YSC3248: Parallel, Concurrent and Distributed Programming

Theory Assignment 2

In this theory assignment, some of the problems refer to an accompanying repository.[1] Feel free to fork it and play with the code. For your solution, though, you only have to submit a PDF with explanations and proofs — no code is required.

Problem 1. Consider a variant of Peterson's algorithm implemented by `ModifiedPetersonLock` in the accompanying repository. Does the modified algorithm satisfy deadlock-freedom? What about starvation-freedom? Sketch a proof showing why it satisfies both properties, or display an execution where it fails.

Problem 2. Programmers at the Flaky Computer Corporation designed the algorithm implemented in by `FlakyLock` class (see the repository) to achieve n-thread mutual exclusion. For each question, either sketch a proof, or display an execution where it fails:

- Does this protocol satisfy mutual exclusion?
- Is this protocol deadlock-free?
- Is this protocol starvation-free?

Problem 3. *Lock contention* occurs whenever one process or thread attempts to acquire a lock held by another process or thread, which might lead to deadlock or starvation. In practice, almost all attempts to acquire a lock are uncontended,so the most practical measure of a lock's performance is the number of steps needed for a thread to acquire a lock when *no other thread is concurrently trying to acquire the lock*.

Examine the "lock wrapper" implemented by the `FastPath` class in the accompanying repository. We claim that if the argument `Lock` instance provides mutual exclusion and is starvation-free, so does the `FastPath` lock, but it can be acquired in a constant number of steps in the absence of contention. Sketch an argument why we are right, or give a counterexample showing why this `FastPath` violates this claim or any other lock properties. Feel free to use the provided tests to play with the lock.

Problem 4. Examine the `Bouncer` class in the accompanying repository. Suppose n threads call its method `visit()` concurrently (as in test `BouncerTest`). Prove that:

1. By the end of the execution, *at most one* thread (of the class `BouncerThread`) gets the value `Some(STOP)` stored in its field `result`.
2. At most $n - 1$ threads get the value `Some(DOWN)`
3. At most $n - 1$ threads get the value `Some(RIGHT)`

Note that the proofs of the last two statements are *not* symmetric.

Hint: try to replicate the resaoning about ordering of events that we used to prove mutual exclusion for various locks in the recent lectures.

[1] `https://github.com/ysc3248/theory-02-examples`