CS 226: Operating systems and networking
Spring Term, 2008

Homework 7

Due: Wednesday 5/7 at 11:59pm

Complete the following. Submit the first problem via `handin` as assignment `hwk7`. The second problem can be submitted via email, on paper, or as a comment in the code for the first problem.

1. (8 points) Take a look at `FastSerialPrimes.java` in the course directory. The primality testing in this program improves upon the version used in `SerialPrimes.java` by only dividing by smaller prime numbers. If you time it, it actually runs faster than the fixed version of the threaded prime finder. (I have placed my version of that program into the course directory as `ThreadedPrimesFixed.java`.)

   Write a partially multi-threaded version of `FastSerialPrimes`. Your version should compute small primes serially and then use two threads to count larger ones. In this context, "small" means at most $\sqrt{2,000,000}$. The small primes can be stored in a shared `ArrayList` while the larger ones should be counted using a private variable in each thread. (I did not store the actual values of the larger primes, but you may do this.) My version achieved a speedup near 2.

2. (4 points) The reason for creating a version of `FastSerialPrimes` that is only partially multi-threaded is that the version that I wrote modeled after `ThreadedPrimesFixed` was significantly slower than the serial version. In addition, it had a race condition. Explain what type of overhead might be slowing this program down and also what race condition would appear when writing a fully multi-threaded version of `FastSerialPrimes`.