Lab 8

GUI client

[Compulsory]

Author: Johannes Schmidt 2018-2023

This lab's goal is to advance your skills in socket programming. You will program a chat client for the chat server from the previous lab. It will have a graphical user interface.

8.1 Introduction

The most basic requirements for a chat client are pretty obvious:

1. be able to connect to, and disconnect from, a server
2. be able to enter a text message and send it to the server
3. be able to receive and display text messages from the server

In principle, you already know how to achieve all this. But, as indicated in the previous lab, there is a problem when using the recv() function the standard way: a call to recv() will wait until there is some data actually incoming. This will freeze your program and the user is not able to interact with the user interface in any way. In particular, you will not be able to enter any text and send it to the server. But a chat client *must* of course allow the user to enter and send messages to the server whenever the user wants to. This means the waiting to receive messages from the server must somehow happen simultaneously, or in the background.

As mentioned in the previous lab, there are several possible solutions to this problem. In this lab we explore the method of *polling* for messages. We

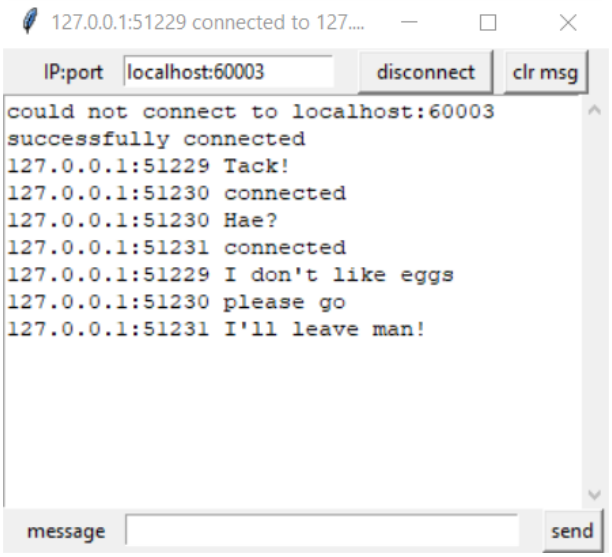
use the recv() function in non-blocking mode. This way we can look whether there are any messages available with a call to recv(), without freezing the whole program. Then, in case there are messages available, we still get them from the recv() function as usual. You set a socket (say `sock`) into non-blocking mode by `sock.setblocking(False)`, which is equivalent to setting the timeout to 0.0 by `sock.settimeout(0.0)`.
When we are in non-blocking mode and we make our call to recv() by something like `data = recv(1024)`, how do we know that there is no data available? In this case the recv() function raises an exception of type `socket.error`. One should of course not let this exception crash the program, but catch it. In case there is data available, we can proceed as usual, `data` will contain the received data.

8.2 Your Task

Download the file *guiclient_ codeskeleton.py* from Canvas and get familiar with the code. It already constructs the GUI for you and even provides a function `pollMessages()` that is called five times a second (try this by writing in its body `printToMessages("hallllo")`). Your task is to fill the four methods `disconnect()`, `tryToConnect()`, `sendMessage()`, and `pollMessages()` with corresponding content so that the GUI client works as intended (confer *most basic requirements* above).

Hints

- When attempting to connect to the server, it may be reasonable to have a non-0 timeout, e.g. 0.1 seconds. When calling recv(), the timeout should be set to 0.0.
- It is desirable, and very helpful for debugging, to print any occurring error/problem into the message field via `printToMessages()`. For instance, if the user attempts to send a message while not connected, it should result in a corresponding message. It could also be helpful to display *why* a connection is interrupted (was it on client's request, server, or some error?). And so on.
- Test your client by connecting at least three clients to your server from the previous lab. Test to connect, disconnect and reconnect them.
- In figure 8.1 is a screen shot of my solution.



Figur 8.1: A screen shot of Johannes' gui client in action.

8.3 Submission and presentation

Submit your python file on Canvas and present your code and program to your lab assistant. Be prepared to explain your code *as well as* the code given in the skeleton.