California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

PROJECT REPORT

Project Title:	Socket Programming Project: creating a server for client chatting
Professor:	Dr. Kulhandjian
Course Title:	ECE - 146 Computer Networks Tues. Thurs. 11:00 am - 12:15 pm
Date Submitted:	April 23, 2019

Prepared by: Hannselthill Camacho, Trien Ngo

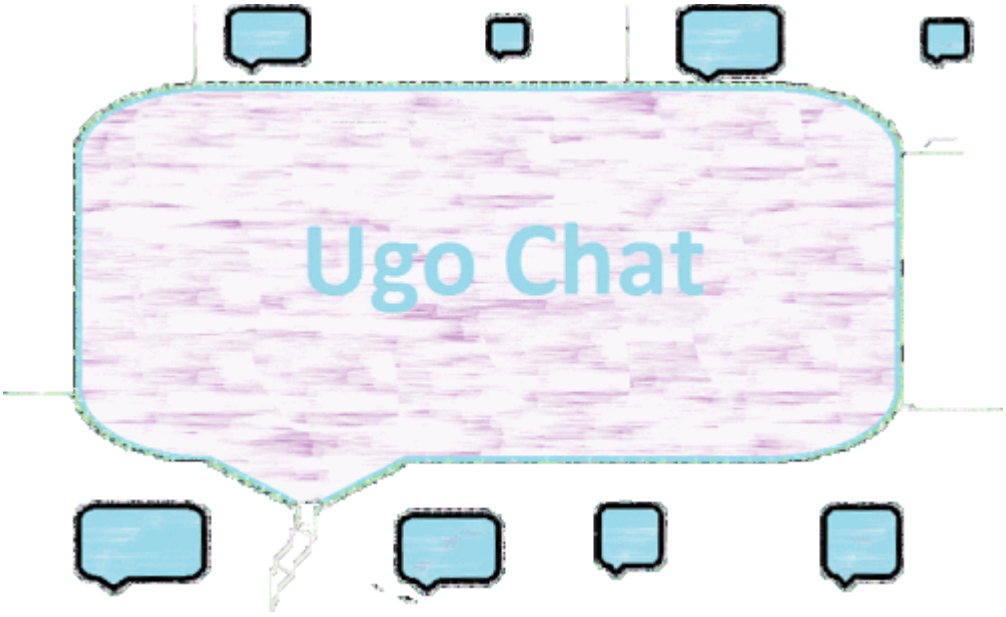

Table of Contents

Theoretical Background

- ➔ Socket Programming

  This is a way to connect two nodes, a client and a server (in this case multiple clients), on the network so that they can communicate with each other. One node/socket, the server, listens on a port on their IP, while the other socket, the client, is given information on what port and what IP to go to communicate with the other node.

- ➔ Threads

  Multithreading is an extremely important application within programming, it works just like multiprocessing in that it is a way to achieve multitasking. With the main difference being how memory is used. Multiprocessing uses separate memory locations/processors, whereas threads use the same memory/processor.

- ➔ GUI

  GUI stands for Graphical User Interface. This allows for more user-friendly user interaction with code. Without a GUI users would have to interface with code via the terminal/command line.

- ➔ Databases

  In order to store and save user data for future use, a database is often used.

Client Side QThreads

On the client side we originally implemented the chat using python's built in threading library and Tkinter for the GUI. This was fine and worked but we were not satisfied with the lack of portability into other programming languages that Tkinter had. So we instead used Qt5's python bindings. A few excellent examples of Qt5 are their use in high end Mercedes vehicles for Dashboard GUI's and Slack is fully implemented in Qt5 as well. This brought its own set of problems.

Mixing PyQt GUIs with python threads can cause locking issues. As such PyQt has their own thread library called QThreads that can run alongside the GUI. QThread class objects have to be initialized like any other class objects. This means you can also pass in variables to offset the work from the main GUI thread. In Ugo Chat we have three different QThreads running on the client side.

The first thread is the main thread. This thread runs for the entirety of the application and handles the showing of GUI objects along with handling events. The second thread is used to receive messages in parallel to sending them through the GUI. The final thread is friendSync which receives a signal to run two commands. One command is to update the online status of a friend and the other command is to add a new friend to your list. Both these commands are triggered by the receiverThread.

Server Side Python Threads

The server takes advantage of the built in python threading library. Since the server only needed to serve as a middleman between clients, no GUI was created. The server required the use of two threads in order to guarantee smooth operation. One thread is in charge of listening in on incoming connections. The receiving and sending of messages for each client in parallel.

Programs/Materials We Used

- **PyCharm**
  - This is the IDE we used, our code has not been tested on other IDEs, but others should work the same
- **DB for SQLite**
  - Only needed if you want to access the databases, not needed if you just want to run the client and server code
- **Computer OS**
  - This code has only been tested and ran on Windows OS and Ubuntu Linux
- **Qt Designer**
  - For our GUI we used PyQt, Qt Designer allows for the easy creation of GUI objects and an extensive library for styling and running multiple threads also known as QThreads in PyQt
- **Home Wifi**
  - server.py should be run on personal wifi, not an institutions (like Fresno State's wifi), this is because on your own personal router you can properly enable port forwarding which is needed for clients to reach the server and this can not be properly done, at least to our knowledge, on Fresno State's wifi without IT permissions
- **Adobe Photoshop**
  - Due to time restrictions the implementation of a GIF to serve as a loading animation after a successful login was never implemented. Adobe Photoshop was used to manipulate the logo created using Microsoft paint. The standalone GIF is available on Github for viewing purposes.
- **Microsoft Paint**
  - Microsoft paint which is built into Windows OS was used to create the ugoChat logo.
- **Pyinstaller**
  - Pyinstaller allows a programmer to quickly create a .exe from python source code. The file can be all in one or can be made into a pseudo installation directory with all the individual modules and images present.
- **Multiple Machines**

- ➢ You can run multiple instances of a client on one machine, but for in real life applications multiple machines should be used

File Break Down

- ➔ **Client Files**
  - ❑ **ClientSideGui.spec**
    - Computer generated a file with the file path, python version, etc...
  - ❑ **ClientSideGui.py**
    - This file acts as a central hub for pulling all other client files together
    - PyQt5 QThreads
    - Handles client-side socket interaction
      - Connecting
      - Sending/Receiving messages
  - ❑ **loginWidget.py**
    - First Gui encountered after TCP handshake is successfully performed
    - Client interface to send login credentials to the server for verification
    - Allows user to call the registration window to register an account
  - ❑ **userNotFound.py**
    - Generic error popup message to indicate a false login or incorrect registration credentials
  - ❑ **loginSuccess.py**
    - Generic error popup message to indicate a successful login or a successful registration
  - ❑ **ugoChat.py**
    - The main application chat window
    - Ability to add friends and view friends that are online/offline
      - Online friends have a green icon
      - Offline friends have a red icon
    - Send/Receive messages
      - Sent messages are highlighted in light blue
      - Received messages are highlighted in light green
  - ❑ **registrationPage.py**
    - Allows user to be added to the servers login.db database
    - Users can return to the login window at any time
- ➔ **Server Files**
  - ❑ **Server.py**
    - The central hub for all the server side files
    - Python Threads

- Server Sockets
    - Binding
    - Listening
    - Accepting
    - Sending/Receiving messages

❑ **userDatabase.py**
- All functions for creating, accessing, and manipulating database information

❑ **Login.db**
- Database in charge of storing usernames and passwords

❑ **[friendsList].db**
- Every user gets their own friend's list database that lists their current friends
- Automatically created upon first successful login

Walk-Through

It is highly recommended to clone our GitHub repository to run our code as opposed to unzipping the attached file and running those.

Github Repo: https://github.com/vincetrien-ngo/ECE146SocketChatting

PyCharm was used as the IDE for this project. CMD line can also be used to run the code if needed. ClientSideGui.py and server.py are the only files that need to be run to use the code and all of the other files just need to be in the same directory. It is important to note that the pyqt5 library will need to be installed in order to run the ClientSideGui.py file. The server.py only uses standard python modules and can be run as is.

Running both Client and Server locally

In the **ClientSideGui.py** on line 308 make sure the host = '127.0.0.1', do the same on **server.py** on line 14, make sure the host is equal to '127.0.0.1'. An example of what your code should look like after changing is shown below.

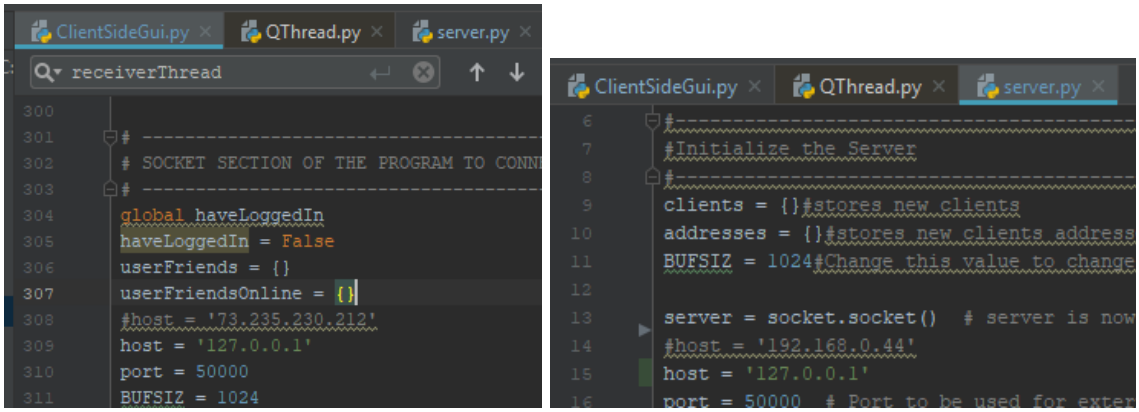

Figure 1: ClientSideGui code (left), server code (right) for local running both

Running Server locally and having Clients log into your server

In order to run over the server over the internet, in **server.py** you must port forward port 50000 for your server machine's local IP address. How to perform this varies router to router. For a Netgear router, this can be accomplished by typing routerlogin.net into your browser and logging into your router. Then go to 'advanced' and find port forwarding/port triggering

On **ClientSideGui.py** you have to insert the public IP address for the server's network. If you attempt to use the internal IP of the server machine then you will not be able to connect to the server. This is because private IP's are hidden to the outside world and two different networks can have the same private IP address allocated for a machine on their separate networks.

Server Listening on port 50000 for incoming connections

The server will continually listen to incoming connections. Python Sockets allow for up to 5 connections at maximum to attempt a handshake at the same time. This is unlikely to happen as the handshake happens rapidly.

```
C:\Users\Hannselthill Camacho\Desktop\ugoChat\ECE146SocketChatting>py server.py
Awaiting new ugo...
```

Figure 2: Server listening for incoming connections

TCP handshake complete and client GUI begins

After the handshake is initiated the server provides the network administrator with the IP address and port after storing it as a Tuple. At this point the initial **loginWidget.py** GUI menu is initiated and shown. The user has the option of filling in their credentials or clicking the REGISTER button shown in the figure below.

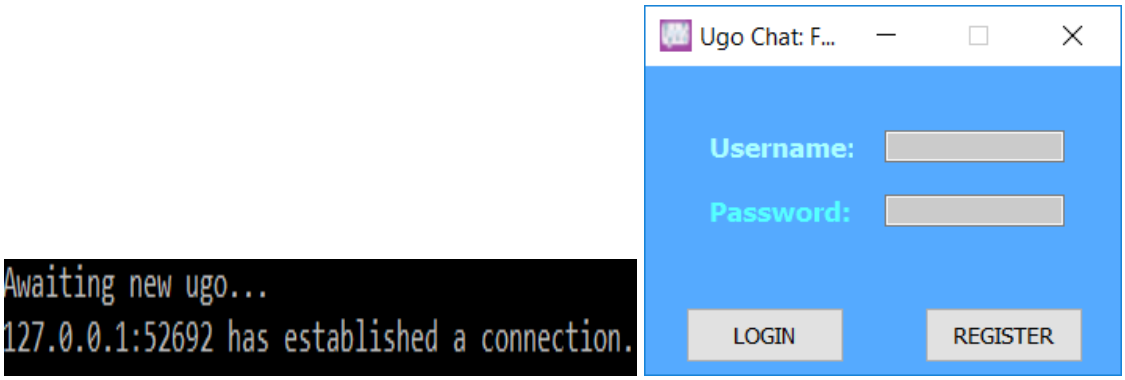

Figure 3: (left) Server terminal after TCP handshake. (right) Client login screen

If the user successfully logs in, the server's terminal will show the user who logged in. When they leave both the chat and the server's terminal will show that the user has disconnected. On the client side a login Success window popups in front of the login widget. Once the OK button is clicked or the space bar is pressed, the client opens the main chat window.

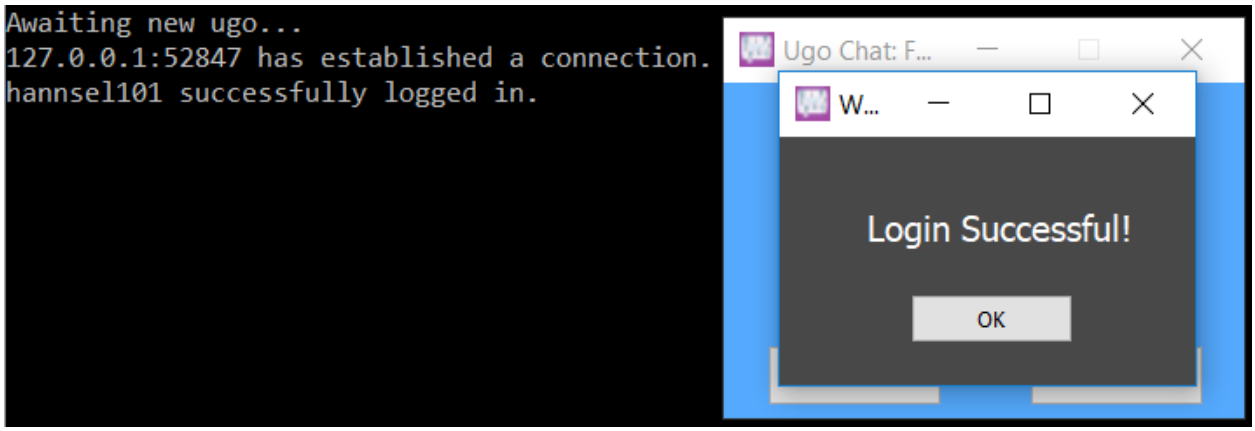

Figure 4: After a successful login

The other two possibilities at the login widget are an unsuccessful login attempt or clicking the REGISTER button to open the registration window shown below. The registration window shares a similar error message and successful registration popup windows as the login widget.

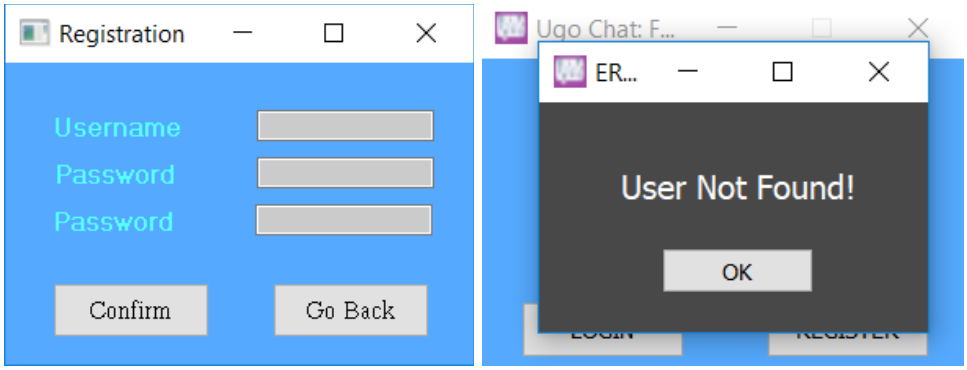

Figure 5: (left)Registration window. (right)Unsuccessful login attempt

When a user attempts to login or register the server runs a quick check through login.db for a matching username. If the username is found then it checks the password column to verify the password used. During registration the same database is used and the username as the primary key is cross referenced against the username attempting to be registered. If they match an entry in the database then an error code is returned by the server to the client.

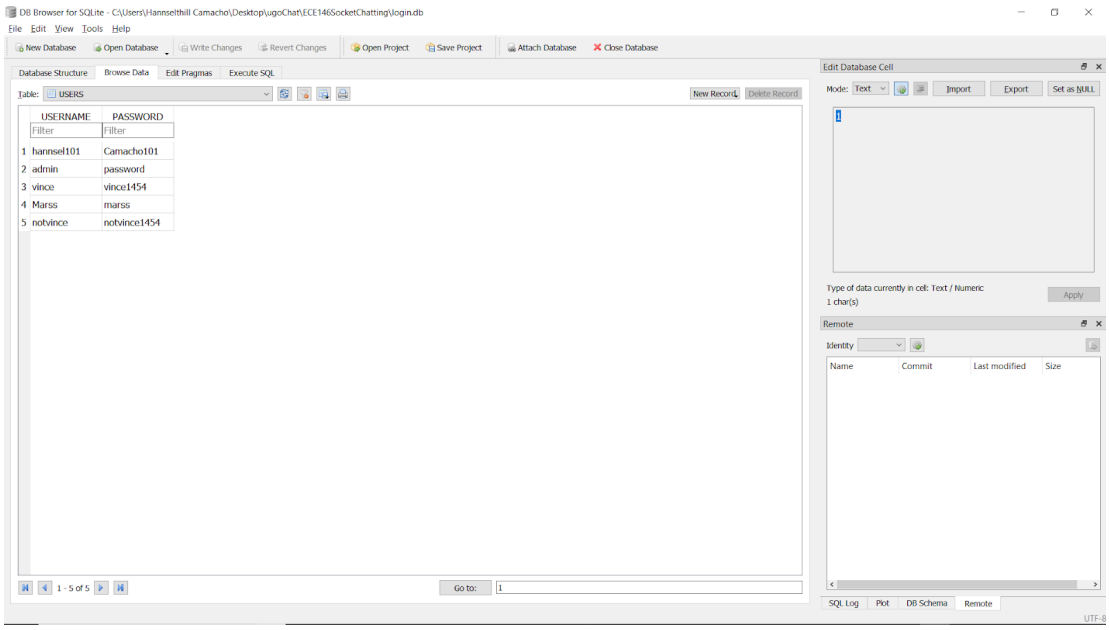

Figure 6: login database viewed through DB for SQlite software

Main Chat Window

The main chat opens up after clicking the OK button on the login success popup window shown in Figure 4 above. To send a message simply type into the bottom left text entry box and hit the “enter” key on your keyboard. Clicking the arrow to the right of the text entry also sends the message. Your sent messages are appended into the chat highlighted

in a light blue color and any received message is highlighted in light green. Every message sent or received is time stamped below the message in 24 hour format along with the month, day, and year as well.
Your friends list populates the left column in the chat and either an “online” green icon or “offline” red icon is listed next to a friends name. The bottom left portion of the chat has a text entry box for adding friends. If a user enters a name and hits the add friend button located next to the entry box one of two things will happen. Either the server will prompt the client with a “//VERIFY ADD FRIEND” message and the new friend will be appended to the left column. Or it will instead send “//CANNOT ADD FRIEND”. Nothing will happen on the client side if the friend cannot be added.

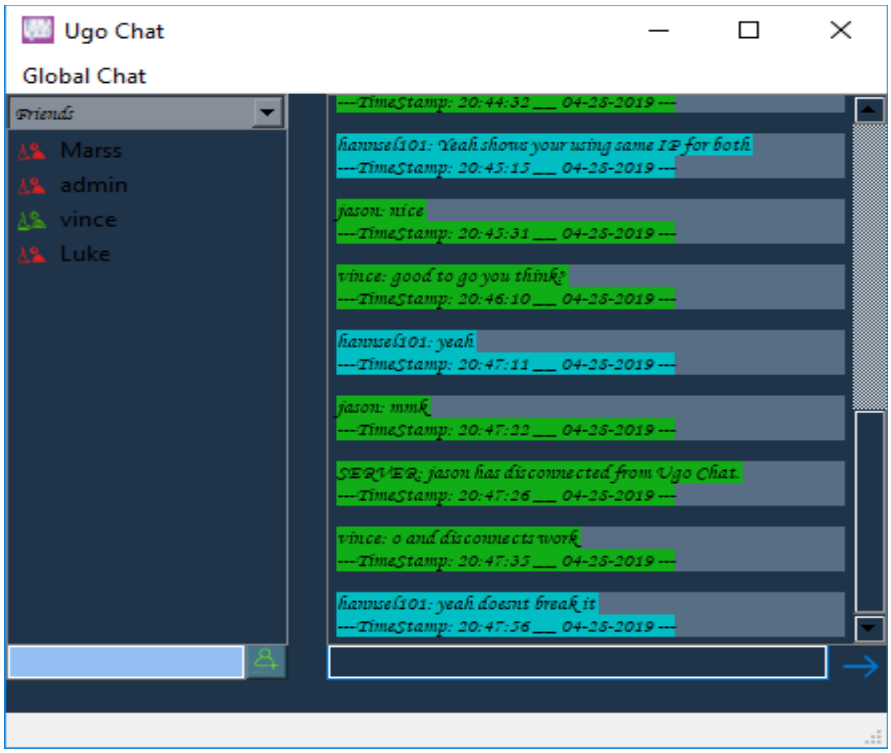

Figure 7: Ugo Chat main window with multiple clients chatting

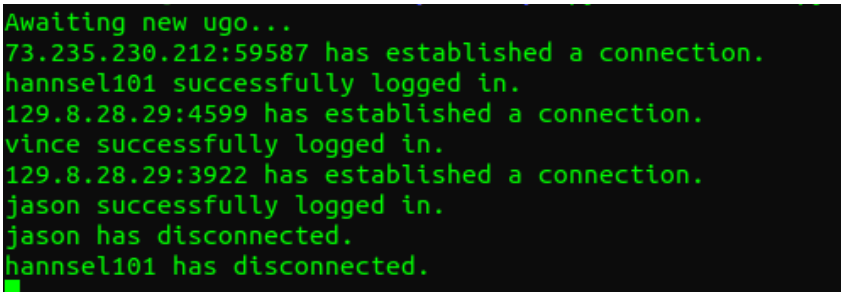

Figure 8: Server side terminal

Client Side Functions and Classes

- **Classes**
    - **myWin**
        - Main Gui window which handles all GUI objects and event driven signals.
        - Central Hub for the rest of the window and widget classes
        - openRegistration function:
            - Event 1: clicking Return button calls returnToLogin function
            - Event 2: clicking confirm sends registration attempt to server
        - confirmReg function:
            - Pulls text from 3 text entry boxes in registration page
            - Event 1: credentials are not correct or empty. Clears text entry boxes of previously entered information and opens registration error message
            - Event 2: Server verifies that username is available. Text entry boxes are cleared and upon successful registration returnToLogin function is called
        - loginCheck function:
            - Retrieves input from both text entry boxes
            - Event 1: If either username or password are empty, then a user not found error popup window is displayed
            - Event 2: Information is sent to the server for verification with login.db database file. If successful then the client's friends are sent over along with their online/offline status. When a "FINISHED" message is received, the client knows to start the application. Otherwise Event 1 is signaled stating credentials were not valid
        - returnToLogin function:
            - Simple function to hide widgets and display the login window
        - appInitialize function:
            - Signals the program to close all GUI objects other than the main chat.
            - Starts the receiverThread and passes the client's username, friends, and online status of friends into the thread.
            - Event 1: clicking the addFriend button calls the addingFriend function
            - Event 2: clicking the sendMessage_Button or pressing enter while focus is in the chat text entry box will call the sendMessage function

        - sendMessage function:
            - Pulls information from the main chat text entry box, clears the text entry box, and then passes the message into the send function
        - addingFriend function:
            - Pulls information from the addFriend text entry box, clears the text entry box, then sends "//VERIFY ADD FRIEND:" with the friends name appended to the end. This signals the server to check login.db to verify the user exist before adding. Also checks if the person to add is already a friend of the client.
    - **myReg**
        - Class that initializes registrationPage.py class object through composition
    - **myErr**
        - Class that initializes the userNotFound.py class object through composition
    - **RegError**
        - Class that initializes the userNotFound.py class object through composition.
        - Modifies the error messages displayed in userNotFound.py to be appropriate for a failed registration attempt
    - **RegSuccess**
        - Class that initializes the loginSuccess.py class object through composition.
        - Modifies the login success message to say "Registration Successful!"
    - **mySuc**
        - Class that initializes the loginSuccess.py class object through composition
    - **mainChat**
        - Class that initializes the ugoChat.py class object through composition
    - **receiverThread**
        - QThread class to run the receiving of messages in parallel to the main GUI thread.
        - Acts as a supervisor for the friendSync QThread by passing in flags to perform certain events.
        - Upon Receiving a message, the Client and Server have a set of strings that act as indicators of what each one should perform next.
            - "//CANNOT ADD FRIEND" signals that the friend request sent to the server is not allowed. (user does not exist in database or user is already your friend)
            - "//VERIFY ADD FRIEND:newFriend" signals that the friend request was allowed. The client reads the substring "//VERIFY ADD FRIEND:" first to set a performAdd status flag to true. Then

the newFriend substring will hold the username of the new friend to be added along with their current online status.
        - To avoid thread locking the new friend to add and their online status is passed into the friendSync QThread
        - To differentiate between a normal chat message and the server triggering "//VERIFY ADD FRIEND:", the server appends "username:" to the beginning of each message for a normal chat message. And for commands no name is appended before the command signal. In this way no user can activate the signal for another user.
    - If ":" is not present in a message then the client knows to update the online status of whoever shares the same name with the message received. Since online status is binary (on or off), if the friend in question is set to "OFFLINE" status then you just change that to "ONLINE" and vice versa.
        - performSync flag is set to True to signal friendSync to change the online status of the friend in parallel to the receiver.
    - If your username is present in the message received and it is part of the substring [0:N-1], with N being the number of characters present in your username, then this was a message you sent to the server and should be appended in blue.
        - Simple HTML/CSS Styling was used to highlight the messages and the background of the chat box.
    - If none of the above conditions are met then the message is considered a normal chat message from another user.
- **friendSync**
    - QThread to handles all friends list operations without burdening the other threads. Two status flag public members are used to signal the two functions present.
        - performSync: status flag that when set to TRUE will change the online status of one friend in your friends list that recently logged in or out of the chat.
        - performAdd: status flag that when set to TRUE will append a new row to the friends list GUI column with either a friend online icon or friend offline icon.
        - If one status flag is mid operation and the other one is called, then the second one must wait for the first one to complete.
        - The two status flags are checked constantly through polling.

Server Side Functions

- **Functions**
    - **portListener function**
        - Listens for incoming connections on port 50000
        - If connection is established then create a new Thread for the new connection
    - **handleClient function**
        - Waits for a command number sent from the client.
        - Command number “1” means a login attempt
        - Command number “2” means a registration attempt
        - After a successful login a loop for receiving messages is ran for the entirety of the client’s TCP connection
    - **Broadcast function**
        - Echos a message sent to the server to every user logged in.
    - **refreshOnlineFriends function**
        - Performs an online status check for a clients friends and sends their name along with online status to client.

Databasing Functions

- **Functions**
    - **createTable function**
        - This was ran once to create the login.db database file but kept in the code as reference for creating new tables
    - **friendsList function**
        - Accepts username as an input and appends “.db” to the username in order to create a new database.
        - If the database exist then “CREATE TABLE IF NOT EXISTS” is a SQlite3 conditional statement to check this edge case.
        - Databases created using the username as the name are known by the server to be the friends list of the user who matches the database name.
    - **updateFriends function**
        - Handles the adding of a new friend into a friends list database.
        - Makes an initial check to ensure the friend does not exist in the database.
    - **checkOnlineStatus function**
        - This function can check the online status of one client. By accepting a client as an input to the function, it can ensure that the user is indeed a friend and change their online status appropriately.

        - Originally implemented into the server.py code but instead was kept as a template for creating the checkAllOnlineStatus function below.
    - **checkAllOnlineStatus function**
        - Same functionality as checkOnlineStatus but accepts a list of clients instead of a single one. This way they can all be checked in one function call.
    - **checkFriends function**
        - Returns a boolean TRUE if the friend exist in friends list. FALSE otherwise.
        - A database function mainly used to be called by other functions for friend verifications purposes.
    - **updateTable function**
        - Updates the login.db database of registered UgoChat users.
    - **checkTable function**
        - Verifies a username and password credentials when logging into the server.
        - Returns boolean TRUE if the credentials match, FALSE otherwise
    - **checkUserName function**
        - Verifies username is not taken when registering a new account.
        - A more efficient way to implement this would be to make the username a PRIMARY KEY in SQLite3. This would force the usernames to be unique.

Conclusion

UgoChat successfully implements TCP sockets to interface a theoretically infinite number of users at the same time. Do to threading everyone can send and receive messages in any order. The GUI interface is user-friendly and the server/client interactions not noticeable by the user with the exception of the logging in. Due to the object-oriented structure of the GUI elements, modifying one does not impact the others in any negative way. This means our software is modular and easily maintainable.

References

[1]"socket — Low-level networking interface — Python v3.0.1 documentation", *Docs.python.org*, 2019. [Online]. Available: https://docs.python.org/3.0/library/socket.html. [Accessed: 29- Apr- 2019].

[2]"SQLite Tutorial - An Easy Way to Master SQLite Fast", *SQLite Tutorial*, 2019. [Online]. Available: http://www.sqlitetutorial.net/. [Accessed: 29- Apr- 2019].

[3]"The Noun Project", *The Noun Project*, 2019. [Online]. Available: https://thenounproject.com/ayasofya/collection/user-interface/. [Accessed: 29- Apr- 2019].

[4]"Python Networking Programming", *www.tutorialspoint.com*, 2019. [Online]. Available: https://www.tutorialspoint.com/python/python_networking.htm. [Accessed: 29- Apr- 2019].

[5]"PyQt Tutorial", *www.tutorialspoint.com*, 2019. [Online]. Available: https://www.tutorialspoint.com/pyqt/index.htm. [Accessed: 29- Apr- 2019].

[6]"PyQt5 thread example", *Kushal Das*, 2019. [Online]. Available: https://kushaldas.in/posts/pyqt5-thread-example.html. [Accessed: 29- Apr- 2019].

Special Thanks to the Noun project for open source images

User by Luis Prado from the Noun Project (online/ offline icon)
User by Wilson Joseph from the Noun Project(message received icon)
add by Roselin Christina.S from the Noun Project(add friend icon)