CS3201 Programming Assignment Report

XUE Kaiwen 55199704

November 2018

1 Overview

This assignment implements a text messenger. Two computers in the same network can communicate send and receive message to/from both sides.

In this assignment, I set a client who initialize the connection and a server who accepts connection from the client as required. The server is identified by its IP address and binded port number.

I use python3 and did not compile the files because python interpreter is sufficient. Since the implementation of bonus part is very different from non-bonus part, I wrote two programs.

1.1 Contribution

Since I work on this project alone, so I contribute in all of it.
Name: XUE Kaiwen SID: 55199704

2 Implementation Details – Basic

2.1 Task 1: Socket Initialization

Server and client programs are seperated in different functions and function will be invoked depend on user input. Python standard library has a networking programming interface socket which provides functions to initialize and operate on sockets. To initialize a TCP socket, simply use

```
clientSocket = socket(AF_INET, SOCK_STREAM)
serverSocket = socket(AF_INET, SOCK_STREAM)
```

and a TCP socket is created. In this step, the server creates a "welcome" socket that applies three-way handshake used to accept connection while the client just creates the socket that initializes the connection. At this point, the client connects to the server by letting user input remote IP address and the port number.

2.2 Task 2: Bind IP and port, listen and accept connections

At this point, the server let user input the IP address and port number which will further be binded. The client connects to the server by letting user input remote IP address and the binded port number. In Python, this step contains two functions: `bind()` on server side and `connect` on client side.

The server marks the socket as accepting connection using `listen()`.

After that, the server accepts the connection using

```
(connectionSocket, clientAddress) = serverSocket.accept()
```

`accpet()` returns a socket that deal with further connection and the IP address and port number of the client.

2.3 Task 3: Connect to the peer with the given IP address and port number

As the description in Task 2, the client uses `connect()` to connect to the remote server.

2.4 Task 4: Send and receive the text

For sending and receiving, python socket library provides `send()` and `receive()` functions. I simply send, encode, decode and receive by normal program flow. If the message is quit, close the socket.

3 Implementation Details - Bonus

The whole idea for this part is to use the python threading `threading` standard library to implement multithreading.

3.1 Sending message continuously

Throughout the chatting process, two things we must do is to send and receive messages. Since receiving is triggered by messages sent from the other side, I put sending in the main thread and receiving in the other using the threading constructor

```
Thread(target=recv_msg, args=(client,)).start()
```

This is implemented both on server side and client side.

3.2 Adding participants

To add more than one participants, the most important thing is connection. Here, I simply set the parameter of `listen()` to 5 so that each client is connected to the server. All messages are sent to server and the server will "broadcast" the messages to all clients connected to it. This is achieved by keeping a dictionary that records a key value pair of information of corresponding connection sockets and client addresses. The rest is just a for loop to send messages to clients.

```
def broadcast(msg, prefix=""):
    """broadcast the message to all clients"""
    if prefix == "":
        to_send = msg
    else:
        to_send = prefix + " says: " + msg
    print(to_send)
    for sock in addresses_dic:
        sock.send(to_send.encode())
```

The other thing needs to be considered is accepting connections. This is also done by putting the accepting function in another thread. This thread is started in the main function of the server.

In conclusion, the server has a main thread for sending and broadcasting (sending function is just by invoking the broadcast function), a accepting thread, and a receiving thread. The client has only two threads, a main thread for sending and a thread for receiving messages.

4 References

Kurose and Ross, Computer Networking: A Top-Down Approach, 7th edition
https://docs.python.org/3/library/socket.html
https://docs.python.org/3/library/threading.html
https://medium.com/swlh/lets-write-a-chat-app-in-python-f6783a9ac170