JusTalk

A Qt's messaging application

Boutin Azarias B00092351

BN013 BSC in Computing

Contents

I. Introduction

The aim of this project is to create a messaging application. Using sockets. We learned how to use the sockets with java, and the java.net.Socket package. But to learn how to use another language and another framework, to have better performances, and a more maintainable code, I chose to use C++ with the Qt framework.

1. The Qt framework

I chose Qt for two main reasons :

1. It is widely used by companies everywhere around the world. Thanks to its history, it is very easy to use and the community it very large.
2. The languages used by Qt is C++. It is one of the most complex object-oriented programming language. By knowing how to smartly use it, I will know how to use others object-oriented language.

Four tools are necessary to run this project :

- A working personal computer
- A C++ compiler (g++)
- The QtCreator IDE to create the Qt application.

2. Installing and running

To be able to run the project. You must first clone the github repository at this address : `https://github.com/AzariasB/JusTalk`. As said above, you will need QtCreator IDE. (Or at least qmake). Qmake will generate the makefile from the .pro files.
The folder contains two "Qt projects". There is the client project and the server project. Both of them must be opened with qtcreator. Once this is done, one can first start the server. Then start as many clients as one wants.

II. Implementation

1. Server

The server implementation contains one class and requires a common file : ActionList (See below for more informations). A server object contains

- a TcpServer. Used to listen any incoming connections.
- a Set of TcpSockets. All the connected clients
- a map of QString for each TcpSockets. It is used to know what sockets correspond to what pseudonym.

The server waits for any incoming connection. And listen for any user's messages.
When one new connection is detected, the socket is added to the set, and the user must send a presentation message to be fully detected by the server as part of the chat room.
When a user is sending a message, this one is tested with regular expressions, and a function is called when the regex matches. The possible messages are :

- /me:pseudo : a user's presentation. The user's pseudo is added to the map. (If it's not already in)
- @pseudo:message : a user whispers to another.
- message : a single-line message. To send to all the users.

The client's GUI is really simple. It displays a list of the connected users. The server can be closed by closing the window, or by clicking the 'quit' button.

2. Client

The client implementation contains one class and requires a common class: ActionList (See below for more informations). A client object contains :

- a TcpSocket. The socket used to communicate with the server.
- a String, for the pseudo.
- a StringList, for all the blacklisted users.

Once started, the client will try to connect to the server (with its socket). And once the connection will be done, it will send its pseudo to the server to give to the chat room the pseudo of the user.
Once this is done, the client just waits for any incoming messages from the server.

When the user send a message, the message is simply written in the socket. When the socket is receiving some data, the data is processed, and depending on the form of the message, differents actions will be executed. Here are the differents forms message a user can receive :

- /users:user1,user2,user3: a list of the connected users.
- pseudo:message : a message from the user with pseudonym : pseudo.

The client's GUI is composed of three main parts.

- The chat room. Where all the messages are displayed. Event those from the user himself. (These messages begin with 'me:')
- The chat bar : where the user can write down a message and send it to everyone. To send the message, the user can press enter or click the 'Talk!' button.
- The user's list : A list of all the connected users. The user can right click on one of these users to either whisper to them or add them to their own blacklist.

3. Common parts

ActionList is a common part of the server and the client. Since the two of them have differents functions depending on the message there are receiving.
The action list is a simple key - value hash. Where the key is the regular expression to match, and the value is the name of the function to call if the regular expression matches.
This class contains two simple functions :

- addAction : register the function name and the regular expression to match.
- triggerAction : see if the string given in parameter matches any regex, and if so, call the function with the given context.

This function makes uses of the really powerful signal-slot system provided by Qt. In Java, it's a bit like the actionListener, but in more concise, and more general. It works not only for GUI, but for any class. It is also possible to create custom signals and slots.

III. Design

The communication protocol used for this application is TCP. Thanks to its error-robustness, the user is able to know exactly what error is happening. The speed of communication is important, but not as important as in video games where the response time really matters. In the case of a chat application, the difference between UDP and TCP is almost imperceptible for the users.
This application is efficient for two main reasons :

- It's developed in C++, therefore it entirely compiled and fast to process
- There are not any big loop (except the main loop for the GUI) and expensive calculations.

Thanks to the signal-slot system of Qt, there is no blocking, and every action triggered gets quickly completed. Therefore the GUI is fluid and is reacting really fast to the user's operations. Here are some screenshots to show how the application looks like.

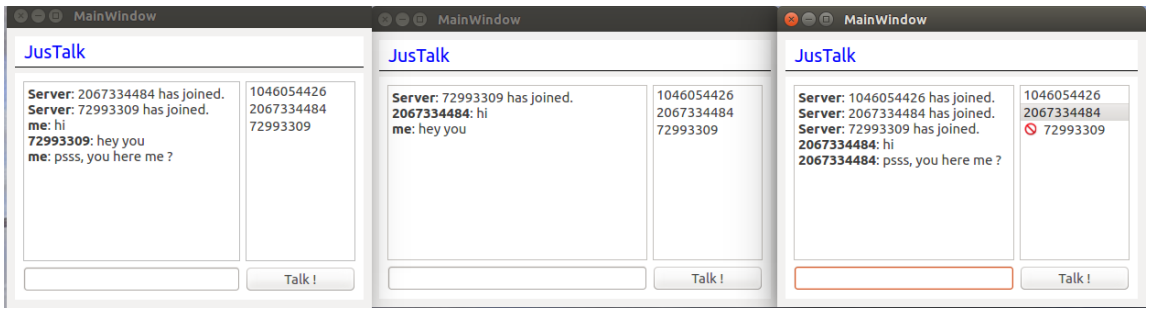

Figure 1: Three clients talking to each others

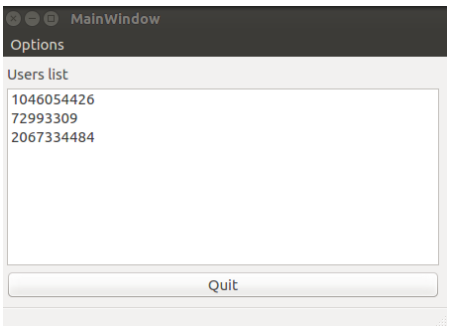

Figure 2: The client's list of users

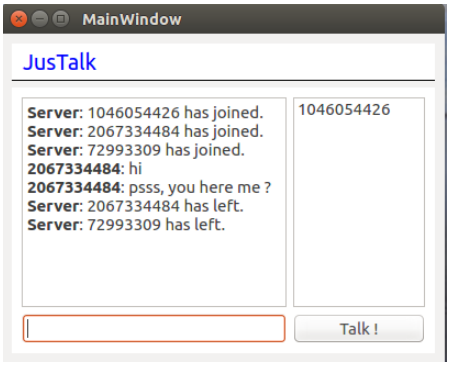

Figure 3: One client is warned when one leaves

IV. Conclusion

To put it in a nutshell : this project is developed in C++ with Qt framework for the GUI. The protocol used for this messaging application is TCP/IP. To run the application, one server must be run, and several clients can connect to this server and then chat with each others. The application's "cool features". Are :

- The possibility for a user to whisper to another
- The possibility for a user to create a blacklist and block messages from other users.