Counting, counting, 1, 2, 3...

Let's say you've got a bunch of files--maybe you've downloaded all of Wikipedia, or Project Gutenberg, or something. You want to be able to find out how many times a word (say, 'circle') appears in this collection of files. In addition, you want your program to be fast!

Fortunately, this task is easy to parallelize across a bunch of threads. We're going to have two kinds of threads: 'mapper' threads and 'reducer' thread(s). Mapper threads will read in files and count the number of times a word appears in each file. Reducer thread(s) will sum up these counts until there's one value--the total number of times the word appears in your files.

To keep from overwhelming the OS by opening a zillion files at once, stick with 4 mapper threads as a start. To coordinate efforts between all our threads, we need two queues:

1. A queue of files that need to be read and counted

2. A queue of word counts

The structure for a mapper thread will go something like this:

1. Lock the file queue, remove the top filename, and unlock the file queue

2. Open the file, count the number of times our word appears in it, then close the file

3. Lock the count queue, push the count we just computed onto it, then unlock the count queue.

The structure for a reducer thread will go something like this:

1. Lock the count queue, remove the top two elements, and unlock the count queue

2. Sum the two elements together

3. Lock the count queue, push the sum onto the queue, and unlock the count queue

Your main function will:

1. Prompt the user for the word they wish to count

2. Set up the queues

3. Launch the mapper threads and wait for them to complete the mapping

4. Launch the reducer thread(s) and wait for them to complete their work

5. Print out the total count!

For this assignment, you will use the C++11 std::thread library (which is much much nicer than pthread). You'll need to compile your code with `g++ -lpthread` (the general Makefile should provide this functionality for you). For your queue, you may use one you've already written, write your own, or use std::deque (double ended queue).

For simplicity and consistency, your program should read in file names from a file called files.dat (one file per line). There are currently two files provided as a starting point (more files may be made available later).

Documentation:

- std::thread -- http://www.cplusplus.com/reference/thread/thread/

- std::mutex -- http://www.cplusplus.com/reference/mutex/mutex/

- std::unique\_lock -- http://www.cplusplus.com/reference/mutex/unique\_lock/

- std::deque -- http://www.cplusplus.com/reference/deque/deque/

Bonus:

- Add a configuration file to allow a user to adjust parameters such as the number of mapper and reducer threads

- Allow multiple reducers to run at the same time as the mappers