COMP4621 Project Report (Multiparty Chatroom)

ZHAO Yu Xuan (20497819)

GitHub Repo: Access Link

Introduction

In this project, I am required to implement a server-based multiparty chatroom to enable the concurrent communication among different users via network. During the course, we have learnt several ways to achieve the concurrent communication for multiple clients. It could be concluded with two categories which are thread/process creation in the server side to handle different users and synchronous I/O multiplexing by using `poll()`, `select()`, and the like. Each method has their own pros and cons, selection should be determined according to the restriction and purpose of the development. For this project, we are mandated to use the `poll()` function to handle multiple concurrent clients.

Task and Implementation

The project tasks are well define in the grading scheme. They are divided into different function blocks with their own **command**, such as REGISTER, EXIT, WHO, #<user_name>:, and so on. I will follow the cases in grading scheme and introduce my own implementation for each part.

- **Register/Login** In this part, the server need to firstly judge whether the user name given by the client is in the `listOfUsers`. If no, it means that this is a new user for registration. Then this user information will be appropriatedly constructed and added into the `listOfUsers`. Moreover, the newly created user socket will be added to the `pfds` for further concurrent communication. And the server will broadcast the welcome message for the new user by the customized function `broadcast_online()`. To write the definition of the function, I created a **target enum** to differentiate the `ALL` and `OTHERS`. `ALL` means the message will be broadcasted to all the online users and `OTHERS` indicates that the client requesting the broadcasting will not recieve the message. For the case of registration, the target parameter is set to be `ALL`. Afterwards, the newly registered user will be assigned to a offline message box which is represented by a txt file with the name of **<user name>.txt**. The txt file is simply created by `fopen()` with the "w" access. This message box will store all the offline message sent from other users line by line.
  If the user name existed in `listOfUsers`, this is the case for login. The most important thing is to update the **state** and the **sockfd** of the existing user. The logged user will recieve all the offline messages which is retrived from his own txt file. The message retrival is done by `fopen()` with "r" access and `fgets()`. After that, a newly created temporary file is created and renamed to the name of the message box after removing the original message box. This is an easy and efficient way to clean the message box. Then the server will also broadcast the message indicating the logged user is online to all other online users by calling `broadcast_online()` with the target `OTHERS`.
- **Direct Message** If the client want to send a direct message to a specific user, the format will be **#<user name>:<message>**. Once the server recieved a message with the symbol `#`, it will recognize this message as direct message and execute the follwing steps to finish this task. Firstly, the server will check whether the target user name is in the `listOfUsers` by function `isNewUser()`. This function will traverse all the existing user in the `listOfUsers` and check if there is a user that has

the same user name with the parameter `name`. If the user is not in the list, it means the user is not existed and the server will sent the message indicating there is no such user to the sending user. Otherwise, the target user exists and the server will further check whether the target user is `ONLINE` or `OFFLINE`. For the online case, the server will directly send the formatted message to the corresponding `sockfd` of the target user. In the other case, the formatted message will be appended to the txt file of the target user by function `fopen()` with "a" access, `fseek()`, and `fprintf()`. An extra message that indicates the message is successfully left to the offline target user will be sent to the sending client by the server.

- **EXIT** If a client type the EXIT command, the server will change the state of this user to be `OFFLINE`. Moreover a message that indicates a user left the chatroom will be broadcasted to other online users by `broadcast_online()` with target `OTHERS`. And the corresponding pollfd of the user will be closed and deleted from the `pfds`. The user could press ctrl+c to terminate its client program after seeing the indication in the client terminal window.
- **WHO** The `WHO` command is straightforward to implement. First step is to construct a empty string messge to the requesting client which it is called `ToClient`. Next, the server will traverse all the existing users except the requesting user in the `listOfUsers`. If the traversed user is online, the user name with symbol `*` will be concatenated to the end of `ToClient`. If it is offline, only the user name will be concatenated to the end of `ToClient`. After each traverse, a tab symbol `\t` is added for `ToClient` to seperate each user. After all existing user's traversation, an extra line showing the meaning of symbol `*` is appended to `ToClient`. Last, the server will send the message `ToClient` to the requesting user.
- **Broadcast** A message that did not match any command above is regarded as broadcast request by the server. Then the server will format the message to add the information of user name of the sending client. Afterwards, the formatted message will be broadcasted by calling the customized function `broadcast()`. Please notice that this `broadcast()` function is different with the one `broadcast_online()` because this broadcast() function will take care of the offline users which means the formatted message will also be left to the message box of the offline users.
- **Other Compulsory Requirements**

Results

Bonus

Future Work