Worcester Polytechnic Institute

CS513 Computer Network Project

A Client-Server Chat Program

Author: Fengrui Zhang *Professor: Thomas Gannon*

June 24, 2019

Abstract

This report outlines the design and development of a program which is a chat program based on the client-server model. The program was written in Python to run under the macOS operating system on Mac. The design and program are modular in nature and make maximum use of abstract data types and of software re-use. Particular attention is paid to Multiple clients, terminal based interface, broadcast message and private chat implementation. The report includes a the test cases used to verify the correct operation of the program, as well as the entire code.

Source code and data can be downloaded from:
https://github.com/zhangfengrui95/CS513_ComputerNetwort_ChatRoom.git

Contents

1 Project Description

This report outlines the design and development of a program which is a chat program based on the client-server model. The program was written in Python to run under the macOS operating system on Mac. The design and program are modular in nature and make maximum use of abstract data types and of software re-use. Particular attention is paid to Multiple clients, terminal based interface, broadcast message and private chat implementation. The report includes a the test cases used to verify the correct operation of the program, as well as the entire code.
The server can operations (such as connect requests and disconnect requests) should be printed out by the server, handle connections and disconnections without disruption of other services, let clients have unique nicknames, inform clients of changes in the list of connected users.
The client can display a list of online users to all users, display connection and disconnection actions of users, display messages from the originating user and other users, receive messages or actions while typing a message, let clients disconnect without disrupting the server.

2 Detailed Design

2.1 Server Design Step1

chat_server.py* client.py*

```
import socket
import select

def broadcast_data(message):
    """ Sends a message to all sockets in the connection list. """
    # Send message to everyone, except the server.
    for sock in CONNECTION_LIST:
        if sock != SERVER_SOCKET:
            try:
                sock.sendall(message) # send all data at once
            except Exception as msg: # Connection was closed. Errors
                print(type(msg).__name__)
                sock.close()
                try:
                    CONNECTION_LIST.remove(sock)
                    index = CONNECTION_LIST.index(sock)
                    CONNECTION_ADDR.pop(index)
                except ValueError as msg:
                    print("{}:{}".format(type(msg).__name__, msg))


def privatechat_data(message, target_add):
    """ Sends a message to special sockets in the connection list. """
    # Send message to specific person
    for sock in CONNECTION_LIST:
        index = CONNECTION_LIST.index(sock)
        add = CONNECTION_ADDR[index]
        if sock != SERVER_SOCKET and add == target_add:
            try:
                sock.sendall(message) # send all data at once
            except Exception as msg: # Connection was closed. Errors
                print(type(msg).__name__)
                sock.close()
                try:
                    CONNECTION_LIST.remove(sock)
                    index = CONNECTION_LIST.index(sock)
                    CONNECTION_ADDR.pop(index)
                except ValueError as msg:
                    print("{}:{}".format(type(msg).__name__, msg))
```

Figure 1: Define two method: broadcast and private chat. In order to broadcast message to every one in chat room, defined broadcast method. For sock in the connected list, send message to all client sockets except server. If there is error, display to server and disconnect the client. In order to send private message to special person, defined privatechate method. For sock in the connected list, compare address of target person and connected sockets, if correct, then send message, hence, special message will not display to others.

2.2 Server Design Step2

chat_server.py* client.py*

```
CONNECTION_LIST = []
CONNECTION_ADDR= [00000]
RECV_BUFFER = 4096
PORT = 1338

SERVER_SOCKET = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
SERVER_SOCKET.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
SERVER_SOCKET.bind(("", PORT)) # empty addr string means INADDR_ANY

print("Listening...")
SERVER_SOCKET.listen(10) # 10 connections

CONNECTION_LIST.append(SERVER_SOCKET)
print("Server started!")
```

Figure 2: Define server socket, set receive buffer and port for server. Any client connects this address can enter this chat room.

2.3 Server Design Step3

chat_server.py* client.py*

```
CONNECTION_LIST.append(SERVER_SOCKET)
print("Server started!")

while True:
    # Get the list sockets which are ready to be read through select
    READ_SOCKETS, WRITE_SOCKETS, ERROR_SOCKETS = select.select(CONNECTION_LIST, [], [])
    for SOCK in READ_SOCKETS: # New connection
        # Handle the case in which there is a new connection recieved through server_socket
        if SOCK == SERVER_SOCKET:
            SOCKFD, ADDR = SERVER_SOCKET.accept()
            CONNECTION_LIST.append(SOCKFD) # add socket descriptor
            CONNECTION_ADDR.append(ADDR[1])
            # Adding \r to prevent message overlapping when another user
            print("\rClient ({0}, {1}) connected".format(ADDR[0], ADDR[1]))
            broadcast_data("Client ({0}:{1}) entered room\n".format(ADDR[0], ADDR[1]).encode())
        else: # Some incoming message from a client
            try: # Data recieved from client, process it
                DATA = SOCK.recv(RECV_BUFFER)
                if DATA:
                    ADDR = SOCK.getpeername() # get remote address of the socket
                    message = "\r avaiable user{}[{}:{}]: {}".format(CONNECTION_ADDR,
                                                    ADDR[0], ADDR[1], DATA.decode())
                    print(message, end="")
                    #public chat
                    if '@' not in message:
                        broadcast_data(message.encode())
                    else:
                    #private chat
                        target_add = message.split('@', 1)[1]
                        privatechat_data(message.encode(), int(target_add))
            except Exception as msg: # Errors happened, client disconnected
                print(type(msg).__name__, msg)
                print("\rClient ({0}, {1}) disconnected.".format(ADDR[0], ADDR[1]))
                broadcast_data("\rClient ({0}, {1}) is offline\n"
                               .format(ADDR[0], ADDR[1]).encode())
                SOCK.close()
                try:
                    CONNECTION_LIST.remove(SOCK)
                    index = CONNECTION_LIST.index(SOCK)
                    CONNECTION_ADDR.pop(index)
                except ValueError as msg:
                    print("{}:{}.".format(type(msg).__name__, msg))
                continue

SERVER_SOCKET.close()
```

Figure 3: Set main program for server. First we need to handle case where there is a new connection. We append sockets and its address into list for later use. Otherwise, if not a new connection, server receive data from client, and display message in the server as well. Then check the message is public or private, if public, use broadcast method we defined in step1, if private, use privatechat method accordingly. If error happen, disconnect client and display message in server, then broadcast this message to every client as well.

2.4 Client Design

chat_server.py client.py*

```
import socket
import select
import sys

if len(sys.argv) < 4:
    print("Usage : python {0} hostname port nickname".format(sys.argv[0]))
    sys.exit()

HOST = sys.argv[1]
PORT = int(sys.argv[2])
nick_name = sys.argv[3]

MASTER_SOCK = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
MASTER_SOCK.settimeout(200)

# connect to remote host
try:
    MASTER_SOCK.connect((HOST, PORT))
except Exception as msg:
    print(type(msg).__name__)
    print("Unable to connect")
    sys.exit()

print(nick_name + " connected to remote host. Start sending messages")

while True:
    SOCKET_LIST = [sys.stdin, MASTER_SOCK]
    # Get the list sockets which are readable
    READ_SOCKETS, WRITE_SOCKETS, ERROR_SOCKETS = select.select(SOCKET_LIST, [], [])

    for sock in READ_SOCKETS: #incoming message from remote server
        if sock == MASTER_SOCK:
            data = sock.recv(4096)
            if not data:
                print('\nDisconnected from chat server')
                sys.exit()
            else: # print data
                print(data.decode(), end="")
        else: #user entered a message
            msg = sys.stdin.readline()
            print("\x1b[1A" + "\x1b[2K", end="") # erase last line
            MASTER_SOCK.sendall((nick_name + ':' + msg).encode())
```

Figure 4: Set Client: get information of system input and set client socket as server. Set main program for the client. In order to get message from server, go through each sockets, if there is data, receive and display to the user, if not, disconnect from server and exit system, then server will automatically display disconnection information to every one still online.

3 Testing and Evaluation

3.1 Server Start

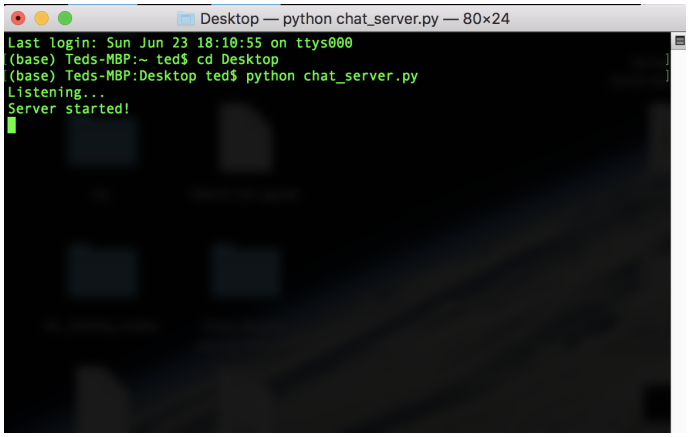

Figure 5: Start Server by typing "python chat_server.py" on terminal. The server start listening

3.2 First User Enter

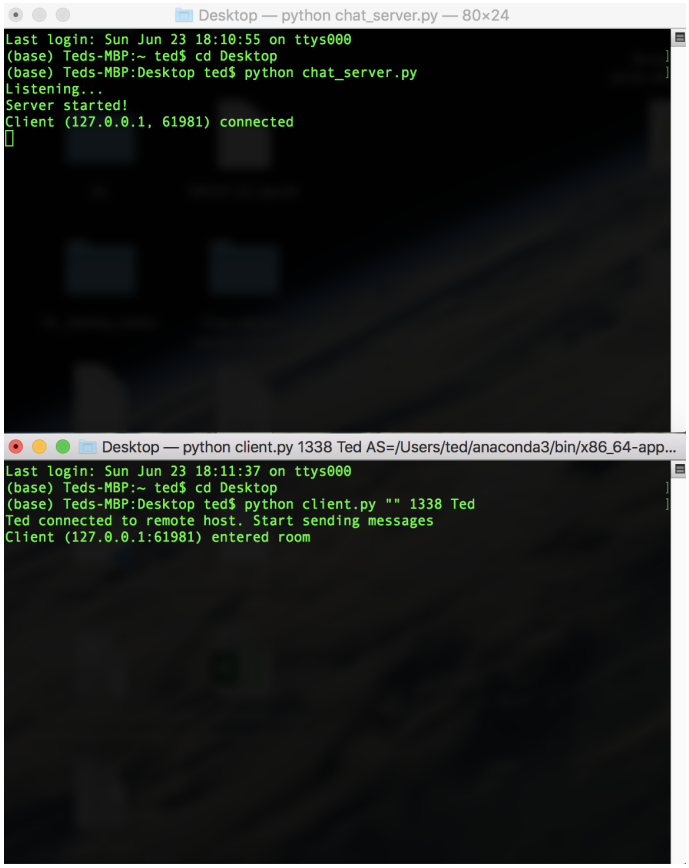

Figure 6: Start client by typing "python client.py "" 1338 Ted" on terminal. The client connect with server, has nick name Ted. Nickname with address is unique and no duplicates in my system. Both server and client display Ted enter room.

3.3 More Users Enter

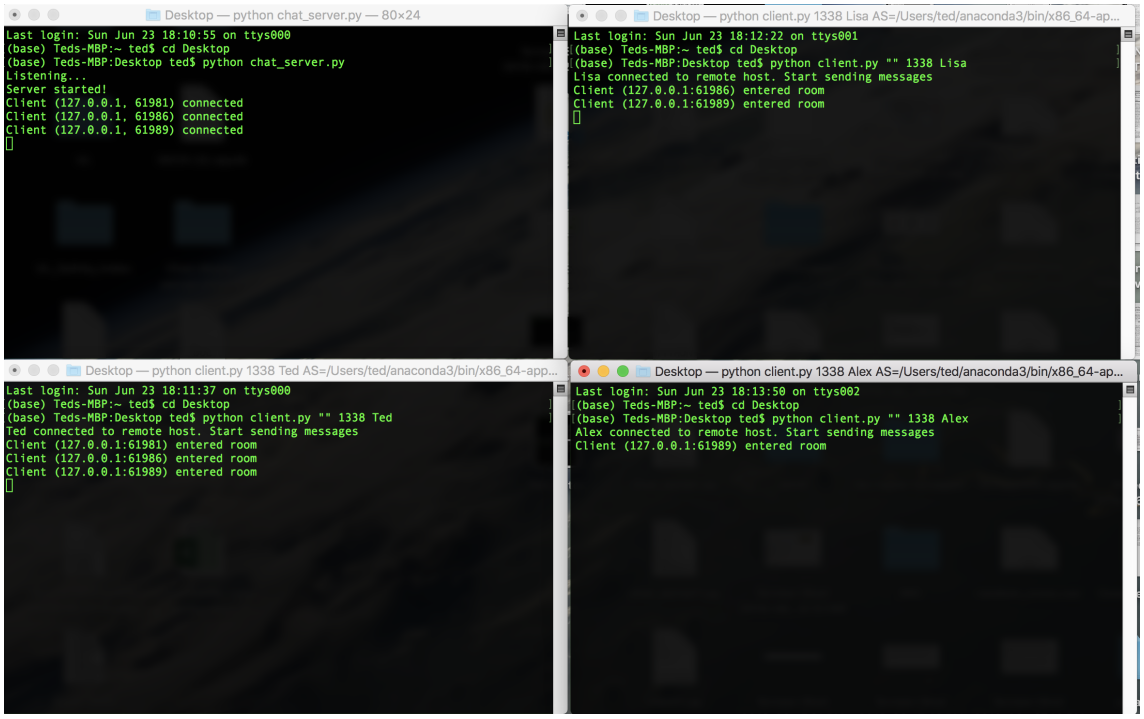

Figure 7: More users enter room. Both server and client display Lisa and Alex enter room. Server handle connections and disconnections without disruption of other services.

3.4 Message to everyone

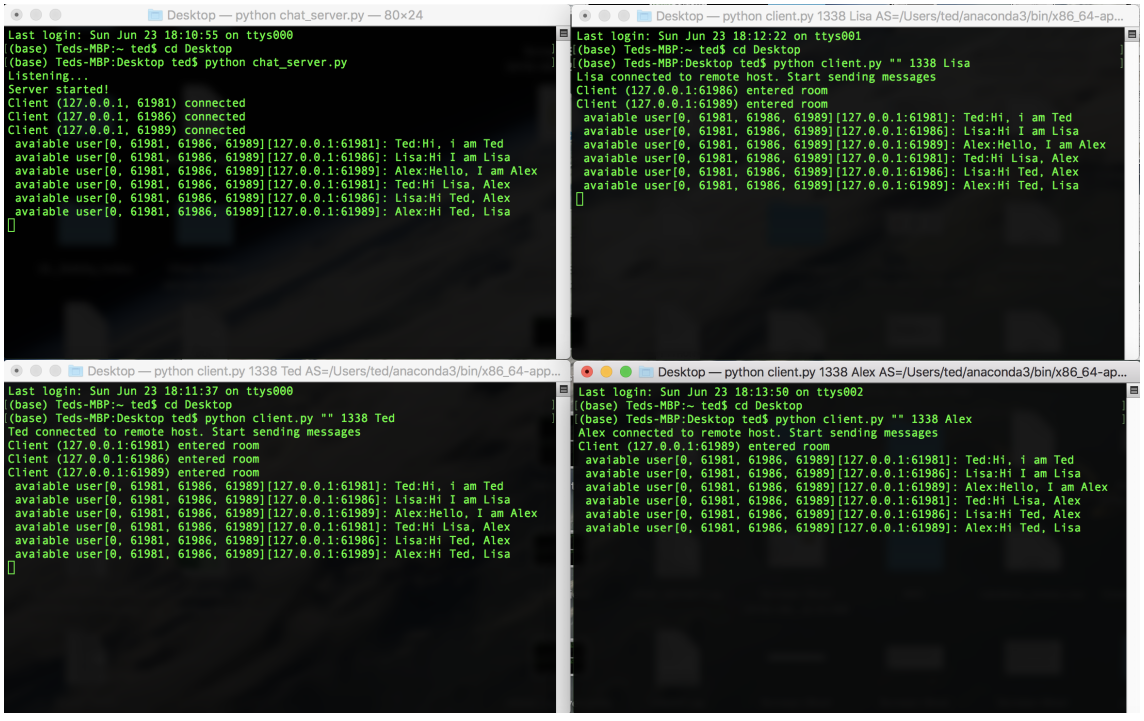

Figure 8: Message broadcasts to everyone in chat room. Everyone know availability of other person.

3.5 Private Message

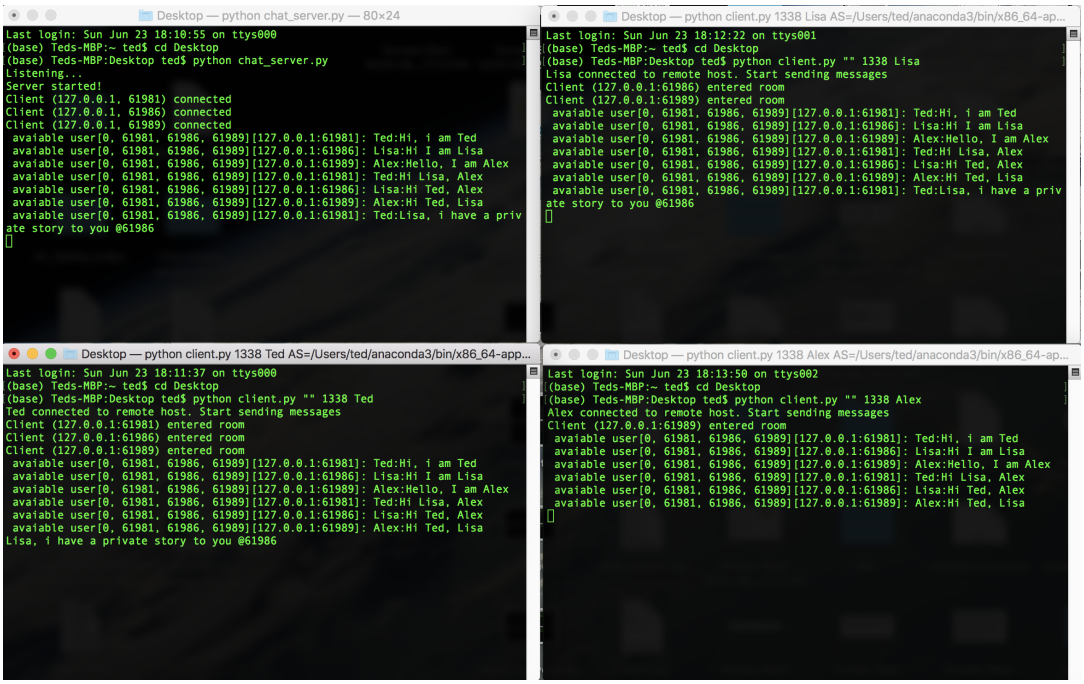

Figure 9: Private message to special person. For example, Ted has a private story to Lisa by typing "Lisa, I have a private story to you @61986", here 61986 is Lisa address, which is display to everyone. And this message will not display to Alex.

3.6 Alex Offline

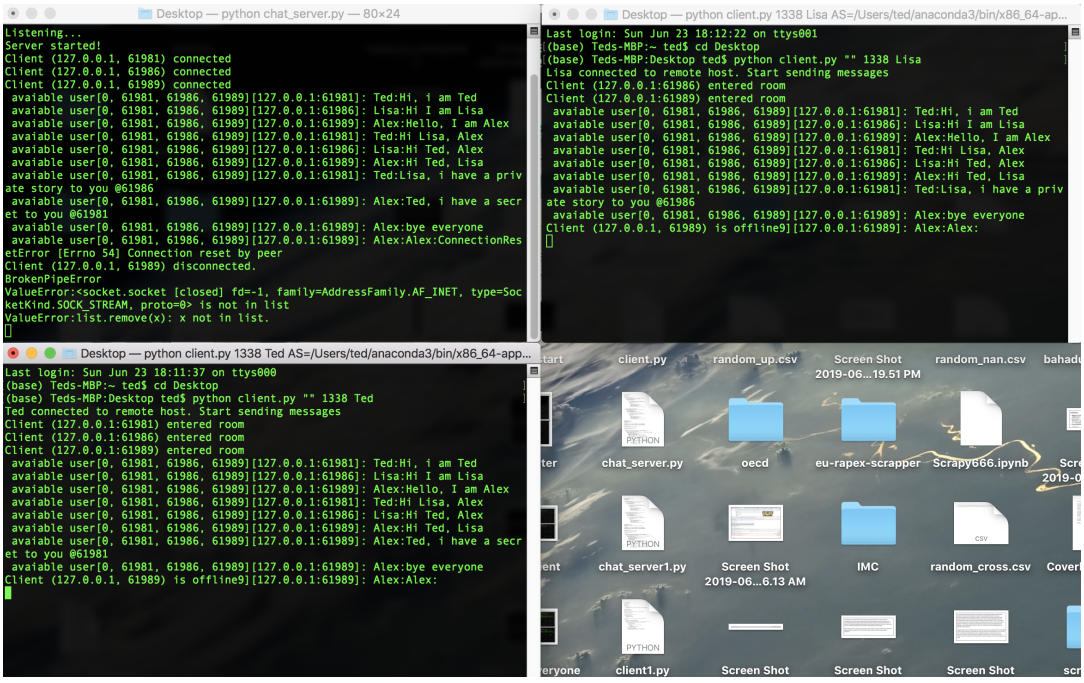

Figure 10: Alex choose to leave, say bye to everyone, then end terminal. All other users, Ted and Lisa display Alex offline.

3.7 Server Offline

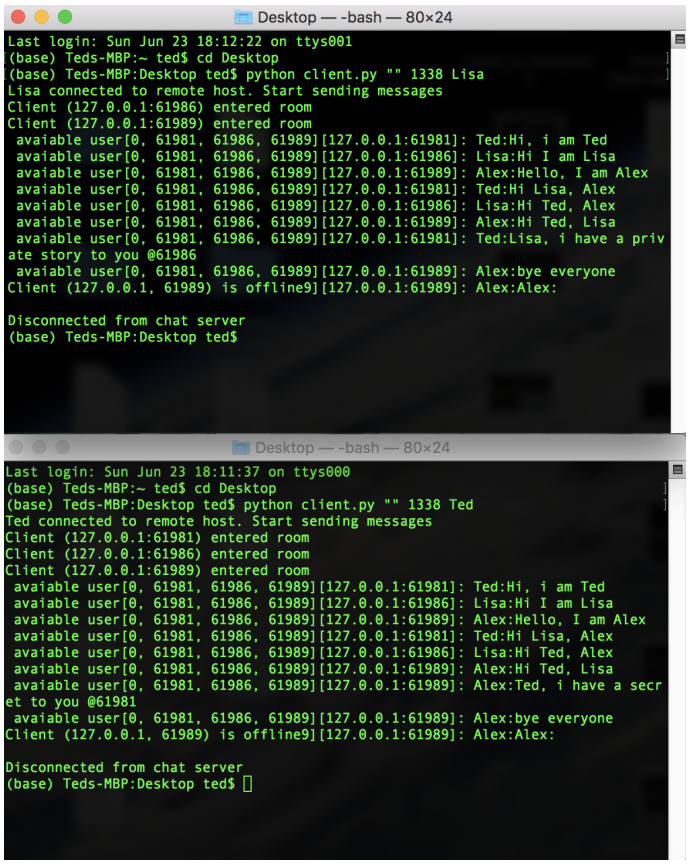

Figure 11: End server. All users, Ted and Lisa, display disconnected from chat server.

4 Future Development

Due to the time limitation of my full-time job in summer, especially twice travel this month, I don't have enough time to implement GUI for client at this time. Sorry, hope you can understand. However, I have finished all functionality as document required. Details can be found in Test Part. Hence, future work is to built GUI for client.

5 Conclusion

This project implemented all functionality of document required except GUI, including Multiple clients, terminal based interface, clients able to choose a nickname, clients be able to "whisper" to each other without having messages displayed to other users. The server can operations (such as connect requests and disconnect requests) should be printed out by the server, handle connections and disconnections without disruption of other services, let clients have unique nicknames, inform clients of changes in the list of connected users. The client can display a list of online users to all users, display connection and disconnection actions of users, display messages from the originating user and other users, receive messages or actions while typing a message, let clients disconnect without disrupting the server.