Anthony Waddell

CSS 430

Program 3 Report

**QueueNode Class**

This class is pretty simple, it just uses some ADT container (I chose vector) to hold thread id’s. The constructor initializes the vector and the class itself contains two utility functions. One that allows a thread to put itself to sleep and one that wakes a thread up. Both of these functions use the keyword ‘synchronized’ to make sure that they are the only one’s who can perform an operation on the thread queue at any given time. For sleepSignal, it monitors if there is anything in the queue before trying to enter and put something to sleep, using wait() to put itself to sleep.

**SyncQueue**

This class makes use of the QueueNode class by including a private array of QueueNode objects of predetermined size 10. It has two constructors, one for a basic initialization of 10 and another that takes a parameter for the number of QueueNodes to instantiate. It comes with three utility functions, one that puts threads to sleep and 2 that wake thread conditions up. enqueueAndSleep() simply checks to see if there if the condition exists within the queue and if so, will return the thread id of the given condition. dequeueAndWakeup takes one or two parameters, the second option being the thread id of the calling thread. Both functions operate almost the same, waking up a specific thread condition but without specifying the thread id as a second parameter, will observe a true FCFS order.

**Kernel.java**

This class was probably the hardest to change and understand just because there was so much given in the .java file. For the first part, we only had to change two code segments for the WAIT and EXIT cases. Going strictly off of what was listed in the program definition, we use some functions from scheduler to get the current thread control block, get it’s tid so we know what condition to put to sleep, and then call enqueueAndSleep on it, putting it in the special waitQueue SyncQueue. EXIT is pretty close but once we have the thread control block we also grab the thread id so we can specify what thread to wake up. In addition, this case differs slightly by calling deleteThread() since the thread is exiting.

In the new Kernel.java, We also have to make changed in the case of RAWREAD, RAWWRITE, and SYNC. I am not quite sure if I got this right since I am having trouble with my program here. I understand that the cses are altered to not allow spin-loops any more and I feel like both my Test3 and the two test classes it tests are all solid but for some reason when running TestThread3b my program enters an infinite loop after the last thread. My assumption would then be that something I am not notifying correctly but then I believe my Part 1 is correct. This makes me think that it has to do something with either the modifications in the kernel.java class or it’s the test classes I wrote. I referred to the FAQ and still am unsure why this loop is occurring. If you have any advice or pointers for me to help me figure this out it would be much appreciated!