Concurrent Programming

COMP 409, Winter 2024

Assignment 2

Due date: Wednesday, February 21, 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 discretized model of a volume of water consists of a $X \times Y \times Z$ 3D grid, representing a horizontal $X \times Y$ area of depth Z. Some number of sea creatures populate the space. Sea creatures are highly abstracted, represented by only a few different shapes and sizes, as shown below: **10**

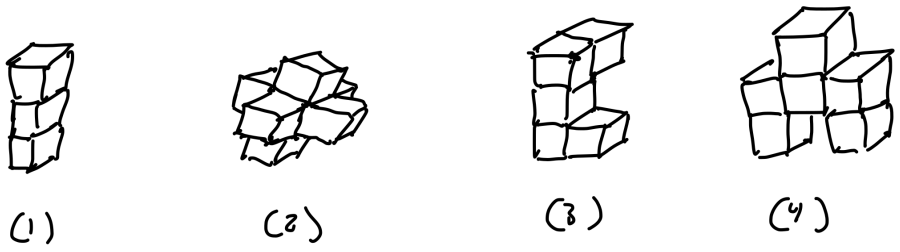


Figure 1: Four sea creatures. Assuming each is positioned as close to the origin as possible, they are composed as follows, with the first coordinate being the reference coordinate:
(1) is 3 cells $\langle (0,0,0), (0,0,1), (0,0,2) \rangle$
(2) is 7 cells $\langle (1,1,0), (1,0,1), (0,1,1), (1,1,1), (2,1,1), (1,2,1), (1,1,2) \rangle$
(3) is 5 cells $\langle (0,0,0), (1,0,0), (0,0,1), (0,0,2), (0,1,2) \rangle$
(4) is 6 cells $\langle (0,0,0), (2,0,0), (0,0,1), (1,0,1), (2,0,1), (1,0,2) \rangle$

These creatures attempt to move around randomly in the space. To keep it simple, they do not rotate and only move by at most one grid cell in each dimension. For example, creature (1) takes up 3 cells adjacent in the Z-dimension and might opt to shift left and rise up, moving from positions $\langle (x,y,z), (x,y,z+$

$1)(x, y, z + 2)\rangle$ to $\langle(x - 1, y, z + 1), (x - 1, y, z + 2)(x - 1, y, z + 3)\rangle$. A moving creature must be guarantee that each of the new cells it wants to occupy are unoccupied (and within the volume bounds) before it releases the cells it currently occupies, and if not then it must remain at its position.

Build a multithreaded simulation. Each grid cell should store a reference to the creature occupying it, and be protected by a unique lock. Any update to a given grid cell only be done while holding the cell lock. Importantly, use **only blocking synchronization**.

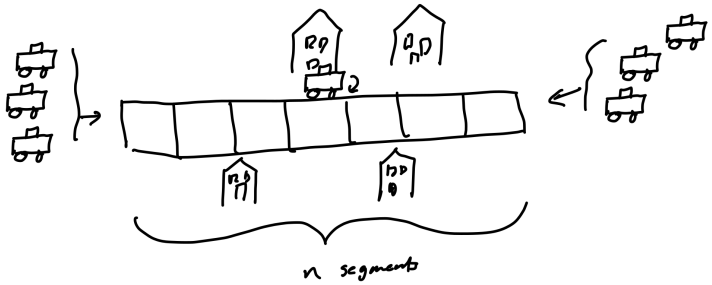
To initialize the simulation, populate the volume with k sea creatures (choosing creatures in a round-robin fashion) in non-overlapping positions prior to starting any additional threads (you may place them randomly or by following some algorithmic or fixed strategy). Give them all unique ids. Once all sea creatures are established in their initial locations, launch k threads, one for each sea creature. Each sea creature repeatedly sleeps for $10 - 50$ ms, and then selects one of the 27 possible movements (i.e., wanting to move in each of X, Y, or X by -1, 0, or $+1$) and tries to move. A movement may fail if it attempts to go outside the volume, or if it finds any of its intended destination cells are already occupied, in which case the creature stays put. Each time a creature moves, whether it succeeds or not, it should emit a one-line message to the console, stating its unique id, creature type (1–4), previous reference coordiante, and destination reference coordinate. After n seconds the simulation should terminate.

Sea creatures must never overlap in position or overwrite each other's locations. Movement must also be atomic in the sense that others can only observe creatures before or after a move. Nevertheless, it should be possible for sea creatures that are not holding or competing for any of the same cells to move concurrently—your solution should allow for maximal concurrency, and operations should not be serialized unless necessary.

Your program, `q1.java`, should accept integer parameters k and n in that order, and should set the sea volume to $5 \times k$ in each dimension. Verify it runs for at least a few seconds for $k \in \{1, 10\}$. Include a sample of the resulting output for $k = 4$ with your submission, along with a text file `q1.txt` which briefly explains how you guarantee your program does not deadlock.

2. A residential block that normally allows 2-way traffic is under construction, converting its normal 2 traffic lanes into a single lane to be shared by both directions. Residents along the street also need to enter and leave their houses. **15**

   Construct a simulation. Traffic shows up and tries to enters from either end of the street, trying to go through the block to the other end. Residents may sometimes try to leave, entering the road at some point within the block, with a specific intention to go in one direction or the the other. Assuming the road is composed of $n > 2$ vehicle-sized segments.



   Every $s \pm 20$ ms a vehicle is introduced in the simulation. It has a 45% chance of trying to enter at one end of the street, 45% chance of trying to enter at the other end of the street, and a 10% chance of trying to enter from a residence; in the latter case it has an equal chance of choosing which direction it wants to go. Absent other traffic, a vehicle takes d ms to move one road segment (and thus at least dn ms to go from one end to the other). Once a vehicle leaves the block it disappears, and a vehicle may not enter if the street is full (vehicles waiting to get in at each end, or as residents, due to not need to queue in any particular order).

Drivers should not pass each other—they pass through all n segments of the road in order if they enter from one side or the other. Residents only enter from one of the interior road segments (i.e., not at either of the ends), but can do so at any available segment.

Use one thread to generate vehicles, and represent each vehicle by a separate thread, giving each vehicle a unique id. Obviously the road can only support traffic going in one direction at a time. Use **only monitor(s) based on blocking synchronization** to model the problem. Ensure your solution is fair to both directions and to residents, but is also efficient, allowing for multiple cars on the road (going the same direction), and does not require drivers to wait unnecessarily (if the road is free).

Your program, `q2.java` should accept $n > 2$, $s > 20$ and $d > 10$ as command-line parameters. Number the road segments $0 \ldots (n-1)$ left to right, consider directions to be ± 1, and print a message to the console:

- each time a car is generated ("`car:`" + id + "," + segment entry point + "," + direction)
- when it enters the road ("`enter:`" + id + "," + segment)
- and when it leaves ("`exit:`" + id)

Experiment with different values and verify your simulation runs for some non-trivial choice of parameters—can you allow good use of the road without it being a bottleneck to traffic? In a separate document, `q2.txt`, briefly describe your synchronization strategy, and justify your choice of parameters.

(nb: No, you do not need to model residents returning to their houses; presumably there's a one-way back-alley for that.)

What to hand in

Submit your declaration and `q1.java`, `q1.txt`, `q2.java`, `q2.txt` 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**.

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. **25**