Jincheng Lu UNI: jl5801

Initially, use this command in the window to start program:
`python3 ChatApp.py`

Command line in program:

`ChatApp -s <port>`: Start server.
The default IP address of server is your local IP.

`ChatApp -c <name> <server-ip> <server-port> <client-port>`:
Start a client.
Target server is usually the server you set up above, so server-ip could be your local IP, and server-port is what you set before. Default IP address of this client is your local IP.

`send <name> <message>`: Send inforamtion to a client that you intend.
Name is your intended client's name (eg. mike, x, client1), that is used to register in the server table. Because any active clients have updated info table from the server, so we can search intened clients' IP and port in that table, according to the client's name.

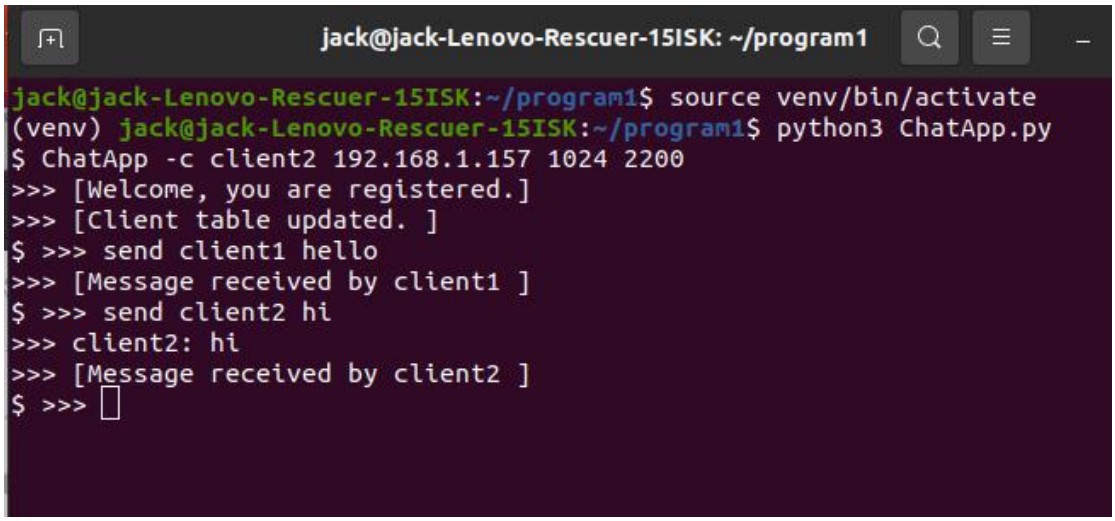
`dereg <name>`: Tell the server that this person in this client is about to de-register.
Before send message to the server, should check name with the registered name in this client.
In the server, it will update this name's status to 'no' in the table, and broadcast all active clients.

`reg <name>`: Tell the server that this de-reg client wants to log back.
Before send message to the server, should check name with the registered name in this client.
In the server, it will update this name's status to 'yes' in the table, and broadcast all active clients.

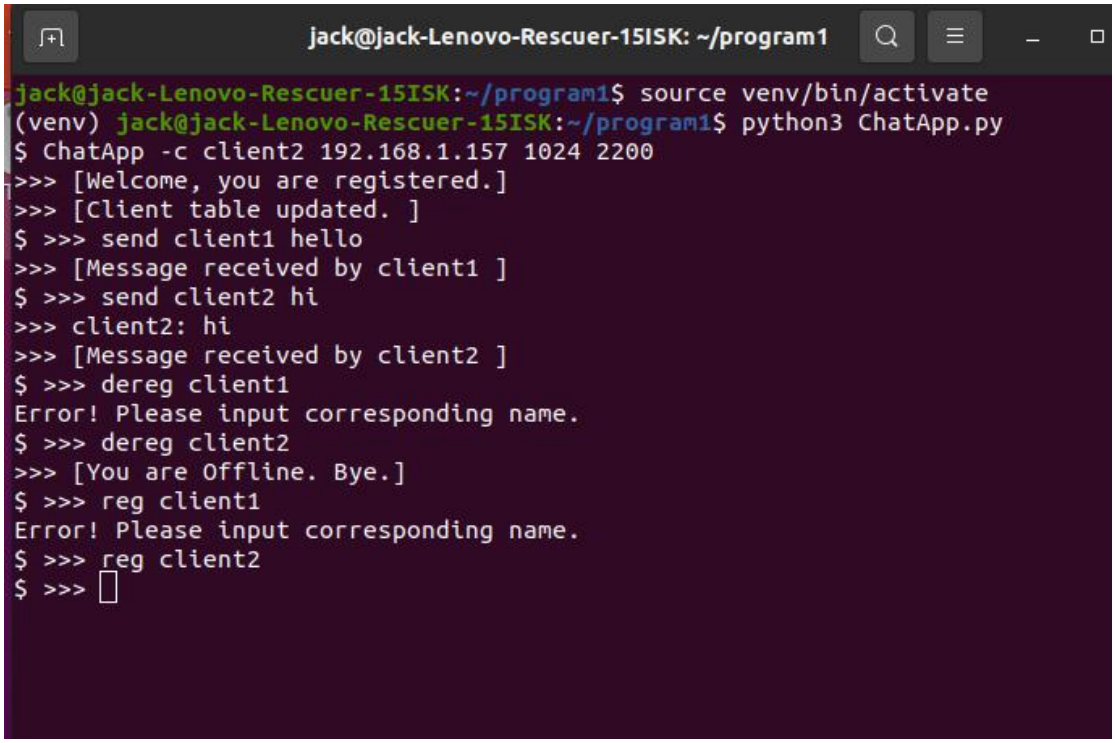
`send_all <message>`: chat with all active people in the channel.

Function:

Client can send itself not crashing



reg / dereg work check name

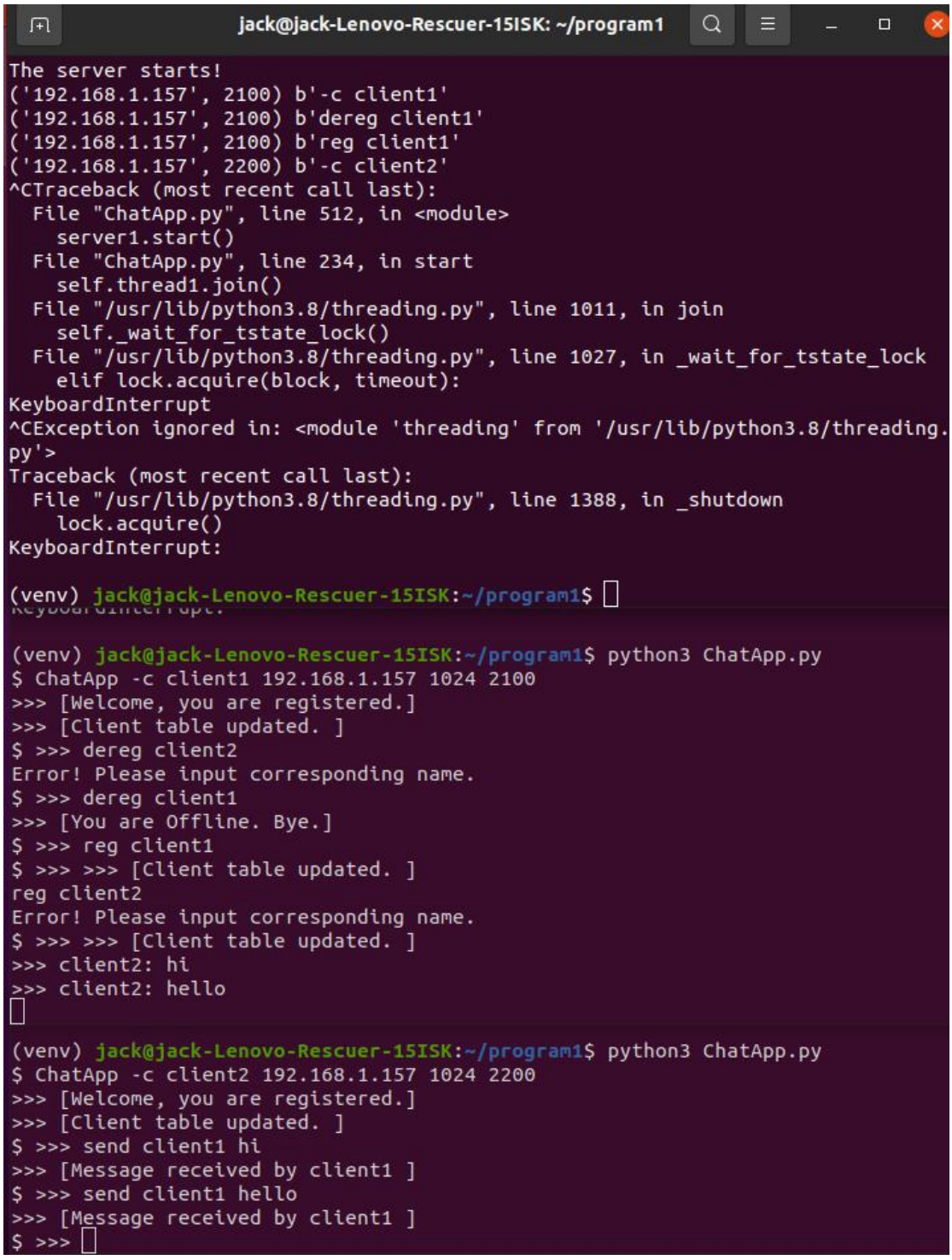


Server down, 5 attempts and conclude

```
def send_msg(self):
    # 发送信息
    while True:
        data_info = input("$ >>> ").strip()
        sentence = data_info.split(' ')
        # de-reg
        if sentence[0] == "dereg":
            # validation
            while sentence[0] == "dereg" and sentence[1] != self.name:
                print("Error! Please input corresponding name.")
                data_info = input("$ >>> ").strip()
                sentence = data_info.split(' ')
            self.clientSocket.sendto(data_info.encode(), (self.serverIP, self.serverPort))
            time.sleep(0.5)
            times = 0
            # 500ms no response retry 5 times
            while Copen is True and times < 5:
                self.clientSocket.sendto(data_info.encode(), (self.serverIP, self.serverPort))
                time.sleep(0.5)
                times += 1

            if times == 5:
                print("$ >>> [Server not responding]")
                print("$ >>> [Exiting]")
```

Active clients chat after server down



Basic project documentation:

First, start program ChatApp.py, and set up the server before you start any clients. After the server, you can initialize clients, the registration will be recorded by the server. If there is any update, (eg. A new client comes in or an active client offline), the server will broadcast updated table with all active clients. Here, 'all active clients' is searched in the updated table, their status is still 'yes' at that time. So you can see everytime a new client is registered, all active clients will obtain updated local table.

With this local table, clients can search destination ip and port according to their name, then they can chat directly, without the server.

But when a client wants to chat in the channel, it has to send message to the server. The server could help client chat with all active people in the channel, and leave messages with all offline people in the channel.

Also, when clients want to talk with someone that is offline, they must tell the server to leave a message. The server will save messages seperately,(corresponding intended clients name) and extract messages in file when intended preson comes back.

If a client wants to leave, he can send dereg command to the server, and server would update his status to 'no', then respond ack to him and broadcast updates. If he waits for ack too long(500ms), he will repeat trying 5 times, after that he will leave directly.

When he wants to log back, just inputs reg command to the server, it will first check if there is leave message for him, then update his status to 'yes' and broadcast updates. Also, after the server reads messages in the file, it has to clear it up.

Program features:

The server should send clients local table if there is any updates in the table. And clients could use this table to communicate with other active clients without the server. That could reduce the server's workload and traffic in the communications.

And the server can help leave messages and broadcast chat in the channel.

Data Structure:

All messages' format is string, and I add different segments in the header of different messages. Which is easy to be identifies.

The format of table is list, and attributes inside are recorded as str. So we can use ' '.join() to convert the table to a string. Which is feasible to transfer between clients and the server.

Also, leave message is recorded as string in a txt file. The file is seperated with intended names.

Except string and list, using dict to instruct the status of files, as we all know, search in hash map cost litte, so we can check much more quickly.