Lab Exercise 1 – Concurrency

CIS 656 – Fall 2017

Due Date: 6pm September 21, 2017
Grade Weight: 15 points (out of a total of 100 Lab points)

Objective

The objective of this project is to give the student hands-on experience with the notion of concurrency in programming. In particular we will look at the consequences of not properly synchronizing the updates of shared data, as well as the costs associated with synchronizing.

Exploring Java Threads

If you have never worked with Java Threads before, or you need a refresher, please complete the following tutorial:

http://docs.oracle.com/javase/tutorial/essential/concurrency/

Write a Java program (Program A) that spawns n threads, where n is a program argument. These threads access a shared counter (initialized as 0) in a loop. In each iteration, each thread reads the shared counter to a local (stack) variable, increments it, and stores it back to the shared counter. When all threads complete 1 million iterations each[1], the program stops and prints the value of the shared counter and the total elapsed time[2]. Do NOT use any kind of thread synchronization in this program.

Prepare a written report that tabulates the output for 2, 4, 8, 16, and 32 threads.

In the report, also address the following question:

1. Is the counter always equal to n * 1000000, where n is the number of threads created? Explain why, or why not.

Next, create a new version of the same program (Program B) that uses one of Java's synchronization primitives to ensure that only one thread updates the shared counter at a time. You can use any Java mechanisms you want for this (e.g., implicit monitor/synchronize, ReentrantLock, etc.).

Tabulate the output for 2, 4, 8, 16, and 32 threads for this version of the program and address these questions:

[1] You can have the main thread pause until all spawned threads are done by using the Thread.join() call. You can also use an ExecutorService and the awaitTermination() method.
[2] You can determine the total elapsed time by calling System.currentTimeMillis() before and after execution.

2. Is the counter always equal to n * 1000000, where n is the number of threads created? Explain why, or why not.

3. Analyze the differences in elapsed time between Program A and Program B. Is there a significant difference? Explain why or why not.

Deliverables

Zip up your Java project for the code described above, and submit along with your written report to Blackboard. If you use github.com you can submit a link to a github repo to Blackboard instead of a zip of the code. However, do make sure I have visibility on the repository (my github name is jengelsma).