CP317 Report – TCP Client Server Communication

Code

The code is broken up into 2 separate files, Server.py and Client.py that each handle the hosting for the Server and Client respectively.

Server.py

The server.py file is responsible for starting the server that all the clients will communicate to. It begins with binding a TCP socket to host the server on the IP address 127.0.0.1 and the port 65432. It initializes listening for future connections using the .listen() function as well as two arrays that will store both the clients and the data cache in memory and prints out some hosting data to display. Do note that it is defaulted to limit only 3 client connections at a time but can be altered by changing the global variable MAX_CLIENTS.

The listen method will constantly be running to poll if a client is attempting to connect to the server and when a successful connection is established the server will begin a new thread to handle the new client's interactions with the server as well as giving each client a designated name. Additionally the server will track the clients by appending them to the "clients" array and begin tracking a cache in memory of all clients that have connected as well as their status of connection time and disconnect time.

Handling new clients is done in a method that each have their own thread monitoring it until the socket is closed to which the thread will die. Each thread manages their own client by constantly listening for any inputs coming from the client whether it be messages (to which those messages are echod in the server along with the respective name of the client that sent the message). Depending on which input the thread sees from the client it will do a variety of different actions including:

- Status to display the cache of all the client connections from memory
- List to display a list of all the possible files from a repository to poll from
- Echo message back when a regular message is seen

Client.py

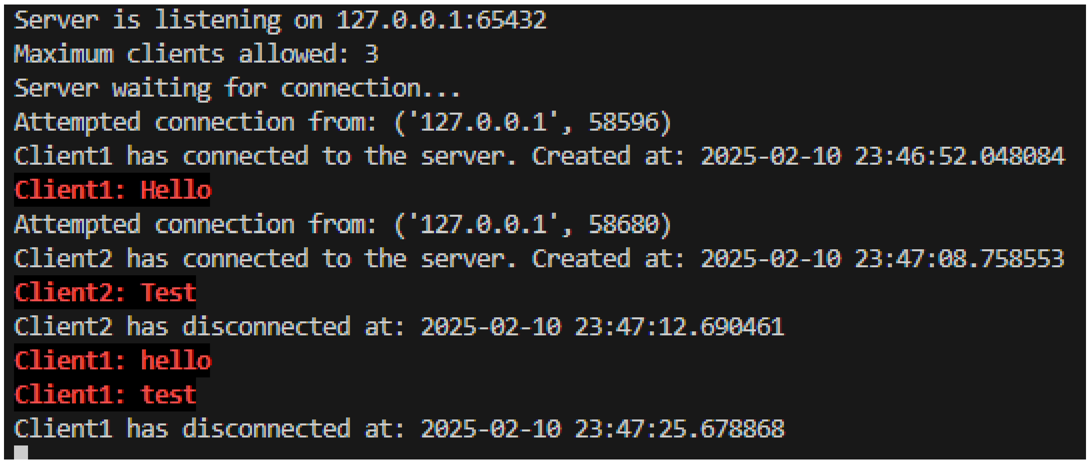
The client.py file is responsible for creating and handling each individual client. It begins by attempting to initiate the socket connection to the server. It contains a simple Boolean locking mechanism to control the inconsistencies with receiving and sending data and the output printing in between. The client starts it's life cycle by initiating the talk_to_server() method where it begins a thread to receive messages as well as an infinite looped send_message() function to allow the client to infinitely send messages to the server. Send messages is involved with the lock logic in order to keep the consistency and uses inputs to prompt clients for messages or commands to send to the server which are encoded and sent over the socket. The receive_message() function is what is repeatedly run within the thread to constantly listen back from the server for any incoming messages as well as using it to reject the client and kill the thread if server is full OR if the user input a “bye” string to close the client which is echo’d back to the client from the server with an indication to end the client.

Struggles

One of the major struggles that we encountered with this program was the timing issues with the client as often times when the client is sending messages to the server which are echo’d back to the client and caught by it’s receive function the messages come out of order since the thread runs on a separate timing set than the send_messages() loop. To solve this problem we used a simple Boolean lock to prevent the send message loop from executing until the lock was released by receive_message() which would indicate that the message was finished being received and the thread returned to listening. This helped to ensure that all messages were displayed in order by the clients.

Test Examples

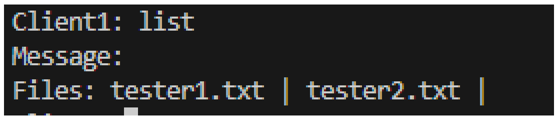
Server.py



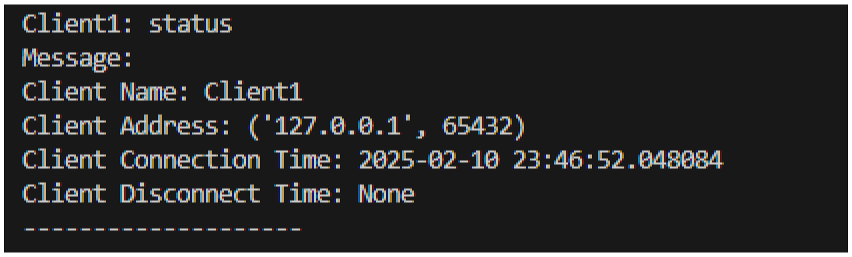
```
Server is listening on 127.0.0.1:65432
Maximum clients allowed: 3
Server waiting for connection...
Attempted connection from: ('127.0.0.1', 58596)
Client1 has connected to the server. Created at: 2025-02-10 23:46:52.048084
Client1: Hello
Attempted connection from: ('127.0.0.1', 58680)
Client2 has connected to the server. Created at: 2025-02-10 23:47:08.758553
Client2: Test
Client2 has disconnected at: 2025-02-10 23:47:12.690461
Client1: hello
Client1: test
Client1 has disconnected at: 2025-02-10 23:47:25.678868
```

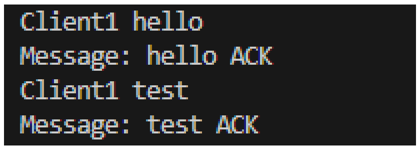
Client.py

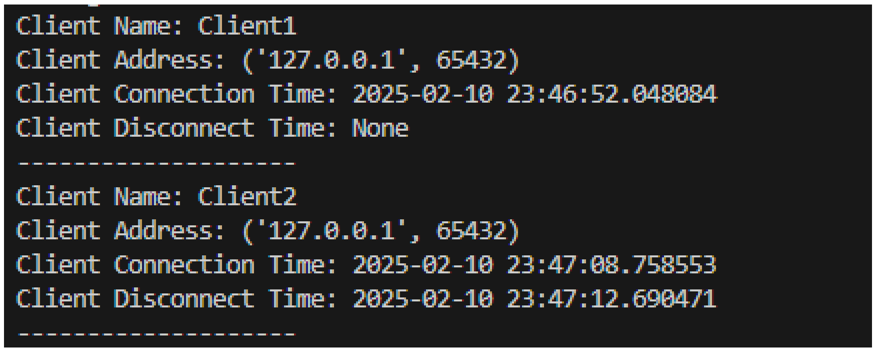
Client 1 Test:



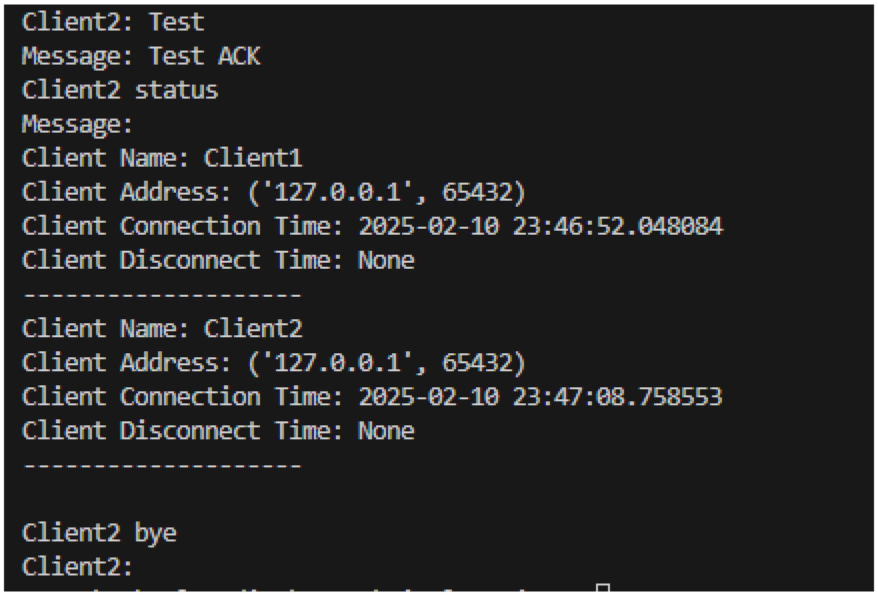
```
Client1: list
Message:
Files: tester1.txt | tester2.txt |
```


```
Client1: status
Message:
Client Name: Client1
Client Address: ('127.0.0.1', 65432)
Client Connection Time: 2025-02-10 23:46:52.048084
Client Disconnect Time: None
--------------------
```


```
Client1 hello
Message: hello ACK
Client1 test
Message: test ACK
```


```
Client Name: Client1
Client Address: ('127.0.0.1', 65432)
Client Connection Time: 2025-02-10 23:46:52.048084
Client Disconnect Time: None
--------------------
Client Name: Client2
Client Address: ('127.0.0.1', 65432)
Client Connection Time: 2025-02-10 23:47:08.758553
Client Disconnect Time: 2025-02-10 23:47:12.690471
--------------------
```

Client 2 Test:


```
Client2: Test
Message: Test ACK
Client2 status
Message:
Client Name: Client1
Client Address: ('127.0.0.1', 65432)
Client Connection Time: 2025-02-10 23:46:52.048084
Client Disconnect Time: None
--------------------
Client Name: Client2
Client Address: ('127.0.0.1', 65432)
Client Connection Time: 2025-02-10 23:47:08.758553
Client Disconnect Time: None
--------------------

Client2 bye
Client2:
```

Improvements

In the future we would like to improve by implementing the file stream system to be able to use the list of repository files as well as adding more GUI functionality to make the program easier to use and understand, perhaps a help function to provide some insight as to the possible commands and use cases of the program as well.