⌠ Coursework 2: Socket Programming ⌡

Introduction Most operating systems (if not all) provide network capabilities to user applications through interfaces called sockets. From a developer point of view, sockets are an API to the transport layer functionalities. While most libraries dedicated to application layer protocols handle the socket creation and management for you, knowing how they work can help when troubleshooting network communication problems. But sockets are not the only thing required in order to build a software capable of network communication, it also requires an application layer protocol and pieces of code to comply with that protocol. This has to be handled slightly differently depending if the software acts as a client (*i.e.* it initiates communications) or as a server (*i.e.* it waits for clients communications).

Goal In this project, you will have to understand the basics of a simple protocol used for achieving real-time group talk (Internet Relay Chat) and implement a basic client and a simplified server for this protocol. The basic client will be a robot which will reply to simple commands sent to group talk (called channels in IRC) such as '!day' giving the day of the week. When receiving private messages, the robot will also reply but with random nonsense or fun-facts. The server you will implement should be a very light version of a real server which will cover only the most basic functionalities of the IRC protocol: clients should be able to connect, join channels and interact with each other either using channel or private messages.

⟨Part 1⟩ IRC and the project

⌊Section 1.1⌉ *The protocol*

IRC is a text based protocol running on top of a TCP stream (eventually secured by TLS but this is not required in the project). The protocol is human readable and so simple that a human can even use it manually, at least when there is not much talk on channels. It is fully described in the IETF RFC 1459 (https://tools.ietf.org/html/rfc1459).
However, the best way to actually get a good first understanding of it is probably to just run locally a simple server such as miniircd (https://github.com/jrosdahl/miniircd) and connect a real graphical client to it such as Hexchat (https://hexchat.github.io/). This client allows you to see all the raw traffic from/to the socket (*i.e.* the TCP stream) by opening the 'Raw Log' window in the 'Window' menu. You can then type '/connect 127.0.0.1' and see

what is happening while playing with the functionalities of the client. Try to join a channel, talk to it, have a second client connect and do the same,...

⌊Section 1.2⌉ *A simple client (bot)*

The focus of the client you will implement is on network communication and not graphical interfaces, thus you are asked to make a robot able of doing the following task:

- Connecting to a server
- Maintaining the connection to the server
- Joining a channel
- Responding to channel messages starting with a '!' character and more specifically:
  - '!day' with the day of the week
  - '!time' with the time
- Responding to each private message by a random sentence (either nonsense or fun-fact)

As a requirement, the robot program must be running on the Windows operating system.

⌊Section 1.3⌉ *A simple server*

There is no need to implement a full IRC server to understand how sockets works on the server side, thus it should only provide the functionalities of the protocol which are strictly required to achieve the following task:

- Allowing clients to connect, choosing there username and realname
- Allowing clients to join channels
- Allowing clients to talk to others users in a channel
- Allowing clients to talk directly to each other in private

As a requirement, the server must be running on the GNU/Linux operating system.

⟨Part 2⟩ Sockets

Since sockets are provided by operating systems, there specific implementation in low level languages (such as C/C++) are usually dependant on the operating systems although some cross-platform libraries also exist. However, higher level of languages such as Python provide natively in there standard library an interface to sockets which is independent of the operating system.

⌊Section 2.1⌉ *With Python*

The reference documentation of Python socket API is available at https://docs.python.org/3.8/howto/sockets.html. A few other good tutorials, sometime better than the reference documentation, can also be found:

- https://medium.com/python-pandemonium/python-socket-communication-e10b39225a4c
- https://realpython.com/python-sockets/
- https://pymotw.com/2/socket/index.html

⌊Section 2.2⌉ *With C/C++*

Things are more tricky with C/C++, depending if you want a cross-platform solution or not. For the Windows operating system, the standard library to access sockets is called WinSock and its reference documentation is available at https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock. Two alternative good tutorial about it can be found at:

- https://www.binarytides.com/winsock-socket-programming-tutorial/
- https://www.codeproject.com/Articles/13071/Programming-Windows-TCP-Sockets-in-C-for

On GNU/Linux, as on many unix like operating system, the socket API is available as a POSIX specification. The following two tutorial gives the basics for using it:

- https://www.binarytides.com/socket-programming-c-linux-tutorial/
- http://tldp.org/LDP/LG/issue74/tougher.html

The reference documentation is available as a classical man page, a copy of which is at http://man7.org/linux/man-pages/man2/socket.2.html. It is not recommanded to look at it until you are well familiar with all the concepts involved. Another good reference which is more understandable for beginners is available at https://www.gnu.org/software/libc/manual/pdf/libc.pdf pages 479 and next.
A cross-platform solution exists for C++ through the excellent Boost library where sockets are provided in the Boost.Asio module. Its reference documentation is available at https://www.boost.org/doc/libs/1_71_0/doc/html/boost_asio.html and a very well written tutorial is available at https://www.codeproject.com/Articles/1264257/Socket-Programming-in-Cplusplus-u