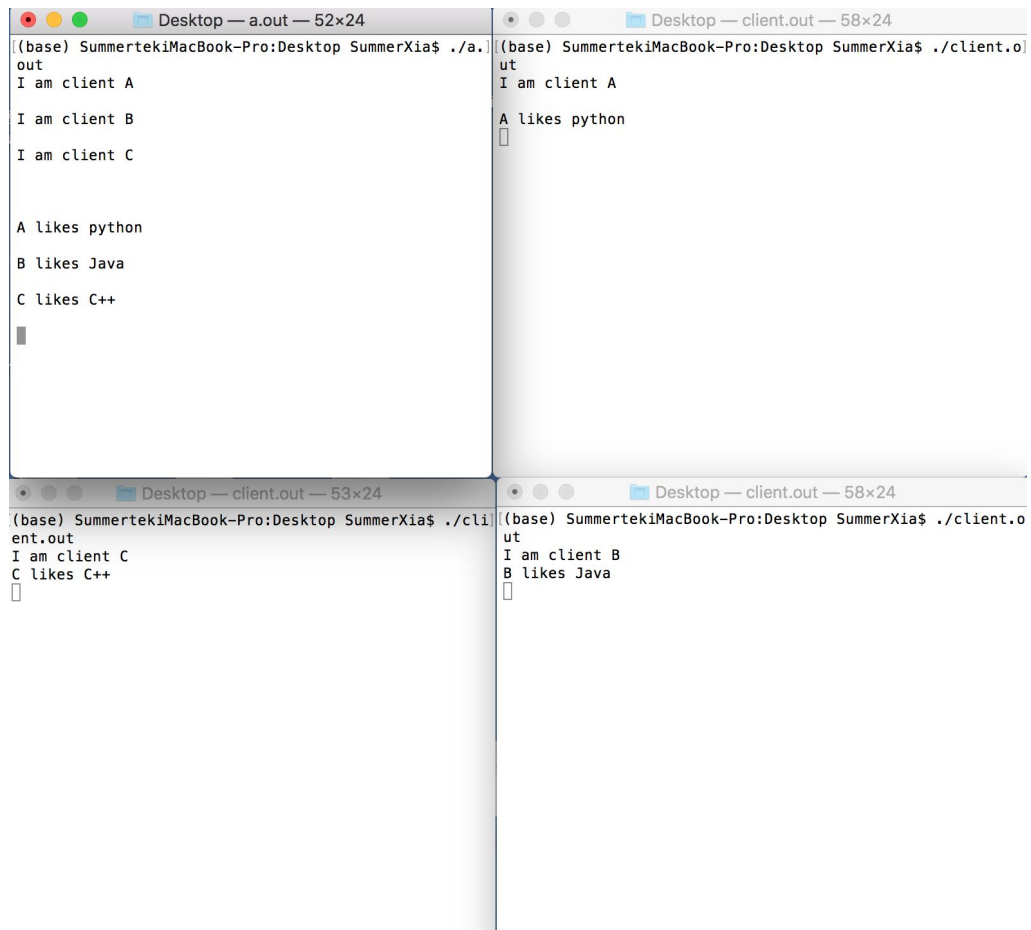


Introduction

In this assignment, our team has implemented a single server with multi-clients network. Our aim is to 1) enable multi-clients to successfully register with server. 2) enable clients to send direct messages to others. By now, we have enabled multiple clients to register successfully with the server. In addition, multiple clients are able to send messages to the central server at the same time.



The image shows four terminal windows arranged in a 2x2 grid, demonstrating the operation of a multi-client network. Each window has a title bar indicating its location and dimensions.

- Top-left window (a.out — 52x24):** Shows the execution of `./a.out`. The output is:

```
((base) SummertekiMacBook-Pro:Desktop SummerXia$ ./a.out)
out
I am client A
I am client B
I am client C

A likes python
B likes Java
C likes C++
```
- Top-right window (client.out — 58x24):** Shows the execution of `./client.out`. The output is:

```
((base) SummertekiMacBook-Pro:Desktop SummerXia$ ./client.out)
out
I am client A
A likes python
```
- Bottom-left window (client.out — 53x24):** Shows the execution of `./client.out`. The output is:

```
((base) SummertekiMacBook-Pro:Desktop SummerXia$ ./client.out)
out
I am client C
C likes C++
```
- Bottom-right window (client.out — 58x24):** Shows the execution of `./client.out`. The output is:

```
((base) SummertekiMacBook-Pro:Desktop SummerXia$ ./client.out)
out
I am client B
B likes Java
```

Design Choice

From the server side, the server has a public known address. All the clients assumably will know this address and will have their own socket connected to this address. The server is implemented by first creating a server file descriptor socket, secondly binding its socket, and then listening to the socket and keep accepting socket. The server stores two variables, which one represents the number of currently registered clients and another one is the maximum number of clients it can listen to at its capacity. If the number of currently registered clients exceeds the limit, the server will notify the last client that the server is running out of socket place. If it

does not, the server will keep listening to other clients and accept clients who has the correct address. For server reading messages from clients, the message content will be stored in a char array buffer. We have set the max message length to be 11. If the message length exceeded the limit, the remaining part of message will be sent in the next message.

From the client side, since the server address is publicly known, we hardcoded that address into our client.cpp. The client can take the address and port to connect to the server. The while loop in client.cpp enables clients to send messages to the server all the time.

Challenges and Open issues

Our design is able to enable multi-clients to register with server. However, we have not been able to implement the feature that enables clients to send direct messages to another.

We understood that for each client, in order to send direct messages to others, it should know other clients IP addresses. One way to implement it is every time when there is a new registered client, the server will store that new client's IP into the list of clients' IP. Then because one new client is registered, the server is responsible to broadcast the updated list of clients IP to every other client. By knowing other client's IP, one client can send a direct message by input the one client IP and his message.

We have tried multiple ways in socket programming, such as creating multiple sockets in client.cpp, using listen socket to listen to requests. But it is still confusing on how one client can send a message to another by type in another's IP.