CS425A: Computer Networks
Server-Client Chat Application

Group-18 : Abhishek Verma(14026), Dipendra Singh(14223), Gourav Katha(14254)

November 12, 2017

1 Introduction

We have implemented a client-server chat application where clients interact with each other using *commands* sent to a server. This model can be used as a simple client-server chat application wherever there is a need of communication between two parties which includes private/group messaging.

2 Objective

Our objective was to implement a basic chat client-server model which would support a list of features like:

- Basic security features of a chat application.
- SignUp/Login/Logout facility.
- Send private as well as broadcast messages.
- Display names of all online users.
- Display names of users who are online and logged in within last hour.
- Blocking and unblocking of users.

3 Assumptions

- Maximum concurrent client is fixed depending on the capacity of the server.
- SSL will be used if there's a scope of man in the middle.
- There are no marks for a GUI.

4 Architecture

The main directory of this application contains may files and an another directory whose details are as follow:

- **server.py**: Contains server implementations and other helper functions.
- **client.py**: Contains client implementations and other helper functions.
- **server_resources**: Contains a file named "user_pass.csv" which stores username ans password. There is a directory too, named "user_data" which contains files corresponding to every user where the server can store their state and messages when they are offline.

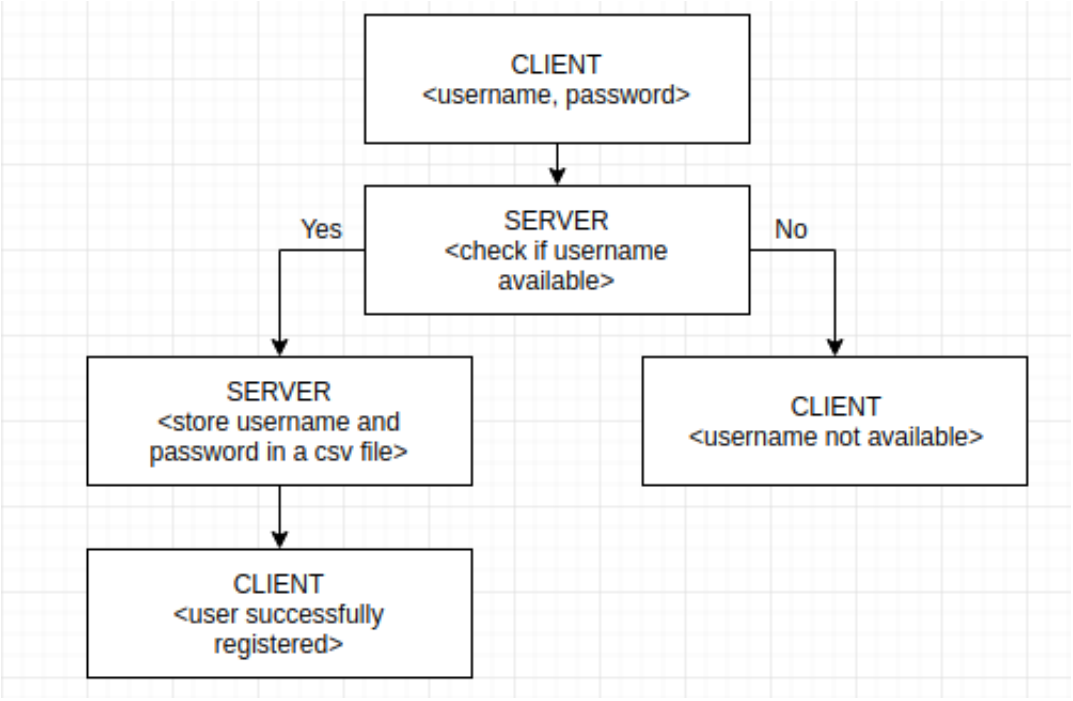

Figure 1: Signup flow

Also, we do not send plain text password over the HTTP, i.e., we send hashed password. Different components of this model are:

4.1 Server

We have implemented a preforked server. Making a concurrent server, i.e., calling 'fork' by the server for every client connection can be costly in terms of resources. So, we prefork some number of children when the server starts and use the pool of preforked processes to handle incoming connections. Here, the server does not calls 'accept' to accept a new connection, but all children are blocked in the call to 'accept' on the same listening socket passed from the parent process. The way it works is that all child processes are blocked waiting for an event on the same listening socket. When a new connection arrives all children are awakened. The first child process to run will make a call to 'accept' and the rest of the processes will be put to sleep on the same call.

4.2 Client

Client is a simple process which when starts connects the client machine to the server by using usual socket connection.

4.3 Feature implementation

We have implemented quite a few features in this application. Details about them are as follow:

- **SignUp**: Client sends a username and password to the server. Server checks if that username is available(i.e., someone else isn't already using that username). If no one is using it, server stores the username and password in a csv file and declares the user to be registered. *Please refer to figure-1.*
- **Login**: Flow diagram is self-explanatory. *Please refer to figure-2.*

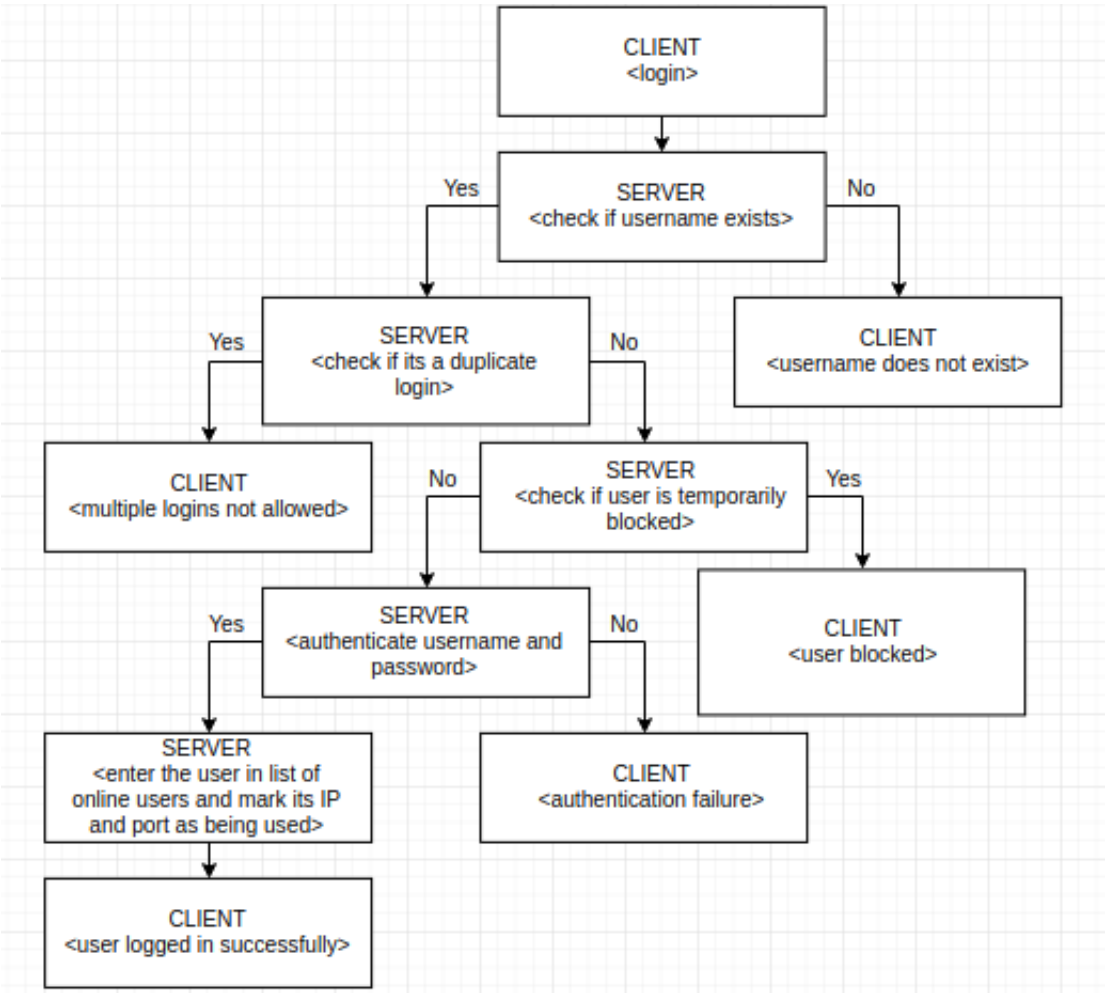

Figure 2: Login flow

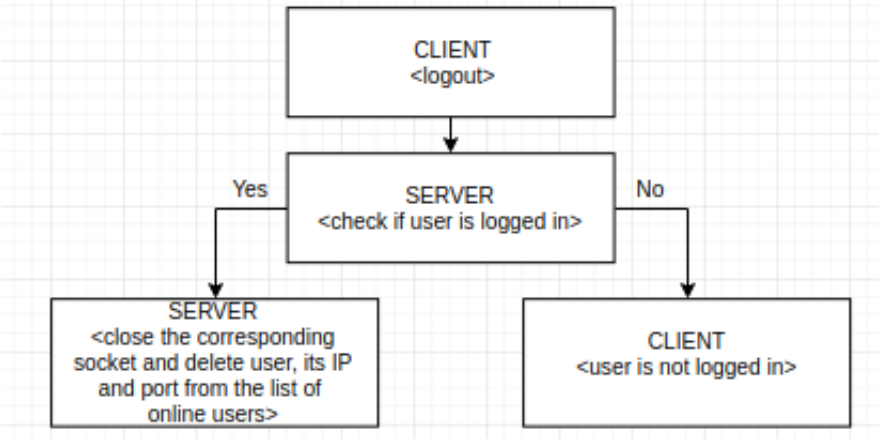

Figure 3: Logout flow

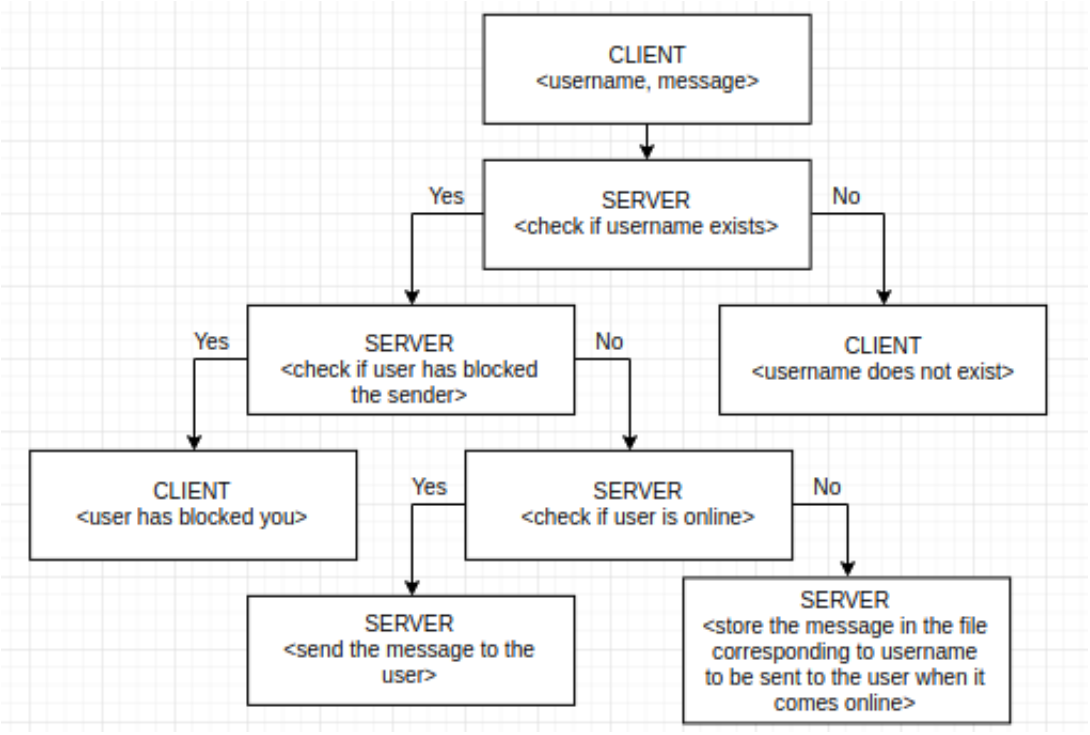

Figure 4: Private messaging flow

- **Logout**: Client sends a logout request. Server checks if user was logged in. If it was, server closes the corresponding socket and delete its IP and port from the list of online users. *Please refer to figure-3.*
- **Users currently online**: When a client requests for all such users, server simply fetches it the list of all online users.
- **Users who logged in within an hour and are currently online**: Server iterates through a list of online users and returns usernames which which have logged in within last hour.
- **Private messaging**: Flow diagram is self explanatory. *Please refer to figure-4.*
- **Broadcast messaging**: Server iterates through the list of all registered users and calls private messaging module for each of them.
- **Block a user**: Server makes a block entry for the user in the file corresponding to the client. *Please refer to figure-5.*
- **Unblock a user**: Server checks if a block entry for user exists in the file of client. If yes, it deletes that entry.

5 Implementation environment

We implemented this application on 64-bit systems with Linux OS. It is expected to work fine with other Operating Systems too. It is implemented in *python-2.7* and uses some of standard python libraries namely, *os, csv, json, errno, signal, socket, thread, optparse, base64 and datetime.* Code is available here.

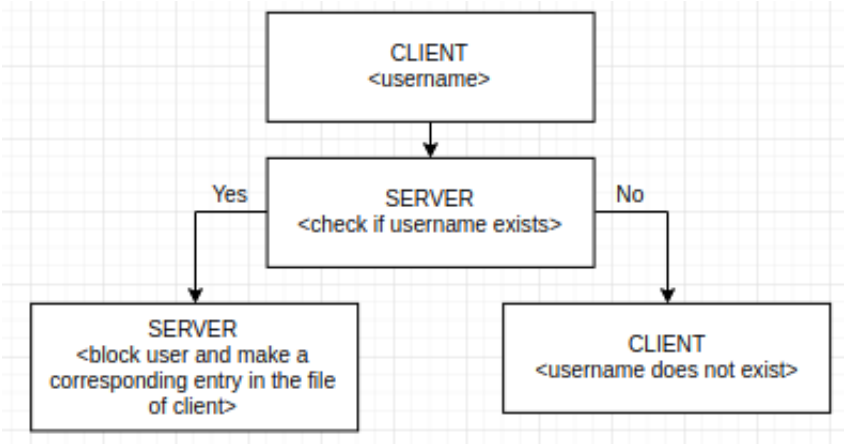

Figure 5: Block user flow

6 Instructions to run the code

On a Linux system,

- You need to have python 2.7 installed.
- For starting server: *python2 server.py –host <host_ip> –port <port_number>*. Default IP and port are *0.0.0.0* and *2000* respectively.
- For starting client: *python2 client.py –host <ip_of_server> –port <port_number_of_server>*.
- Rest is self-explanatory.

7 Summary

We have successfully completed all our objectives. In addition to that, we have implemeted some more features:

- Password is communicated over HTTP in a hashed form.
- After 3 consecutive unsuccessful login attempts, the blocks the Port of that IP for that user for 60 seconds. We don't block the whole IP as testing the application would be simpler if we block that port only.
- Prohibit simultaneous duplicate logins.
- If a user is not available, offline message are displayed when the user logs in next time, i.e., it supports asynchronous messaging too.