2019 Computer Network Final Project cnMessage Report

B06902127 鄭人愷
B06902037 吳柏成
B06502149 張琦琛

1. User & Operator Guide

a. Server Operations

i. Actually, you can just leave our server alone after start running it, while it will output some system messages.

b. Client Operations

The followings are all operations, and you can see the details by typing "help" after running client.py

i. Send: send the message to other online users.

send [receiver] [message]

ii. Broadcast: send the message to all other online users.

send [all] [message]

iii. Blacklist: if we add some users in the blacklist, we won't receive their messages anymore, and we can also remove them from the blacklist.

blacklist [add/rm] [receiver]

iv. History: the instruction will return the latest messages. In a special case, if you enter '0' as the second parameter, it will return the messages received when the user was off-line.

history [# of the latest messages received]

v. List_users: the instruction will return both online user list and offline user list.

list_users

vi. File transfer: the instruction will send the files to the server and inform a client to get the files from the server.

sendfile [receiver] [file_num] [file path 1] [file path 2] ...

vii. Getfile: getting files from the server.

   getfile [file_num] [file path 1] [file path 2] ...

viii. Exit: user leave the chatroom

   exit

ix. Sign up: after typing “signup”, we will ask user to enter username and password. If the signup is successful, user will automatically login.

   signup

x. Login: after typing “login”, we will ask user to enter username and password. If the username not exists or password is wrong, we will print error messages to user.

   login

2. Instructions on how to run server & clients

a. Environment: python 3.x

i. Package: os, sys, threading, json, cmd, time, struct

b. Run the program

i. We don’t need to run the build script

ii. Default server ip:’127.0.0.1’ port:12416

iii. python server.py

iv. python client.py

3. System & Program Design

a. Files and Folders

i.

```
└─06_final
          ├─src
          │      ├── server.py
          │      └── client.py
          ├─build.sh
          └─Report.pdf
```

ii. Server side

```
.
└─data
│      ├─file_received
│      └─PasswordTable
```

```
└──msg
      └──history of messages
```

iii. Client side

```
.
└──client_data
          └──username
                 └──file_received
```

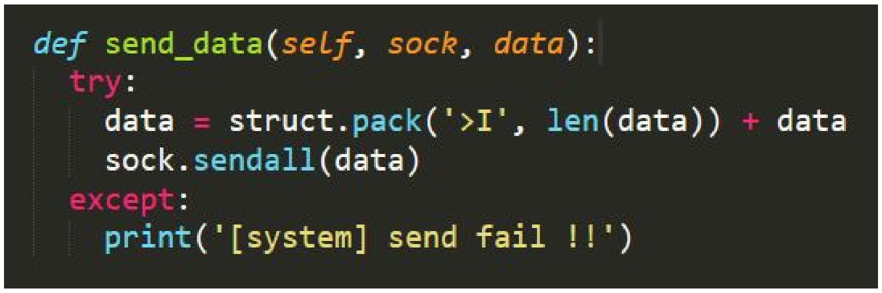
b. Scenario

i. After user running client.py, we will create two socket connections to server; one is recv_msg_socket, another is send_msg_socket, and it will enter a while loop to wait for user's command. If user login successfully, we will create a new thread called recv_msg_thread, which is responsible for receiving messages from server; otherwise, user will create a new thread called send_msg_thread, which uses send_msg_socket to send messages, for each sending request. Furthermore, we use locking mechanism to avoid multiple write the same socket file simultaneously.

ii. After server running server.py, the server will run a daemon thread to send ACK message to all online users to detect whether they are still alive; in addition, it will wait for the new connection request; for each request, server creates a new thread to handle it. By this way, we can address with multiple requests simultaneously.

c. Some details for function implementation

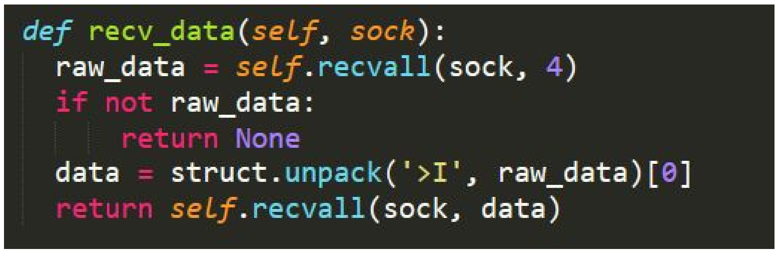
i. Send data



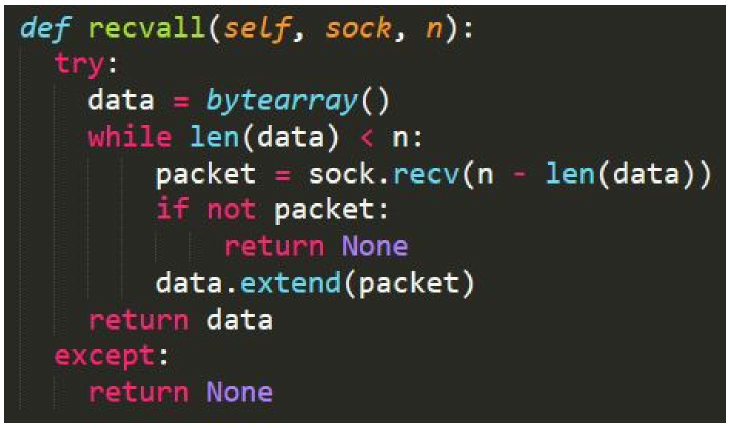
```python
def send_data(self, sock, data):
    try:
        data = struct.pack('>I', len(data)) + data
        sock.sendall(data)
    except:
        print('[system] send fail !!')
```

When sending data to others, we will compute its size first and append the information with 4 bytes in the beginning of data, and we use sendall() method to send it. By this way, despite that the size of data is larger than the buffer, we can send it to others completely.

ii. Receive data



```python
def recv_data(self, sock):
    raw_data = self.recvall(sock, 4)
    if not raw_data:
        return None
    data = struct.unpack('>I', raw_data)[0]
    return self.recvall(sock, data)
```



```python
def recvall(self, sock, n):
    try:
        data = bytearray()
        while len(data) < n:
            packet = sock.recv(n - len(data))
            if not packet:
                return None
            data.extend(packet)
        return data
    except:
        return None
```

Since we have append the 4 bytes' data size in the beginning of the data, thus the receiver will know the whole data size and receive it thoroughly.

iii. history

Every user has a file to store the messages sending to him/her in the server. When a user requests for his own history of received message, the user sends an instruction to the server and the server replies the history to the user. There is a number in users' requests for history and it stands for the number of messages that users want. In a special case, if the number is zero, the server will return the messages(not including broadcast) sent to the user when it was offline.

iv. send files

To send multiple files at the same time, we open a new thread and a new sending socket for each file. The files will first be transferred to server, and the server will save these files in the specific directory according to who these files are sent to. Then server will inform the receivers that there are files sent to them and that they can get the files from it.

v. get files

After the client was informed that there are files sent to them, client can use “getfile” command to download files from server. The server will check whether the file names in the argument are exist, and then transfer the requested files one by one.

vi. Sign up

All the (username,password) pairs are saved in PasswordTable.json file. When the server receive a signup request that contains username and password, server will check if the username is used. If username was not used before, the two attributes above will be hashed by SHA-1 and saved in PasswordTable.json file. If username is used, server will return error message to user.

4. Other things

This project can be said to be overwhelming. Our group read related articles and tried to write a simple version of the code before the start of the final exam. However, due to improper division of labor, a huge amount of code was hard to be integrated. The problem, especially the variable names, is really quite complicated, causing obstacles in understanding. Later, one person was responsible for writing the basic data transmission, specifying the message format, and the others were in charge of completing the rest of the functions.

[Bonus]:

1. **Auto-reconnect:**

   For server side, server will run a daemon thread to send ACK message to each online user every 3 seconds; if server fails to send the ACK message, then the server will consider the user as offline, and join it to offline list. For client side, client will also run a daemon thread to detect whether the recv_msg_thread was exited, which means that the user was unconnected. When the daemon thread detect such condition, we will run the reconnect() method automatically and the user don't need to login again. In addition, we use do_break(), which will close the recv_msg_socket and send_msg_socket of the user, to simulate disconnection.

2. **Broadcast:**

   If one needs to send messages to lots of users, entering the almost same instructions for countless time is a nightmare for every user. It will be much more convenient for users to have a function to send a message to all the users online.

3. **List_users:**

   Since users may have the demand to know who is online to transfer file or other messages, knowing who's online or offline is very helpful.

4. **Blacklist:**

   It is annoying to receive useless messages from specific users. To save users from a screen full of trash, blacklist can help to block messages from any user you dislike.