CSc 415 Project 3

(This document available on iLearn.)
due Wednesday 10/16/2019 11:59pm
(standard 48-hour grace period, for 75% credit)
(6% of your grade)

This is an individual assignment. Work on your own!

All the files you need are here: http://unixlab.sfsu.edu/~whsu/csc415/P2/

Recall for Project 2, we looked at fileStats.c, similar to the Unix utility wc, and rewrote fileStats.c so that each input text file is handled by a different POSIX thread. Project 2 was assigned before we learned about synchronization primitives (mutex locks and semaphores); we could solve the problem without synchronizing. All child threads wrote to different FileInfo structs, and the main thread waited for all threads to finish before computing overall totals.

For Project 3, we'll write a multi-threaded implementation of fileStats.c, using synchronization primitives. Follow these steps (separated into Main thread and child threads for clarity, major differences from Project 2 in red):

Main thread:
- creates N concurrent threads
- using a semaphore, waits for signal from last child thread to finish
  - (no pthread_join()!)
- reports overall total counts for all files (as computed by child threads)

Each child thread:
- computes and records counts for one file
- prints its thread ID using pthread_self()) and its counts to stdout
- updates overall count totals
- if current thread is the last child to update overall counts
  - signal the main thread using a semaphore

Some implementation rules (first two rules same as Project 2):

1. Make sure there are N concurrent threads! Otherwise points will be deducted.
2. Make sure that your code works with any reasonable N. That is, if you need an array with N elements, allocate the array dynamically, not statically.
3. Main thread must *not* use pthread_join() to wait for specific threads to finish!
4. If a child thread is the last to update overall counts, it must use a semaphore to signal the main thread. (Hint: only one semaphore is necessary. One thread calls wait, and one thread calls post/signal.)
5. You should use *minimal* global/shared variables. Points will be deducted for having too many global/shared variables.
6. Efficiency matters! Use mutex locks and semaphores only when necessary.

Suppose the multi-threaded version of fileStats, with synchronization, is called msyncFileStats. The output produced (similar to mtFileStats):

```
unixlab: ./msyncFileStats Oliver.txt Pessoa.txt Davis.txt
Thread 53587 Oliver.txt: 7 lines, 85 words, 453 characters
Thread 53588 Pessoa.txt: 1 lines, 8 words, 55 characters
Thread 53589 Davis.txt: 10 lines, 123 words, 723 characters
Total: 18 lines, 216 words, 1231 characters
unixlab:
```

The counts for each file should be accurate, though the threads may report counts in a different order.

Submission:

Submit a .c file (or, if there's more than one file, a tar/zip file containing all source code and headers) using the iLearn submission link. Each source file should have a header with accurate instructions on compiling and running your code on the Unix command line. If your instructions don't work perfectly, you may get a zero on the project.