Discussion for Mini Project 2

In this project we used Threads and Synchronization to solve the bounded buffer problem.

***Java:***

For the java implementation, we used the scanner method to input both the amount of slave threads as well as the maximum number of jobs which can be done.

In the main method we used two simple for loops to create the threads object as well as calling the generateRequests method.

We created our master and slave threads by implementing the built in Runnable interface which also had the generateRequests and Thread\_routine methods respectively . generateRequests took in the Queue object and created new requests, added them to the queue and started the thread. The length of each thread was randomly determined .

Our Queue class used an arraylist to store the request objects and has methods for pushing, dequeuing etc.

To implement the consumer part we used a while loop to run the thread\_routine method. Thread\_routine dequeues the first awaiting thread and puts the thread to sleep for its respective time period.

The program terminates after the queue is empty and there are no longer any jobs to finish.

**C++:**

C++ does not support Monitors as Java does. While implementing this in Java is as simple as using the *Synchronized* keyword to ensure mutual exclusion, the C++ implementation has to rely on a semaphore class that is created with a conditional variable and a mutex.

So while C++ would seemingly be more difficult to implement, there were some issues with generating threads in the Java version. This may be due to a misunderstanding of how the Thread/Runnable objects work compared with a simple pThread fork to generate the consumer threads.

The tradeoff of programming difficulty really comes down to knowledge and comfort with object oriented programming and abstracting thread forks. C++ has a higher bar for passing variables around, but once comfortable it is quite intuitive to handle lower level processes than Java.

Finally, the C++ implementation differs with respect to the Java, where there is a inputted number of requests to run. In this version, requests will be added until it reaches the limit for the int type ( that could be handled and cleaned up in a more robust version ).