
2. Init()
3. Do\_create(…)
4. Do\_kill()
5. Do\_suspend()
6. Do\_resume()
7. Do\_dispatch()
8. Do\_handleInterrupt()

The main idea in this module is dealing with the creation, killing, suspending, resuming and finally scheduling of Threads.

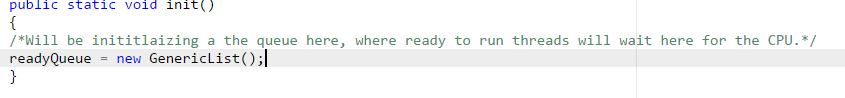
**Approach used:**

There are various ways a thread can be scheduled such as FCFS (First Come First Serve), RR (Round Robin), MLFQ (Multi Level Feedback Queue) and so on.

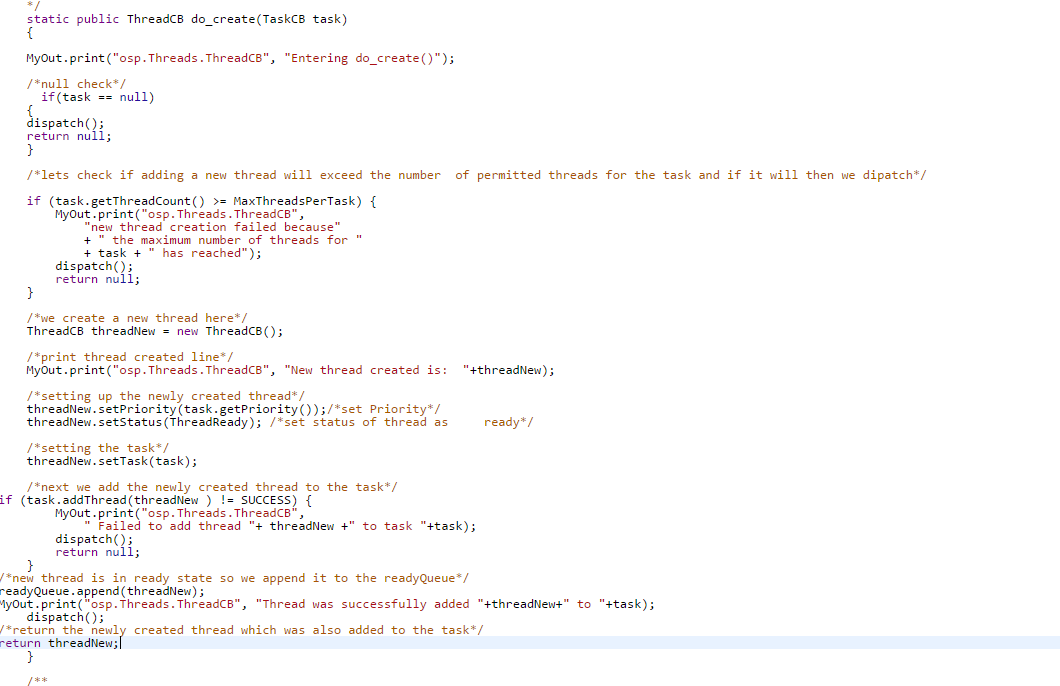
In this project we will be using Round Robin and FCFS. Let me explain the implementation in the order of occurrence in the program.

To begin with, a GenericList (provided by OSP2) is created. This will be used to contain the  
ready queue for the threads. And further we will be initializing the in init() method of ThreadCB class.

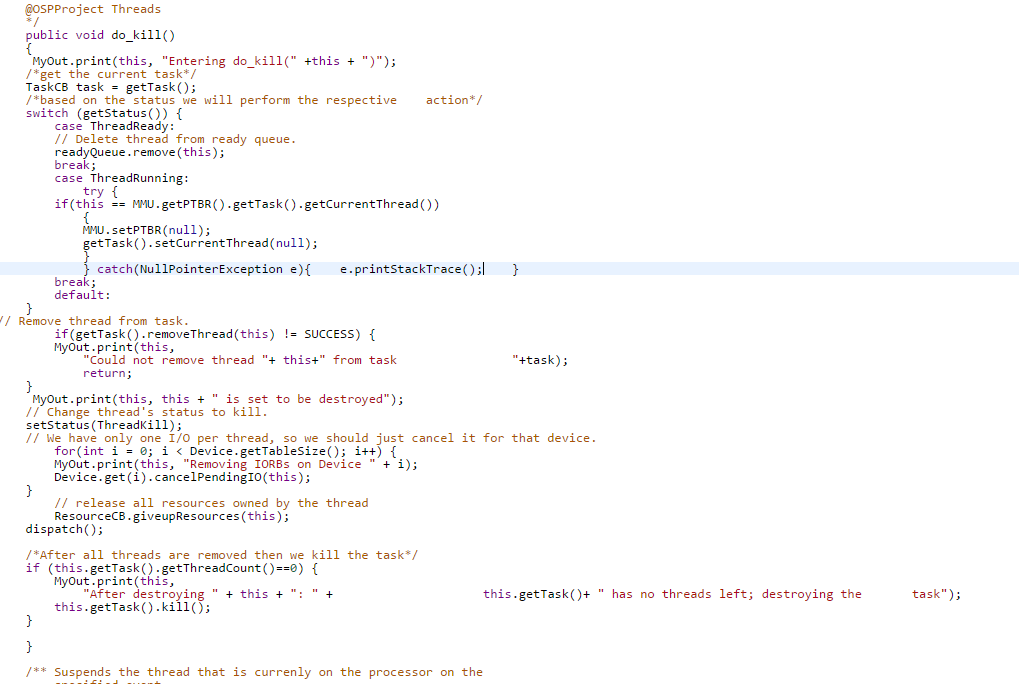




Next, we shall be implementing the do\_create() method as shown below:



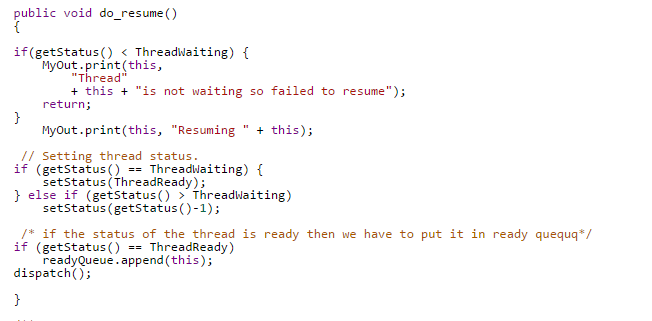
We make sure that if the task is null or if the task cannot handle anymore threads, we do not make a new thread and instead dispatch a thread. If it didn’t fulfill these two conditions, then we made a new thread, and set the priority (which we get from the existing task), set the status as Ready. We then add the newly created thread to the task and then append it to the readyQueue and we make sure to dispatch the thread, no matter what conditions occur. Finally if everything goes well then we return the newly created thread.

Next, the do\_kill() method implementation is as follows:  
  


Based on the status of the thread, we have to switch. 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.

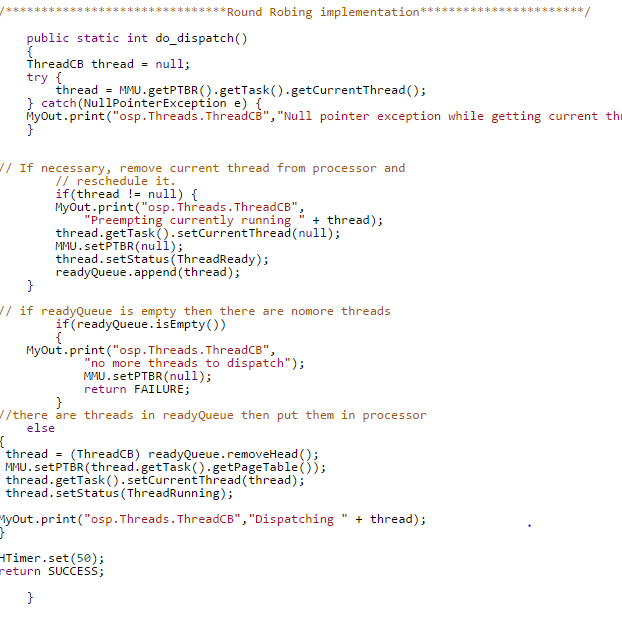
The do\_suspend() implementation is as follows:  


This part was a little tricky as I did not want the method to check if a thread was both running followed by waiting. So the particular ordering was followed.If the thread is already waiting when we call do\_suspend() and then 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, if yes then we set PTBR to null and remove it from the current task. From there, we make sure that the thread is not in the readyQueue, add it to the event queue, and dispatch.

The do\_resume() method implementation:  
 This is quite simple and here we check if the status is less than that of waiting. If it is then the thread is not waiting so we cannot or need not resume it. If the thread is waiting then we change the status to that of ThreadReady. Next if the status is waiting then we decrement the status value by 1. But still wnough to keep it waiting. Then last if the thread is in Ready state then we append it to the readyQueue and finally dispatch.

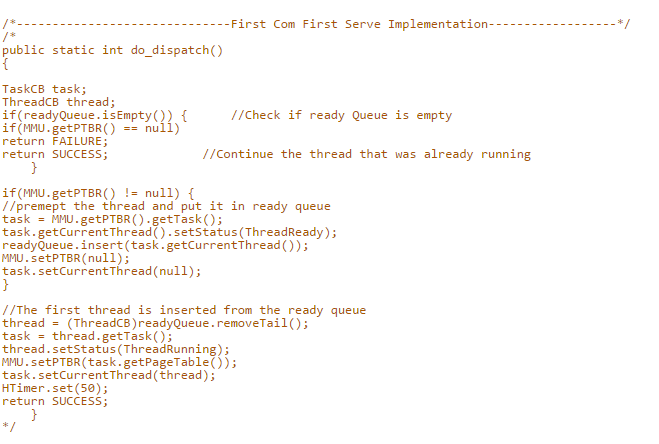
The do\_dispatch() method is the main place where all the thread scheduling happens. The SCheuling policy is coded in here. I have successfully finished RoundRobin and FirstComeFirstServe algorithms. I have tried MultiLevelFeedBackQueue Scheduling but it was not complete.

Let us first see the Round Robing Scheduling:



In this implementation, we first get the current running thread of the task from PTBR. Surround it with a try catch block to handle null pointer exceptions. If it does exist then we we get the thread, remove it from PTBR, set the status as ThreadReady and finally append it to the ReadyQueue. Basically we finished half of pre-empting the thread.

Next we just check to make sure the readyQueue is not empty. If it is then no scheduling is needed. If there are more threads in the readyQueue then we remove it form the Head (removeHead). Put it in PBTR. Get the current thread and then status as ThreadRunning. Next we set the time quantum to 50. Resetting the times can also be done in the TimerInterruptHandler() method TimerInterruptHandler.java. 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.

Next, I also implemented FCFS and the code is:   


The main difference is the place from where we remove the thread from the readyQueue. In FCFS we remove it from the tail.

After all the implementation we navigated to the project path and

* Compiled the .java file using ‘javac –classpath .;OSP.jar \*.java’ and
* Later used ‘java -classpath .;Demo.jar osp.OSP’ command to run the simulation.

OSP.log file which was generated during the execution is also attached.