Concurrency:
Multi-core Programming and Data Processing

Assignment 01

Professor: Pascal Felber
Assistant: Isabelly Rocha

March 7, 2019

1 Shared Counter

1. Write a Java program that spawns multiple threads. All threads access a shared counter (initialized to 0) in a loop (of i iterations). All threads read the counter to a local variable (on the stack), modify the variable (as follows) and then store it back to the counter. The threads are split in two groups with respect to the modification of the variable: threads in first group will increment the variable, threads in second group will decrement it. We define n, number of threads in first group, and m, number of threads in the second group. n, m and i will be given as runtime arguments to the program. When all threads finish, the program prints the value of the shared counter and the duration of the execution. (Check the value for equal number of incrementing and decrementing threads, i.e. $n = m$, and for $i = 100000$ iterations). Save the source to Ex1NoSync.java.
2. Modify this program using the keyword synchronized to protect the counter. The counter must be 0 at the end of the execution. Save your code to Ex1Sync.java.
3. Modify the first program to use java.util.concurrent.locks.ReentrantLock class. The counter must be 0 at the end of the execution. Save your code to Ex1ReentrantLock.java.
4. Run the 3 programs with equal sets of 1, 2, 4, and 8 threads and report the results in a textual table in a file named *Ex1.txt*. Report the runtime in milliseconds, the speedup and your machine specification.

2 Prime Numbers

The goal of this exercise is to identify all prime numbers between 1 and n^{10} (where n is the number of cores in your computer). Implement two variations of a program which distributes work between n threads. The first variant should have each thread i testing numbers in the range $[(i-1)*n^9+1, i*n^9]$. The second variant should have a shared counter used by threads to pick the next number to test. Ideally each thread should take approximately the same time processing its n^9 numbers.

1. Run a program with 1 thread that identifies all prime numbers between 1 and n^{10} and measure how long the program takes to execute.

2. Run the first and second variants described above with n threads and measure how long they take to execute.

3. Report the results in a textual table in a file named *Ex2.txt*. Report the runtime in milliseconds, the speedup and your machine specification.

3 Producer-Consumer Problem

Suppose we have two types of threads: Producers and Consumers, sharing a circular buffer. Each Producer deposits data at a suitable position in the buffer, denoted by a variable *in* (i.e, the next position available in the buffer) and advances the variable *in*, while each Consumer retrieves the data item at the position denoted by the variable *out* and advances this variable. A producer cannot deposit its data if the buffer is full. Similarly, a consumer cannot retrieve any data if the buffer is empty. Write a program that can correctly coordinate the producers and consumers and their depositing and retrieving activities. For simplicity, the number of Producer threads is equal to the number of Consumer threads which is denoted by t (program argument). The buffer will have n integer elements (program argument).