"Network Applications and Design" Homework Assignment #5

"File Transfer in Chatting App." (5+2 points)

Due date: June 10th (Wed) 2020, 11:59 PM.
- Submit softcopy on the server.

In this homework assignment, you will implement and add a '**file transfer**' feature to the simple server-client chatting application that you have already implemented for your previous assignment. All the requirements from the previous assignment still stays (e.g. there can be multiple connected clients, and everyone can chat with everyone else). In addition, any client should be able to send a file to everyone else.

Here are the descriptions and requirements on what you'll need to do.

You must implement two programs, FileChatTCPClient.py for the client, and FileChatTCPServer.py for the server. Your client-server program pair provides 'file transfer' functionality in addition to 'chatting' functionality.

All requirements related to the chatting feature are same as the previous 'chatting' assignment. Below is a brief summary for your convenience but refer to the previous assignment for details.

- Server accepts connections from any number of clients.
- The client can set its nickname(username) when it starts using a command line argument. However, if the server detects duplicate nickname, it should refuse the connection with a message "*duplicate nickname. cannot connect*".
- When a client connects to the server, the server will send back '*welcome*' message to the client, and the client should print this message on the screen. (see previous assignment for details)
- Everyone can chat with everyone: When you type a chat message and press `Enter`, that message should be delivered to everyone (every client) connected, except yourself of course.
- On the client, while running, if the user presses '**`Ctrl-C`**', the client should send an exit message to the server before quitting so that the server knows that you're gone. (see previous assignment for details)

All '**Chat room commands**' from the previous assignment must be supported. In addition, you need to support two more commands for 'file transfer';

- **`\fsend <filename>`** // send <filename> to everyone
- **`\wsend <filename> <nickname>`** // send <filename> to <nickname> only

"**`\fsend <filename>`**" command sends a file <filename> to all connected clients.

- Of course, a file with <filename> must exist in the current directory of the sender. If not, print out an error message "*file does not exist*" to the sender.
- When the sender sends a file, a message "*<nickname> is sending file <filename>*" must be printed out as a chat message to all connected clients.
- All other connected clients should receive this file, excluding the sender.
- However, we will need to change the receiving filename because (in our case) two clients might be on the same server, same directory. So, the receiving filename must be
  - '*fsend_<sender_nickname>_<receiver nickname>_<filename>*'.
  - For example, if ironman sent 'hello.txt' and if captain-marvel received it, the receiving filename will be '*fsend_ironman_captain-marvel_hello.txt*'.
- If the same filename exists in the receiver's current directory, you may overwrite that file.

    - [OPTIONAL]: Instead of overwriting, you may put a sequence number (e.g. (1), (2), etc.) at the end of the <receiving filename> to distinguish the duplicate filenames. This would be great! However, this is optional, and not mandatory.
- When each client receives a file, a message "*file <receiving filename> received from <nickname>*" should be printed out. Note that this message will be different per client due to <receiving filename>.

"**`\wsend <filename> <nickname>`**" command sends a file <filename> to client <nickname> only.

- Of course, a file with <filename> must exist in the current directory of the sender. If not, print out an error message "*file does not exist*" to the sender.
- Of course, a client with <nickname> must exist in the chat room. If not, print out an error message "*<nickname> does not exist*" to the sender.
- When a sender sends a file, a message "*<sender nickname> is sending file <filename> to <nickname>*" must be printed as a chat message ONLY to the client with <nickname>. Only <nickname> should receive this file.
- However, we will need to change the receiving filename because (in our case) two clients might be on the same server, same directory. So, the receiving filename must be
    - '*wsend_<sender_nickname>_<receiver nickname>_<filename>*'.
    - For example, if ironman sent 'hello.txt' to captain-marvel received it, the receiving filename will be '*wsend_ironman_captain-marvel_hello.txt*'.
- If the same filename exists in the receiver's current directory, you may overwrite that file.
    - [OPTIONAL]: (same as \fsend)
- When each client receives a file, a message "*file <receiving filename> received from <nickname>*" should be printed out to the receiving client <nickname>.

The 'file transfer' feature that you implement should (ideally) support files of any sizes. However, just to limit the memory/storage usage and help you a little, let's limit the maximum file size to 5MB. That is, you will get points as long as you can send files that are 5MB or smaller. If you can support something bigger, that would be great, but not mandatory.

A few important things to think about:

- When sending commands from the client to the server, should you use a string (e.g. "\fsend", "\list") to represent a command? Or should you encode that into binary information (e.g. 10 for "\fsend", 11 for "\wsend" and so on)? What are the advantages and disadvantages of these? (string vs. binary)
- For each error message, who will generate the messages? Server or the client?
- When sending a large file, how will you fragment the file? At what sizes?
- How will you encode each file fragments? What header will you put on each file fragment? How many bits will you use for the sequence number? What if some part of the file is missing at the receiver?

In this homework assignment, a lot of this is up to you; you have quite a bit of freedom to implement it in your own way. However, please think about the advantages and disadvantages of each design choice carefully.

Additional note and requirement regarding command line argument:

Ideally, it would be nice if both your programs, FileChatTCPClient.py take IP address and port number as command line arguments so that we can run them on any machine and any port without modifying the code. However, if I do that, it would be very difficult to grade your submissions. For this reason, (although it is not nice), you should hardcode the 'server port number' in both programs using your personal designated port number. Also, you should hardcode the server address (nsl2.cau.ac.kr) in your client program. For your client sockets, you must not assign a port number or use null (0) port number which will let the operating system assign a random port number to your client's socket. The only command line argument is the 'nickname' for the client program. That is, the client should run using the command "**`$ python3 FileChatTCPClient.py <nickname>`**", and the server should run using the command "**`$ python3 FileChatTCPServer.py`**".

You should not use any external libraries, commands (e.g. scp, ftp) nor system calls to do this assignment. File transfer must work within your own program, by sending messages (possibly multiple) through your own socket.

Other additional requirements:

- You must implement and run using Python 3. Do not use Python 2.x
- Your program must run on our class Linux server at nsl2.cau.ac.kr.
- Running your client on nsl5.cau.ac.kr and your server on nsl2.cau.ac.kr should work.
  - ✓ Both should work without ANY code modification.
- Make sure that you use your own designated port number for the server socket.
- **For the client socket, you should either not set the port number, or must use null (0) port number which will let the OS assign a random port number to your socket.**
- Your code must include your name and student ID at the beginning of the code as a comment.
- Your code should be easily readable and include sufficient comments for easy understanding.
- Your code must be properly indented. Bad indentation/formatting will result in score deduction.
- Your code should not include any Korean characters, not even your names. Write your name in English.

What and how to submit

- You must submit softcopy of FileChatTCPClient.py and FileChatTCPServer.py programs.
- Here is the instruction on how to submit the softcopy files:
  - ✓ Login to your server account at nsl2.cau.ac.kr.
  - ✓ In your home directory, create a directory "submit_net/submit_<student ID>_hw5"
    (ex> "/student/20149999/submit_net/submit_20149999_hw5")
    (do "pwd" to check your directory)
  - ✓ Put your "FileChatTCPClient.py" and "FileChatTCPServer.py" files in that directory. Do not put any code that does not work! You will get negative points for that.

Grading criteria:

- You get 5 points
  - ✓ if all your programs work correctly, AND if you meet all above requirements, AND
  - ✓ if your code handles all the exceptional cases that might occur, AND
  - ✓ if your code is concise, clear, well-formatted, and good looking.
- Otherwise, partial deduction may apply.
- You may get optional extra credit of up to 2 point if you do the optional extra credit task.
- No delayed submissions are accepted.
- Copying other student's work will result in negative points.
- Code that does not compile or code that does not run will result in negative points.

[Optional] Extra credit task: (up to 2 point)

- Do the same thing as above also in **C**.
  - ✓ For submission, your file names should be same as Python counterpart except the file extension (.py → .c) Your C programs should compile with gcc.
  - ✓ You should provide a Makefile that can compile both your server and client.
- If you do this, not only your C client should work with C server, but your Python programs should also work with your C programs. That is, you should be able to mix and match C and Python programs for server and client, in any combination. Do not submit if it does not work. You will get negative points for that.