CS 341L Fall 2009 Lab 6

Assigned: Monday, 9 November 2009

Due: 10:00am, Monday 16 November 2009 by attachment as an email to labturnin.cs341l@gmail.com

You should attach your program with your last name and the lab number in the name of the file, e.g., "knockel-lab6.c".

No late assignments will be accepted unless circumstances warrant an extension for the whole class (e.g., server is down all weekend, etc.). You are expected to do your own work as an individual effort, copying the code of others is considered cheating.

The purpose of this lab is to familiarize you with multi-threaded programming and using semaphores to write thread-safe programs.

Overview:

The pthreads library provides the following primitives:

- *pthread_t – a POSIX thread object*
- pthread_mutex_t – a mutex object that can be locked or unlocked
- pthread_cond_t – a condition variable used for allowing a thread to wait until a condition is true

The POSIX semaphores library provides the following primitives:

- *sem_t – a counting semaphore object*

The pthreads library provides the following functions:

- *pthread_create() – create and run a new POSIX thread*
- *pthread_join() – wait until given POSIX thread has exited*
- pthread_mutex_lock() – if unlocked, lock given mutex, else wait until mutex is unlocked
- pthread_mutex_unlock() – unlock given mutex
- pthread_cond_wait() – wait until given condition is true

The POSIX semaphores library provides the following functions:

- *sem_wait() – if given semaphore is > 0, decrement, else wait until semaphore is > 0*
- *sem_post() – increment given semaphore*

Note: The italicized stuff above we are using in this lab.

Note: ALL of the above functions create memory barriers.

For more detailed information, look up these functions' man pages.

Assignment:

You will be given code that is not thread-safe as well as sample bad output from it on a single-processor system. Note that the code you are given may behave differently on your machine and is not even guaranteed to terminate as-is.

Your first task is to examine the */usr/share/dict/words* file, which is being used as input, and the given bad code and the given bad output, and understand what the code is doing and why it is generating the bad output. Then write, in C comments, at the top of the program, a short paragraph explaining the behavior of the program and why it is not thread safe.

Your second task is to then fix the bad code so that it is thread-safe, namely so that it is guaranteed to take a stream of characters from *stdin* and capitalize them and then terminate. You may not make any alterations to the code except to make two calls to sem_wait() and two calls to sem_post(). In addition to being graded on thread safety, you will also be graded on performance, i.e., maximize the amount of concurrency in the program such that it still guarantees thread safety.

Note: This code is not multibyte safe, so it may not capitalize multibyte characters correctly, but do not worry about this. In other words, this code may output the wrong characters when you input some foreign characters, like Greek or Chinese.

Compilation:

gcc -Wall -Wextra -lpthread knockel-lab6.c -o knockel-lab6

A test (passing will not guarantee your program is thread-safe but at least on the right track):

./knockel-lab6 < /usr/share/dict/words