Stephen Onnen
245599375
Coms 319 HW 01

For my assignment I decided to use Golang for both the client and server. I have made basic web-servers in Go serving static files, so I was already somewhat familiar with the http library. After reading through the assignment I decided that using a WebSocket for the client/server communication would probably be the easiest and require the least amount of handling of http requests. The only design decision I really had to think about was how I would prevent the server from sending incoming messages to all clients. This is because my initial design was to have two threads: one handling incoming connections, upgrading them to a WebSocket and continuing to listen for messages, and another thread taking messages posted to a go "channel" and handling send them to all current connections. Since all my active connections were being stored in a map, and then iterated through to send all messages to all clients, I decided to make the key of the map a UUID. This allowed me to attach the UUID of a connection to all its messages. With the UUID embedded in the messages dropped into the msg channel, I was able to create logic that would not send a message to a active connection that matched the UUID of the given message.

A big friction-point of this assignment was trying to get the output to be pretty on Window's console emulators, something I found to be impossible without extensive third-party librarys. Unlike OSX and Linux, Windows did not recognize the escape sequences I was using to flush a message as it was written to Stdout. On Linux/OSX this was done to format Stdout and present it in the terminal in color with the username present. I finally gave up on making it pretty on Windows and settled with double printing all messages typed to Stdout. This error is not present when the client/server are running with OSX, Linux, or Docker.

*The following screenshots are displayed with the server being in the top terminal, and two separate clients running in the bottom two terminals.*

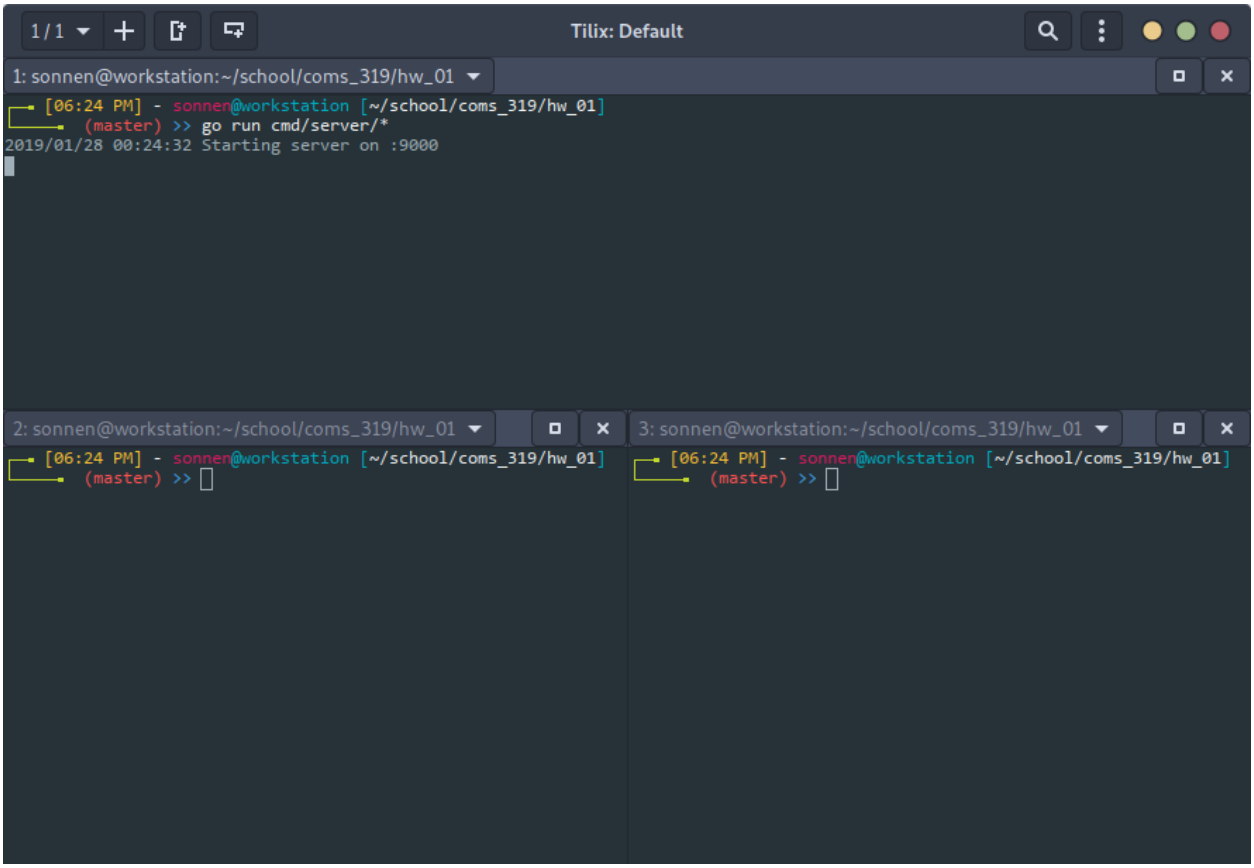


*Figure 1 Server starting*

Stephen Onnen
245599375
Coms 319 HW 01

1.1 Connect to Server

a. When client starts, it should come up with *Enter your Name: prompt.*

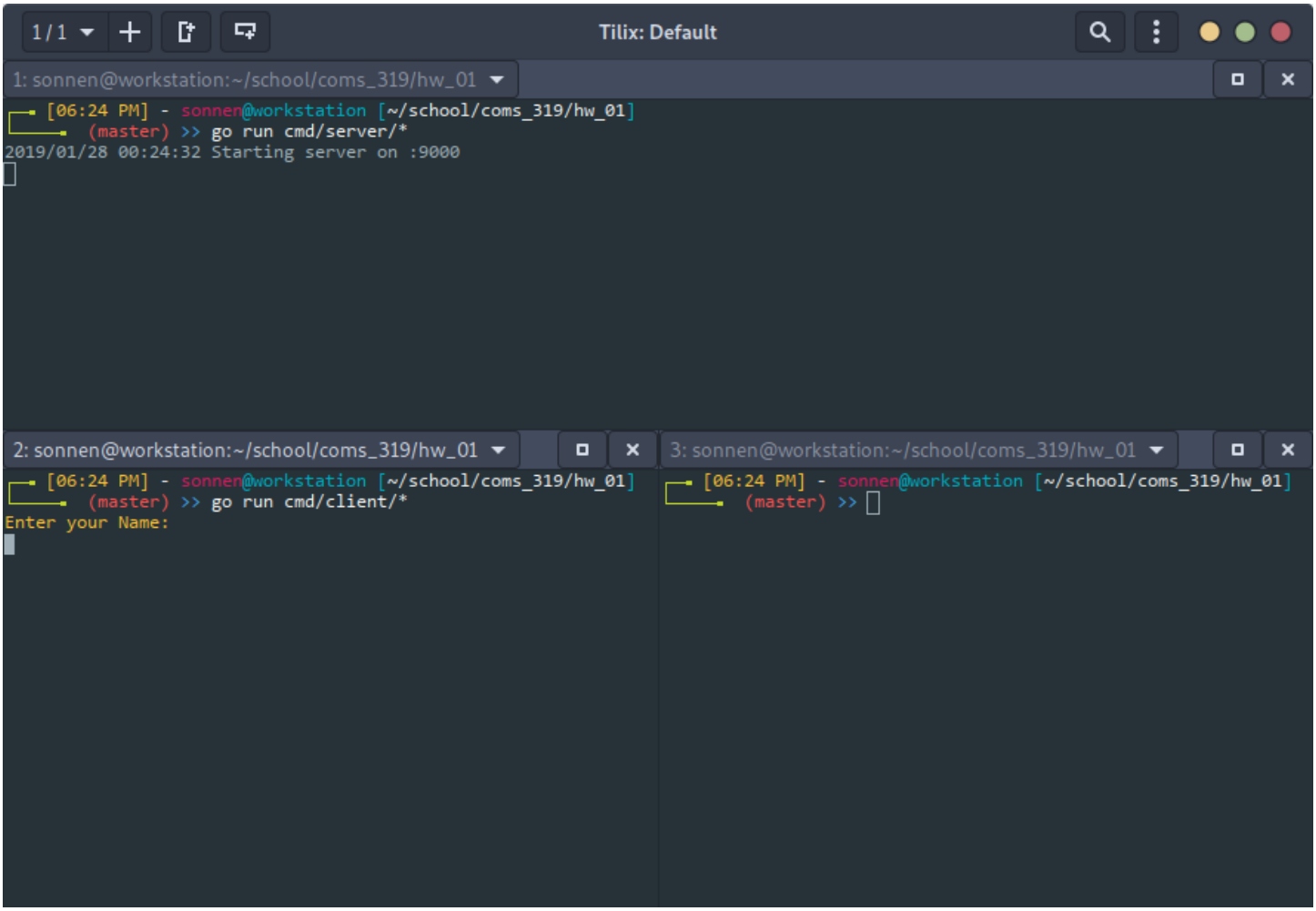


*Figure 2 Enter name prompt, bottom Left*

b. After entering a name, the client should connect to server.

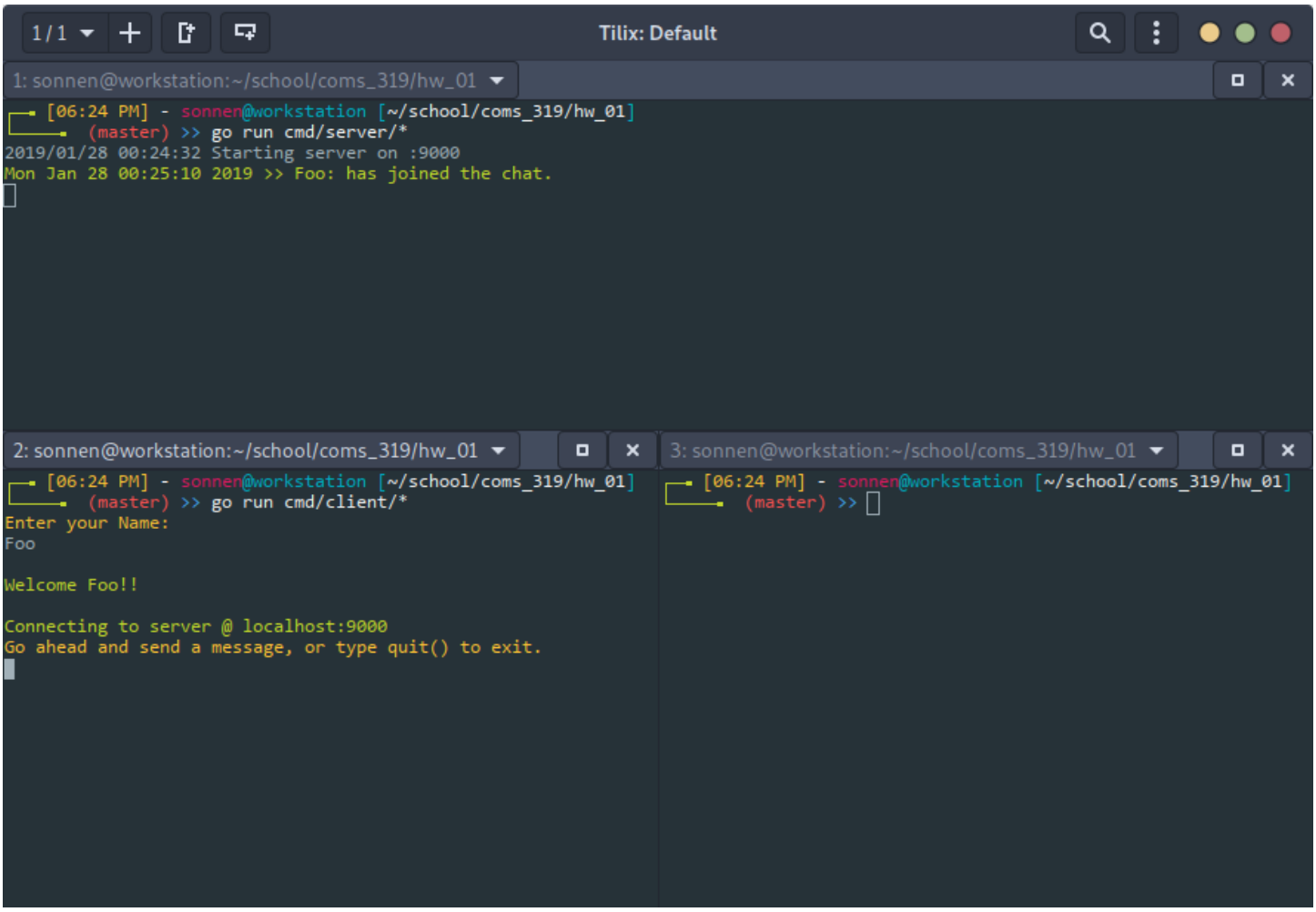


*Figure 3 Client One connects*

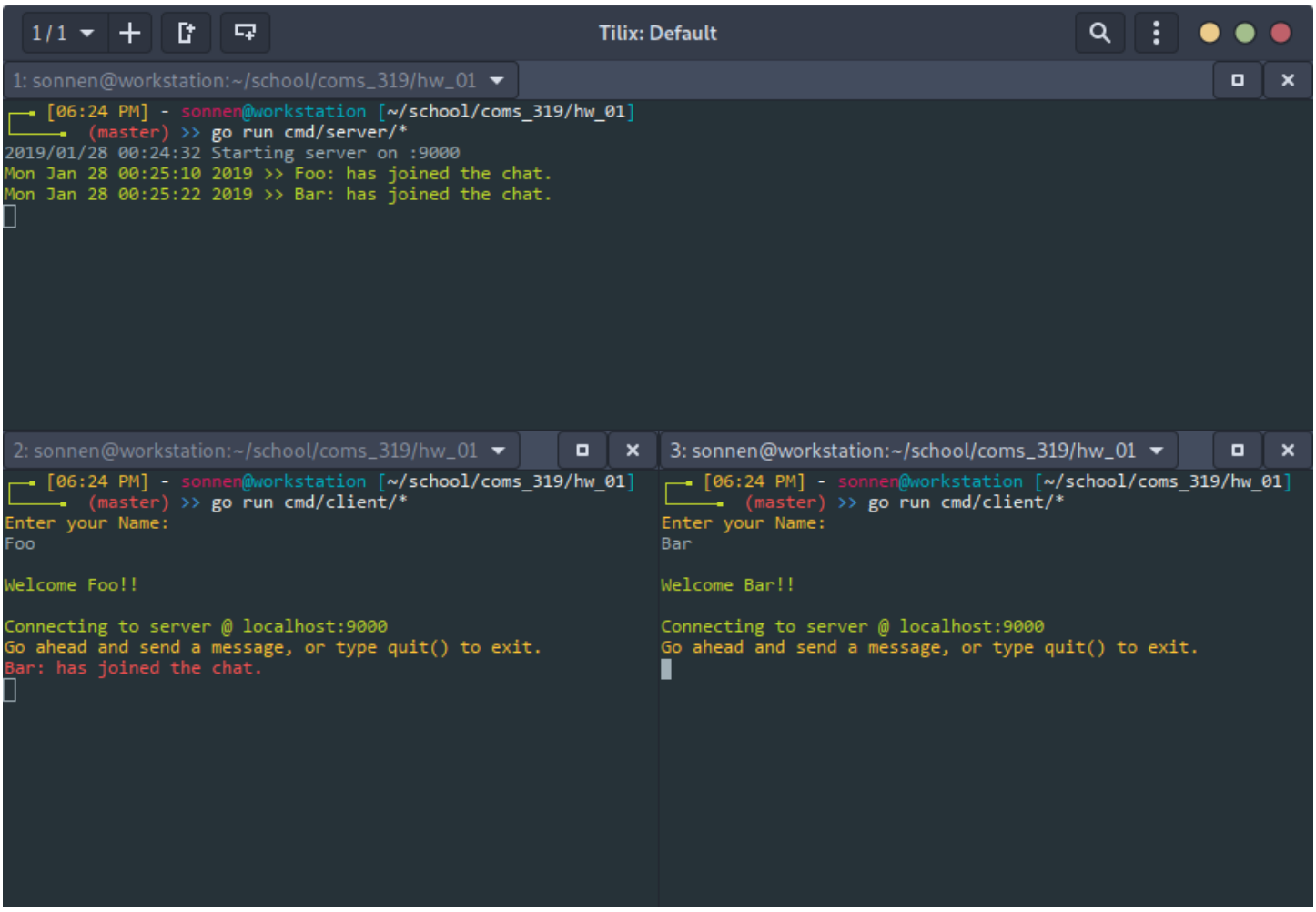


*Figure 4 Client Two Connects*

1.2 Send a text message to the server

a. Send username and chat message to server.

Each message is sent to the server WebSocket as a JSON object with username and text fields.

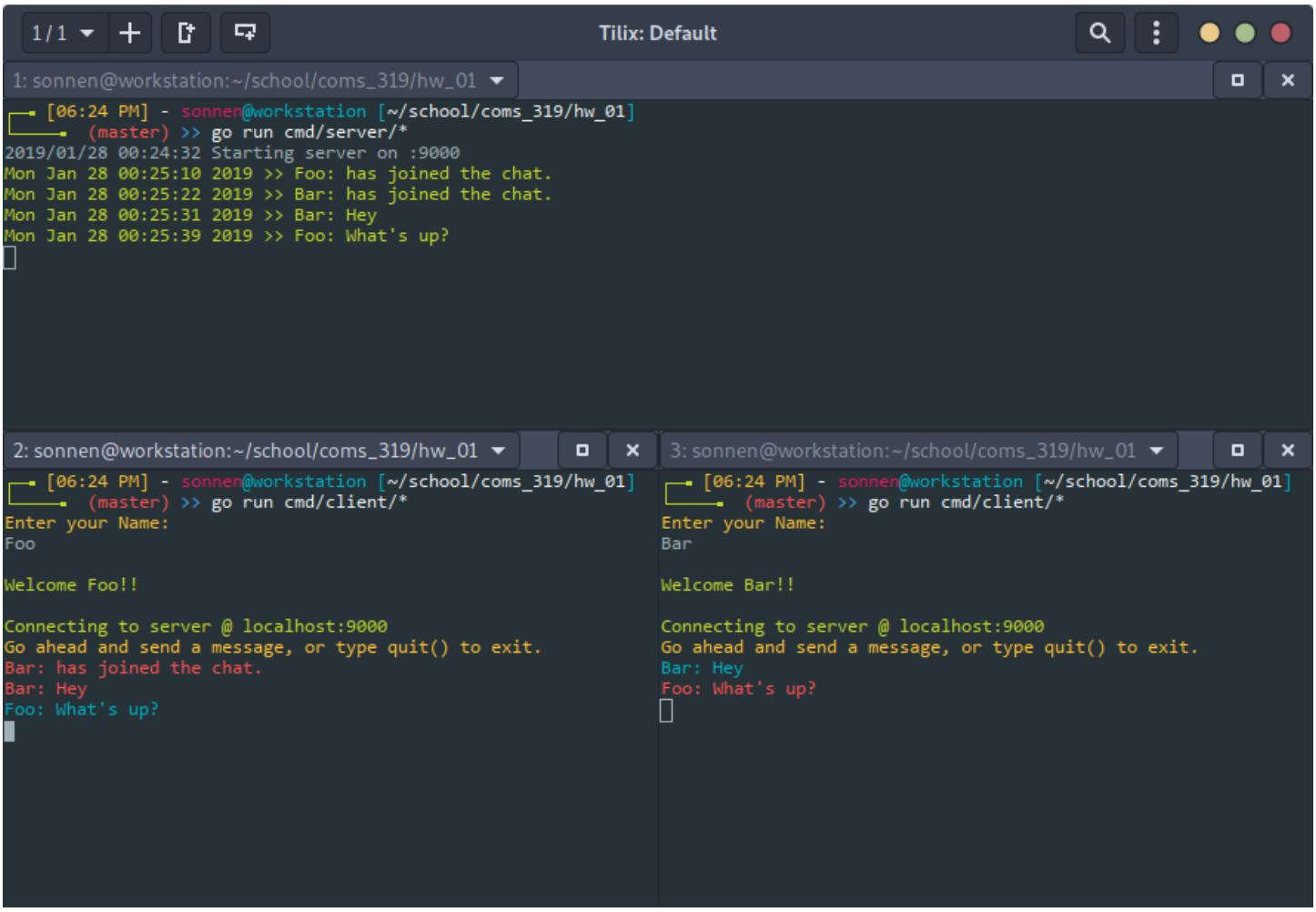


*Figure 5 Messages back and forth*

b. Server broadcast's Clients message to every client other than sending client.

See figure 5, all messages are logged to the server's Stdout with the time the message was sent. Clients only receive messages that did not originate by them, this ensured by checking each message against a UUID assigned to each of the server's active connections.

c. Message printed in each client's console and server's console.

See figure 5. Each of the messages are printed in color with incoming messages appearing in red and outgoing messages in blue.

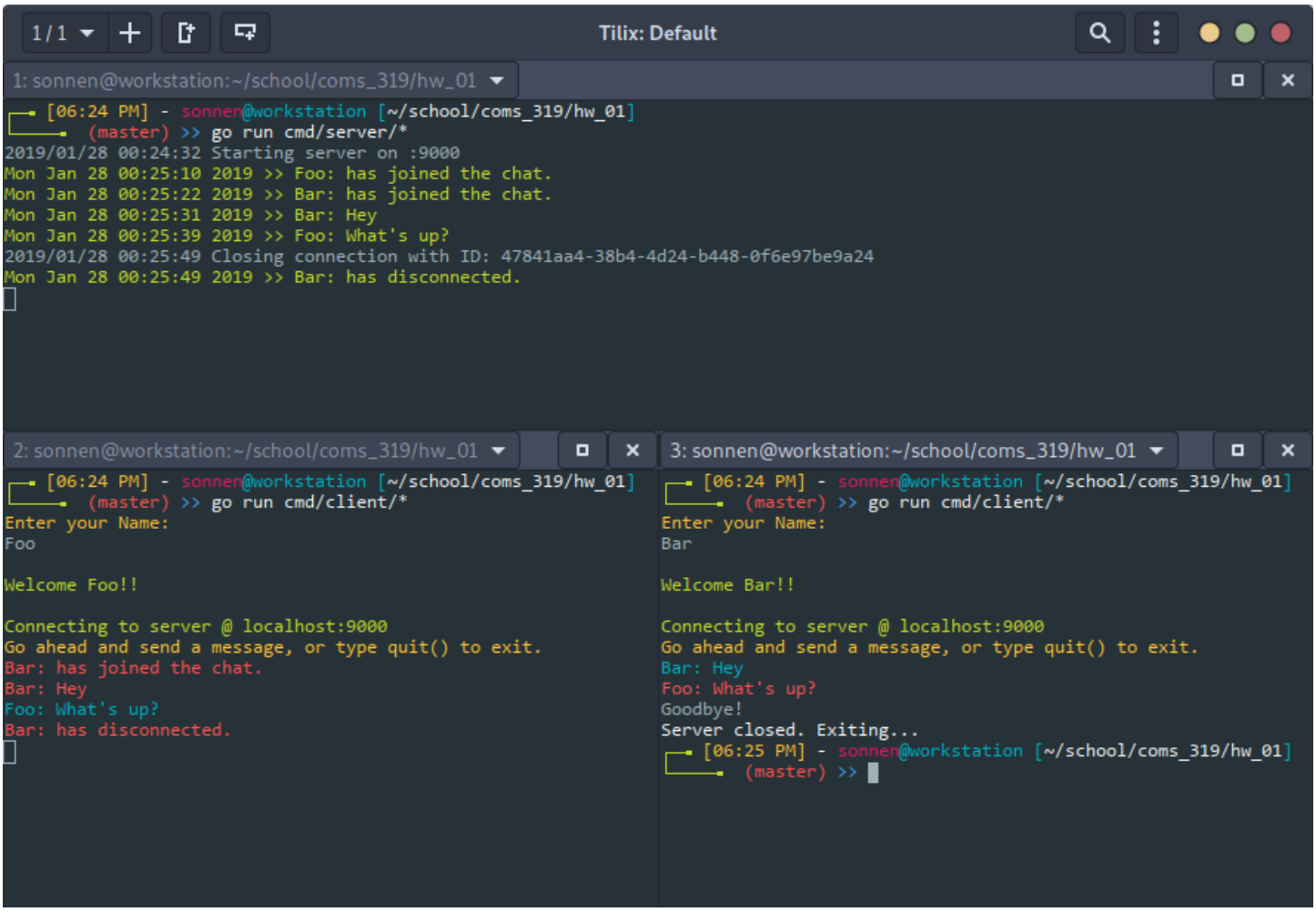


*Figure 6 Client Two Disconnect*

After a client types quit() they send two messages to the server: the first is a JSON message with their name and “has disconnected”, and the second is them closing their WebSocket. When a WebSocket is closed the server prints to Stdout the UUID of the client that closed their connection.

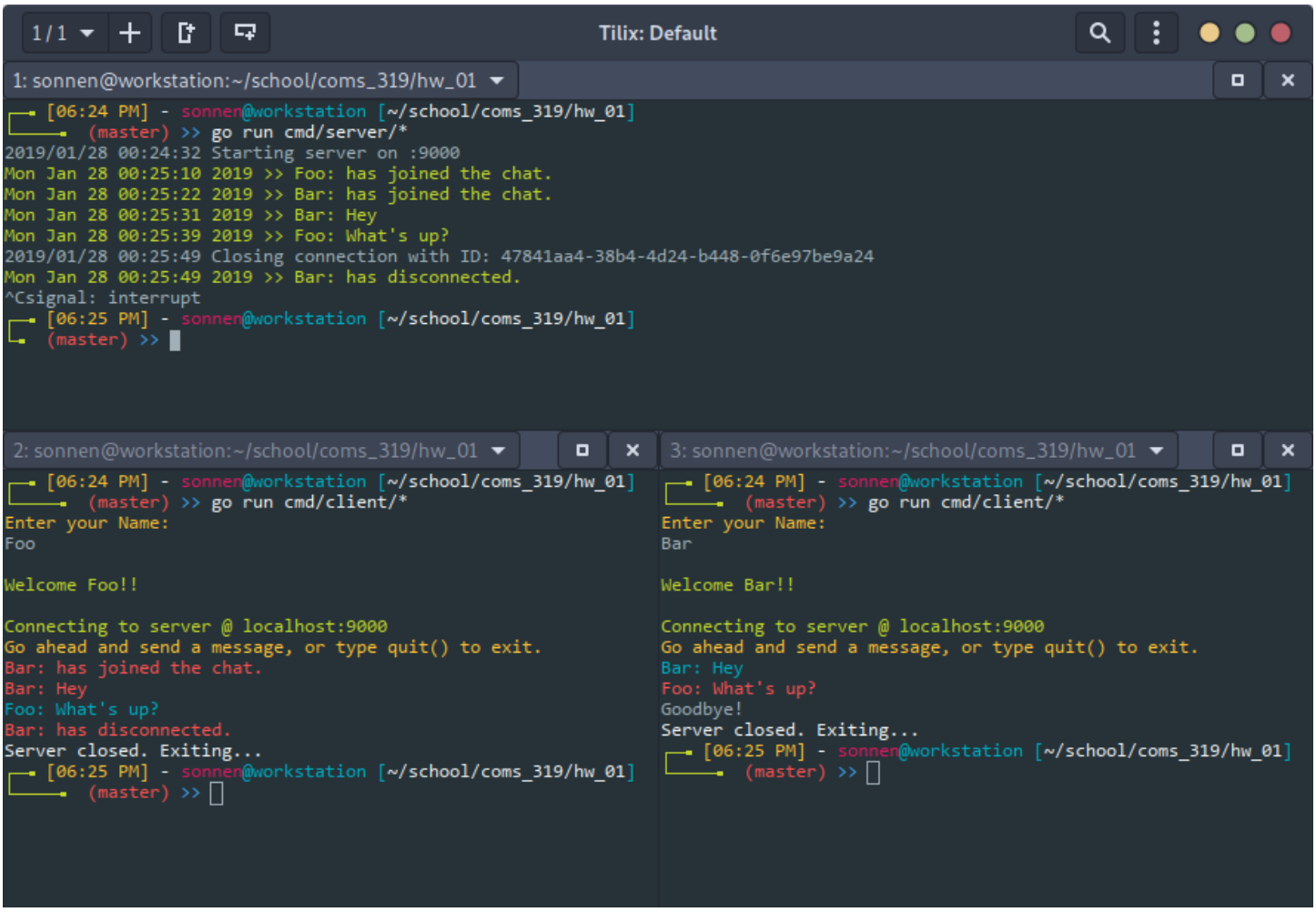


*Figure 7 Server exited*

If the client loses connection with the server, it will print to Stdout saying the server has closed, and will exit the client application.