Part 1

Part 1 includes a modified Kernel.old.java, SyncQueue.java, and QueueNode.java. Kernel.old's WAIT and EXIT cases were modified to make use of new functionality provided by SyncQueue and QueueNode that allows threads to be put to sleep and woken up conditionally, which is a finer grained approach than the monitors provided by Java. SyncQueue allows for different conditions to be signaled, and can wakeup and put threads to sleep by utilizing a particular QueueNode in a list of QueueNodes that is particular to the condition being signaled.

Part 2

Part 2 includes an improved Kernel.java that instead of busy-waiting for i/o, it goes to sleep until an i/o operation can be executed and again until the operation is completed. The performance increase from this improvement can be seen in the details in the table below, as measured by Test3.java. Test3 executes between 1 and 4 pairs of threads. The first thread in a pair is a cpu bound operation defined in TestThread3a.java, and the second thread in a pair is an i/o bound operation defined in TestThread3b.java. The cpu bound work in TestThread3a is a simple recursive fibonacci function that has a fixed limit of fibonacci(50). Its only purpose is to tax the cpu while other i/o bound operations are executed. TestThread3b in turn reads and writes to a fixed number (750) memory locations. Its only purpose is to create busy work for the program to measure i/o performance.

Test3 performance results (in milliseconds)

Thread pairs	Kernel - busy wait	Kernel - async i/o
1	55665	58748
2	89998	82688
3	132865	126469
4	176372	168311

Discussion about performance results

The async i/o kernel was demonstrably faster than the busy-wait kernel for 2 or more thread pairs. The only exception was that the busy-wait kernel was faster when there was only one thread pair, but this can be rationalized in that the pair had no other threads to compete with -- only itself -- and neither was there any overhead of switching between threads on wakeup and notify.