Part 1

Part 1 was comprised of building the WAIT and EXIT cases in the kernel using a Queue array that stored a vector queue of sleeping threads. The overall goal of part 1 was for the Shell and the loader to wait for the termination and EXIT of a process.
This was done by creating a class called SyncQueue that would create an array of QueueNode objects when created. SyncQueue works as a monitor for the SysLib.join() call. It will put current thread to sleep and keep track of thread. Once created, SyncQueue has two methods for adding and removing threads from the QueueNodes. Those are enqueueAndSleep and dequeueAndWakeUp; they use QueueNode sleep and wakeup as need to store the threads inside the SyncQueue array's QueueNode.
QueueNode when created makes a Vector that will hold an integer representing the Thread ID. With the two functions mentioned earlier, it can put threads to sleep and wake them up while in the critical section.
When SysLib.exit() called. Kernel will utilize the queue to wake up the thread waiting under the condition which is equal to current thread's parent ID. So that parent thread will be notified.
The results of this part were to make sure that the loader waited for the termination of the Shell. The shell was waiting for the terminated of Test2. This was visible when the Shell[n] would prompt after thread[d] completed. Then when we typed exit into the shell, the loader prompted.

```
gimsuyeon-ui-MacBook-Pro:ThreadOS sooyunkim$ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
shell[1]% l Test2
l
java.lang.ClassNotFoundException: l
shell[2]% Test2
Test2
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=2)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=2)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=2)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=2)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=2)
Thread[b]: response time = 4009 turnaround time = 5041 execution time = 1032
Thread[e]: response time = 7019 turnaround time = 7532 execution time = 513
Thread[c]: response time = 5014 turnaround time = 8075 execution time = 3061
Thread[a]: response time = 3009 turnaround time = 8179 execution time = 5170
Thread[d]: response time = 6014 turnaround time = 12221 execution time = 6207
shell[3]% exit
exit
-->q
q
```

Part 2

Part two was comprised of building Test3. Test3 was built to stress the CPU with a computational thread and stress the DISK with a variety of Read/writes. When the disk busy waiting, user thread needs to keep checking the disk serve state in a spinning loop. So, I use enqueueAndSleep to put the thread into wait, and let CPU to server other read thread. These two threads are done in pairs in which the user specifies how many pair to create when running Test3. These tests were broken into two classes, TestThread3a and TestThread3b. Test3.java will call TestThread3a.java (computation thread) and TestThread3b.java (disk thread) number of argument times and keep track the beginning and ending time for all the thread finished.
TestThread3a was responsible for stressing the CPU by doing a series of computations. The value that it creates is irrelevant and the algorithm it uses is a random creation. The main goal of this class was to do a series of computation in the thread to use the CPU. After it will cout "done comp" and exits.
TestThread3b was responsible for stressing the DISK with read/writes to that DISK. This was done by going through each block in the DISK (1000) and WRITING and READING from that location. After that loop, it cout "done read write" and exits.

Spinning Kernel

```
gimsuyeon-ui-MacBook-Pro:ThreadOS sooyunkim$ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test3 3
l Test3 3
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=1)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=1)
done comp
done comp
done comp
done disk
done disk
done disk

elapsed time = 251457 msec
-->q
q
```

Non-spinning Kernel

```
gimsuyeon-ui-MacBook-Pro:ThreadOS sooyunkim$ java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test3 3
l Test3 3
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-9,2,main] tid=3 pid=1)
threadOS: a new thread (thread=Thread[Thread-11,2,main] tid=4 pid=1)
threadOS: a new thread (thread=Thread[Thread-13,2,main] tid=5 pid=1)
threadOS: a new thread (thread=Thread[Thread-15,2,main] tid=6 pid=1)
threadOS: a new thread (thread=Thread[Thread-17,2,main] tid=7 pid=1)
done comp
done comp
done comp
done disk
done disk
done disk

elapsed time = 248232 msec
-->q
q
```

According to the elapsed time results. The non-spinning kernel works better than the spinning one with 3 pairs of threads. And this advantage will be more distinctly when running large pairs of threads. This because when put the thread into waiting queue, it will wait until be waked up instead of keeping check the ready state. In this way, CPU resources will be relinquished, and other ready thread can use these freed resources.
When running the Test3 in both the spinning Kernel and the non-spinning Kernel, I noticed that the non-spinning Kernel (the one that does not implement spinlocks and allows the thread to sleep) finishes faster than the spinning kernel (that implements Spinlocks).
It is good to note that this data will vary based on that machine due to the number of cores and threads that the CPU has. But I conclude that the kernel that does not implement spinlocks runs faster than the one that does.