COMP3331 21T1 Assignment Report

Author: Demiao Chen
zid: z5289988
Programming language: Python3, version 3.7

Overview

The program is consisting of two files: server.py and client.py.
The program is a simple chat server can hold multiple clients. Clients can login the server by username and password. After login, clients can view, send, delete and edit message in the server, also are able to check active client list in the server. Clients are able to send each other files via P2P UDP protocol.

Start Up

To use the program, firstly run server.py by following command in terminal:
$ python3 server.py <server_port> <number_of_consecutive_failed_attempts>
server port specify the port number the server wish to be running.
number_of_consecutive_failed_attempts specify the maximum attempts times for user to enter incorrect password.

After running server, multiple clients can be created by command in terminal:
$ python3 client.py <server_IP> <server_port> <client_udp_server_port>
server_ip is the server's IP address. By default, the server is running on localhost 127.0.0.1, user of the program can edit server's IP address in server.py at the top to be able to running on a public IP address.
server_port is the port number server running.
client_udp_server_port is the port number which the client will wait for the UDP traffic from the other clients.

If number of parameters is not correct when running the program, an input format will prompt to terminal.

Design

server.py

After server is initialised, server will be listening to client's connections on a TCP socket, then read client's username and password from credentials.txt, which must in the same directory with server.py. Each user in credentials.txt will be stored as a dict in python, and having the following format:
user = {
'username': username,
'password': password,
'failed_attempts': 0,
'login_timestamp': 0,
'last_block_timestamp': 0,
}
Keys other than username and password initialised as zero. Each user dict then will be append a global variable user_list. Server is able to connect with multiple clients by using threading.

The program use special message format send between client and server. Each message contains type part and body part. Type part send from client to server is the command issued, from server to client inform some actions in the client side. Body part is the main part, e.g. it is the message content send when client issuing MSG command.

After a client is connected to the server, server allocate a new thread for the client to receive the client's responses. Server will be waiting for client to input username and password. Server then compare the client's input username and password with user_list's user information. Four different type of messages type will be sent to user: login successful, username does not exist, password wrong, block username. Server will then update user's login_timestamp information in user dict, and add login information in userlog.txt. Updating login_timestamp and last_block_timestamp in user dict when user input wrong password or being blocked.

Blocking period by default is 10 secs. This can be modified at the top of server.py.

After a user successfully login, the server is waiting for the user's command. There are six command available in server.
MSG for post message on server, which will update messagelog.txt
DLT messagenumber timestamp to delete a message if the user has sent the message
EDT messagenumber timestamp message to edit a message if the user has sent the message
RDM timestamp will send message that sent after timestamp to user
ATU will send the user active user list
OUT will logout the user, remove it from userlog.txt

client.py

After a client is initialised, client will establish TCP connection with server, and wait server to send a welcome message to indicate connection has been established, then client will prompt username and password wait user's input and send it to server for authentication.

After authentication successful, client will establish a UDP socket in a thread for incoming p2p transfer. After that, client will prompt message for available command and wait user's input to send to server. Client will listen to server's response and display its body part. When a response type is 'OUT', client will display the goodbye message and shut down, When a response type is 'ATU', client will update its user_list which store a list of user's username, IP address and UDP port number. This list is used to found a username's IP address and UDP port number to be able to send file to the user via UDP connection.

Users are only able to choose a user to transfer file after issuing ATU command to server to obtain active users information, and only be able to transfer file to an active online user.

UPD username filename is used to transfer a file specified by filename to a user specified by username. A user cannot receive multiply p2p transfer at the same time. Receiving p2p file will has the prefix of the sender name in file name.

Limitations

- User cannot register a new user to credentials.txt via client.py or server.py, all username and password can only be added by editing credentials.txt.
- username is the only identifier of a user, duplicated usernames are not allowed.
- Blocking login for a username applies globally, attempts from all IP address will be blocked once a username is blocked. So a username cannot login if someone is deliberately blocking the username by guessing the wrong password of the username.
- A username is able to login multiple times at the same time. If the same username login concurrently, userlog.txt use appear the same amount of login times of the username.
- A user exit unexpectedly (not via OUT command) will not be able to update userlog.txt information.
- UDP receiver have to wait at least for 2 seconds for a confirmation that file is received. This is for the tradeoff in the scenario where datagram has lag more two seconds.
- Clients cannot receive more than one p2p file transfer at the same time, this will cause all files received corrupt due to no threading made for each p2p client. This is the tradeoff for the simplicity of the code, as it assumes user will only transfer small file via UDP because of the unreliability of UDP protocol, small files can be transferred very fast so threading is not made.

Improvements

- User register can be added. This can be implemented by adding a new command for client. By issuing the command, server will write new username and password to credentials.txt.
- Users can be able to reset their password via command. This can be implemented by adding new reset command and overwriting credentials.txt.
- Userscan be able to receive multiple p2p transfer at the same time. This can be implemented by using threading at UDP server side. In the UDP client side, sender can firstly send special message indicates this is a new connection, so UDP server will make a new thread for the connection and receive file from.
- User can be able send DM to an active user. This can be implemented by adding improving application layer message on UDP p2p transfer, making it not only be able transfer files, but message display as well.

- Automatically obtain active user information when a user login. This can make an intending p2p transfer faster, not need send a ATU command first. This can be implemented by adding code to issue ATU command once a user login

.txt File Format

credentials.txt format

username password
jess 123
derek 1211

userlog.txt format

1; 25 Apr 2021 05:26:30; jess; 197.83.62.1; 9999
2; 25 Apr 2021 05:26:32; steve; 123.0.1.22; 7777

messagelog.txt format

1; 25 Apr 2021 05:33:41; jess; i want to leave; no
2; 25 Apr 2021 05:33:46; steve; not valid tho; yes
3; 25 Apr 2021 05:33:48; jess; lol; no

userlog.txt and messagelog.txt will be cleaned up when restart server.py