Introduction

For Assignment 6, we designed one server with multiple clients network. This network is able to 1) allow multi-clients to **register** with a server. 2) allow **direct communication** between two clients in the network. In the sections following, we will explain how the network works, what our personal design is, how users use it, the full use case, and the challenge we still are facing in this stage.

Design Choice

This assignment is splitted into server.cpp and client.cpp. From the client side, we used the *fork* function to support that a client can always do both **listening** and **writing** messages. From the server side, it will keep **accepting** new clients, **receiving** and **reading** each client's messages, and **redirecting** messages. In order to allow one client to send direct communication, clients will first send a message to the central server, then the server will redirect that message to the corresponding client.

The message format is designed in our personal choice. Every massage should be 1) started with the name of the client you want to send to. 2) Having an "<-" pointing to the name on the left. 3) Creat your message after the arrow. For example, "David<-How are you?" means sending "How are you" to David. Then your message prefixed with your name and "->" will be received by David. For example, your name is Eva, the message in David's window will be shown like "Eva->How are you?", meaning Eva sent you a message "How are you?". The more comprehensive example will be shown in the next section.

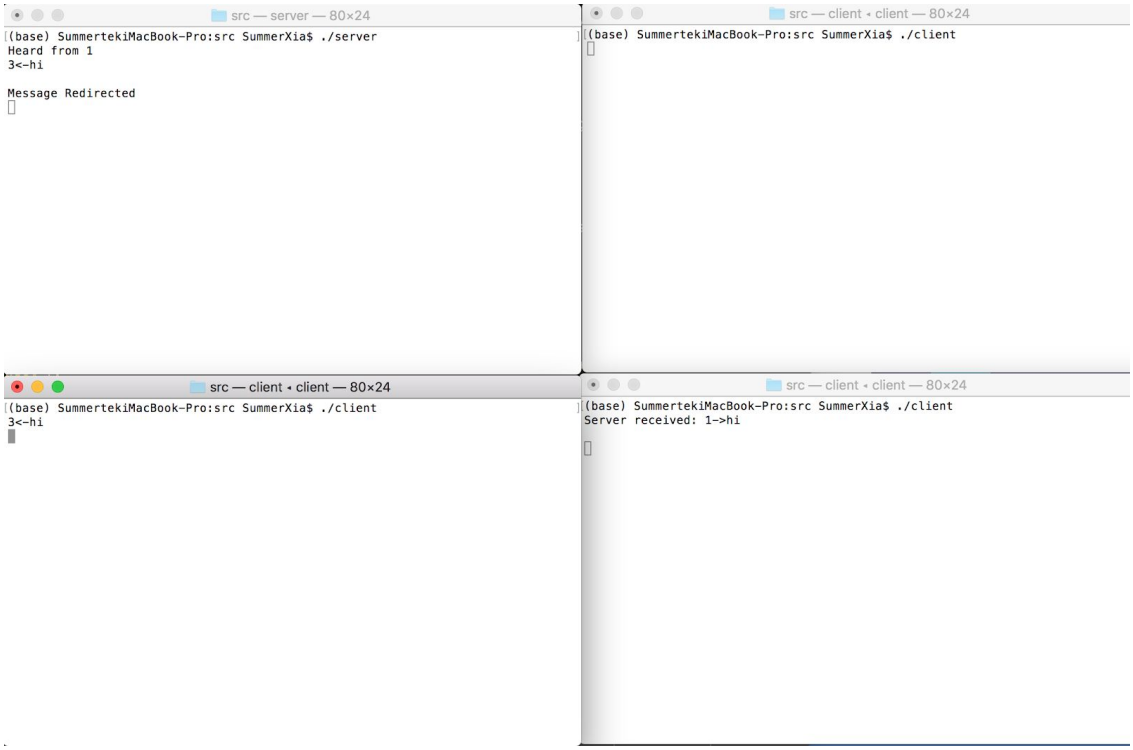
Instruction for User

Inside of zip file, the server.cpp is the server, and the client.cpp is the client. Make before run it in the terminal. Run ./server in one terminal and run ./client in all other terminals. If the number of client are not exceed the limit, all clients can be successfully registered without any error message. Then follow the message rules to send direct messages. The message rules are informed in the section above.

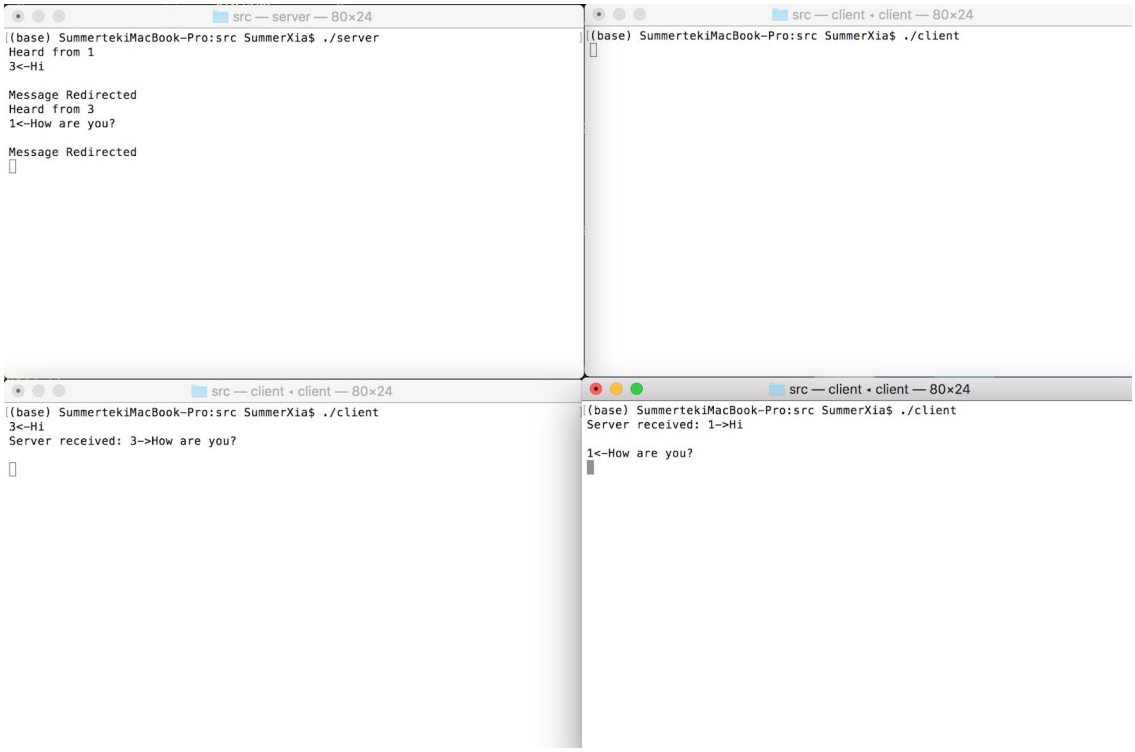
Use Case

Here it is a use case for 1 server VS. 3 clients. For simplicity, three clients' names are "1", "2", "3" respectively. For reference, the top left window is the **server**, the bottom left is 1, top right is 2, bottom right is 3.

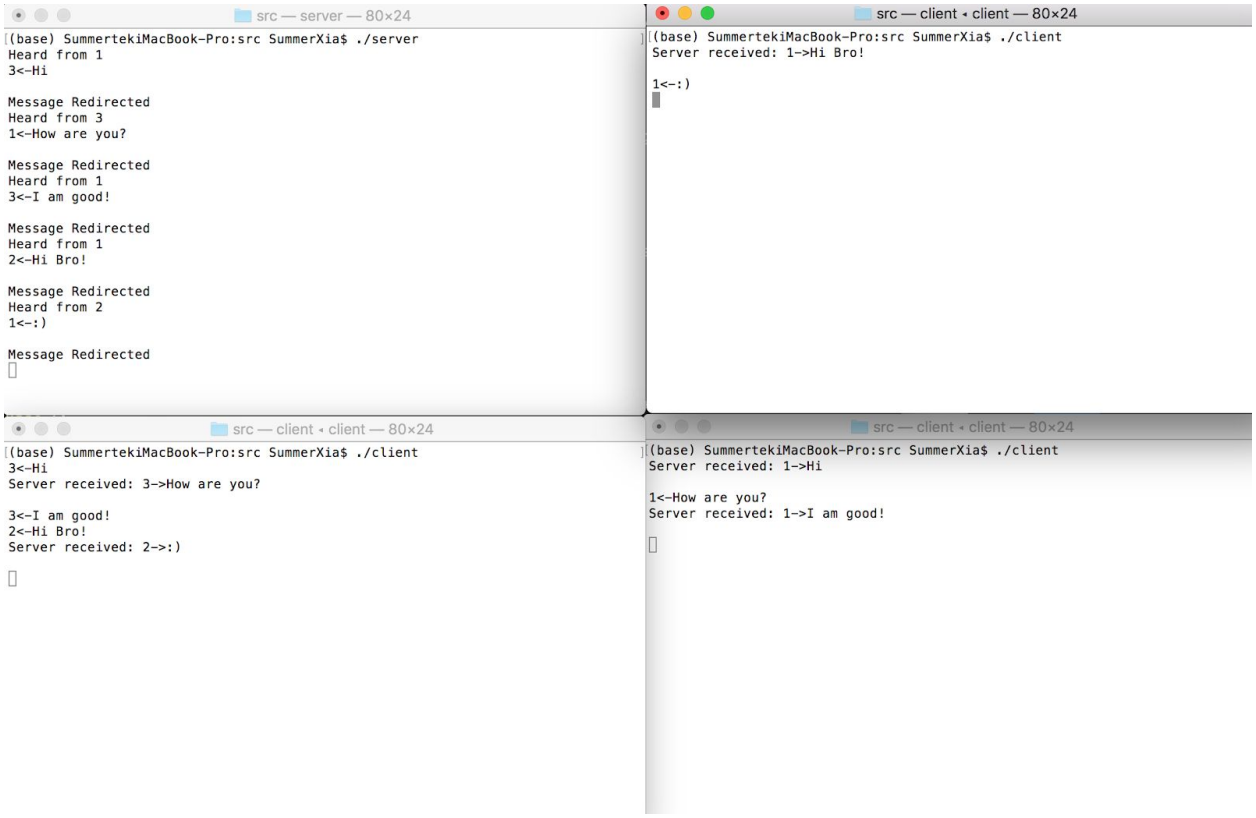
1. Successfully registered and sent "Hi" from 1 to 3.



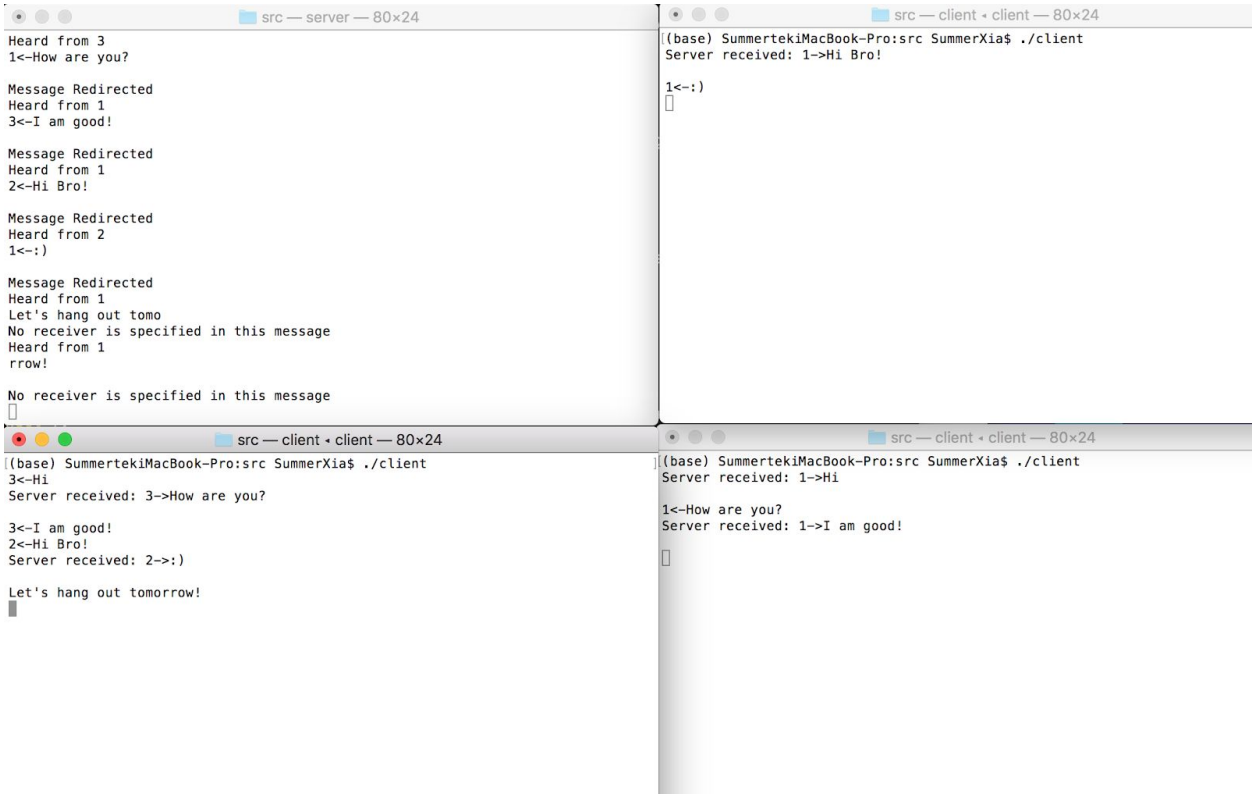
2. Replying back from 3 to 1 with a message.



3. More reply examples and sending msg to new client 2.



4. If the message did not follow the format, the server will not redirect the message. (Wrong message is in the bottom left window. Error message is shown from the server window).



Challenges and Open issues

The problem we are going to solve is we need to improve our design in API so that we can minimize code duplication in our server.cpp and client.cpp. Essentially each client can be seen as a server. Thus our code redundancy will largely be reduced if we can have a more efficient. API. Another problem to solve is we need to reduce hardcoding in our existing code. We are going to utilize the hashmap to store each client's nickname paired with its socket number. If we did that, our network would be more scalable and efficient.