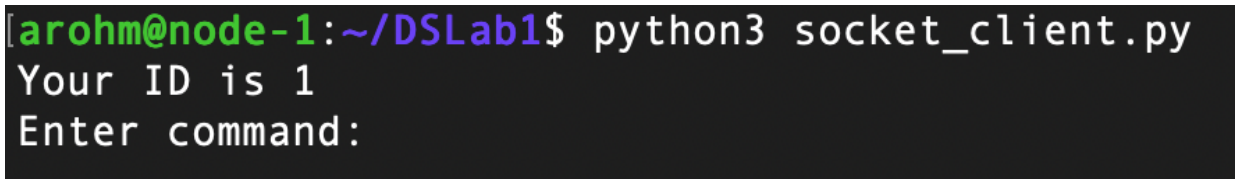
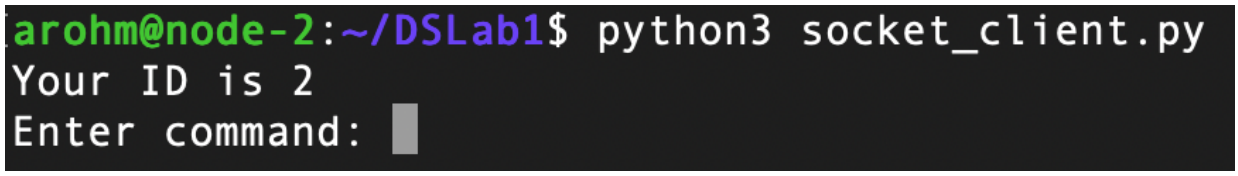
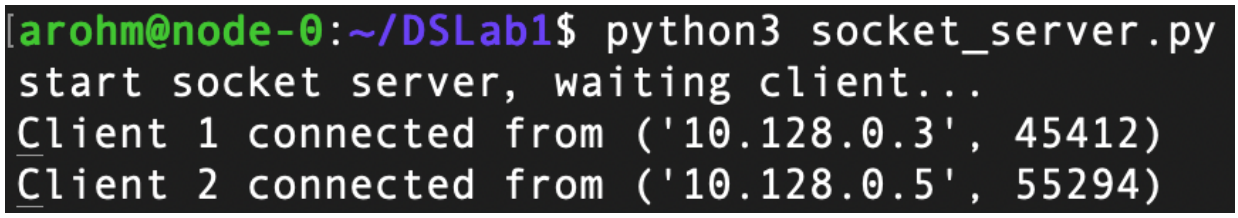
Lab 1 - CISC 5597 - Aidan Rohm - October 6, 2025

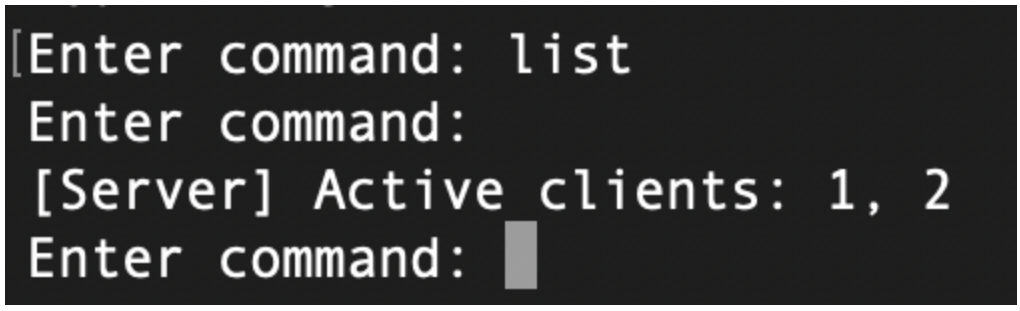
Writeup/Report for Lab 1

Demonstration Screenshots

1. Initial setup
    1. 

    2. 

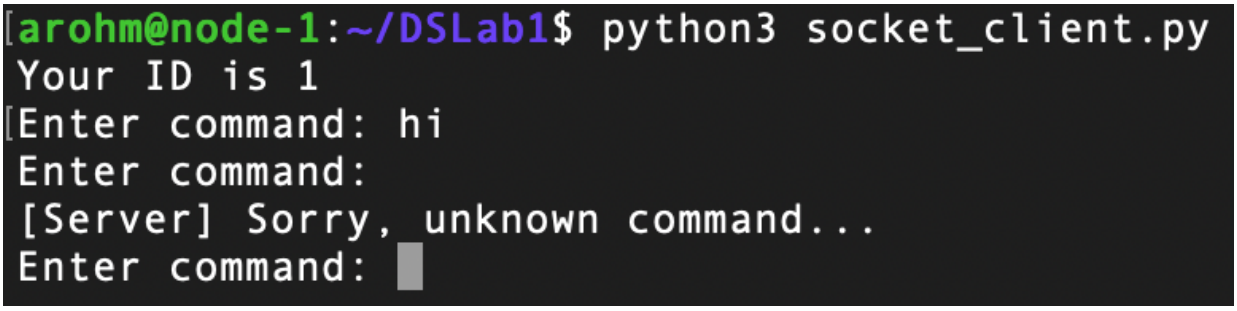
    3. 

        1. Clients 1 and 2 communicate over node0. Node0 executes the server script and demonstrates that the clients both connected. This is demonstrated from the internal ip and port number being displayed from both connections
2. List command testing
    1. 

        1. The list command is used on client1 and the active client ids are listed. It took me a while of trouble shooting to realize that "Enter command: " is displayed again simply because the client reprints the prompt before the server has time to respond
3. Forward command testing
    1. node1:

1.
```
Enter command: forward 2 Hello from client1!
Enter command:
[Server] Message forwarded to 2
Enter command:
```
2. node2:
    1.
```
arohm@node-2:~/DSLab1$ python3 socket_client.py
Your ID is 2
Enter command:
[Server] 1: Hello from client1!
Enter command:
```
    2. node1 is used to forward a message to node2. Node2 automatically displays this message
4. History command
    1.
```
Enter command: history 2
Enter command:
[Server] 1: Hello from client1!
Enter command:
```
        1. "history 2" is run on client 1 and it demonstrates the history of their communication.
5. Exit command
    1.
```
Enter command: exit

[Server] Goodbye!
arohm@node-1:~/DSLab1$
```
    2.
```
arohm@node-0:~/DSLab1$ python3 socket_server.py
start socket server, waiting client...
Client 1 connected from ('10.128.0.3', 48578)
Client 2 connected from ('10.128.0.5', 45224)
Client 1 disconntected!
Client 2 disconntected!
```
        1. The first screenshot here demonstrates the nodes correctly disconnecting from the server. The last screenshot demonstrate that the server acknowledges the disconnection

6. Error
    1. 

        1. The command "hi" is not supported or implemented, so a basic error message is thrown. Any command other than the previously demonstrated, will not work and an error message will be thrown.

Brief Report

Challenges:

There were a few main challenges that I encountered while developing the code:

1. The first challenge I encountered was finding a way for the second client node to display the message after the first client forwarded one through the parent. Originally, I only had the single thread used for communication, but I quickly realized that I needed to use a second thread to explicitly handle all cases of receiving messages. This way the client is able to adequately print the message before continuing.
2. For a while, there was some confusion as to why "Enter command: " was showing up after I would enter a command on either of the client nodes. I soon realized that this was simply because this prompt would print before the server got a chance to respond. As a result, the "Enter command: " prompt will display immediately after a command is entered. This honestly made sense considering we discussed how this communication between the client and the server can take time, and sometimes the client is already ready for the next message before the server responds. I decided not to modify this functionality simply because it seems more accurate. In my scenario, the server does respond very fast, but in the real world this is not guaranteed. We would not necessarily want to wait for the server to respond before continuing with another command that may be completely unrelated to the other node.
3. Finally, I had a little bit of trouble getting the exit command to work gracefully. What was happening frequently was that the node would disconnect itself from the server before the server had a chance to say goodbye. This was similar to the issue mentioned previously. The node is too fast for the server and the message is therefore not given a chance to display. I decided to add a short timer to allow the message to display before the node quits.

Code explanation

1. Server:

    1. The server listens on node0's internal IP as this is the address that each of the clients will be forwarding messages to. Each client is assigned a unique id upon connection to the server. The server then supports commands such as list (show active clients), forward (to send a message to another client), history (to retrieve past messages between two clients) and exit (to close the connection). These messages are saved in a histories dictionary. The threading.Lock ensure that the thread has safe access to the shared information. Finally, if and when a client disconnects with the exit command, the server removes the client from the active list and closes the socket.
2. Client:
    1. The client connects to node0's internal ip using TCP and receives its unique id from the server. It then runs two loops, one of which is a listener thread that is waiting to hear from the other clients (and then prints the messages that are sent). The other is the main input loop that allows the user to enter a command. When the client sends exit to the server, both of these threads stop, the socket closes, and the program terminates.

How to Test the Code

1. Firstly, node0 should run socket_server.py while nodes 1 and 2 run socket_client.py, this will initiate the basic setup
2. Running "list" on either of the client nodes (node1 or node2), will list the active client ids
3. Running the "forward" command on either of the clients (using the syntax "forward {id} {msg}) to adequately forward a message through the server to the intended id
    1. then check the window of the receiving node to see the message being forwarded
4. Run the "history" command on either of the clients (using the syntax "history {id}) to display the messages that have been communicated between these two nodes
5. Run the "exit" command to simply allow the client to disconnect from the server
6. Any other words typed will be treated with an error message. Only the above listed commands are given implementations.