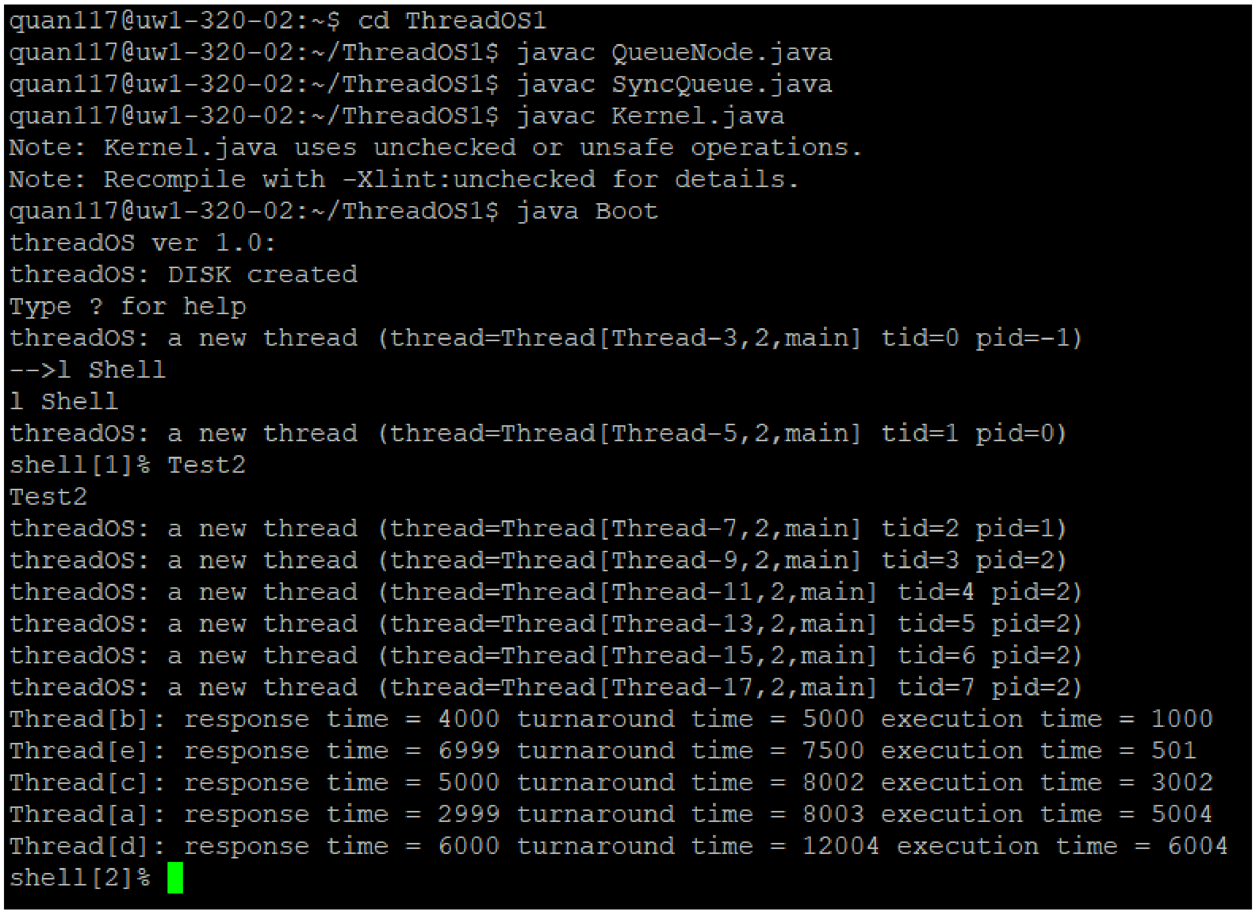
Part 1



```
quan117@uw1-320-02:~$ cd ThreadOS1
quan117@uw1-320-02:~/ThreadOS1$ javac QueueNode.java
quan117@uw1-320-02:~/ThreadOS1$ javac SyncQueue.java
quan117@uw1-320-02:~/ThreadOS1$ javac Kernel.java
Note: Kernel.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
quan117@uw1-320-02:~/ThreadOS1$ java Boot
threadOS ver 1.0:
threadOS: DISK created
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]% 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 = 4000 turnaround time = 5000 execution time = 1000
Thread[e]: response time = 6999 turnaround time = 7500 execution time = 501
Thread[c]: response time = 5000 turnaround time = 8002 execution time = 3002
Thread[a]: response time = 2999 turnaround time = 8003 execution time = 5004
Thread[d]: response time = 6000 turnaround time = 12004 execution time = 6004
shell[2]%
```

The SyncQueue functions as a monitor for the SysLib.join() call. The SyncQueue uses the QueueNode to manage threads waiting for a given condition. SyncQueue will put current thread to sleep and keep track of it by storing it into the waiting queue. When SysLib.exit() is invoked, Kernel wake up the thread waiting in the queue under the condition, which is equal to the current thread's parent thread ID. Parent thread will be notified.

For more in depth detail about the specification/algorithm, go to the comments of the code.

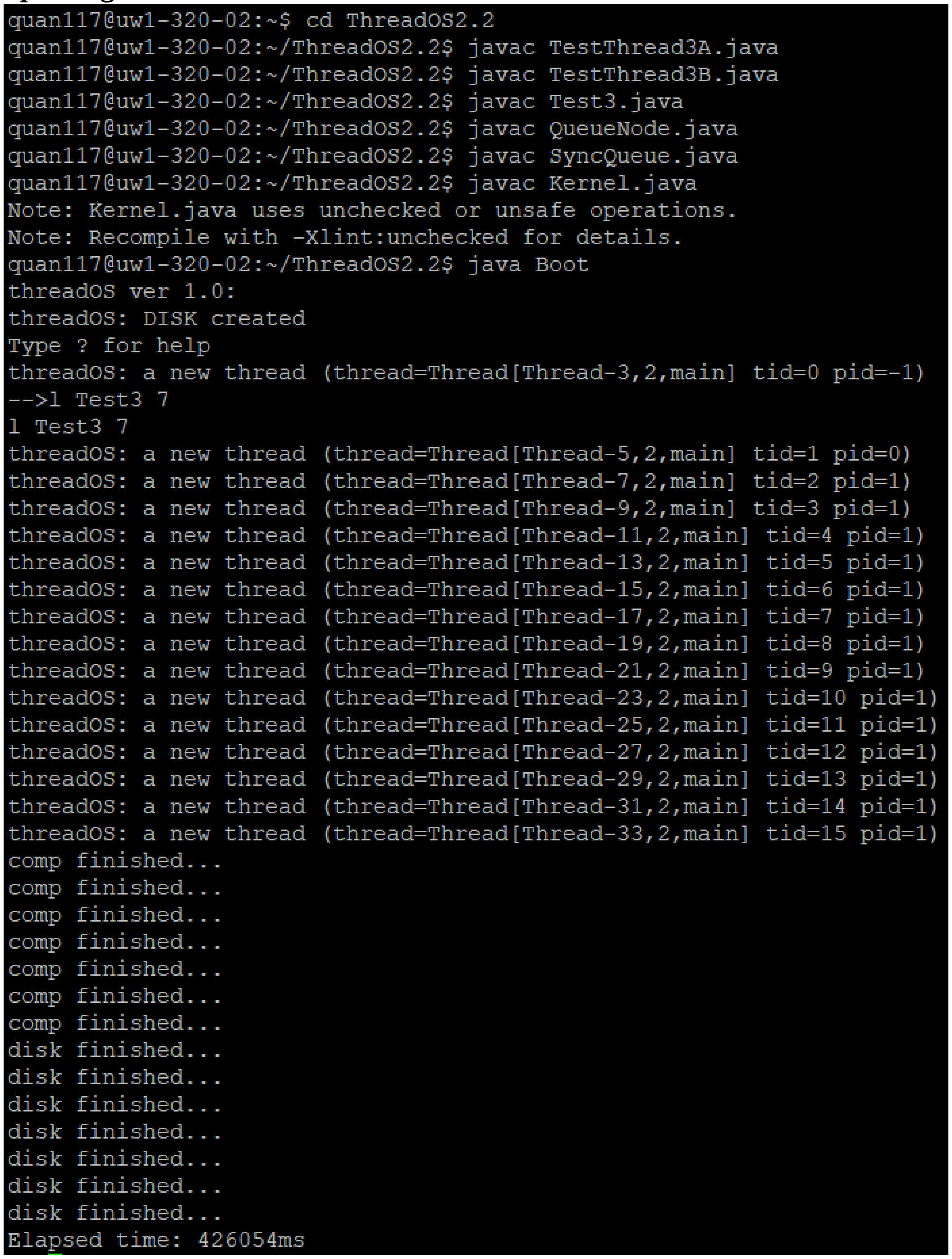
QueueNode
sleep(): put calling thread to sleep until monitor have another thread, then invoke notify().
Return the parent or -1.
wakeup(): add the thread ID to the waiting queue. Then wake up the parent thread via notify().

SyncQueue
enqueueAndSleep(int condition): enqueue the calling thread and put to sleep until given condition is satisfied.
dequeueAndWakeup(int condition, int tid): dequeue the calling thread, and wake up for given condition. If there is no tid, assumed that tid = 0.

Part 2:

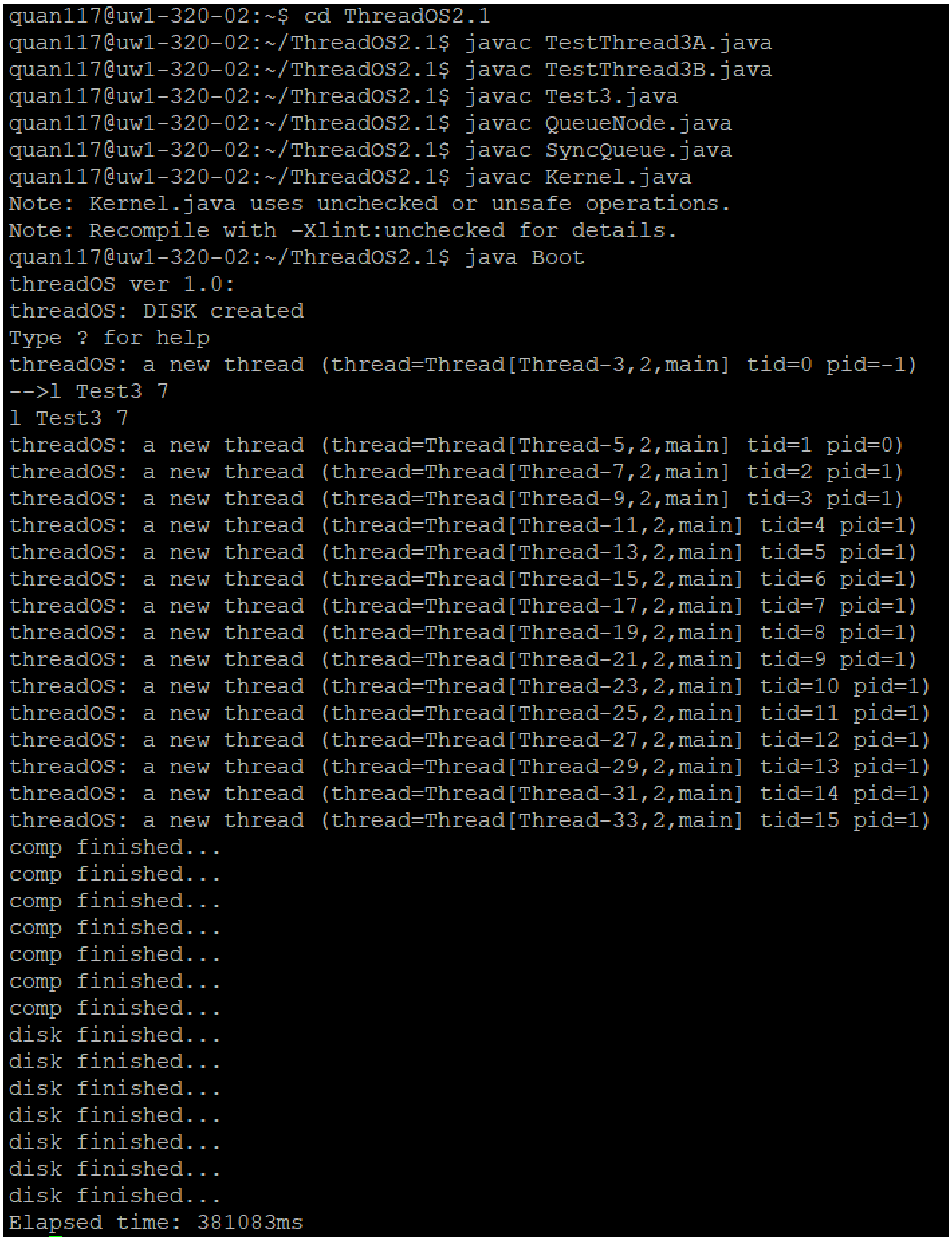
Kernel_old: Spinning Kernel. Use busy wait. Modified only case WAIT and EXIT to implement the SyncQueue. Also known as busy waiting.
Kernel_new: Non-spinning Kernel. Don't use busy wait. Enqueue the threads and put them to sleep. Further modification of Kernel_old in cases RAWREAD, RAWWRITE, SYNC to implement the SyncQueue. Also uncomment the INTERRUPT_DISK case to dequeueAndWakeup threads. Also known as interrupt.

Spinning Kernel Result:



```
quan117@uw1-320-02:~$ cd ThreadOS2.2
quan117@uw1-320-02:~/ThreadOS2.2$ javac TestThread3A.java
quan117@uw1-320-02:~/ThreadOS2.2$ javac TestThread3B.java
quan117@uw1-320-02:~/ThreadOS2.2$ javac Test3.java
quan117@uw1-320-02:~/ThreadOS2.2$ javac QueueNode.java
quan117@uw1-320-02:~/ThreadOS2.2$ javac SyncQueue.java
quan117@uw1-320-02:~/ThreadOS2.2$ javac Kernel.java
Note: Kernel.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
quan117@uw1-320-02:~/ThreadOS2.2$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test3 7
l Test3 7
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)
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=1)
threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=1)
threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=1)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=1)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=1)
threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=1)
threadOS: a new thread (thread=Thread[Thread-31,2,main] tid=14 pid=1)
threadOS: a new thread (thread=Thread[Thread-33,2,main] tid=15 pid=1)
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
Elapsed time: 426054ms
```

Non-spinning Kernel Result:



```
quan117@uw1-320-02:~$ cd ThreadOS2.1
quan117@uw1-320-02:~/ThreadOS2.1$ javac TestThread3A.java
quan117@uw1-320-02:~/ThreadOS2.1$ javac TestThread3B.java
quan117@uw1-320-02:~/ThreadOS2.1$ javac Test3.java
quan117@uw1-320-02:~/ThreadOS2.1$ javac QueueNode.java
quan117@uw1-320-02:~/ThreadOS2.1$ javac SyncQueue.java
quan117@uw1-320-02:~/ThreadOS2.1$ javac Kernel.java
Note: Kernel.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
quan117@uw1-320-02:~/ThreadOS2.1$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test3 7
l Test3 7
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)
threadOS: a new thread (thread=Thread[Thread-19,2,main] tid=8 pid=1)
threadOS: a new thread (thread=Thread[Thread-21,2,main] tid=9 pid=1)
threadOS: a new thread (thread=Thread[Thread-23,2,main] tid=10 pid=1)
threadOS: a new thread (thread=Thread[Thread-25,2,main] tid=11 pid=1)
threadOS: a new thread (thread=Thread[Thread-27,2,main] tid=12 pid=1)
threadOS: a new thread (thread=Thread[Thread-29,2,main] tid=13 pid=1)
threadOS: a new thread (thread=Thread[Thread-31,2,main] tid=14 pid=1)
threadOS: a new thread (thread=Thread[Thread-33,2,main] tid=15 pid=1)
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
comp finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
disk finished...
Elapsed time: 381083ms
```

Test3
Takes in an argument containing X times to run computation and disk threads. Execute TestThread3A and TestThread3B with X number of times using for loop. Afterward, two for loops to join the threads, print out the corresponding finish statement for computation and disk threads. The elapsed time will be calculated based on the difference of the submission time at the start of the run() and the completion time at the end of the last for loop. Print out the result and exit.

TestThread3A
Perform the computation. Run a mathematical problem for 10000 times with factorial of 15. Basically, this class is designed to tax the system computation ability. Exit when done.

TestThread3B
Perform the read operation and write operation from disk. Created a byte array of 512, represents one disk block. For 1000 times, it will write and read from the disk. Exit when done.

Runtime Discussion
The output is based on calling the creations of 7 pairs of threads. The elapsed time of the old kernel is longer than the elapsed time of the new kernel. The reason for this is because the new kernel implements interrupt and perform context switch. The old kernel implements busy waiting and keeps the thread awake and keeping check the ready state for the lock. The new kernel put the thread to sleep and place it into waiting queue. Thread will be wake up when ready instead of staying awake checking the ready state. In this way, CPU resources will be relinquished for other ready thread to use these freed resources.