

Lab 3 – Chat Room

Assistants

Junchao Wang (J.Wang2@uva.nl)

Rex Valkering (r.a.valkering@uva.nl)

Daniël Louwink (daniel.louwink@student.uva.nl)

Daan Kruis (daan.kruis@student.uva.nl)

Lab dates:

Sep. 16 and 21 2016

Deadline

Sep. 22 at 23:59 CEST

Total points

25

This lab must be done individually

Abstract

This assignment focuses on client-server architecture. You will learn how to multiplex sockets to handle multiple simultaneous connections and how to combine socket I/O with other operations.

Preparation

You should already be familiar with the socket module, but you will now also need the select module. You can read the documentation of select at <http://docs.python.org/library/select.html>. You should at least understand the basic use of select: to discover which sockets (or other file descriptors) are ready for writing and reading.

Submission

IMPORTANT: Make sure your code is PEP8 compliant.

You can check your code by installing a PEP8 checker such as the command-line utility `pep8` or <http://pep8online.com>. Points will be subtracted if your code is badly formatted.

Submit your working code (always check if your code works one last time before submitting it, this will save you and the teaching assistants a lot of time) in an archive called `lab3-<last_name>_<first_letter_of_first_name>.zip` (or `tar.gz`). For example, `lab3-Louwink_D.zip`. It should contain the following files:

- `server.py` (task 1 + 3)
- `client.py` (task 2 + 3)

Please do not forget to include your name, your student number and a short description of the program at the top of each file.

Assignment

Instant messaging is *hot*. In this assignment, you will be building a very simple chat room of your own that uses a client-server architecture.

The main assignment is to program a server for a chat room that can handle multiple clients at the same time. You also have to program a client application to communicate with the server. In lab 2 you learned how to make a server that can accept one client at a time. In this lab, you will use the `select` function

from the `select` module to deal with multiple sockets at the same time. The `select` function determines which sockets are ready to read from, ready to write to, or have errors.

On some platforms such as Linux there are more efficient alternatives to `select()`, but these are not cross-platform. Therefore you *must* use `select()`. Conceptually there is no difference, as they all have the same functionality.

Protocol

The client and server communicate using a simple line-based text protocol where clients send commands to a server. Commands are strings terminated with a single `\n` (newline character). The server does not send commands to the client, so the client simply prints out whatever the server sends. Keep the logic in the client to a minimum.

You must implement all of these commands:

<code>/nick <user></code>	Sets the client's nick name, unless it is already taken. The server must also notify all other users of the name change. Initially clients have no name, in which case the server should choose a name for them.
<code>/say <text></code> or just <code><text></code>	Every user receives the chat message.
<code>/whisper</code> <code><user></code> <code><text></code>	Only the specified user receives the chat message.
<code>/list</code>	The user gets a list of all connected users.
<code>/help</code> or <code>/?</code>	Print a useful help message listing the possible commands.
<code>/me <text></code>	Prints an action. For example <code>/me goes to the store</code> would become <code>"<my username> goes to the store"</code> .
<code>/whois</code> <code><user></code>	Print information about a user. You should at least print their address.
<code>/filter</code> <code><word></code>	Filter every message containing <code><word></code> , excluding server messages.

When a client connects, disconnects or changes their name, the server notifies all users.

Task 1 – Server (10 pts)

Remember to use the `select` function to find readable and writable sockets. Use the `server.py` file as a base.

You are free to design the structure of the messages yourself. Remember that the client will need to know when the message was sent, by whom and what their name is. You should provide a basic description of the message format in the comments of your code. Make sure that the structure is such that it is easy for the computer to parse.

Tips

- If a listening socket is readable, it means a client is trying to connect.
- If a socket is readable, but `recv` returns no data (zero length string), it means the other side has disconnected.
- You might find it easier to write your server in an object-oriented style. This will teach you about writing OOP code in Python and it will make your program much easier to read.

- Check the CPU usage of your program. If it is running at 100%, you might be busy-looping, which is not correct in this case.

Task 2 – Client (5 pts)

Program a chat client to communicate with the server using the protocol described above. Instead of using a command line interface, you must use the simple GUI interface provided by `gui.py`. You cannot use blocking socket I/O as that would block the GUI event loop. Therefore you should use the `select` function here as well. Alternatively you can set the socket to non-blocking.

The reason for using a GUI instead of the command line is that the client must do multiple things at once: Receiving/displaying incoming messages as well as letting the user enter text, and that is not as easily handled by a command line. Use the `client.py` file as a base. In the end, you should be able to open several clients at the same time and chat with yourself.

Task 3 – Encryption and Security (10 pts)

To protect your users from eavesdropping, add encryption to your chat program. For example, you might be familiar with the RSA asymmetrical cryptosystem from a course on algorithms or encryption.

Implement wrapping your sockets in a TLS layer like we did with the SMTP server (if you did that part of the previous assignment). As with the SMTP task you won't have to implement it yourself. Just import the `ssl` module and use the `wrap_socket` function from there.

To create a more functional chat protocol, implement the functionality to kick and ban people from the server. This should be a right reserved for certain admin users only.