Concurrent Programming

COMP 409, Winter 2024

Assignment 1

Due date: Wednesday, January 31, 2024
Midnight (23:59:59)

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. This means all shared variable access must be properly protected by synchronization: any memory location 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. A metro system is managed by a single overseer, but consists of 4 metro stations. The metro has a maximum capacity of n people, so each metro station needs to keep track of the number of people entering and leaving the station. To keep the tracking efficient, each station only maintains its own local count. In order to verify overall capacity is not exceeded every q milliseconds the overseer tells each station to stop letting people in or out of the station, gathers up all the station counts and computes the total sum. If they find the total is $< n$ then they let each station know they can let people in/out again, and re-check q milliseconds later. If, however, the total is $\geq n$ they tell each station to disallow people entering and only let people out, re-checking/re-counting every $q/10$ milliseconds until the total is $< \lfloor 0.75n \rfloor$ before allowing people to enter again (and returning to their regular q ms checking periodicity). **10**

   Model this scenario using 5 threads: one for the overseer, and 4 for the metros. Each station thread models a passenger passing through its entrance/exit every 10ms, with 51% of the time that passenger trying to enter the metro and 49% of the time they are leaving. Note that negative station and global counts are possible!

   Your solution should aim at allowing maximal concurrency, while still ensuring correct behaviour. Use Java's `synchronized` keyword to coordinate between threads and ensure the overseer's total count is accurate. Stations, however, should operate independently from each other as much as possible—it should be possible for people to enter/leave at the same time from 2 different stations, as long as the overseer hasn't frozen all metro access.

   Provide a program `q1.java` that models this simulation. Your program should accept integer command-line parameters, $n \geq 10$, and $q \geq 10$ in that order. You can assume q is evenly divisible by 10. Each time the overseer computes a total print out (to the console) the count reported by each station and sum.

Choose a reasonable q value and run your program for a few minutes with $n = 5$, $n = 10$, $n = 100$, and, $n = 1000$.

2. This question requires filling an image with multiple sub-images (icons) using multiple threads. An empty/blank image is created, and multiple threads then repeatedly try to copy a given (thread-specific) icon into random places in the image. Icons should not overlap (or go outside the image boundaries), and (non-transparent) pixels may not be erased once copied in, so threads must coordinate somehow in ensuring their copies do not overlap[1] during or after creation, while still allowing multiple threads to create icons when there is no possibility of interference. **15**

   Implement a simulation that allows the threads to copy in icons and fill the image as much as possible. Template code and icons are provided to demonstrate constructing a blank image of given dimensions, loading a single icon, and copying the icon into the image. The output image has a fixed name ("outputimage.png") but the program accepts 5 command-line arguments: i, h, w, t, and a. The first 3 should be ok to leave with default values—thread icons are represented by separate image files, and the i parameter specifies the basename of these files (so if the basename is "cat" then thread 1 loads "cat1.png", thread 2 loads "cat2.png" etc.), and w and h specify the output image dimensions. t is the number of threads to use, and a is the maximum number of times a thread may fail to write an icon because it would overlap with a icon already in the image before the thread gives up and terminates. Your program should support 1–6 threads (6 icons are provided) and *must demonstrate speedup over the single-threaded case for some* $t > 1$.

   In order to verify speedup, add timing code (e.g., using `System.currentTimeMillis`) to time the actual work done by all thread(s), not including setup such as the initial output image creation or file I/O to read the icons. The program should emit as output a single integer (long) value, the total time taken in milliseconds.

   Now, plot total time versus t. For some reasonable a value (and perhaps output image dimensions) verify that $t = 1$ takes measurable time (10s to 100s of milliseconds, but not more than a few seconds). Then try increasing values of t. 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 version over the single-threaded version for the different choices of t. You should be able to observe speedup for some value of t, but perhaps not all values. This of course does require you do experiments on a multi-core machine.

   Either as text included in the plot file, or as a separate `.txt` file (and specifically *not* just as code comments), briefly explain your results in relation to your synchronization strategy and the number of cores available on your test machine. Note that achieving speedup is not necessarily trivial: beware of sequential bottlenecks from different methods and APIs you may use or implement, and choose an appropriate strategy.

   Provide your source code (`q2.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).

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 special permission: **do not wait until the last minute**. Assignments must be submitted on the due date **before midnight**.

[1]Icons are represented by rectangular images with some number of transparent pixels surrounding. You can either consider overlap in terms of just the non-transparent pixels in the icons or based on the bounding box of the icon.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files, but do include a `readme.txt` of how to execute your program if it is not trivial and obvious. 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. $\overline{\textbf{25}}$