CS 513 Class Project

A Client-Server Chat Program

Xinjie Hao(568968665)
Date of submission:7/2/2015

It is all me work unless marked to the contrary.
Signiture: Xinjie Hao

Abstract

This report outlines the design and implementation of CS 513 course project. In this project, a chat program based on the client-server model is implemented. The project is implemented in Java under Mac OS by taking advantages of frameworks like multithreads and sockets. 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 the stability and robustness of the server-client connection, disconnection and reconnection. The report will describe the design, implementation and test in detail.

Contents

Project Description

This project is implemented to build a server-client modeled chat program. Multithreads programming is used to realize multi-clients connect to a single server and communicate with each other bypass server. The date transmitted between server and clients take advantage of protocol of socket. As the framework on multithreads and socket is well designed in Java, this project is programmed in Java language. The most challenging thing in this project is to manage the connection, disconnection and re-connection operations for both server and client sides. Only fixing all the exceptions during dealing with these stuffs can one say he have a initial understanding of multithreads and socket programming.

Detailed Design

1. Design pattern

(1). As it is a server-client model, server and clients usually are deployed in different computers. So the program is separated into two independent parts: server side and client side. They communicate with each other by socket.

(2). For both server and client sides, the program uses MVC design pattern which separate data, GUI and operation as much as possible.

2. Overall structure

The over all structure of the program is shown in Figure 1.

For the server:

(1). A server object is created and started on a particular port p.

(2). The server then listen to port p, wait client to request connect to it. When it receives a connection request, it will create a server thread for the client to use

and wait for the next client connection request.

(3). The server can stop its service temporarily, the clients connected to it will then disconnect with server. However, when the server restart, all the clients connect to it will re-connect to the server and using the service as normal.

(4). The server can quit and then stop its service forever, the clients connected to it will have to quit also.

For the client:

(1). A client should use the server's address and port number to send connection request to the server. When the server accepts its request, it will build input and output stream with its corresponding server thread to transmit data with server back and forth.

(2). When the client disconnect to the server, the server thread will close the socket and go to end.

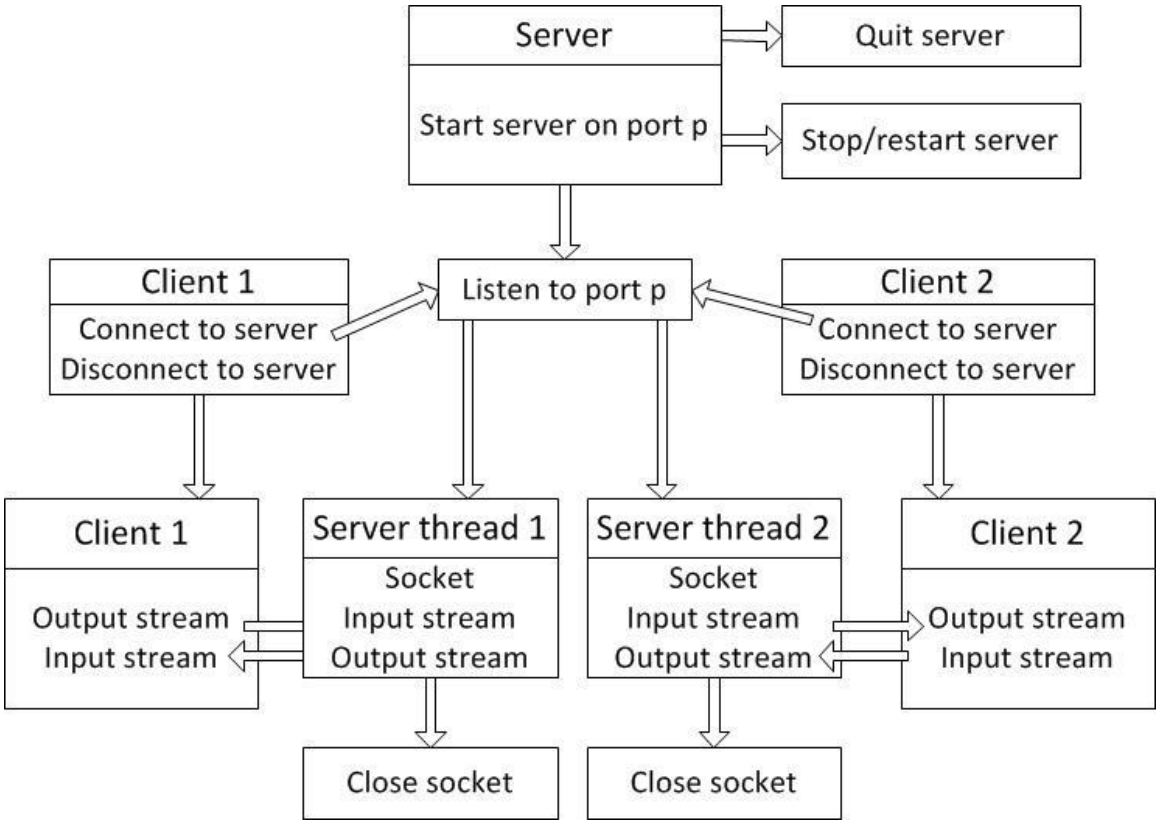


Figure 1. The over all structure

3. Server side design

The server side is implemented in package Server.

The classes be related to the model part of the server is shown in table 1.

Table 1. Classes of server related to model part

Class	Function
ClientInfo	Record a user's nick name and socket
ClientMap	A hash map to store all the online users' information

The classes be related to the view part of the server is shown in table 2.

Table 2. Classes of server related to view part

Class	Function
ServerView	The server's GUI witch can input port number and exchange between stop and start service

The classes be related to the controller part of the server is shown in table 3.

Table 3. Classes of server related to controller part

Class	Function
ServerController	Take care of the operations of start server, listen to connection request from clients and create threads for them, stop/restart server and quit server
TransmitMessages	Make sure the messages sent to the right receivers
UpdateClients	Update the online clients list for all the clients whenever there's change on the client map

4. Client side design

The client side is implemented in package Client.

There's no classes related to the model part of the client. Because a client doesn't need to store the online clients list or message histories in this initial version of chat program.

The classes be related to the view part of the client is shown in table 4.

Table 4. Classes of client related to view part

Class	Function
Login	GUI for client to input nick name and type the host name (address) and port number of the server to connect to
ClientView	GUI of a client when logged in. Check online clients, show received messages and send messages all worked on this GUI. It is also a thread to listen to the data sent from server thread.

The classes be related to the controller part of the client is shown in table 5.

Table 5. Classes of client related to controller part

Class	Function
ClientController	Manage the connect and disconnect to server

Critical Implementation milestones

1. Building connection between server and client

Solution:

Official document (https://docs.oracle.com/javase/tutorial/networking/sockets/) of Oracle described details on working with sockets in Java. This program just followed the instructions of the document.

2. Multi-clients connect to a single server

Solution:

Creating a server thread for each user connected to the server. The reason of doing this is to prevent slow user from blocking server, so the server can respond as quickly as possible to each user.

3. The protocol of messages between server and clients

Solution:

In order to focus on main purpose of this project, a simple and efficient protocol of message based on data structure of String is designed. A typical message is constructed as below:

Keywords##sender##receiver##msg

Keywords is to indicate the type of the message, sender and receiver are the massage's sender and receiver which can be none for some special messages, msg is the actual text message which can also be none, '##' is the token to differentiate the three part when receiver parsing the message. Table 6 is the message types used in the program.

Table 6. Message Types

Message Type	Instruction
server stopped	Inform all the clients server has stopped
server restarted	Inform all the clients server restarted
server quit	Inform all the clients server quit for ever
client login##sender	log in request sent to server by sender
login confirm	Inform the client it has already connected to the server
duplicate nickname	Inform the client it should change another nickname before using service
client logout##sender	Inform the server sender will log out
create list##sender	All clients send this message to sender to inform sender they are online
add one##sender	Inform all the clients sender logged in
remove one##sender	Inform all the clients sender logged out
message##sender##receiver##msg	sender send msg to receiver on client side
private##sender##receiver##msg	server send msg to receiver and inform him the it was sent by sender
public##sender##msg	server send msg to bulletin board and inform it was sent by sender

Given the messages, it is straightforward to implement broadcasting messages, whispering to each other, checking duplicate nick name and updating online clients list.

4. Client log out

Solution:

The client send message to inform server thread it will disconnect with server. Then the server thread stop listening the socket(jump out of the while-loop) and then close the socket, input stream and output stream.

5. Client re-log in

Solution:

On the client view, construct a new thread and start the thread. Then it will send request to the server and server will create a new thread for the client.

6. Server stops temporarily

Solution:

Send a fake connection request to server by force to help server stop stuck on listening port(serverSocket.accept()), then use a flag to help server stop listening the socket(jump out of the while-loop) and then close the socket, input stream and output stream. However, the trick is instead of closing the all the clients' sockets, just prevent clients from sending messages (disable "send" button). So, when the server restarted, all the connected clients don't need to re-connect server, just enable "send" button.

7. Server re-start

Solution:

Re-create the server socket and start to listen to port. Then inform every former client the server is restarted and let them using service by enable "send button".

8. Server quit forever

Solution:

First, stop the server. So it will not accept new client.

Second, inform every clients to disconnect with server.

Testing and Evaluation

The program is tested base on designed scenarios to evaluate the satisfaction of project requirements and robustness of the program.

Scenario 1: Start server
Testing: Shown in Figure 2 - 3

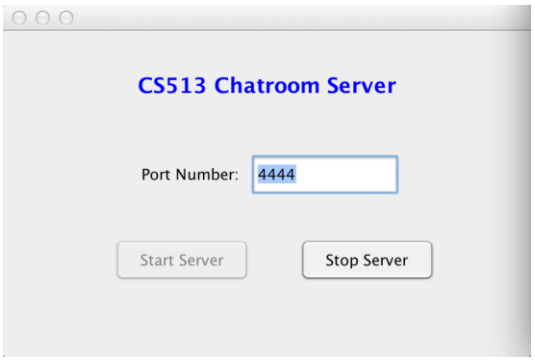


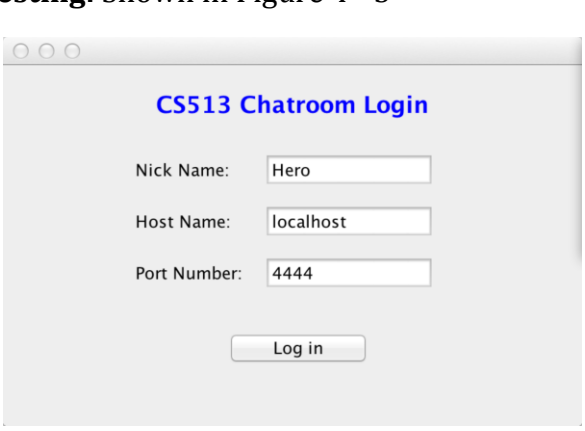
Figure 2. Server GUI status

```
The server is started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=4444]
```

Figure 3. Console status

Evaluation: Server stated properly.

Scenario 2: Two clients want to login the chat room, they both want to have nick name "Hero", however, the chat room will block the second client and ask him to choose another nick name.
Testing: Shown in Figure 4 - 5



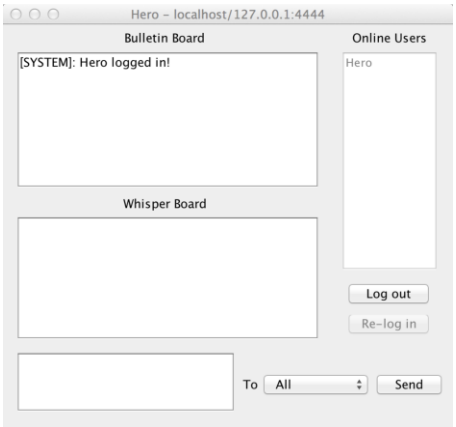


Figure 4. First client using nick name "Hero" successfully logged in

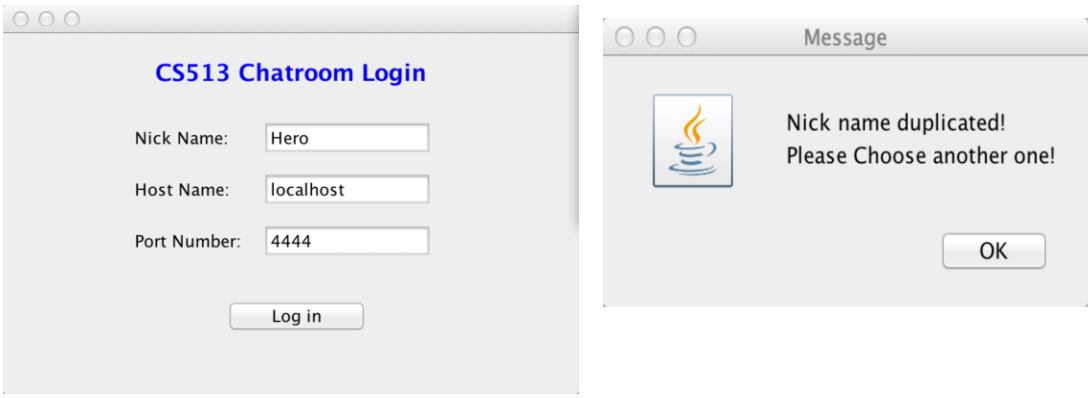


Figure 5. Second client using nick name "Hero" could not log in

Evaluation: The program successfully detects the duplicate nick name.
Scenario 3: Four clients talking on bulletinboard and whispering to each other
Testing: Shown in Figure 6

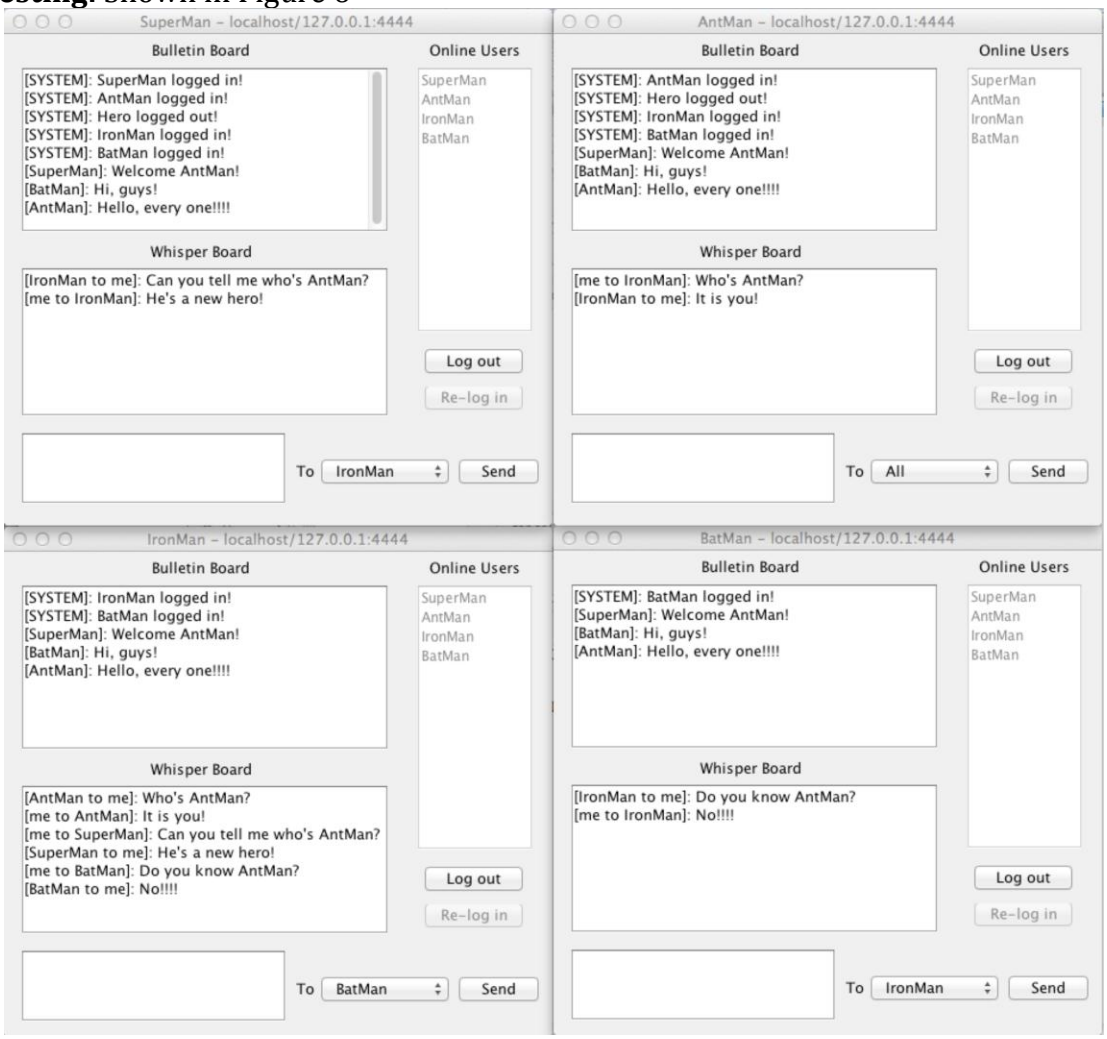


Figure 6. Four clients broadcasting and taking to each other

Evaluation: The messages are sent correctly to their destinations and parsed errorless. To send message to BulletinBoard, just select to "All", while to whisper, just select the specific client you want to whisper to. It is easy to operate and prevent the scenario of whispering to a non-existing client or itself from happening.

Scenario 4: One client log out and re-log in, check the stability of the program.

Testing: Shown in Figure 7- 8

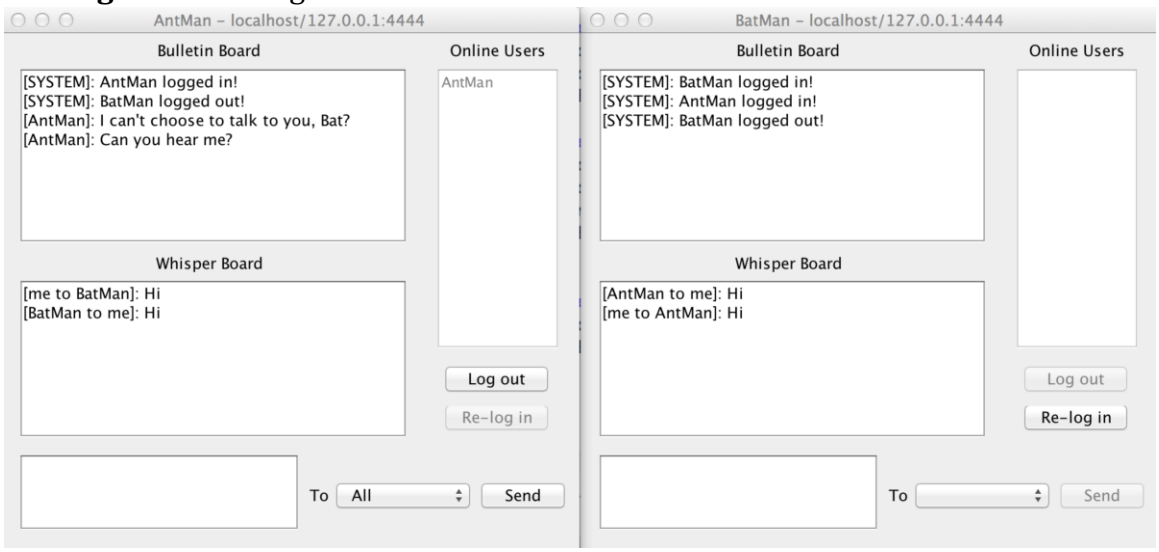


Figure 7. System informed every client that BatMan logged out. AntMan's online Users list updated. AntMan cannot choose BatMan to whisper to. BatMan cannot receive any message from server.

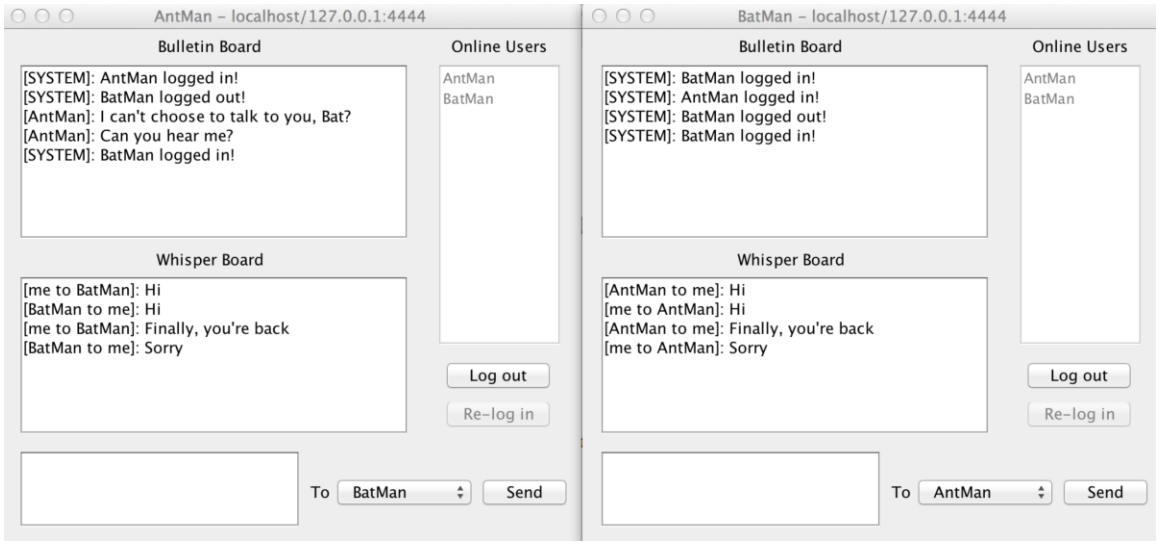


Figure 8. System informed every client that BatMan logged in. Both AntMan and BatMan's Online Users List updated. They can whisper to each other again

Evaluation: Whenever a client logout or re-login to the server, the system is stable without any exception. The online users' list updated timely and correctly. After logging out, the socket is closed completely, the client no

longer receive any message from server. And <u>clients can go on talking to each other from the moment he logged out</u>.

Scenario 5: A client want to connect to a server with port number = 444 which is not existing.
Testing: Figure 9

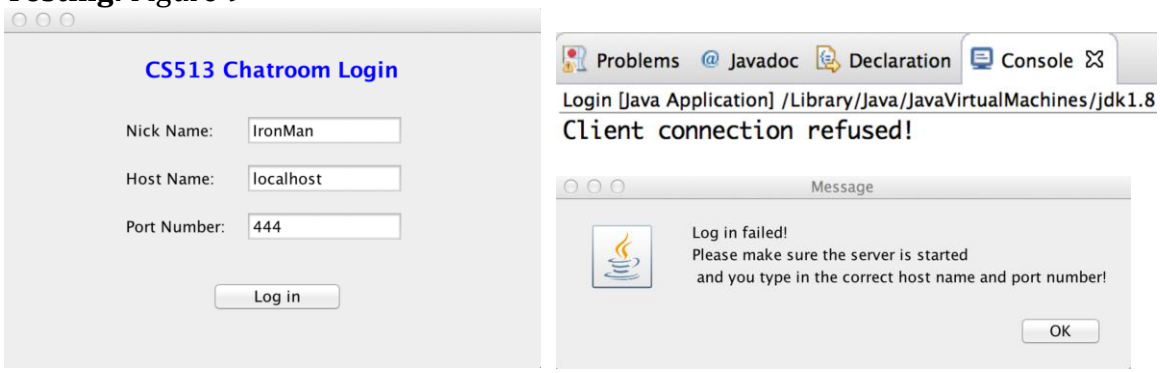


Figure 9. When connecting to a non-existing server, a warning message is given.

Evaluation: Checking this exception correctly and handle it properly.

Scenario 6: System maintain stable during server stopping temporarily and restart
Testing: Shown in Figure 10 - 13

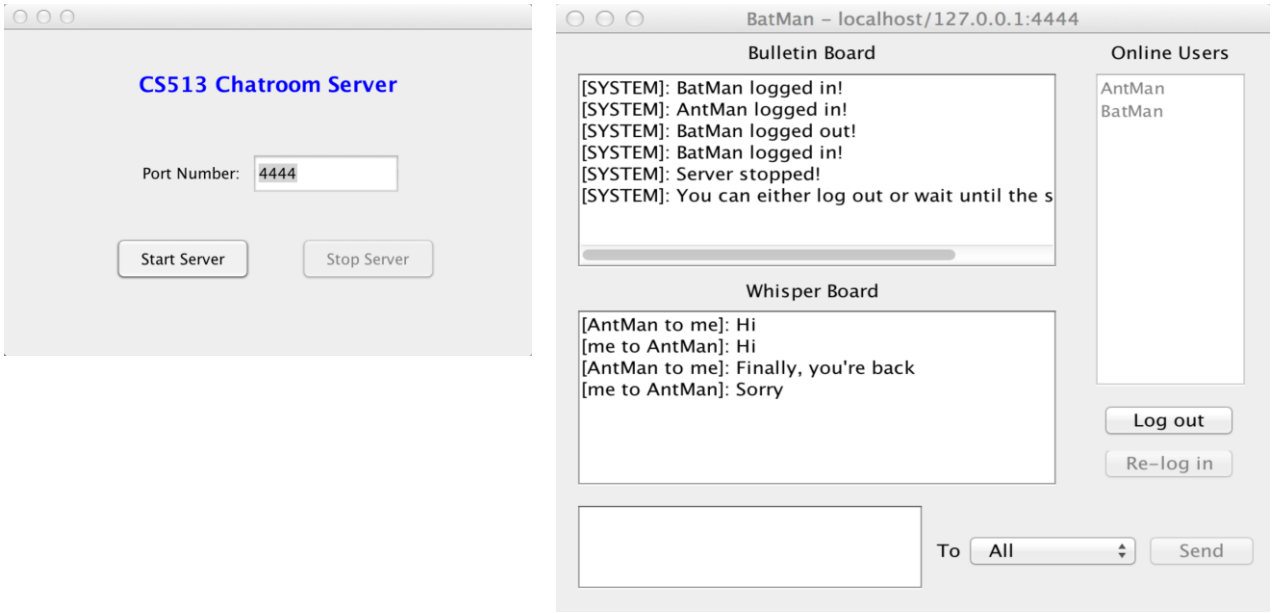


Figure 10. After server stopping, the client got the message to wait or logout and the "send" button is disabled. But actually, it is still socket with server and just wait the server restarting.

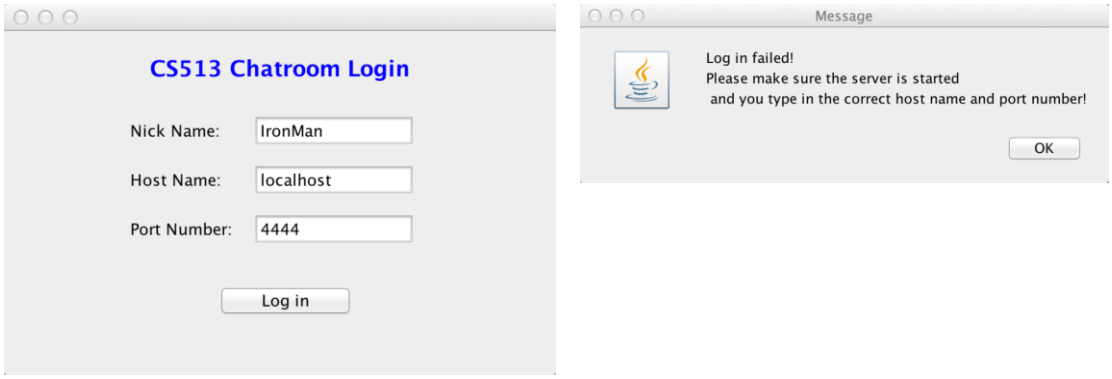


Figure 11. After server stopping, a new client cannot connect to it

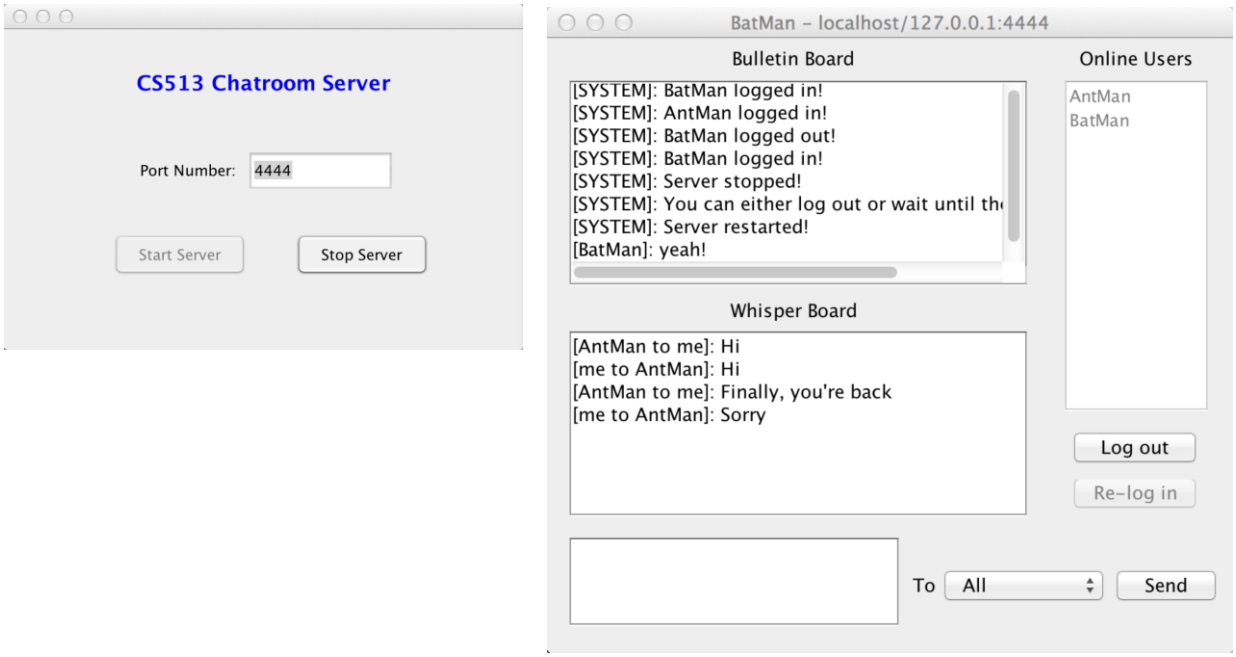


Figure 12. After server restart, the former clients talk from the moment server stop

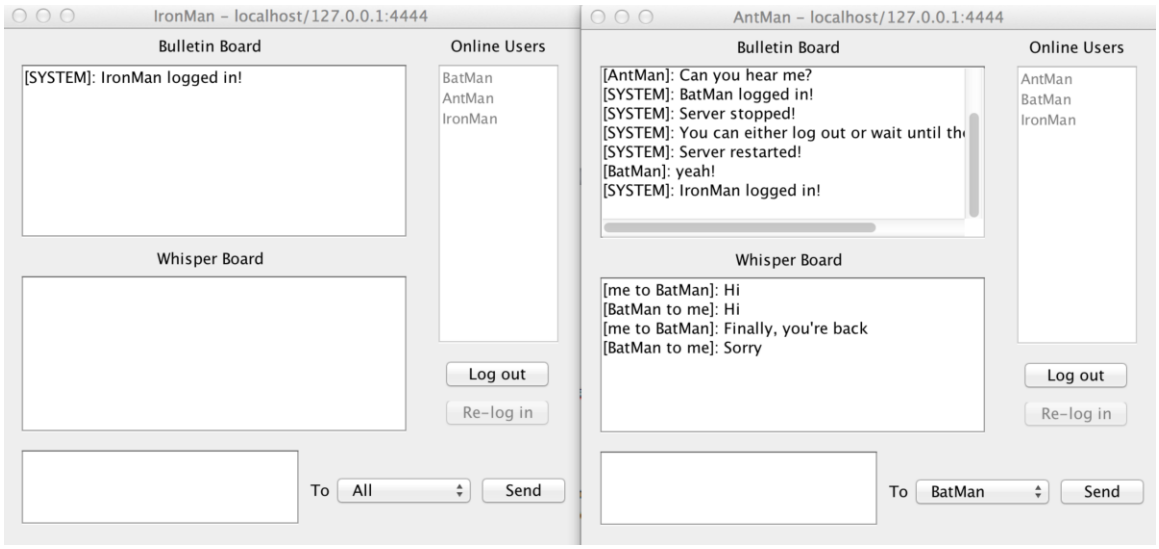


Figure 13. After server restart, a new client can connect the server properly

Evaluation: The system works stable and properly after server stop and restart.

Scenario 7: System maintain stable when server quit forever
Testing: Shown in Figure 14

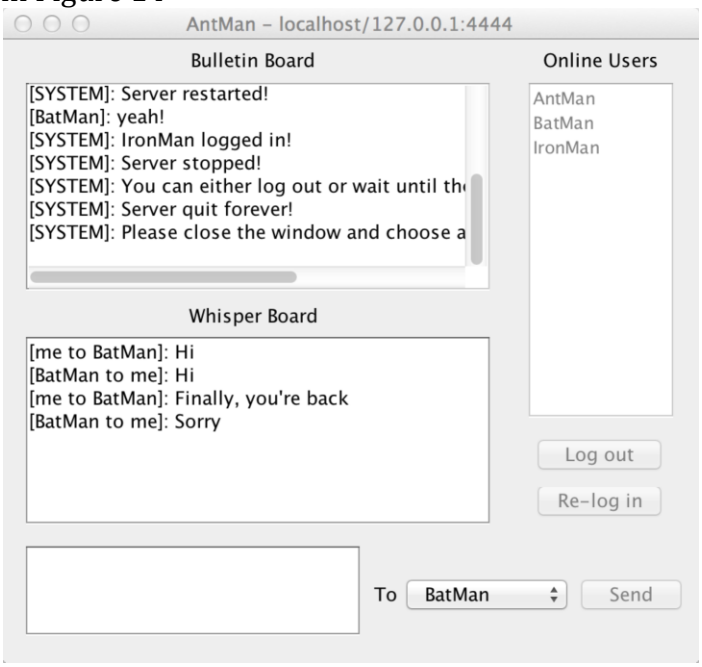


Figure 14. After closing the server window, the client got the message that the server quit forever, all it's button is disabled and it can only quit.

Evaluation: The system maintain stable after server quit forever.

Future Development

a. Build the model part of client, so the client can store their chatting history and friends information at local mechine.

b. Create account(nickname, passwords, birthdate,.etc) for each client.

c. Design more complicated protocol for massages.

Conclusion

This report describes and analyzes the whole project including the design of the program, difficulty and overcomes during the project and testing the program. From the project, I get a well understanding of network-programming like socket and multithread programming. And the most important part of the project is to give me a deep insight on networking which I cannot obtain from lectures. Thank the professor to design such a good project for this CS513 class.

Appendices

The source code is posted on my Github-site:
https://github.com/xjhao/CS513---Chat-Room.git