Created by Tom Burns and Andrew Dulichan
05/06/2019
CS 214 Systems Programming
Asst3 README

Breaking Down Asst3 Into Subprojects

1. Creating the multithreaded WTFserver.
2. Creating client commands as specified in writeup.
3. Design manifest and allow for its creation and maintenance in WTFclient commands.
4. Design protocol and send commands to server.
5. Interpret commands from the client on the server side and perform required actions.
6. Create and maintain mutexes across threads to maintain synchronization of the WTFclients and their communication with the server.
7. Create a WTFtest executable which builds and runs all testcases specified by creating processes using fork() and exec().
8. Complete writeups – readme and testcases.txt describing project, methodology, functionality, and design decisions.

To begin this project, we created an example client and server set of files. The two files managed to connect to each other and send a simple message to the client, which was then received, and the confirmation of this reception was sent back to the client. After creating and testing this we moved on to adding functionality for sending a filename from the client to the server, having the server read that filename, open it, and send its contents byte by byte back to the client, and finally having the client print it out on its side. This was also successful and so we moved on to designing this functionality in the WTF project.

We moved on from a simple client/server connection to making a multithreaded server in c that would host multiple connections and pass each of these new connections to a thread, while listening for more connections and continuing this process. We managed to get this working and had a server which ran continuously as connections came in and were passed off to a connection_handler function that a new thread would run each time a connection was created. The last part of the server that we managed to complete was the creation of a global mutex which we planned to have lock the critical section of the code, where projects would be read/written to/from during each connection to the client. We planned to make a mutex for each thread that would lock based on the project that was being worked on but did not manage to get that far. The server also catches the Ctrl+c input and handles the exit of the server side WTF client gracefully.

For the WTFclient commands, we managed to complete only the configure method, while starting many of the others and not finishing implementing their functionalities. The wtf client does not connect to the server but pushes the given IP and port into a .configure file. The

WTFclient also has implemented a connect method that only is called after checking if the command for the client is configure or one of the commands that requires a connection to the server. This command checks the configure file, pulling out the IP and port and connecting based on its given values.

If we were to complete more of this project, we would have followed the outline given at the beginning of this readme, finishing the wtfclient commands, then moving on to ensuring the protocol works with sending and receiving files between the client and server, and maintaining the projects and their manifests that lied on the server side. The thread synchronization would have been integral in this part of the code execution, and even with a global mutex for the client connections would have avoided deadlock while protecting critical sections. Creating the wtftest would have taken a decent amount of time to implement as well, since we would have had to write code for each of our testcases and then write processes that would then be forked and executed by the wtfserver and different wtfclients connecting to it.