Spring 2017 ECEN 4313, CSCI 4830-010

Concurrent Programming

Homework 5

Assigned: 04/13/2017; Due: 04/25/2017, before class.
Please submit your homework through Moodle.

Problem 1 (40 points)

A multiplayer history-based real-time strategy game consists of creating and defending an empire by constructing different types of buildings. There are two opposing teams - the `Home` team, consisting of n_1 players, that attempts to construct buildings and the `Enemy` team, consisting of n_2 players, that attempts to destroy the buildings.

(a) Consider that there is only one type of building that can be built and that the land area available for building the empire is limited to 20 such buildings. Implement a bounded concurrent queue for this scenario. Create an `Empire` class which provides two methods, `build()` used by the `Home` team, and `destroy()` used by the `Enemy` team. The `Home` team is said to win when all the land area available has been built upon, and the `Enemy` team is said to win when there is not a single building standing. A tie occurs when neither of these conditions are met within 5 seconds. Create an separate `Observer` class that has a single method `result()` that continually observes the game and prints out the result. Implement your code for three cases: 1) $n_1 = 64, n_1 = 32$, 2)$n_1 = n_2 = 32$ and 3) $n_1 = 32, n_2 = 64$ and print the result of the game (`HOME TEAM WINS`, `ENEMY TEAM WINS` or `TIE`.) for each.

(b) Now consider that the land available is now infinite. Implement an unbounded lock-free queue for this scenario with the same classes and methods as above. The `Home` team gains 5 points by constructing a building and the `Enemy` team gains 5 points for destroying a building. The first team to reach XP points is said to win and a tie occurs when no team reaches 200 points within 5 seconds. Implement your code for the three cases as in Part (a) and print the result of the game for the three cases.

Problem 2. (30 points)

For a dance competition, a school attempts to pair boys and girls using a concurrent stack. In this system, g number of girls share a counter `girlCount` which is initially at 0. Each girl increments the counter atomically, writes the resulting integer value on a paper and pushes it onto a stack. At the same time b number of boys attempt to pop a paper from the stack are paired with the girl corresponding to the integer value on the paper which they pop. In addition, the competition rules state that each pair must belong to the same grade (that is a sixth grade girl can only be paired with a sixth grade boy and so on.) Consider that the school consists of grades 6 to 10.

Implement a `LockFreeStack` for this scenario for $g = b = 20$ (with four boys and four girls from each grade). Provide a `push()` method for a girl and a `pop()` method for a boy. If a boy pops a paper that belongs to a girl from a different grade then the `pop()` method would fail and return no value. Note that you would need to modify the `Node` class to hold two values (integer value assigned to a girl and her grade).

Problem 3. (30 points)

The `EliminationBackoffStack` uses an `EliminationArray` in which threads pick random array entries in order to meet and cancel with complementary calls. If a `push()` or a `pop()` arrive at the same slot before either one times out, then they exchange values and return.

Now consider an alternative algorithm which consists of two arrays: array **A** for the `push()` functions and array **B** for the `pop()` functions. Instead of picking slots randomly in the two arrays, threads choose slots sequentially (starting from slot 0). This is implemented by threads sharing two counters, one for `push()` (called `pushCounter`) and one for `pop()` (called `popCounter`), which are incremented by each thread (atomically using `getAndIncrement()`) when it picks a slot in array **A** or **B** respectively. If arrays **A** and **B** are both occupied at slot i, then the respective `push()` and `pop()` exchange values and return. Discuss the advantages and disadvantages of this scheme in comparison to the "textbook" scheme of picking slots randomly in a single array. Which scheme has more parallelism? Which scheme makes it more likely for `push()` and `pop()` to meet?