CSC 0845

Advanced Computer Networks

Professor: Hao Yue

Socket Programming

Team/App Name: Mini-Chat

Team Members:

Zhuojun He

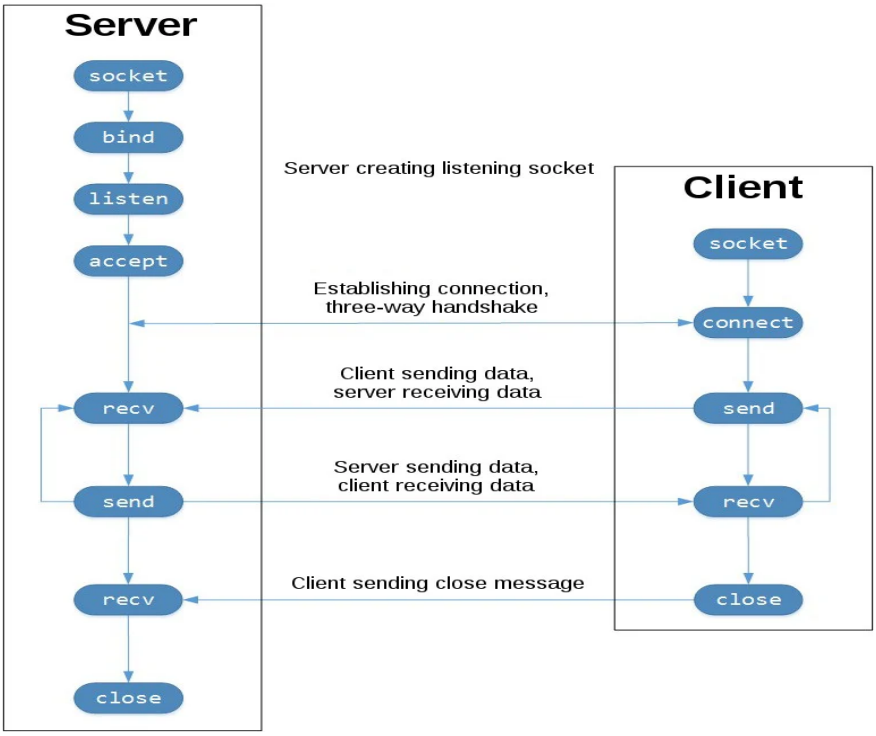
Swetha Govindu

Date: May 20th, 2021

Introduction

Socket communication is a technology for exchanging data between two networked computers. It is a convention or a way of communicating between computers. Through the socket agreement which is also called a protocol, a computer can send or receive data from the other computers. Taking our life as an example, we can get electrical power from the grid by plugging it into the socket. Similarly, to transmit data to a remote computer, we need to connect to the Internet, and a socket is a tool used to connect to the Internet. As a result, learning sockets is one of the ways to learn how computers talk with each other.

Today, the most commonly used network model is the TCP/IP protocol. And we also planned to use the TCP/IP model and its methods to develop our chatting app. The TCP/IP family all shared some basic methods. Here is a typical flowchart[1] about the structure of the TCP/IP model below.



The socket() function is used to create a socket, determine various attributes of the socket, and then the server-side needs to use the bind() function to bind the socket to a specific IP address and port. In this way, the data which flows through the IP address and port can be handled by the socket for processing. For the server-side, after using the bind() function to bind the socket, then it uses the listen() function to make the socket enter to the passive listening state, after this it will call the accept() function to respond to the client's request at any time. The passive listening state means that when there is no client request, the socket is in a "sleep" mode. Only when a client request arrives, the socket will be "awakened" to respond to the request. Accept() returns a new socket to communicate with the client which contains information about the client's information such as IP address and port number and so on. Similarly, the client-side will use the connect() function to establish a connection with the server, it uses the same

parameters input just as the accept function on the server-side. After the connection is established successfully, we have to use different functions to send and receive socket data depending on what system you have. Because Linux does not distinguish between socket files and ordinary files. It uses the write() function to write data to the socket and uses the read() function to read data from the socket. Windows is different from Linux. Windows distinguishes between ordinary files and sockets, and it defines special objects for receiving and sending data. It uses the send() function for sending data from the server and uses the recv() function for receiving data from the server. At the end of the communication, it is required to run the close() function to terminate the connection in respect to the socket for preventing memory leaks and many other crash problems.

The motivation of this project is to practice all those TCP/IP methods I discussed above, and create a simple chatting app. And We think the implementation of the chatting app is a good exercise to learn socket programming. So after completing this project, we can learn knowledge about socket programming, how two computers communicate with each other, and how to use a programming language to implement it. We can also learn how to write practical programs by using some basic TCP/IP methods, and also some simple and common interface to send and receive data.

To create a chatting app is the final goal of our project. We will use python to create client and server apps for text sending and receiving, and also instant chat between clients locally. So, in general, This app will contain a server and a client program. The server side holds all the account and password information for authentication purposes. It also handles different requests from the client. The client app is for different users to connect to the server. This app will have two functions for the users: text messaging through email asynchronously and instant chatting synchronously.

Project Description & Methods

The goal of the project is to allow users to communicate over offline messaging, instant messaging, and email by providing different options to logged-in users.

0. Connect to the server

This method handles socket connection between client and server. After the successful connection between client and server, the client has to log in with a username and password to use other functionalities in the project.

1. Get the user list

This method fetches all the users from the server flat database. Only logged in users can use this method to know other users that are registered in the server.

2. Send a message

Logged-in users enter the recipient's user name and message to send. The server receives the message from the client and acknowledges/responds to the client that the message was received successfully.

3. Get my messages

All the offline messages are saved on the server. Once users are logged in, they can retrieve all the messages that are received when they are offline.

4. Initiate a chat with my friend

This method allows initiating an instant chat by giving a specific port number. Once the other client is connected with the same port number. It allows to communicate live and communication can be terminated any time with message bye by either user.

5. Chat with my friend

This method allows instant chat by connecting to the same port number which has been used to initiate the chat by another client. It allows to communicate live and communication can be terminated any time with message bye by either user.

6. Send an email through Gmail

This method directly connects to the Gmail server from the local client through SMTP protocol. However, users have to log in with Gmail credentials before sending an email. The message has been sent to the receiver Gmail after successful authentication of the sender's Gmail credentials. Recipient Gmail has to enable low secure app settings in his account to receive email from the client(local).

7. Disconnect

This method allows the client to disconnect from the server.

Results & Demo

https://drive.google.com/file/d/1DdeuY7MiuMH449f4LRrWGAEs90zDkMLb/view?usp=sharing

Conclusion and Future Work

There are still a lot of challenging problems we need to consider in the future. For example, how to deploy this app online. Currently, this chatting app is able to run locally, but we find a lot of issues when we test it online. Swetha and I are all able to open three or more terminals and test the app in our local network. But we try one person to run the server and the client code, and the other person runs only the client code, and we hoped that the connection would still succeed, but are not. So there are still many steps we need to go forward with. Secondly, there are some reconnect problems that exist in our app. If a client doesn't close the app properly, for example, he just terminates the app by using ctrl+c or just manually closes the terminal window. Then his account may be held on the server. The Server still thinks his account is still alive. The next time the same client tries to log in, the server may give him some error message. As a result, we plan to improve our code which can handle those zombie accounts in the future. Last but not least, we are trying to make our app connect to the Gmail server. But still, there are a lot of technical issues that need to be overcome, such as there are some security standards that need to be met when our app tries to connect to the Gmail server. You have to turn on the "the less secure app access allow" function in your Gmail account if you want to use this to send email through the Gmail function. According to Google, if an app or site doesn't meet its security standards, Google might block anyone who's trying to sign in to your account from it. So if we have time, we will improve the security level of our app, and hope that users who send email through the Gmail function can be used by all Gmail accounts normally.

To make a short conclusion, during the practice of this project, we gain better insight into socket programming. We practiced applying some basic TCP/IP methods to develop a simple chatting app from starch. This project enriches and updates my knowledge and vision about the computer network today. Other than that, we find that communication is the key to making your job successful. And people, your teammate is the major reward after the project. So in general the experience we learned from this project is weighted more worthy than whether the app is good or not. And I am glad to have a chance to practice building a chatting app in this course.

References

[1] Nathan Jennings. *Socket Programming in Python.* https://realpython.com/python-sockets/
[2] https://docs.python.org/3/howto/sockets.html