B5 - Advanced C++

B-CPP-500

Babel

Let the people speak!

3.1

EPITECH.

Babel

binary name: babel_client, babel_server
language: C++

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

The project consists in a client/server architecture voice over IP application, similar to Skype or TeamSpeak.

Generalities

Platforms

The project **MUST** be OS independent. It has to compile and run in a similar manner on **Unix** system and **Windows** systems.
It **MUST** be built using a CMake and dependencies **MUST** be handled using conan.
These, and only these, conan repositories may be used:

- conan-center: `https://bintray.com/conan/conan-center`
- epitech: `https://bintray.com/epitech/public-conan`
- bincrafters: `https://bintray.com/bincrafters/public-conan`

The build of you project will be done in the fashion (for Unix systems):

Terminal

```
~/B-CPP-500> mkdir build && cd build && conan install .. && cmake .. -G "Unix
Makefiles" && cmake -build .
```

PROTOCOL

The **Babel** project aims to create an **SIP**-like Voice Over IP (VOIP) protocol. It MUST be usable over the Internet (no multicast or anything LAN specific).

To test your protocol you MUST write a server and a client implementation.

The protocol MUST be a client/server protocol but voice transport MUST be client to client (the server can have a proxy mode for conference calls or NAT-ed clients).

The protocol MUST be a binary protocol, not a text one. For instance, TCP is a binary protocol and HTTP is a text one.

LIBRARIES

Any non-explicitly authorized library is explicitly forbidden. However:

- You are NOT allowed to use any **SIP** or other VOIP library.
- You MUST use the PortAudio (v190600.20161030) library for anything sound related
- You MUST use Opus (1.3.1) for the compression codec.
- You MUST use Qt (5.14.2) for the client's graphical user interface, or any implementation detail on the client side.
- You are NOT allowed to use any **Qt** library on the server side.
- You are allowed and encouraged to use Boost libraries.
- For storage you are allowed to use sqlite3 and/or sqlite_orm
- **PortAudio** and **Opus** are **C** libraries, so you MUST create your own abstractions to them.

For Qt5, **and Qt5 only**, you can consider that it is already installed on the system. If you still get it working using Conan, you will earn bonus points.

Have a look at Boost ASIO for networking on the server side.

A good idea may be to implement your network abstraction in terms of **Boost Asio** on the server side and in terms of **Qt Network** on the client side. Your network abstraction could also be hand written on each side, but it won't change your grade. Use the tools at your disposal.

Mandatory part

Contents

The following items are mandatory at the end of the project:

- A documentation of your binary protocol for your communications (mandatory for follow-ups!)
- A fully UML compliant class diagram for client and server (mandatory for follow-ups!)
- A network abstraction implemented in terms of **Boost** or custom-made server side, and implemented in terms of **Qt Network** or custom-made client side.
- A **Qt** GUI client side.
- A C++ abstraction of **PortAudio** and any sound-related code.
- A C++ abstraction of **Opus**, and the transmission of compressed sound.
- A contact list.
- The ability to make a call.
- The ability to hang up.

To help you when evaluating your abstractions, think of them this way: could you change your implementation without impacting the rest of the program? For instance, could you use a different audio library and only have to change your sound abstraction's implementation? Another example: could you get rid of **Boost::Asio** and still only have to change your network abstraction's implementation? If the answer is "no", your abstraction is not good enough.

General setpoints

You are (more or less) free to implement the client and server any way you please. However, here are a few restrictions:

- The only authorized functions from the **libc** are the ones that wrap system calls (and don't have C++ equivalents!)
- Any solution to a problem **MUST** be object-oriented.
- Any not explicitly authorized library is explicitly forbidden.
- Any value passed by copy instead of reference or pointer MUST be justified.
- Any member function or method that does not modify the current instance and is not **const** MUST be justified.
- Any code that is deemed unreadable, unmaintainable or with unnecessary performance costs **WILL** be sanctioned. Be rigorous! Write code you'll be proud of!