COMP28112 Exercise 2 Report

March 23, 2023

1 Description

The most important part of the protocol is handling the message. For both client and server we need manage the message they received. In my protocol, for the client I set a register part before everything. For this part, I set a while loop with the state of registration and let the user type only the name they want and send this with format: reg name, then the server will receive this and record this socket in a dictionary which the key is the name and the value is the socket. If there's no repeated key in the dictionary the server will send a message to tell the client that the registration is successful. I also have a state to judge if the server send the state in client. If the server sends the registration is unsuccessful, the loop will let the user to type a new name until the user regist successfully.

```
Display "message received: " + message
If the message is "the user name is already used, please redo":
    Set the registration status to False
Else:
    Set the registration status to True
Return the message

While the registration status of the client is False:
    # Prompt the user to enter a registration name
    Set reg to "reg " + input("enter register name: ")
    # Send the registration message to the client
    Send the encoded version of reg to the client
```

```
(command, sep, parameter) = message.strip().partition("if command == 'reg':")

if command == "reg":
    if parameter in self.dict:
        socket.send("the user name is already used, please redo".encode())
    else:
        socket.send("you have registered successfully".encode())
        self.dict[parameter] = socket
```

For the part of sending a message to other client, I set a while loop to make it run all the time. Due to in registration part I stored the user name in a value, so that we can ask the user to send a message to server directly. For the server, if the server doesn't identify any commands in the message, it will send the message to all the users registered except the sender.

```
(command, sep, parameter) = reg.strip().partition(":")

Display "if you want to sending a message to all users, just type your message"
Display "if you want to check the list of registered/connected users, type 'check users'"
Display "enter the message by following form: messagelname of person you want to send message."
Display "if you want to quit, type 'quit'\n"

while client.isRunning():
    mes = input()
    client.send((parameter + ":" + mes).encode())
    if mes == "quit":
        client.stop()
        break
```

```
# Loop through the dictionary of registered users
for i in self.dict:
    # If the current user matches the specified name
    if i == name:
        # Send the message to the user
        self.dict[i].send((name + ":" + text).encode())
        # Return True to indicate that the message was sent
        return True

# If no user with the specified name was found, return False
return False
```

For the part of sending message to a specific person, I let the user type the message and type a verticle bar and the name user wants to send after the message. For this part I use another way to seperate the message and this makes the name unique. The server will always try to find the name but if the name is empty, the server continue run and once the server identify a name, then the server will start to find the name in the dictionary and if there's no matches in the dictionary, the server will send a message that the user has not register back to the sender.

```
# If the parameter is "pass", do nothing
if para == empty:
    pass
# If the parameter is in the dictionary of registered users
elif para in self.dict:
    # Send the message to the specified user
    self.dict[para].send((speci_name + " (private) ").encode())
    # Return True to indicate that the message was sent
    return True
else:
    # Send an error message to the user if the specified user is not connected
    self.dict[name].send("the user you want to send message is not connected".encode())
    # Return True to indicate that the message was sent (even though it's an error message)
    return True
```

The user can also try to get a list of registered users by sending a message "check users". Once the server identify this command the server will start to send all the keys in the dictionary to the user to show their names.

```
# If the text is "check users"
if text == "check users":
    # Get the list of keys (usernames) from the dictionary of registered users
    key = self.dict.keys()
    # Loop through the keys and send a message to each user with the list of connected users
    for i in key:
        self.dict[i].send(("the list of connected users is:\n" + ", ".join(key)).encode())
    # Return True to indicate that the message was sent
    return True
```

At last we should also allow a user to quit. User can type "quit" to quit. The clint will send a message "quit" to the server and the client will stop at the same time. When the server receive the signal to quit the server will delete the user in the dictionary immediately.

```
# If the text is "quit"
if text == "quit":
    # Remove the user from the dictionary of registered users
    del self.dict[name]
    # Display a message indicating that the user has quit
    print(name + " has quit")
    # Return True to indicate that the message was processed
    return True
```

```
# Loop while the client is running
while client.isRunning():
    # Get input from the user
    mes = input()
    # If the input is "quit", stop the client and break out of the loop
    if mes == "quit":
        client.stop()
        break
    # Otherwise, send the message to the server
    client.send((parameter + ":" + mes).encode())
```

2 List of the commands

At the beginning, you can register by only type you name. You can see following commands in the instruction below.