CS3201 Programming Assignment Report

Wenbo YANG, 55203027
Chengyu Sun, 55202891
Deheng Zhang, 55199998

1. Design of the software

1.1. Message Format

1.1.1. Message from clients to server

Each message should be ended with the string "END", followed by one or more line breaks, i.e., "\r\n". The beginning of one message should not have any blank line.

1.1.2. Message from server to clients

The first line of message should be "*address*:", followed by one or more blank space, then followed by one line break. Each message should end with "END", followed by one or more line breaks, i.e. "\r\n".

1.2. Modules of the software

To make the development and testing easier, the software is divided into several parts shown in Figure 1 and Figure 2.

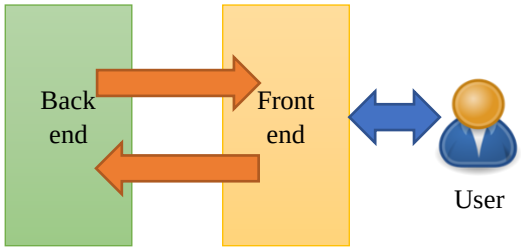

FIGURE 1 CLIENT PROGRAM

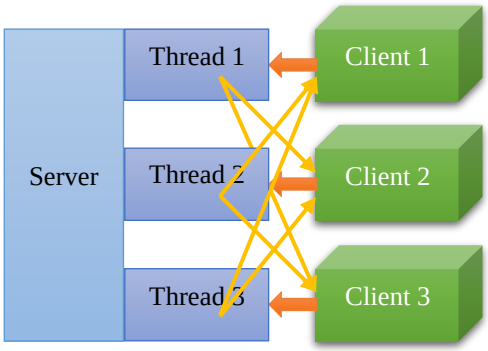

FIGURE 2 SERVER-CLIENT COMMUNICATION

The client has two parts, the back end is in charge of communication to the server, and the front part if in charge of communication to the user. When front end want to send message to the back end, it just call the `.send()` method of that back end, and if one back end want to send message to the back end, it can call the `.send()` method of the front end.

The server creates several threads, each of which serve one client. When (the back end of) one client send a message to the server, the server will broadcast its message to all other clients, which implements a chat room. Using this technique, one client does not need to wait for others' reply before sending message, which fulfil the requirement of the bonus part.

In case of one client's input process may be interrupted when another client's message is received and presented in the command line, a user interface has been designed to separate client input and message output, which is implements the back end. The sending and outputting behavior is further triggered by user behavior and `.send()` method of back end.

The main function in the program let user choose an option to be a server or a client. Upon the instruction of creating a server, it will create a `Server` object and initialize it with corresponding IP address and port number, with a `Client` object and a `UserInterface` object created and binded with the `Server` object representing the server.

Upon the instruction of creating a client, the program creates a `Client` object and a `UserInterface` object and further bind client with the server through IP address and port number.

2. Guide to Use the Program

2.1. Start the program

The execution start point is the `Main.main()` method. After starting this program, a promotion in command line will be shown as the Figure 3. User can type ‘c’ to connect to a server, type ‘w’ to act as a server, or type ‘q’ to quit the program.

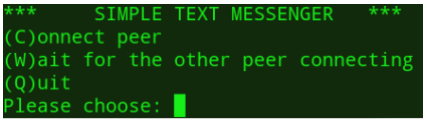

FIGURE 3 WELCOMING MESSAGE

2.2. Server

At first, type “W” or “w” in the command line to choose the server. Next enter the IP address and port number you want to bind. Then a server-side chatting GUI will be created. The basic operation is same as client, but if you quit the GUI, all the clients will be forced to exit. At the end, you should type “END” in the command line to completely quit the program. When a client connects to the server or a client leaves a server, there will be a promotion printed in the command line.

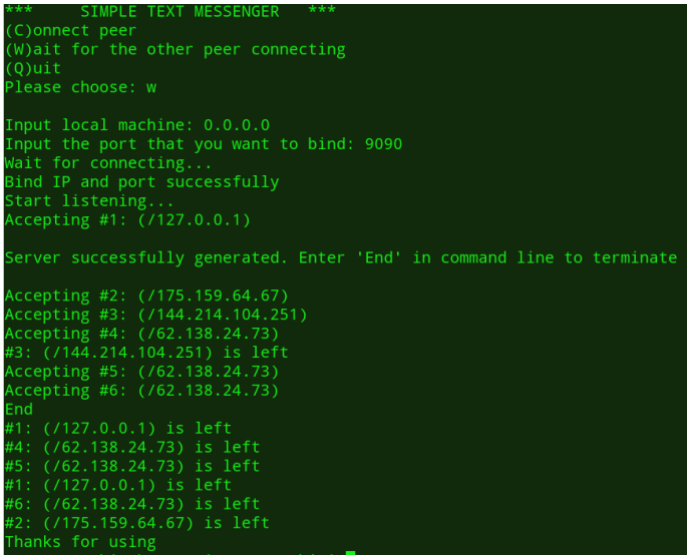

FIGURE 4 SERVER

2.3. Chat window

In the chat window, as displayed in Figure 5, one can press `Ctrl+Enter` to send message.

3. Implement Details

3.1. Client

3.1.1. Creating sockets, binding and connecting

In the `.connect(host, port)` method of `Client` object, `socket = new Socket(host, port);` creates a socket, bind it to an ephemeral port and then try to connect to server with given host address or name and port number.

3.1.2. Send and receive texts

After successfully connecting and starting the client's back end, it will start a new thread (`waitingThread`), waiting for the server to send messages. When the front end calls its `.send()` method, the back end will send this message to the server.

3.2. Server

Server-side codes are mostly in the Server.java file, which contains a Server class.

3.2.1. The work principle of server

As a bonus version, the Server class enable group chatting and bidirectional message sending. Besides, each message can be multiple lines.

- How to achieve bidirectional pipelining message sending (sender doesn't need to wait)?

  In order to achieve group chatting, the server cannot directly join the chatting, and the way to enable server to join the chatting will be introduced in the 4th bullet point.

- How does server wait for client and receive the message from client?

  There is a serve thread (not the main thread) created by server to keep on waiting for the incoming connect socket from new client. Each time a new client connects to the server, the server thread will create a new thread to keep on listening the message sent by the client.

- How does server achieve group chatting? (most important part)

  The role of server is a broadcaster. Each time, an arbitrary client sends a message to server, the server broadcast the message to all clients (stored in a hash map). Then the client just displays the message received from server.

- How does server-side user join the chatting?

  This is done by main thread. If a user choose server, then a server will be set up. At the same time, a local client-side application will run to achieve the server chatting. (details can be seen in the user interface part)

3.2.2. The implementation detail of server

- Socket initialization and bind the IP and port

  This is done by constructor of Server. Which accept 2 variables (IP and port) and throws Bind Exception.

```
try {
    this.serverSocket = newServerSocket(
        port,MAX_USER,InetAddress.getByName(ip)
    );
} catch (BindException e) {
    throw e;
```

```
}
```

- listen and accept connections (bonus part)

  In constructor, the thread wait for connections from clients will be called

```
Thread t = new Thread(() -> addClient());
this.waitForConnection = t;
t.start();
```

  In the `.addClient()` method, there will be a statement wait for connection and block the thread.

```
Socket incoming = serverSocket.accept();
```

- Store the client (bonus part)

  The client sockets will be stored in a concurrent hash map (thread safe data structure). Key is the socket, value is the IP address of the socket.

- Receive the text (bonus part)

  There will be a new thread created in the `.addCient()` method once a new client connects to the server ,which used to keep on listening the message from client.

- Send the text (bonus part)

  Each time server received a message, it will send the message to all of the clients. (Use the iterator of hash map)

- Quit the program

  In the `.stop()` method, the serve thread which is waiting for connection will be interrupted, all sockets in the hash map will be deleted.

  User can evoke this stop method by typing "`END`" in the terminal.

3.3. User Interface

3.3.1. Components

The `UserInterface` class implements Window components defined in `java.swing` and `java.awt` packages, which contains two `JTextArea` objects implementing the output field and input field and one `JButton` object implementing the "Send" button. Layout and detailed structure design will be ignored in this report, but the general look of the interface is shown in Figure 3.

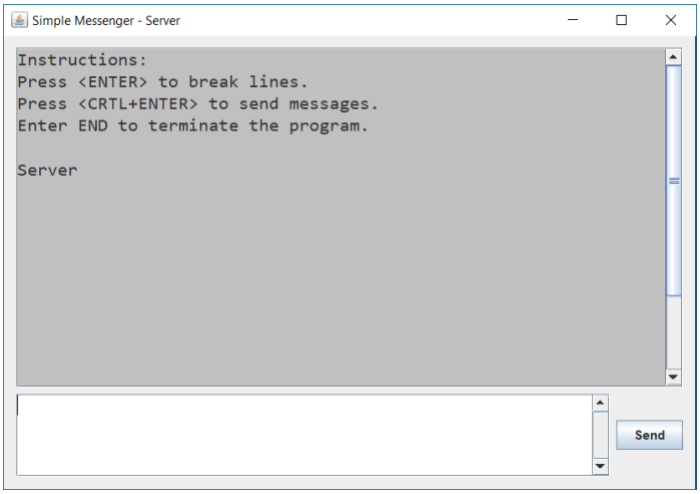

FIGURE 5

3.3.2. Binding with Client

In the `.bindTo(chatBack)` method, the user interface will refer back end (i.e. saving that client in the instance field) to the client bond and define the further `.send()` method through the corresponding back end.

3.3.3. Presenting Messages

Upon receiving messages sent by the server, the interface will call the `.pushToOutput()` method to append the receiving String message to the output field defined in the `JFrame`.

3.3.4. Sending Messages

When the client presses "`CTRL+Enter`" or the Send button, the interface will read the input value in the input field defined in the `JFrame` with suitable input manipulation done. Then it will call the `.send()` method defined in back end to send out messages.