Computer Science 435
Project 4: A Concurrent Computation of Π

Due: Fri., Dec. 9, at the beginning of class. You may implement your solution either in Java (using Jthreads) or C (using Pthreads).

In class we discussed a Monte Carlo simulation method for approximating the value of π. For this project, you are to write a multi-threaded program to perform this simulation and periodically print out the results.

Your program should take three command line arguments, which will all be positive integers:

1. The number of threads that will perform Monte Carlo experiments.
2. The total number of experiments to be performed.
3. The number of experiments between the printing of a new approximation.

You should structure your program with three kinds of threads:

The main thread initializes shared variables and creates the other threads.

One or more simulation threads generating and check the random points, and share data with and wake the printing thread as required by the third parameter.

A single printing thread calculates and prints out the approximations provided by the simulation threads.

Since the simulation threads will be producing data used by the printing threads, you will need to create a list of some type to share this data. In addition you will need appropriate variables to guarantee both atomic access to the list, and proper conditional synchronization between the simulation threads and the printing thread.

Simulation threads operate in a loop. One each iteration, a thread should generate a point in the unit square, determine if that point is on the interior or exterior of the unit circle, and update appropriate counters and accumulators so that the approximation of π can be calculated. If a thread performs an experiment that is a multiple of the third argument, it should place the

information needed to calculate the current approximation in the shared list and then wake the printing thread. When awoken, the printing thread should print all the approxiations in the list.

When a simulation thread detects that the requested number of experiments have been performed, it should put place one last piece of information in the list, wake the printing thread, and then terminate itself and all the other simulation threads.

Strive to make your program as concurrent as possible: Do not perform any actions atomically unless mutual exclusion is actually required for correct execution. Do not delay an threads unnecessarily. Here is an example of what the output of your program should look like when run with the parameters `2 100 24`:

```
 24: 3.333333254
 48: 3.250000000
 72: 2.888888836
 96: 3.000000000
100: 3.039999962
```

What to turn in: When you have finished with the program, email a copy of the source file(s) to me (as a tarball if your program consists of multiple source files), along with any instructions I will need to compile and execute it. Submit a hard copy of your program within 24 hours of submitting your electronic copy.

A note on generating random numbers:

- If you are programming in C, use the random number generator available in the standard library. `Void srand(unsigned int seed)` initializes the generator, and `int rand()` returns a random number in the range [0, RAND_MAX]. RAND_MAX is a constant defined in `stdlib.h`. Note that the standard library version of `rand()` is not thread safe: You will need to ensure that each call is atomic by using a mutex variable.

- If you are programming in Java, use either a `java.util.Random` object or a `java.util.concurrent.ThreadLocalRandom` object. See the Java API for a discussion of the performance and use of these objects in concurrent programs.