**Networked Spell-Checker:**

A multi-threaded spell-checking server implemented over a local network.

Program acts as a server for multiple clients and distributes worker threads to each for simultaneous execution. When a client connects to a server it is placed into a waiting queue to receive worker thread services. Thread pools allow for execution to seem concurrent and with more time mutexes would be included to provide synchronization. A separate thread was created to track all other threads and write to log.

**Idea for Implementation:**

The idea of the main thread is to, while in a continuous loop, accept a new socket descriptor and add it to the client buffer. While doing this, in order to maintain synchronization it utilizes mutexes to restrict access to the client buffer. When entering the buffer, it first locks it so that no other thread may enter during execution. The main thread then checks to see whether the buffer is full. If the buffer is full, it unlocks the client buffer and consumer is allowed to proceed. If the client buffer is not full it proceeds to accept a new socket descriptor and add it to the client buffer. When adding a client to the buffer the thread it is critical that the thread increment the size of the client buffer. Once this is done, the thread unlocks the client buffer and signals to waiting threads that the first client in the buffer is allowed to proceed.

The idea of the worker thread is very similar to the main thread however it is where the spell checking actually occurs. After being signaled by the main thread and while in a continuous loop, a worker thread will enter the client buffer, locking it behind them. Next the thread reads a word from the client and uses various functions (functions i was unable to complete) in order to determine whether the word was spelled correctly. It is worth noting that the program is able to be passed a specific dictionary to be used when cross referencing words but also that a default dictionary is included in the code. After cross referencing the word within the function, the thread must return to the client whether or not the word was spelled correctly. The thread is then responsible for writing to the log which keeps track of all words sent to worker threads by client and whether or not the aforementioned was spelled correctly. Writing to the log utilizes a separate buffer which also requires mutual exclusion to ensure that no more than one worker thread is writing to it at once. The log buffer utilizes the same method for mutual exclusions both previously explained instances, i.e. entering the buffer and locking it behind them in addition to unlocking the buffer and signaling to other threads when execution is completed.

Finally the log writer thread utilizes a separate buffer and mutexes in order to maintain mutual exclusion and synchronization. After being released from the queue of threads waiting to write to the log buffer, the thread enters the log buffer if and only if no other thread is currently writing to the log, and locks the mutex behind them. The thread writes what word was cross-referenced and whether or not it was spelled correctly into the log buffer. Once this is complete, the thread leaves the log buffer and signals to other threads waiting on a queue that they may enter the buffer and write to the log file.

**Execution:**

Execution did not go as smooth as designing the implementation. In the current execution you will find that the code does not compile due to incomplete code. I was able to implement the part of the main thread where an argument may be passed to the main thread to determine which dictionary to be used. I also was able to open a listening port and accept socket descriptors. I was unable to finish the functions necessary to spellcheck the word. With more time, functions would’ve been written to actually spell check the words and write to the buffer but the base code is there. All macro and micro variables, including the struct for the server thread, and declarations for functions are included in a header file titled “network.h” which is included in the main source file. Problems that I encountered include inexperience with networking code and several segmentation faults.

**Testing:**

Testing for this program as far as debugging goes, included using several print statements in order to determine where exactly in the program a bug was located. In addition, using the debugger in my IDE proved helpful at times. If I was able to compete my code I would’ve used the loopback network address to continuously run my code and ensure that the results were correct and the proper data was written to the log file.