CS 436 - Project: Network programming (Chatroom)
100 points + 10 points extra credit

This assignment is to be done in two-person groups. Do not show your code to other groups and do not look at other students' code. Prevent other students from accessing your code (do not put it in a public directory). However, you are free to use any tools you like (library, search engines), to discuss problems with others, and to seek the help of the instructor.

1. Project

The following is the project I assign to the class. You can either do this project or define another project. Some suggestions to define a project are given below. If you plan to define a project, you need to submit a proposal to the professor and completely describe your project, 5 days before the due. **If the professor approved your proposal, then you can work on your proposed project**, and your grading will be according to your approved proposal. If you don't get the professor's approval before the project due and submit any project other than the one that has been defined here, your grade will be 0. The professor encourages all students to define their own projects.

1. Chatting between multiple clients through a server. This is not a chatroom. Instead, a pair of clints chat together. Client 1 chats with client 2, and client 3 chats with client 4. Clients 1 and 2 will not see the chats between clients 3 and 4. The server should keep track of the client pairs and forward each message only to the paired client.
2. Communication between an IoT device, like an Arduino or Raspberry Pi, and the client program to exchange data through a server on the cloud. For example, client 1 sends a request to your server and the server will forward it to the IoT device. The request could be a command to turn on/off an LED on an Arduino or a camera on a Raspberry Pi to take a picture and send it to the client.
3. A blockchain application where it communicates with the blockchain server.
4. Secure communications between two IoT or mobile devices.
5. A program that automatically communicates with public servers like GitHub or YouTube. You should define an application for such communication as well. For example, a client can get a list of avatars at bungie.net server or search an avatar name, select one of them and download it from the server. You implement the client in Python. You should use tkinter or a similar module to create a GUI.

2. Python3

You will implement this project in Python3. How to learn Python3? There are many great tutorials on the Internet. The following is a link to a very easy and helpful tutorial for beginners with hands-on examples in the popular PyCharm IDE. Study the first 2 hours of the video.
https://www.youtube.com/watch?v=_uQrJ0TkZlc&t=16696s

The playlist tutorial by Corey Schafer is also great to learn Python with examples.
https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU

3. Socket programming in Python

Refer to the socket programming lecture and video. Also, a simple socket programming example is given in the project section of the course. Download the attached folder. I have also provided an instruction to run the files in the project section. We implement this project with **TCP**. A reference for an initial chatroom in Python is given in the following webpage.
How to create a chatroom in Python?
https://www.askpython.com/python/examples/create-chatroom-in-python

Important note: If you simply copy-paste this code to Python files, run them, take snapshots, and submit them to your project assignment, your grade will be 0 because this code is not your effort. You should make the changes as given in your project description. But this link is a good initial example to start with.

4. Project Objective

In this project, you will develop a client-server application that implements a simplified chatroom application using socket programming in Python language.

5. The structure of the message

In this project, the structure of a message is as following. Each message contains all these fields. If a field does not apply to a message, the value for that field will be 0 or blank depends on the type of the field. For example, in a report request message, the FILENAME is empty because the user is not uploading any file in that message.

Note: You can add more fields or slightly change the values that each field accepts if that make the implementation easier for you.

1. REPORT_REQUEST_FLAG: If this is a report request message, this field is 1. Otherwise, it is 0.
2. REPORT_RESPONSE_FLAG: If this is a report response message, this field is 1. Otherwise, it is 0.
3. JOIN_REQUEST_FLAG: If this is a join request message, this field is 1. Otherwise, it is 0.
4. JOIN_REJECT_FLAG: If this is a join reject message, this field is 1. Otherwise, it is 0.
5. JOIN_ACCEPT_FLAG: If this is a join accept message, this field is 1. Otherwise, it is 0.
6. NEW_USER_FLAG: If this is a new user message, this field is 1. Otherwise, it is 0.
7. QUIT_REQUEST_FLAG: If this is a quit request message, this field is 1. Otherwise, it is 0.
8. QUIT_ACCEPT_FLAG: If this is a quit accept message, this field is 1. Otherwise, it is 0.
9. ATTACHMENT_FLAG: If this message sends the contents of a file, this field is 1. Otherwise, it is 0.
10. NUMBER: If this is a report response message where REPORT_RESPONSE_FLAG=1, this field will be set to X, where X is the number of active users, which is in range [0-3]. Otherwise, it is 0.

11. USERNAME: If in this message JOIN_REQUEST_FLAG=1 or JOIN_ACCEPT_FLAG=1 or NEW_USER_FLAG=1 or QUIT_ACCEPT_FLAG=1, this field is the username. Otherwise, it is empty.
12. FILENAME: If this message sends the contents of a file where ATTACHMENT_FLAG=1, this field includes the filename and its path, e.g. “attachment/file1.txt”. Otherwise, it is empty.
13. PAYLOAD_LENGTH: The length of the payload. The length is 0 for messages that have no payload and X for a string message of length X characters, including \n.
14. PAYLOAD: The contents of the message or the file that the sender sends to the receiver.

6. Chatroom Implementation

In this project, we implement a chatroom.

1. You can use Python tkinter module to create a GUI, but it is optional.
2. We assume at most 3 active users can chat in this chatroom. The server keeps track of the users and their messages. If a fourth client requests to join the chatroom, the server will not accept it. The usernames must be unique among the active users.
3. We implement two programs: a client and a server. We name them “client.py” and “server.py”. Use port number 18000 for the server. While you are testing your program, if you got any issue with this port number, feel free to change it to 18001 or above.
   Note: In this project description, when we say client, we refer to the client.py program that is running on a terminal. When we say user, we refer to the person who runs client.py on a terminal and enters some commands and interacts with the client program.
4. First, we run the server program on a terminal. Notice that no user directly interacts with the server.py program. But while we are testing our program, we expect your server prints on the screen a brief report of what it is doing.
5. Next, we run the client program. The client program displays the menu as given below to the user and the user enters one of the options.
   Please select one of the following options:
   1. Get a report of the chatroom from the server.
   2. Request to join the chatroom.
   3. Quit the program.
6. Assume the user selects the first option: “1. Get a report of the chatroom from the server.”
   a. Then the client program sends a report request message to the sever where REPORT_REQUEST_FLAG=1.
   b. Then the server responds with a report response message where REPORT_RESPONSE_FLAG=1. Its NUMBER=X, which is the number of active users, which is in range [0-3]. Its PAYLOAD is a list of active users’ usernames, IP addresses, and port numbers at the time they joined the chatroom. If there is no active user, the PAYLOAD is empty.
   c. Then, the client program prints this response on the screen. For example, if there are 2 active users, the client prints something like the following. After printing this message, the client prints the menu again.

There are 2 active users in the chatroom.
1. Cool at IP: 192.168.1.5 and port: 3546543.
2. Fun at IP: 192.168.1.6 and port: 726513.

7. Assume the user selects the second option: "2. Request to join the chatroom."
    a. Then the client program prints a message on the screen that "Please enter a username:". Then the user enters a username. Then the client sends a join request message to the server where JOIN_REQUEST_FLAG=1. Its USERNAME=XYZ, which is the username that the user has entered.
    b. When the server receives this request, it first checks whether 3 active users are in the chatroom. If yes, the server responds with a join reject message where JOIN_REJECT_FLAG=1. Its PAYLOAD is "The server rejects the join request. The chatroom has reached its maximum capacity." Then the client prints this message and the menu on the screen.
    c. If less than 3 active users are in the chatroom, the server checks whether any active user is using this username. If yes, the server responds with a join reject message where JOIN_REJECT_FLAG=1. Its PAYLOAD is "The server rejects the join request. Another user is using this username." Then the client prints this message and the menu on the screen.
    d. If the username is not being used by any other user, the server accepts the join request, adds this user to the list of active users, increments the number of active users by 1, and responds the client with a join accept message, where JOIN_ACCEPT_FLAG=1, USERNAME=XYZ, the username that the user has entered, PAYLOAD contains the history of the chatroom. This history is a list of all text messages and the contents of the attachment files that the users have sent to the chatroom, the username who has sent each message, and each message's timestamp when the message has been sent by the sender user. It also includes the server's announcements that a user has joined or left the chatroom.
    e. Notice that the server should store all chatroom messages with the username who has entered that message and the message's timestamp. For the announcements, the user is the server who sends the message.
    f. Then, the client welcomes the new user and prints the history of messages on the screen.
    g. Then the server sends a new message to all clients to announce that a new user just joined the chatroom. In this message NEW_USER_FLAG=1, USERNAME= XYZ, the username of the user who just joined, and PAYLOAD= "XYZ joined the chatroom." This message will be stored in the history of the chatroom on the server. All clients will display this message in their chatrooms.
    h. Now the user can start sending messages to the chatroom, and when other users send new messages to the chatroom, this user can see those messages on its screen in this format "[timestamp] username: text message" for each new message.
8. During the time a user is in the chatroom, there are three options available for the user.
    a. The user can type a message and press enter. The client will send this message to the server. The server first checks whether this user is in the list of active users. If

yes, the server will store the message in the history of the chatroom, and forwards it to all active users, and the clients will print this message on their screens.

b. The user can type lowercase 'q' and press enter to quit the chatroom. Then, the client sends a quit request message to the server where QUIT_REQUEST_FLAG=1. Then, the server removes the user from the list of active users, decrements the number of users by 1, and sends a quit accept message to all clients where QUIT_ACCEPT_FLAG=1, USERNAME=XYZ, the user who just quitted, and PAYLOAD="XYZ left the chatroom." Then all clients receive this message and print the payload on their screens. This message will be stored in the history of the chatroom on the server. All clients will display this message in their chatrooms. Then, the quitted client closes its connection with the server and prints the menu on its screen.

c. The user can type lowercase 'a' and press enter to upload an attachment file to the chatroom. For simplicity, we assume the files are .txt files and each client stores them in its "attachments" folder in the folder where it stores client.py file. Then the client prints "Please enter the file path and name:" Then the user enters that. For example, if the user wants to upload coolfile1.txt, she will enter "attachments/coolfile1.txt", which means "coolfile1.txt" file in "attachments" folder. Then, the client takes this file and sends it to the server. Then, the server will receive it and stores it in the server's "downloads" folder. Then, the server forwards the file to all clients, and the clients will store the file in their "downloads" folder. Then, the clients print the content of the file on the screen as well. When you test your program, you should take snapshots of the "download" folder before and after the file is sent to the client to show that initially the file does not exist in the client's "download" folder and after the file is sent by the server, it shows up in the client's "download" folder. Notice that after this process, all active users are still in the chatroom and can keep chatting. For this step, you can use the Python OS module or other related modules. A good reference is: https://youtu.be/tJxcKyFMTGo

9. 10 points extra credit: The attachments are expected to be images. The users can upload an image from their machine to the chatroom and other users will see the image in the chatroom and the image will also be downloaded to their machines. To implement this feature, you need to implement a GUI for your chatroom project. One suggestion is to use Python tkinter module to create a GUI. Notice that if you only create a GUI for the chatroom but do not display the uploaded images in the GUI, you will not get the extra credit. In this GUI, the chatroom has a text input to enter the message, a text input to enter or search the filename, and buttons to post the texts and files and quit the chatroom. Also, enter a profile avatar for each user in the chatroom. For example, enter a cool smiley avatar for each message that user "Cool" sends to the chatroom. The users upload their avatars at the time they request to join the chatroom.
10. An example of running the client for a user named "Fire" who is the third active user in the chatroom is as following. The highlights are to better understand the interactions between the user and the program. Your program is not expected to be highlighted. The yellow highlighted texts are the ones that the user who works with this terminal types.

Please select one of the following options:

1. Get a report of the chatroom from the server.
2. Request to join the chatroom.
3. Quit the program.

Your choice: 2
Please enter a username: Fun
The server rejects the join request. Another user is using this username.
Please select one of the following options:

1. Get a report of the chatroom from the server.
2. Request to join the chatroom.
3. Quit the program.

Your choice: 2
Please enter a username: Fire
The server welcomes you to the chatroom.
Type lowercase 'q' and press enter at any time to quit the chatroom.
Type lowercase 'a' and press enter at any time to upload an attachment to the chatroom.
Here is a history of the chatroom.
[20:32:40] Server: Cool joined the chatroom.
[20:32:45] Cool: Hi.
[20:33:14] Server: Fun joined the chatroom.
[20:33:25] Fun: Hi Cool, are you there?
[20:33:40] Cool: Hey Fun, glad that you joined. We can chat now. *This is the chat history.*
[20:33:55] Fun: Let's wait for Fire. He wanted to join.
[20:34:00] Cool: Ok
[20:36:06] Server: Fire joined the chatroom.
Hey guys! I just joined. *This is what user Fire types in the terminal.*
[20:36:22] Fire: Hey guys! I just joined.
[20:36:28] Cool: What's up Fire? *This is the chat after user Fire joins.*
[20:36:36] Fun: Hey Fire.
Guys, I want to attach a cool file!
[20:36:52] Fire: Guys, I want to attach a cool file!
a *User Fire types "a" in the terminal intending to upload a file, and then enters its path.*
Please enter the file path and name: attachments/coolfile1.txt
[20:37:55] Fire: I am on a light diet: I eat in daylight, I eat in moonlight, and sometimes, I eat in refrigerator light! *This is the content of coolfile1.txt file that user Fire sent.*
[20:38:16] Fun: Hahaha *Note: user Fire sends the file, not the text of the file.*
[20:38:23] Cool: That was cool! :)
Alright guys. I should leave now. Buy.
[20:38:42] Fire: Alright guys. I should leave now. Buy.
q *User Fire types "q" in the terminal intending to quit the chatroom.*
[20:38:48] Server: Fire left the chatroom. *Note: user Fire will not see this message.*
Please select one of the following options:

1. Get a report of the chatroom from the server.

2. Request to join the chatroom.
3. Quit the program.

Your choice:

At this point, if this user selects option 2 and enters username “Fire” again, the server will accept it because “Fire” has already quitted the chatroom and the new “Fire” is not being used in the chatroom anymore.

7. Notes to test the program

Run the following steps and **take a snapshot of your server and clients terminals at each test step**. Zoom in before you take the snapshots to make them more readable. Make a Report.docx file and enter the snapshots of each step to the report file. The program should run on Python3.

Before testing the program, at the folder where your programs are stored, create two folders named “downloads” and “attachments” and one file named coolfile1.txt in “attachments”. This file contains this text: “I am on a light diet: I eat in daylight, I eat in moonlight, and sometimes, I eat in refrigerator light!”.

8. Test your program as following:

1. Run the server.py program. Use port 18000 at the server. Remember that your snapshots of the server should always show a brief report of what the server does at each step.
2. Run the client.py program. The client program displays the menu. The user selects option 2 and enters username “Cool”. Cool enters one message “Hi” to the chatroom.
3. Run the client.py program again. The client program displays the menu. The user selects option 2 and enters username “Fun”. Fun will get a history of the chat and observes that Cool has entered “Hi” to the chatroom. Then, Fun enters “Hi Cool, are you there?” They both see this message in the chatroom. Then, Cool enters “Hey Fun, glad that you joined. We can chat now.” They both see this message in the chatroom. Then, Fun enters “Let’s wait for Fire. He wanted to join.” They both see this message in the chatroom. Then, Cool enters “Ok”.
4. Run the client.py program again. The client program displays the menu. The user selects option 2 and enters username “Fun”. The server detects that Fun is being used by another user and rejects the request. The reject message will show up for the user. Then the client prints the menu again. The user selects option 2 and enters username “Fire”. Fire will get a history of the chat and observes the chats between Cool and Fun.
5. Run the client.py program again. The client program displays the menu. The user selects option 2 and enters username “Sad”. The server detects that 3 active users are in the chatroom and rejects the request. The reject message will show up for the user. Then the client prints the menu again. The user selects option 1. The server sends a report of the 3 active users, their usernames, IP addresses, and port numbers to this client. The report will show up for the user. Then the menu will show up. The user selects option 3 and the program exits.

6. Then, Fire enters “Hey guys! I just joined.” Then Cool enters “What’s up Fire?”. Then, Fun enters “Hey Fire.” Then Fire enters “Guys, I want to attach a cool file!” Then Fire enters lowercase “a”. The client displays a message “Please enter the file path and name:” Fire enters “attachments/coolfile1.txt”. Then Fire will upload the contents of the file to the server. The server stores the file in its “downloads” folder and forwards it to all clients. All clients store the file in their “downloads” folder and display the contents of the file on the screen. Your snapshot should include the clients’ folders to show that they downloaded the file. Then all clients display the contents of the file in the chatroom. Extra credit: Instead of coolfile1.txt, upload coolfile1.png or any other image extension. Then display the image in the chatroom. Also, each message in the chatroom should include the user’s profile avatar.
7. Then, Fun enters “Hahaha”. Then Cool enters “That was cool! :)” Then Fire enters “Alright guys. I should leave now. Buy.” Then Fire enters lowercase “q”. The client sends a request to quit message to the server. The server removes Fire from the active users and sends a message to everyone that “Fire left the chatroom.” Everyone will see this message. Then, the quitted Fire client will close its connection to the server.
8. Then Cool enters “Buy Fire.” Fire does not see this message. Then, the quitted Fire client program displays the menu. The user selects option 2 and enters username “Fire”. The server accepts this request because at this moment Fire is not an active user in the chatroom. The new Fire will get a history of the chatroom, containing all messages including the last “Buy Fire” message from Cool and the messages from the server indicating that users joined and left the chatroom.

9. Submission

Submit a zip file containing the following files:

1. server.py
2. client.py
3. Report.docx

The Report file should include:

1. The team members names.
2. The steps of the test and snapshots of each step of the test.

Put all the code files in a folder named Project1_YourTeamMembers and compress it into a .zip file. Submit your report and zip files. Make sure your report file contains at least 8 snapshots, one snapshot of both server and clients for each step. **It is important to follow the test instruction given above. Otherwise, you will miss points.**