Jonas Mogilnicki - 169034191

Tobias Cheung - 210860220

Heider Ali

CP 372-C

2025 February 10

Assignment 1: Socket Programming

Code Description:

The requirement was to create a server-client connection using TCP sockets with two python programs named Server.py and Client.py. These files use TCP sockets to create a simple chat application. Below is an outline of these files and a description of their code.

Server.py:

The server class is initialized with a host, a port and a max client limit, set to 3. The start() method is used to create a TCP socket which binds it to the host and port and also listens for incoming connections. If a client tries to connect when there are already 3 connections, “server full” is printed and the connection closes. If space is available, the server assigns the client a name, records a connection time and sends this name to the client. Using the handle_client() method, the server continuously listens for messages from the client. Additionally, the server supports a few simple commands that let a client check its connection status, view available files, or request file transfers. All other messages are treated as standard chat messages and are echoed back with an acknowledgment. Then, when a client exits the server, the server updates the client's record with the disconnection time, closes the socket and removes the client from an active list.

Client.py:

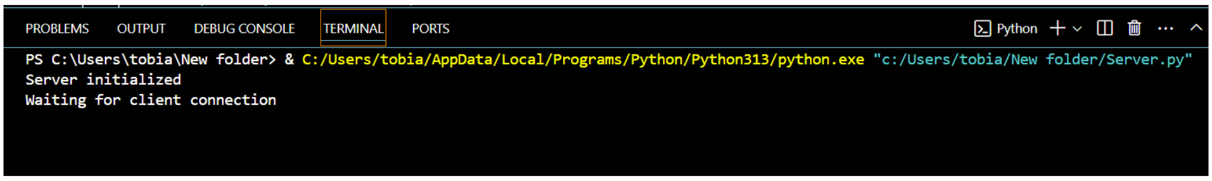
The client class is initialized with a host and port, and upon starting, it creates a TCP socket and connects to the server. Once connected, it receives and displays a welcome message that includes its assigned name. The client then enters a loop where it prompts the user for input, sends messages to the server, and displays the corresponding responses. It supports a few simple commands that allow the user to check the connection status, list available files, or request file transfers, while all other messages are treated as standard chat messages. When the user exits, the client closes the socket and closes the connection.

Difficulties faced:

A difficulty that we faced was managing multiple client connections without them interfering with each other. We ended up using pythons threading module to spawn a separate thread for each client connection. This let us isolate client communications and manage the connection limits.

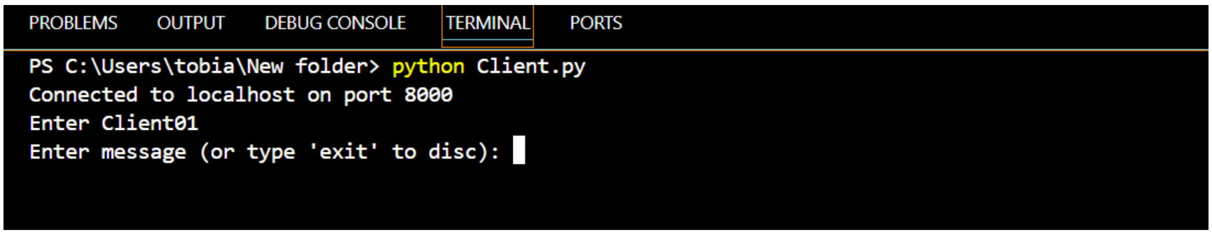
Test Results:

1. Launch Server

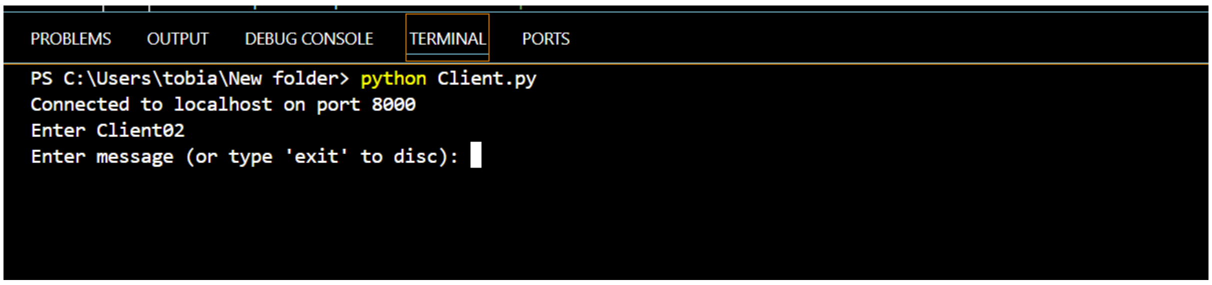


```
PS C:\Users\tobia\New folder> & C:/Users/tobia/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/tobia/New folder/Server.py"
Server initialized
Waiting for client connection
```

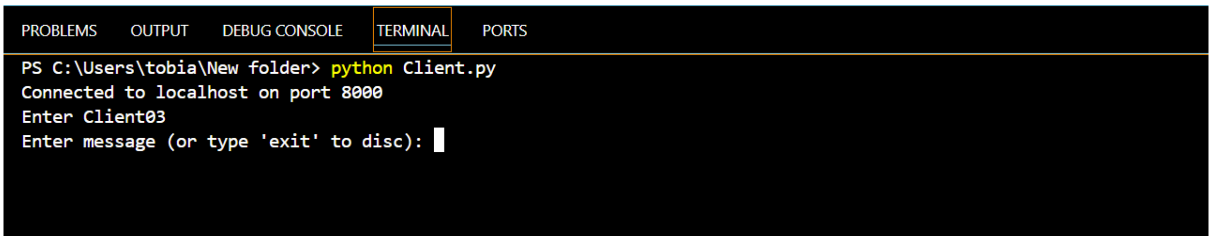
2. Clients connected and names up to 3



```
PS C:\Users\tobia\New folder> python Client.py
Connected to localhost on port 8000
Enter Client01
Enter message (or type 'exit' to disc):
```

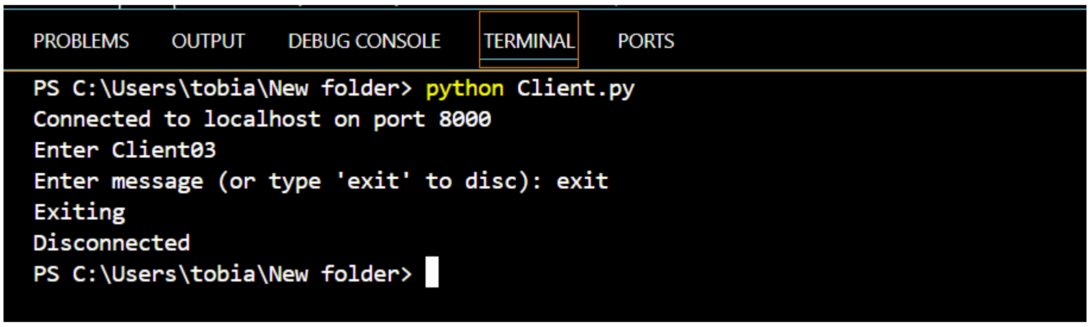


```
PS C:\Users\tobia\New folder> python Client.py
Connected to localhost on port 8000
Enter Client02
Enter message (or type 'exit' to disc):
```

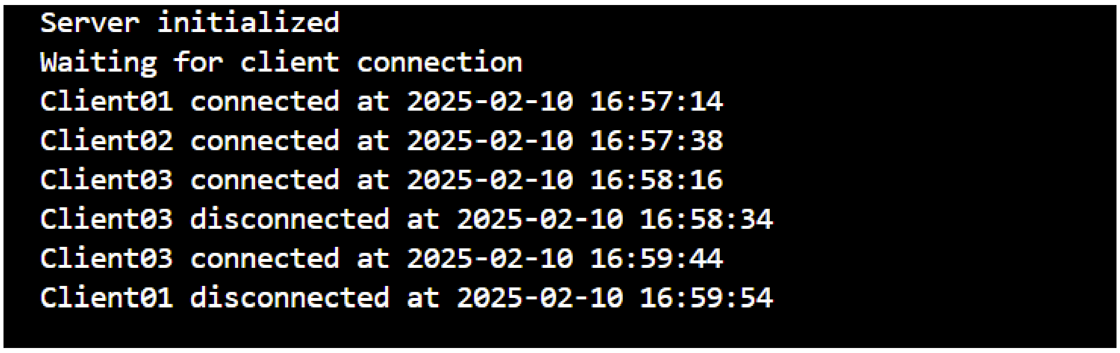


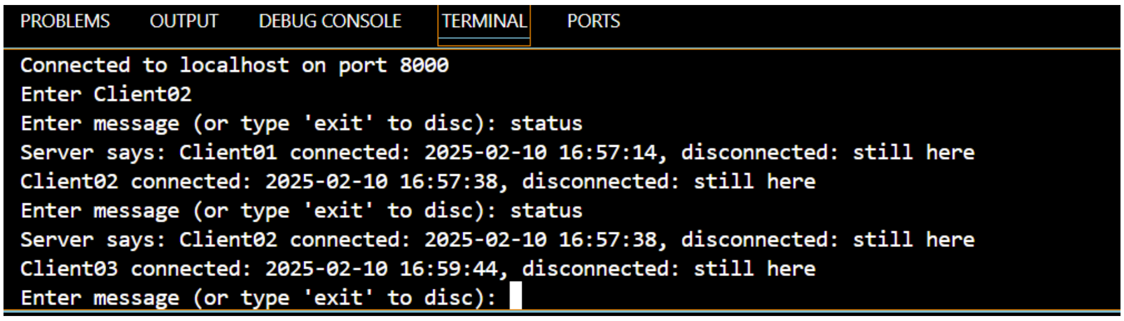
```
PS C:\Users\tobia\New folder> python Client.py
Connected to localhost on port 8000
Enter Client03
Enter message (or type 'exit' to disc):
```

3. Connect / disconnect time status / Exit and make space for new client



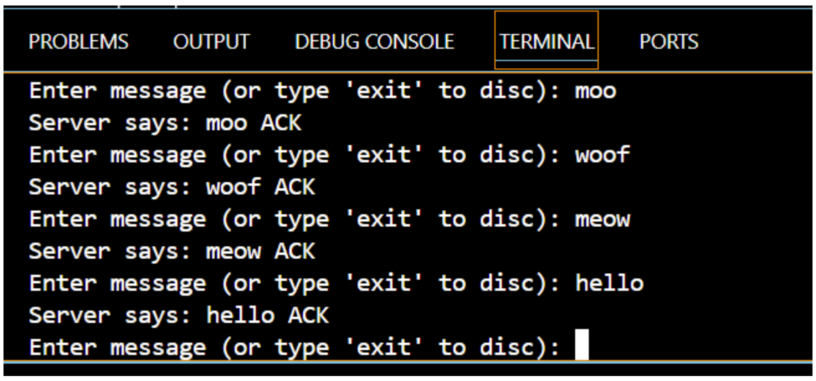
```
PS C:\Users\tobia\New folder> python Client.py
Connected to localhost on port 8000
Enter Client03
Enter message (or type 'exit' to disc): exit
Exiting
Disconnected
PS C:\Users\tobia\New folder>
```


```
Server initialized
Waiting for client connection
Client01 connected at 2025-02-10 16:57:14
Client02 connected at 2025-02-10 16:57:38
Client03 connected at 2025-02-10 16:58:16
Client03 disconnected at 2025-02-10 16:58:34
Client03 connected at 2025-02-10 16:59:44
Client01 disconnected at 2025-02-10 16:59:54
```



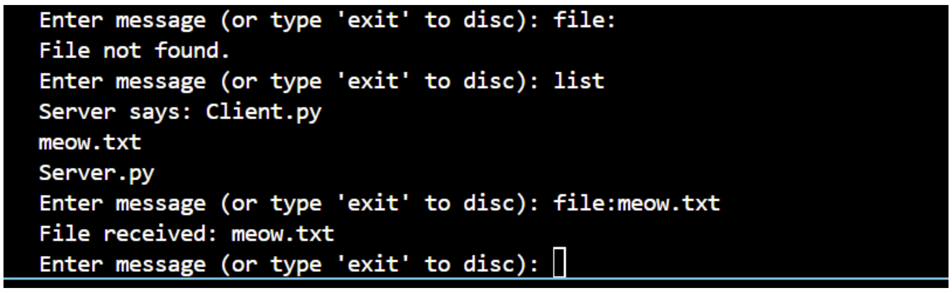
```
Connected to localhost on port 8000
Enter Client02
Enter message (or type 'exit' to disc): status
Server says: Client01 connected: 2025-02-10 16:57:14, disconnected: still here
Client02 connected: 2025-02-10 16:57:38, disconnected: still here
Enter message (or type 'exit' to disc): status
Server says: Client02 connected: 2025-02-10 16:57:38, disconnected: still here
Client03 connected: 2025-02-10 16:59:44, disconnected: still here
Enter message (or type 'exit' to disc):
```

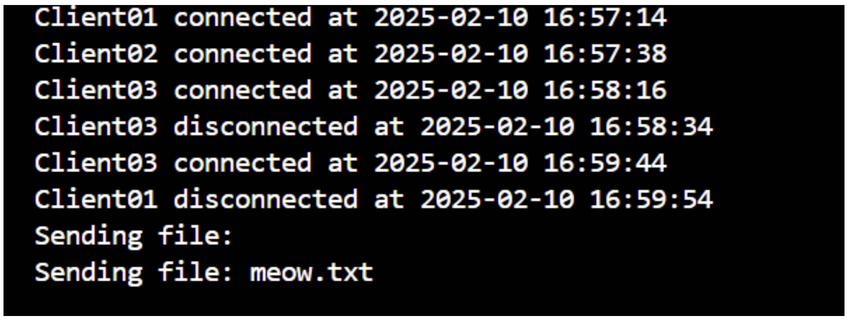
4. ACK



```
Enter message (or type 'exit' to disc): moo
Server says: moo ACK
Enter message (or type 'exit' to disc): woof
Server says: woof ACK
Enter message (or type 'exit' to disc): meow
Server says: meow ACK
Enter message (or type 'exit' to disc): hello
Server says: hello ACK
Enter message (or type 'exit' to disc):
```

5. File / List


```
Enter message (or type 'exit' to disc): file:
File not found.
Enter message (or type 'exit' to disc): list
Server says: Client.py
meow.txt
Server.py
Enter message (or type 'exit' to disc): file:meow.txt
File received: meow.txt
Enter message (or type 'exit' to disc):
```


```
Client01 connected at 2025-02-10 16:57:14
Client02 connected at 2025-02-10 16:57:38
Client03 connected at 2025-02-10 16:58:16
Client03 disconnected at 2025-02-10 16:58:34
Client03 connected at 2025-02-10 16:59:44
Client01 disconnected at 2025-02-10 16:59:54
Sending file:
Sending file: meow.txt
```

Possible Improvements:

With more time, we could enhance the file transfer process and improve error handling. Also, we would add a basic user interface with more visuals and interactive messaging for the client and explore simple ways to support more connections.