Concurrent Programming

COMP 409, Winter 2020

Assignment 1

Due date: Thursday, February 6, 2020
6pm

General Requirements

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 very generously deducted for bad style or lack of clarity.**

There must be no data races: all shared variable access must be properly protected by synchronization. Any variable that is written by one thread and read or written by another should only be accessed within a synchronized block (protected by the same lock), or marked as volatile. At the same time, avoid unnecessary use of synchronization or use of volatile. 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.

Your assignment submission **must** include a separate text file, `declaration.txt` stating "This assignment solution represents my own efforts, and was designed and written entirely by me". Assignments without this declaration, or for which this assertion is found to be untrue are not accepted. In the latter case it will also be referred to the academic integrity office.

Questions

1. The goal here is to use multiple threads efficiently to draw rectangles on an image. Rectangles should be axis-aligned, drawn with a 1-pixel thick black border, but filled in with a random colour. **20**

   Each thread repeatedly attempts to draw a random rectangle. It chooses a random spot to start (a corner or center, up to you) and a random size (within the image limits), and ensures the resulting rectangle at that location will not overlap with other in-progress rectangles, and draws it. Once a thread starts drawing a rectangle it must fully complete the process. You may not use any built-in line, rectangle, or filling primitives to do this.

   Provide a program, `q1.java` that accepts the following command-line arguments, w, h, n, and k. It should launch n threads to draw a total of k rectangles on a $w \times h$ image. Note that n may or may not evenly divide k.

   Choose w, h, k such that the program typically takes a few seconds (or at least several hundreds of milliseconds) to run with $n = 1$. Add timing code (using `System.currentTimeMillis`) to time the program from the point the threads are launched to the point when all threads have completed their work. Once all threads have completed, the time taken in milliseconds should be emitted as console output and the image output to a file, named `outputimage.png`.

   Plot performance (speedup) versus the number of threads, keeping w, h, k fixed. Given the randomness, you will need still to do several runs at each thread-count (at least 5) averaging the timings (you may opt to discard the initial run as a cache-warmup). Provide a graph of the relative speedup of your multithreaded versions over the single-threaded version for 2, 3, and 4 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. Note that achieving speedup is not necessarily trivial: beware of sequential bottlenecks from different methods and APIs you may use or implement.

Example/template code that creates an image of a given size and writes it to a file is provided with this assignment description.

Provide your source code (`q1.java`), and as a *separate document* your performance plot with a brief textual explanation of your results (why do you get the speedup curve you get).

2. This question disallows lock-based synchronization—you may not use `synchronized`, nor `mutex`es, nor any of the hardware atomics (TS, FA, CAS). You are allowed only to use basic R/W atomicity; threads should not spin/retry or be blocked (other than as described below)—this means you should not be attempting to build your own locks either. You must still avoid data races of course. **10**

   Develop a program, `q2`, in which 3 threads act on a leaf-threaded binary tree. The tree initially consists of 3 nodes, a root node and left/right child nodes. To identify them easily (for humans) nodes (leaf nodes at least) will need (heuristically) unique names: use short random strings, giving the same string in all lower-case to a left child, and in upper-case to its matching right child. The leaves of the tree should be in-order threaded (singly linked), with a separate, static head and tail nodes to store the beginning and end of the threaded list.

   Thread 0 scans through the leaves using the threading. At the head node it prints out a "*", and then (on the same line) prints out the names of the nodes it encounters (on the same output line, space-separated), sleeping 50ms between outputting each individual node-name. Once it reaches the tail, it prints a newline, pauses for 200ms, and then starts again from the head.

   Thread 1 also repeatedly goes through the leaf list. In this case, however, at each node it encounters, and with a 1/10 chance, it may choose to expand the node, giving it two new children (with a random name, again lower-case for the left child and upper-cased for the right child). Children should be properly linked into the tree, including fixing up the threading to ensure it still represents a leaf-threading of the tree. Whether children are added or not, the thread sleeps for 20ms before moving to the next node. Once it reached the end of the thread, it also pauses for 200ms and starts again.

   Thread 2 repeatedly performs depth-first searches through the tree from the root to count the number of nodes in the tree. At each node it enters it increments its count and pauses 10ms. After returning to the root it emits the total count (and a newline). It then waits 200ms before beginning a new count.

   The simulation should run for 5s of execution. Once 5s is reached, wait for each thread to complete its full traversal (reach the tail or return to the root). Print out a newline, the final count from thread 2, another newline, and the final contents of the leaf threading (each name space-separated, followed by a newline).

   Note that the activities of these threads necessarily conflict. You should nevertheless ensure that threads do not crash, and that the tree structure and leaf threading do not end up corrupted.

   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 declaration and assignment files 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. Avoid .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**