Week 5 Exercises: Farm meets multithreading

Excellent job on making it past the halfway point of the quarter! We're so proud of all of your progress, and we hope you feel accomplished as well!

The goal of this week's exercise is to get you thinking about multithreading material and to help you ask questions about lecture material that still feels confusing. Please ask questions! The exercise is designed to be light in order to give you time to focus on Project 1, and should take approximately an hour.

Due date: Saturday, May 16, 11:59pm (Pacific time)

We can extend this deadline for you if you are having a really rough week. Message us letting us know what's up, and we'd be happy to work with you!

Ping us on Slack if you are having difficulty with this assignment. We would love to help clarify any misunderstandings, and we want you to sleep!

Getting the code

The starter code is available on GitHub here.

Part 1: Farm

This week, we'll take a stab at implementing a simplified version of `farm` from your CS 110 assignment 3. One twist: we'll use multithreading instead of multiprocessing!

This version of `farm` will receive the numbers to factor using command line arguments instead of reading from `stdin`. It can be run like so:

```
🍉 cargo run 12345678 12346789 34567890 45678902
    Finished dev [unoptimized + debuginfo] target(s) in 0.29s
     Running `target/debug/farm 12345678 12346789 34567890 45678902`
Farm starting on 8 CPUs
12345678 = 2 * 3 * 3 * 47 * 14593 [time: 420.687768ms]
12346789 = 7 * 13 * 19 * 37 * 193 [time: 678.612537ms]
34567890 = 2 * 3 * 5 * 7 * 97 * 1697 [time: 1.176596148s]
```

```
45678902 = 2 * 433 * 52747 [time: 1.812403818s]
Total execution time: 1.812585221s
```

You can implement this version of `farm` in three steps:

- First, establish the queue of numbers to factor. You will want to use the `get_input_numbers()` function that we have provided. You may draw on lecture examples to make this queue safe to access from multiple threads.
- Spawn `num_threads` worker threads. In each thread, using a loop, pop a number off the queue and factor that number (by calling `factor_number`) until no numbers remain on the queue.
  - You should be careful to think about when the thread holds the mutex. You'll need to hold the mutex while accessing the queue, but you should not hold it when factoring a number to avoid serializing the work. We highly recommend writing a helper function that borrows a reference to your `Mutex<VecDeque<u32>>`, pops a number from the front of the queue, and returns `Option<u32>` (note that `VecDeque::pop_front` conveniently returns `Option<u32>`). This way, the lock is only held for the duration of that function. You can tell the queue has become empty when that function returns `None`.
- Wait for all of your threads to finish by calling `thread.join()` on each of them. If you don't do this, your program will compile fine, but you may have problems where some numbers are missing from the output because the program exited before the threads could finish.

It may be helpful for you to reference the ticket agents Attempt 3 code from Lecture 10.

When you run your program, pay attention to the total execution time printed at the end of the program. If it is significantly longer than the time taken to factor any one number, you may be accidentally serializing your program by holding a lock for too long.

Fun fact: This implementation is made significantly easier by the fact that we have all the numbers to factor up front (from `argv`) instead of receiving them at some point via `stdin`. Your CS 110 `farm` implementation needed to contend with the fact that an empty queue does *not* mean workers can exit (they need to stick around in case another number arrives later via `stdin`). Here, workers can pull numbers directly off the queue (there is no main process broadcasting numbers to workers), and workers can exit as soon as the queue becomes empty, since there is no way for additional numbers to come in. In the coming weeks, you'll learn how to manage worker threads in cases where additional work might arrive after a queue becomes empty, and you'll implement such a program in CS 110 assignment 6.

Part 2: Weekly survey

Please let us know how you’re doing using this survey.

When you have submitted the survey, you should see a password. Put this code in `survey.txt` before submitting.