Program 3: Synchronization

# Drew Pierce

# Part 1: Implementing SysLib.wait( ) and SysLib.exit( )

Kernal.java was modifed to use SyncQueue.java, I implemented SyncQueue based on SyncQueue.class behavior and spec. When running Shell.java, the joins now wait for the threads to finish and the Shell prompt does not get displayed before the child threads end. Ran into a few issues, first where SysLib.join() was not returning -1 and Shell.class was in an endless while loop waiting for -1 to be returned. Second issue was SysLib.exit() was not terminating the current thread and so the thread was creating issues in Scheduler.java, because it was in a done state but was not set to terminated in the TCB wrapper class..

**Confirmations:**

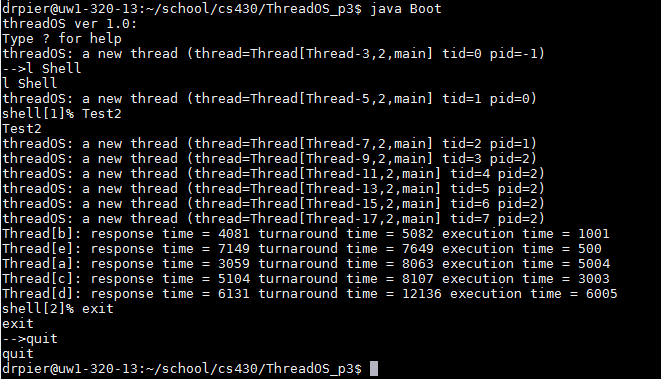
1. PASS: Tests2.java waits for termination for all its five children
2. PASS: Shell.java waits for the termination of Tests2.java for the prompt
3. PASS: Loader.java waits for the termination of Shell.java before displaying prompt

See Figure 1 for confirmation ouput results.

## SyncQueue.java & QueueNode.java

SyncQueue.java was created to create a monitor for the tread ids. I created QueueNode.java which was a Queue of Child Thread ids, this is based on reflection methods of QueueNode.class. SyncQueue.java’s enqueueAndSleep creates a wait and returns the first child id in the queue for that partent this is used in the SysLib.join(). dequeueAndWakeup nodifies other child threads to wait up this is used in the SysLib.exit() so other threads are woken up.

Figure Tests2 Output



# Part 2: Implementing Asynchronous Disk I/Os

I added the recommended ioQueue references in Kernal.java and wrote a custom Test3.java which started two IO processes and two CPU/math processes both. I setup a Linux virtual machine to preform my tests because when running the tests on uw1-320 machines the last Disk process would hang and not return results.

## Test3.java, TestsThread3a.java & TestsThread3b.java

Tests3.java is a simple class that spins up two pair of processes of 1 CPU processes in TestThread3a.java and 1 IO processes in TestThread3b.java. TestThread3a.java just runs math operations in a for loop. TestThread3b.java reads 500 blocks of disk.

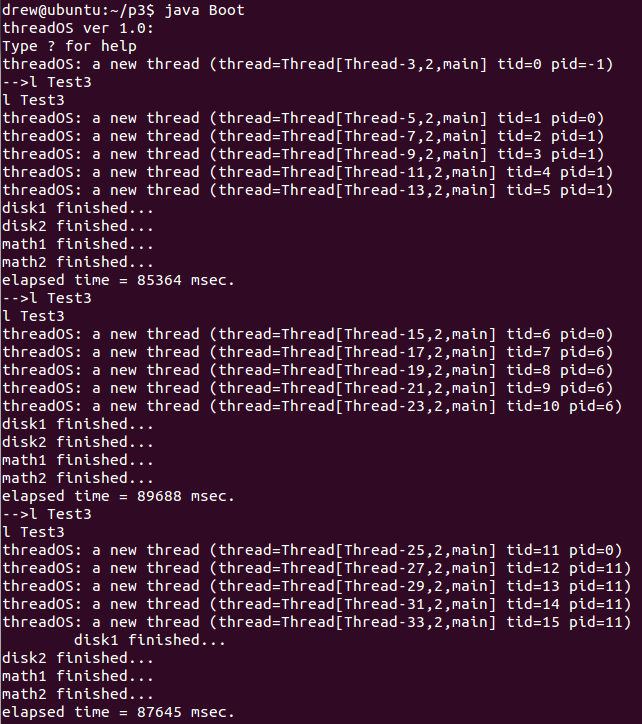
### Performance results

For performance, on average there the ioQueue was about 1 second faster than the busy waiting in the IO. See Table 1 for the results of Kernal.old vs. Kernal.java.

Table - Performace data

|  |  |  |
| --- | --- | --- |
| Try | Test3 ioQueue | Test3 no ioQueue |
| 1 | 85364 | 89735 |
| 2 | 89688 | 87191 |
| 3 | 87545 | 88288 |
| Average | **87532** | **88405** |

Figure Performace Chart comparing results

Figure Test3 IoQueue test output

## Discussion

In reflection of this program; part 1 was very troublesome, knowing that SyncQueue.java’s enqueueAndSleep had to return -1 if it was not able to find a thread to wait on was trick. Part 2 was very clear after reading the description for the fourth time. The main time was spent trying to code the Test classes. I did see a clear difference on the ioQueue but I think if I added more pairs to be kicked off like the directions state, I would have gotten better results with a clear data on the difference of busy waiting vs. monitoring threads. I did run into threading issues on the uw1-320 linux machines so I was not able to observe the results using multiple cores.