Final report

DB928120 Zhang Jiajin

1. Instruction

The process follows one specifical rule, which is to run the server before running the clients. Otherwise, it will cause errors like failure to connect or refuse to connect. In detail, we need to run server.py in one complier and client.py in another one. If you want to add more users, we may need software that could deal with multiple processes. And the test environment is Windows 10.

1. Explanation

There we design the project into two parts, one is file transmit, the other is a multiuser chatroom. To run the code on one single computer, I decide to use two compilers, one is Visual Studio Code, the other is PyCharm. In most cases, I will run the server file on PyCharm because it will show the running state with the sentence “Process finished with exit code …”, so I can justify the result easily. On the other hand, I will run the client on Visual Studio Code. For file transmit, client.py and server.py are created separately. When running the code, I suggest running server.py first and running the client.py later. After launching the server, “Waiting for clients to connect...” will show in the terminal of the server.py. On the terminal of client.py, we are required to “Please enter the filename you want to send:” to type in the file name and format you want to transmit. Notice that the file you want to transmit should be in the directory of the client.py if you don’t want to type in the detailed path of the file. When the server accepts the connection, the IP address and PORT will be on the terminal and the process of receiving the data will be told. Once the transmit process is done, the server will close automatically. We use TCP to design our file transmit project, In server.py, we use the socket() function to create a socket, bind the address (host, port) to the socket, under AF\_INET, the address is expressed in the form of a tuple (host, port). Then we use listen() to start TCP listening. Then the server passively accepts TCP client connections, (blocking) waiting for the connection to arrive. We use recv() to receive TCP messages. Then we read the file path and rewrite it under the path of client.py. If the writing process is done, the socket closes and ends the process. In client.py, we use os.path.split() to separate the file name and path and use os.path.getsize() to find its size. Then we set our IP address and PORT. It uses open() to open a certain file and use a socket.socket() like in the server to create a socket. The next issue that should be considered is that we need to build a stable connection between the server and the client. So, we directly use connect to build it. When connected, we read the file and send it to the server. After the transmitting, the socket will be closed and counted the time used during the process. For chatroom, the server should run first, then the clients are free to join the server. The first thing we need to do after joining the chatroom is type in your name to make distinctions between different users. On the server’s terminal, we can find detailed chat content history and how many clients are connected to the server and their IP address and PORT. For clients, we can find who connected to the server and what he/she typed in. All clients will disconnect if the server is closed. In server.py, I use TCP/IP to build a socket. In the server.py, I used modular programming techniques to make my program look more concise. In server.py, I created main() to handle basic text output, newclient() to handle the response of a new client joining, broadcasting() to send information, and removeClient() to handle user interrupts. At the same time, everything becomes clear in client.py, I use receive() and send() to deal with messages sending and receiving problems. In addition, we consider the failure possibility in both sections, so we use except: to handle this. Moreover, there is an utility.py in the folder which is used to encode() and decode() messages we send.

1. Comments and Discussion

This project successfully realizes the purpose of multiple user chat and file transmitting processes. After the test of three users, our chat room can send and receive messages smoothly, and there is no unexpected situation, which makes me very pleased. And during file transfer, most (9/10) files have been transferred.

1. Limitations

However, in the file transfer part, when the file content is in Chinese, there may be the possibility of file transfer failure or garbled characters. The problem when transferring Chinese is more serious. Not only will the transfer process not stop, but the source file after the transfer is forcibly stopped. It will become garbled, which makes me very confused, I have not been able to solve this problem.

Note: I and my partner Wang Boyu(DB928232) did the coding process together. I did the server.py in file transfer and client.py and utility.py in chatroom. And he did the rest part.