z5163204 Joshua Mclellan COMP3331 Assignment Report

Technical Details

- Written in Python3
- Working on CSE and Mac OS

Program Design

I used the existing python3 server and client code from the lectures on WebCMS3 to start my project. For the server design, there is a main loop that connects a new client to the server, and once done, creates a thread that handles all authentication and command processing for that client, while waiting for more clients to join. This loop is within a try except statement, as my solution for the SHT command involved having to catch a socket error, in order to terminate the server fully. The client side is a bit more simple, involving one loop which checks if the client is logged in (i.e. authenticated by the server), and if so handles the commands. The majority of command handling and error checking is done on the server end, as it made more sense for me that the client knows as little as it can of the ways the server checks content sent to it by a client. However, the client does process some commands before sending them off, predominately the UPD, XIT and SHT command, as all three involve some sort of client action when called (opening a file, and closing the socket/loop respectively). The basic pattern of the loop is that the Client receives and then prints the input command prompt from the Server, the User client inputs a command which is then sent to the server, and then the client receives and then prints the response from the server. The opposite of this pattern (ie send recv send) is mirrored by the Server. For authentication, an AUTH string token is sent to the client if they have logged in successfully, and a number of variations on this AUTH string token are used in areas where the client needs to verify something before processing the response, and vice versa. For example, an SAUTH (Shutdown AUTH) string token is sent if the client sends a SHT command with the correct password. Most messages that are sent between the client and server are formatted as strings. I think that the way i implemented this is quite rigid, and if I could start over I would have perhaps had multiple connections, or monitored the I/O with the select python library, so that the loops wouldn't have to adhere to such a strict and fragile process. I would have also designed a more comprehensive message format, for added verification that the messages are being delivered and received properly. However, my implementation is secure in that every interaction is handled the same way, and it makes adding threading quite easy (for the most part).

Known issues

- SHT **does not** work for multiple users. Unfortunately, in part due to my inexperience with threads and in part due to my implementation, I was not able to work out an effective non-blocking user input solution for the client-side. Hence, when a user issues a shutdown command, the other clients will most likely be stuck waiting for an input from the user client. I believe that the ways to fix this would be to use multiple threads/sockets

for receiving and sending, and using the select library to implement a non-blocking input using stdin.

- Occasionally the client/server will get packets mixed up, and thus print or wait for input when it shouldn't be. To combat this, I added a very small amount of artificial delay (time.sleep(0.001)), which gave more room between messages that were often being sent back to back. From my testing, this has fixed the problem 95% of the time. However, it is by no means an ideal solution, and i think a better solution might be to either number the messages being sent, or have a stricter explicit ACKing system in which the server sends a REC message to the client, so that the client knows when to send the next bit of information, and vice versa.

Code used from online resources:

- https://stackoverflow.com/questions/26362193/stop-server-program-inside-a-threading-module-python I used the basic principles of the answer to this question for getting SHT to work with threads with my server.