Python version 3.7

Program Design

- Socket specification
    - Both Client and Server Use the design strategy: Blocking + Multiple Threads
    - Every new accepted connection would create corresponding socket thread
- Server Function
    - User Authentication
        - Try to match the username/password in local file with user input
        - Offline message check after every successful user login attempt
        - Login attempt's number check after every login fail
    - User Blocking
        - After user attempt more than 3 times, the server would create a thread which take a server specific sleep time. After that, every login attempt would check the alive of corresponding block thread
    - Timeout detection
        - Right before server start to receive the message, there's a time recording thread, this thread would send timeout message to client after server specify time.However, any receive message would trigger a thread event in that thread, which makes time recording thread quit without send timeout message
    - Parse command
        - Function would extract first keyword of receiving message and try to match the pattern. If nothing match, a Invalid Command would send to the client
    - User login and logout
        - Software observation design pattern is used in this section. Each user login/logout would trigger the update of information to all other current users
- Client Function
    - Two parallel threading
        - Client part has two parallel threads, one is for receiving and another for sending. The main reason for this design is to let the client has the ability to process the received message at the time of waiting for user input.
- P2PMessage
    - Client p2p socket
        - Except ordinary server communication socket, the client also setup extra p2p connection thread, used to monitor incoming p2p request.
    - Fetch information and share information
        - When the user want to have p2p connection with other online users, server would help user fetch ip address and port of specific user. Then, user would send tcp request and self information directly to the desired user

- P2PFileTransfer
    - The Implemented p2p file transfer work in following order
        1. When the server receive the p2p download command, sever would notify all online users to check their local directory.
        2. If user has the file, they would divide file into specific chunks and register them on the server
        3. Once server do receive any registration from users, it would send the corresponding file chunk list to the user
        4. User would randomly choose one of the chunks that haven't been downloaded before. Then, fetch the owner list of that chunk from the server.
        5. User would randomly choose one of the users. Start p2p session to that user and request to send that chunk back
        6. Once that chunk is received, it would register ownership to server. Check the file is completed or not. If not, repeat the procedure from step 4
        7. If the all chunks is gathered. User would open a file and write the binary data to it.
    - File encode
        - Due to socket send don't allow binary data. Encoding method, ISO-8859-1 is used in this case and when the whole chunks are together, they will be decoded and write to file in the form of binary

Further Improvement

- Encrypted message
    - One of the biggest security problems in this system is that all the message, no matter command or pure text are in the form of plain text. For further optimisation use use AES encryption for the socket communication
- Hash table for command matching
    - The biggest time consuming activity in server is to match the command to suitable function. In this case, lots of if statements are used here which is totally inefficient and condition would get worse as the system grow. Using hashtable might be an adapted solution for this problem
- File transfer limitation
    - Currently, the size of p2p Download is pretty limited, which is around 20k. One of the reasons is assignment specify that only allow 10 segments. For later use, the system can increase the allowed segments or chunks' number. Having robust and correct way to zip the message is a good solution as well.

Design Consideration:

- In this case. Block setting is used in this system, make the system more robust, because failures or errors occurred in one of the threads don't affect other threads, don't make the whole system crash for single error, allow system has higher fault tolerance.

  However,this kind of method do have tradeoff. It's consume much more power than single thread operation, non-blocking mode. Instead of letting some of threads keep looping infinitely, the "select" method in non-blocking mode would only response to incoming detected buffer

Application message format

Desired function	Corresponding massage format
Check current users online	whoelse
Check users from certain period	whoelsesince <time>
Message to online/offline user	message <user> <message>
Block or unlock user	block/unblock user
Broadcast	broadcast <message>
logout	logout
Start private session with online user	startprivate <user>
P2p message	private <user> <message>
Stop private session with User	stopprivate <user>
Binary file download *note* may have size limitation	p2pDownload <filename>

NOTICE
P2p download is done in this case. However, for each p2p file transformation function shown in specification might not independently work well, as they are slightly modified in order to fit with final download command:

p2pDownload <filename>

It has been tested well on small text and binary file