CSE 5/7343 Program 4

Make Your Own Shell (part 2) Using a Thread Pool

Due Monday, April 2 by 11:59pm

Program three was based on the boss/worker model. The primary thread, the boss, accepts input for the entire program. Based on that input, the boss passes off specific shell commands (in our case *frand* and *fsort*) to a worker thread that was dynamically created for the sole purpose of carrying out that task. Once the task has completed the worker thread terminates. (All other commands are carried out by the boss.) The drawback of this approach is the overhead of constantly creating and destroying new threads.

To avoid potentially time-consuming thread creation/deletion, program 4 will be based on the thread pool model. A thread pool consists of a fixed number of worker threads that are created at program start-up and exist until the program terminates. The primary thread (producer) will accept shell commands from the keyboard and place every command in a shared buffer. The worker threads (consumers) remove commands from the shared buffer for the purposes of carrying out the command. Because the buffer is shared you must deal with mutual exclusion and process synchronization This is achieved by using POSIX *mutex* variables and *condition* variables.

Program 4 supports the same functionality as program 3. Write your program so that it is easy to change the message buffer size and the thread pool size.

Code should be in 'straight' C using the gcc compiler. Remove all compiler-generated warning messages.

Devise a prompt that will uniquely identify to the user that your shell is being used.

Submit your .c file through Canvas.