COMP3331 Assignment Report

Program Design

The program is distributed across four different files; client.py, client_helper.py, server.py and server_helper.py.

client.py acts as the main interface for client usage. It handles the creation of a TCP socket with the server, as well as prompting the user to input commands for the forum.

client_server.py handles the semantics of sending and receiving messages to the server.

server.py acts as the main interface for server usage. It handles the creation of TCP sockets with all connected clients, as well as implementing multi-threading to accommodate for multiple clients.

server_helper.py handles the semantics of sending and receiving messages to its respective client. It also maintains state about a given server, and interacts with files needed for server operation.

Application Layer Message Format

The application layer message format I chose for this assignment is utilising dictionaries. Dictionaries have the advantage of being able to relay strings, integers and Booleans flexibly. Additionally, due to the method that information is retrieved, the purpose of the information is apparent to anyone reading through the code. Finally, the json library efficiently converts dictionaries to and from bytes, preserving ordering and values.

How It Works

A server must initially be started, requiring a port number to run on, and assuming that localhost will be used as the IP address. I have chosen not to require an admin password, which will mean that any client can shutdown the server with any password they provide. After the server is running, clients can begin to connect.

Clients require an IP address and port number to connect to, where the IP address in this case is assumed to be localhost (otherwise they wouldn't be able to connect to this specific server). Multiple clients can connect concurrently, and can all interact with the thread. However, clients cannot alter the thread at the exact same time, which is why a threading lock is used.

The server will always listen for information sent over TCP from its clients. According to the command, the server will alter its state and appropriate files. Most of the time, it will be altering a text file which stores a thread, although may also include sending and receiving files. It will additionally respond to the client, with a message that should be displayed to it.

Considerations

I considered utilising simple strings for the application layer format initially. The advantage is that it's very simple to implement, and requires the least setup and familiarisation (given we all work with strings very often). However, there are several disadvantages. It requires a lot of sending and receiving over the TCP connection to convey the same amount of information. It may also be more difficult to interpret for other programmers, as the communication may be nuanced. It will be difficult to alter for any future amendments.

I considered controlling server state through a Forum class. The advantage is that it will be applicable and modularizable for future similar programs. It will also be quite easy to interpret in the interface files. The major disadvantage in this assignment is over-complexity. The emphasis for this assignment was on the network communication component, rather than the efficiency and optimisation of the programming language. Thus, I thought it was unnecessary to go to this extent for this specific implementation.

Improvements + Extensions

The biggest improvement would be a more intuitive and enticing user interface. Designing and implementing a simple app for each client to connect to the server, as well as provide all the commands and receive messages from server would make it easier for all clients.

An extension would be to implement direct interaction between clients. Clients can currently only message and exchange files through a thread on the server. The communication could still go through the server, but it would be useful if two clients could communicate directly.

Lack of Functionality

The only function that is not fully capable in multi-threading is shutdown. A client can send a shutdown command, and will successfully shut down the server (adhering to the specification). It will also send the appropriate response to the respective client that issued the shutdown command, and force that client to exit as well. However, the specification requires the server to send a message to ALL clients, which I have not implemented. Instead, when a client attempts to interact with the server after shutdown, it will display a message that it has been shut down, and will force the client to exit.