Concurrent Programming
COMP 409, Winter 2019

Assignment 1

Due date: Wednesday, February 6, 2019
6pm

These instructions require you use Java. All code should be well-commented, in a professional style, with appropriate variables names, indenting, etc. Your code must be clear and readable. **Marks will be <u>very generously</u> deducted for bad style or lack of clarity.**

In this assignment you may use only basic synchronization constructs; in Java, this includes the `synchronized` and `volatile` keywords, `wait`/`notify`, and the `java.util.concurrent.atomic.*` classes, as well as basic Thread methods, such as `start`, `join`, `sleep`, `currentThread`, `yield`.

All shared variable access must be properly protected by synchronization—there must not be any *data races,* but also no unnecessary use of synchronization. Unless otherwise specified, your programs should aim to be efficient, and exhibit high parallelism, maximizing the ability of threads to execute concurrently. Please stick closely to the described input and output formats.

1. Various image effects can be created by performing a *convolution*. Given an input image, an output image is constructed by computing each output pixel value as a weighted summation of the corresponding input pixel and the pixels forming its immediate neighbourhood in the input image. If weights are stored in an array (a *kernel*), the following pseudo-code summarizes a naive, but simple approach: **20**

```
for each row:
   for each pixel in the row:

      set r,g,b accumulators to zero

      // Logically center the kernel on the given pixel.
      // A sum is computed by adding the product of each kernel
      // weight and its corresponding input pixel value (per channel).
      // This sum is the output pixel value (per channel).

      for each kernel row:
         for each value in the kernel row:

            if the input pixel offset by the kernel displacement is
                         within the image dimensional bounds, then:

               for each channel r,g, and b separately:
                  multiply the pixel colour value by the corresponding kernel value
                  add result to accumulator
            endif

      constrain r, g, and b values to be within 0..255
      set output image pixel to final r,g,b value (with alpha of 0xff)
```

Template code is provided that reads in an image, creates an empty output image, and writes it to a fixed filename. The program accepts 1 command-line argument, the number of threads to use (defaults to 1).

Implement a basic kernel computation. To test it, use the kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

and verify that you generate an image the same as the `exampleoutput.png` provided. Feel free to experiment with different kernels as well, but use this kernel in your submitted code and in doing performance tests.

Your submitted code should use multiple threads (as specified on the command line) to improve performance.

Add timing code (using `System.currentTimeMillis`) to time the actual work of the program (including all threads, but excluding all I/O). The time taken in milliseconds should be emitted as console output. No other console output should be included.

Plot performance versus the number of threads. Note that when timing the program you should execute the exact same execution scenario several times (at least 5), discarding the first timing (as cache warmup), and averaging the rest of the values. Provide a graph of the relative speedup of your multithreaded versions over the single-threaded version for 1–8 threads. You should be able to observe speedup for some number(s) of threads, but perhaps not all values. This of course does require you do experiments on a multi-core machine.

Submit your solution as a file `q1.java`. Include your performance analysis, consisting of a speedup plot and a textual explanation of your results (relating them to your synchronization strategy and machine used for testing) as separate documents.

2. This question disallows lock-based synchronization—you may not use `synchronized`, nor `mutex`es, and must rely entirely on using basic R/W atomicity. You must still avoid data races of course. **10**

   Develop a program, `q3`, in which 3 threads compete to traverse and modify a circular, singly linked list of single-characters. The list is initialized to contain 3 items "A", "B", and "C".

   Thread 0 scans through the list printing out the character it encounters (on the same output line, space-separated) at each node, sleeping 100ms between each output.

   Thread 1 scans through the list, removing each entry it encounters with a 1/10 chance, and sleeping 20ms before moving onto the next entry. The original starting items, "A", "B", and "C" may never be deleted, ensuring the list will always contain at least 3 items.

   Thread 2 scans through the list, inserting new entries (for random single characters which are not "A", "B", or "C") after each entry it encounters, also with a 1/10 chance of doing so, and sleeping 20ms before moving onto the next entry.

   The simulation should run for 5s of execution, and then stop all list modifications and print out on a separate line the final contents of the linked list. Note that the potential for conflict between threads 1 and 2 mean that some nodes may not be fully or successfully inserted or deleted. Nevertheless, the integrity of the circular linked list must be guaranteed—it must remain circular, and threads should not crash or end up unable to traverse the list.

   Ensure concurrency is maximized—it should be possible for all threads to be performing their actions at the same time.

What to hand in

Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files. For any written answer questions, submit either an ASCII text document or a .pdf file. Do not submit .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.
This assignment is worth 10% of your final grade. **30**