**Chandler Ford | February 24th, 2019**  
**CSS 430 Operating Systems | Program 3 Report**

**Part 1: Specifications/Descriptions**  
The three main files involved with Part 1 were Kernel.old, SyncQueue.java, and QueueNode.java.

Kernel.old  
The main changes here occurred in the WAIT and EXIT blocks, which represent System.join() and System.exit(). In WAIT I got the current TCB and the current thread ID. The latter was passed into the enqueueAndSleep method called by the waitQueue object created from SyncQueue.java. WAIT returns the value returned by this method call through the use of an integer variable childTID. EXIT gets the current TCB, current thread ID, and the parent ID. Depending on the current thread ID value one of the two dequeueAndWakeup methods is called via the waitQueue.

SyncQueue.java  
This file implements the two constructors, enqueueAndSleep, and two dequeueAndWakeup methods specified in the instructions. The enqueueAndSleep method sets a new QueueNode to the position in the index represented by the condition. In this spot is calls the sleep method in QueueNode.java. It returns the child thread ID from sleep() to the Kernel. The two other methods dequeueAndWakeup call the QueueNode.java wakeup() method at the spot specified by the condition and pass the child thread ID.

QueueNode.java  
This file uses a vector of integers along with wait() and notify() to track the child thread IDs. The sleep() method first checks if the vector is empty. If it is the thread is put to sleep with wait(). When woken the first integer from the vector is set as a return value and then removed from the vector. The wakeup() method adds the child thread ID to the vector and calls notify, which alerts the sleeping thread.

**QueueNode.java**

sleep

wakeup

**SyncQueue.java**

enqueueAndSleep

dequeueAnd  
Wakeup

dequeueAnd  
Wakeup

**Kernel.old**

WAIT

EXIT

**Part 2: Specifications/Descriptions**  
The four main files involved with Part were Kernel.java, Test3.java, TestThread3a.java, and TestThread3b.java.

Kernel.java  
The main changes that occurred were in the RAWREAD, RAWWRITE, and SYNC blocks. These blocks had previously caused busy waits but this is averted with the ioQueue. Like waitQueue above, it provides a way to track sleeping threads. The given code had to modified to account for the three different actions. The given code parameters also had to be changed to the ones available in the file.

Test3.java, TestThread3a.java, and TestThread3b.java  
These files were modelled off Test2.java and TestThread2.java. Test3.java creates and executes two threads, one being TestThread3a and the other TestThread2b. TestThread3a represents the thread that should read/write to many blocks randomly across the disk. Multiple disk call blocks were created that had messages of the correct size to be held by the buffers. TestThread3b represents the thread that conducts numerical computation. I created an integer value that is modified by three different for loops using different forms of arithmetic.

**Part 2: Performance Results**The Part 1 Kernel.old ran slightly faster than the Part 2 Kernel.java with a time of 84395 msec compared to 84448 msec. I didn’t see a performance improvement - I believe this may be because my test threads weren’t complex enough. The Part 1 spin loop Kernel might be more efficient for handling less complex/shorter tests than the Part 2 asynchronous disk I/Os Kernel.