Chat client
Bonus assignment (250 points)
Computer Networks X_400487
Vrije Universiteit

Dennis Andriesse Jesse Donkervliet

February 8, 2018

1 About the assignment

This assignment is an optional part of the course. You can choose to complete the assignment in exchange for points.

The assignments is checked by a teacher or TA as a pass/fail after the assignment deadline. Because there is no resit opportunity for the lab assignments, it is important to verify that your solution works and meets the requirements specified in the assignment.

You can hand in the assignment by uploading it to Canvas. The deadline for the assignment can be found on Canvas and in the course syllabus. Your submission must have the following properties:

1. If the deliverable is only a report, the file format of the report must be PDF. Do not forget to include your names and student numbers in the report.

2. If the deliverable is a report and additional material, such as code, you should hand in a `.zip` or `.tar.gz` archive.

3. If the deliverable includes code, your archive must contain a `run.sh` file in the root of the archive that compiles and runs your code without parameters. If your program requires parameters, it is fine to run the program with default parameters hard-coded in the `run.sh` script.

If you have any questions which you cannot figure out on your own, do not hesitate to contact us. You can post questions on the Canvas discussion forum, or send them to cnvu@googlegroups.com.

2 Introduction

This assignment is about the application layer of the network architecture. In it, you create a simple chat client application using C, C++, Java, or Python. Java offers an easy-to-use Socket API. A short presentation introducing Java Sockets can be found on Canvas.

3 The assignment: a simple chat client

The application you must make is a very simple chat client. The corresponding server has already been made, and is running on IP `52.59.206.71` port `5378`[1].
You can use this server to test your client program.

Your client program must implement a simple communication protocol in plain text. Each message consists of a line of text, which ends with a '\n' symbol (this is automatically so if you use `println()` to send your messages).

[1] You can also find this URL on the Canvas course page.

First, the client must open a TCP connection to the server on the specified port. The server will say nothing until your client sends the first message.

This first message must have the format `HELLO-FROM <name>`, where <name> is any single alphanumeric string which you want to use as your name. If all is well, the server will reply with `HELLO <name>`.

Additionally, the following errors can occur. If any error occurs during the "hello" phase, the server will close the connection after sending you the error message.

If your request had a bad header (i.e., not `HELLO-FROM`), you will receive a response containing `BAD-RQST-HDR`. Similarly, if the payload of your request was invalid, you will receive a `BAD-RQST-BODY` response. This happens if you specify a <name> which does not meet the required format. Also, the name may already be in use, in which case you receive an `IN-USE` response. If the maximum number of clients has been hit, you will receive a `BUSY` response. This should never happen, unless one of you decides to try a denial of service attack, in which case we will have logged your credentials. :-)

Once you have succesfully logged in on the server, you can send two types of messages. Any other type will trigger a `BAD-RQST-HDR` response.

If you send a message containing the line `WHO`, the server will send you a response containing `WHO-OK <names>`, where <names> is a comma-separated list of names of people who are logged in.

If you send a message containing a line `SEND <dest> <msg>`, where <dest> is the name of an online user, and <msg> is a message to send to that user, the server will attempt to forward your message to the correct user. If this succeeds, you will receive a `SEND-OK` reply.

If your `SEND` message does not meet the expected format, you will receive a `BAD-RQST-BODY` response. Also, if you try to send to a user which is not online (anymore), you will receive an `UNKNOWN` response.

If you are logged in on the server, someone may send you a message. In that case, you will receive from the server a `DELIVERY <src> <msg>` message, where <src> is the user who sent you the message, and <msg> is the message that was sent to you.

The goal of the assignment is to make a simple interactive chat client which allows a user to specify a name to log in with on the server, and then allows the user to see who is online, send messages to other users, and receive messages. Don't start from scratch. Use the file `ClientStub.java`, available from blackboard, and fill in the "todo" parts.

To ensure that you can always test chatting to another user, one special user is always online. This is `echobot`, which will echo back to you any message you send to it, so that you can test if your send/receive functionality is working.

In addition, you can use a program like `telnet` to experiment manually with the communication protocol.

4 Requirements

On startup your client must ask the user for a username. It must keep asking for a username until a correct one has been entered. After the user has been logged in, there are three supported commands:

- `!who`. This command prints the list of active users on the server. The output of this command **should be printed on one line**.
- `@<username> <msg>`. This command send a message to the mentioned username, e.g. `@echobot Hello World` should send the message `Hello World` to the user `echobot`.
- `!exit`. Closes the connection to the server and quits the client.

Submit your solution as a zip file. Your submission must include two bash scripts in the top-level directory (i.e. must not be in a sub-directory in the zip):

- `compile.sh` which compiles your program, and installs any dependencies you may have.
- `run.sh` which runs your program, taking two parameter: `host` and `port`. I.e. your program will be run using `./run.sh 52.59.206.71 5378`.

For your convenience, we have set-up a skeleton project for Java which includes a client stub. For this skeleton you do not have to modify the shell scripts. You can find this skeleton at: `https://github.com/VU-Programming/CN-ChatClient`. We will run all submissions on Ubuntu 16.04 LTS.