CSC 345 – Project #4: Network Programming

Due: May 8, 2019, before midnight

Objective:
Design and implement a multi-user chatting program using socket library on Linux.

Due Date:
May 8, 2019, 11:55pm

Project Details:
In this project, you will implement a command line-based chatting program using socket library. We provide a skeleton program (chat_server/client, chat_server_full/client_full) to demonstrate basic behavior of the chatting program. Please note that the programs are not in a complete form, and may have some bugs. To test the programs, download and extract the zip file, and run make to compile. Open two terminals, and in one terminal, type

```
$ ./chat_server

OR

$ ./chat_server_full
```

and in the other terminal, type

```
$ ./chat_client 127.0.0.1

OR

$ ./chat_client_full 127.0.0.1
```

and see how they work. The difference between chat_server/client and chat_server_full/client_full is that the latter server can accept multiple clients. Thus, you can open yet another terminal, and try to connect to the server_full with another instance of client_full. Then, the server_full will route message from one client to the other (actually it is supposed to broadcast the message to all connected clients).

You can use these programs as your starting point. Create your own program, named [main_server] and [main_client] and submit only your programs.

Program Requirements (70 pts):

- (10 points) main_server can display the up-to-date list of connected clients. When a new client connects or any client disconnects, the up-to-date list will be printed in the server-side terminal. It should work exactly in the following fashion:

  [in server terminal]

$ ./main_server

[in client terminal]
$ ./main_client *IP-address-of-server*

- (10 points) main_server accepts multiple main_client connections and broadcast messages from one client to all connected clients properly.
- (10 points) main_client can specify user name when connecting to the server. The server will maintain the user names and broadcast the names to clients properly.
- (15 points) main_server allows multiple chatting rooms running simultaneously. For this purpose, the client behavior will be slightly changed. Client can request whether it will open a new chatting room or join an existing room. If a client wants to open a new chat room, then it should connect to the server as

  [in client terminal]
  $ ./main_client *IP-address-of-server* **new**

  If the connection was successful, then the client will be assigned a room number from the server and display it on the screen, e.g.

  [in client terminal]
  $ ./main_client *IP-address-of-server* **new**
  Connected to *IP-address-of-server* with new room number XXX

  If the client wants to connect to an existing room, then it can use a command as

  [in client terminal]
  $ ./main_client *IP-address-of-server* **XXX**

  You can assume that all rooms are numbered. main_server is responsible for maintaining the rooms and clients therein properly. If a main_client requests for non-existing room number, it should properly block the request. You can optionally set the maximum number of rooms to be managed by the server, but it should be at least 3.
- (10 points) Allow random joining of the existing room if the client requested

  [in client terminal]
  $ ./main_client *IP-address-of-server*

  assuming the feature above was implemented. If there is no room, then a new one should be created and assigned to this client.
- (15 points) Instead of explicitly specifying or random joining the room, let the client retrieve the list of the rooms available currently and choose from the list as below:

  [in client terminal]
  $ ./main_client *IP-address-of-server*
  Server says following options are available:
  Room 1: 2 people
  Room 2: 1 person
  …
  Room 10: 5 people
  Choose the room number or type [new] to create a new room: _

If no room is available at the moment, it should automatically join to the new room (thus implicitly using new command above).

Extra Credit Requirements (20 pts):

- (10 points) Allow file transfer. A client can send a local file to a remote client with a special command [SEND]. The syntax is like below

  [in client terminal]
  [*this-client-IP-or-user-name*]: SEND *IP-address-of-receiving-client file-name*

  The receiving client should be notified of the transfer and agree on receiving the file (by Y/N question).
- (10 points) Use a random, unique color for each client in a room. It is okay if clients are using different sets of colors for the same room information, as long as individual users are distinguishable with different colors.

Formatting Requirements

- Comment the part of code you made changes.
- In a separate file (**report.pdf**), clearly describe which requirements you implemented.
- In another separate file (**discussion.pdf**), attach your group discussion log including date/time.
- Put all your files into the zip file.

  ```
  ./project4.zip          // your zip submission
  ./project4              // extracted subdirectory inside your zip file
  |- report.pdf           // your project report
  |- discussion.pdf       // your group discussion log
  |- main_server.c        // your server program
  |- main_client.c        // your client program
  |- Makefile             // makefile (REQUIRED!)
  |- (all other files)
  ```

  and put your implementations inside.

What to turn in

- Zip the folder containing your source file(s) and Makefile, then submit it through Canvas by the deadline. There will be a penalty for not providing any working Makefile.