Task

Write a Qt graphical user interface (GUI) application that receives values of some special functions through a named pipe and draws the corresponding curves.

Application ICS0025SpecialFunPipeServer

Console application *ICS0025SpecialFunPipeServer* is implemented by the instructor. For calculations it needs file *Alglib.dll* (downloaded from http://www.alglib.net/) that must be in the same folder with *ICS0025SpecialFunPipeServer.exe.* You can find the both files in folder *Coursework 3*.

Duplex (i.e. able to read as well as write) pipe created by server is named as \\.*pipe**ICS0025*. The server and the client (i.e. the student's software) must run together on the same computer. The server has no command-line parameters.

The special functions supported by the server are:

- Sine integral (http://mathworld.wolfram.com/SineIntegral.html).
- Cosine integral (http://mathworld.wolfram.com/CosineIntegral.html).
- Bessel functions from the first kind J_0 …. J_{10} (http://mathworld.wolfram.com/BesselFunctionoftheFirstKind.html) .
- Fresnel integral S (http://functions.wolfram.com/GammaBetaErf/FresnelS/introductions/FresnelIntegrals/ShowAll.html).
- Fresnel integral C (http://functions.wolfram.com/GammaBetaErf/FresnelC/21/ShowAll.html).

The server accepts commands from the client in the following format:

1. The first four bytes of a packet are for storing the packet length (i.e. the total number of bytes in packet). It is a regular C/C++ integer.
2. The following bytes specify the type of special function. It is a regular Unicode C/C++ (*wchar_t*) string with terminating two zero bytes at the end. Allowed expressions are:
    a. *"Sine integral"*
    b. *"Cosine integral"*
    c. *"Bessel function"*
    d. *"Fresnel integral S"*
    e. *"Fresnel integral C"*
3. The last bytes are for specifying the range of arguments (regular C/C++ double numbers), the number of points to calculate (regular C++ integer) and in case of the Bessel functions the order (regular C++ integer).

Example: the command for calculation 200 points of Bessel function J_1 in range 0 … 20 must be as follows:

1. Bytes [0:3] – integer 60 as the total number of bytes in package (0x3C, 0x00, 0x00, 0x00).
2. Bytes [4:35] – *wchar_t* string *"Bessel function"* (0x42, 0x00, 0x65, 0x00, ….., 0x6E, 0x00, 0x00, 0x00).
3. Bytes [36:43] – double number 0.

4. Bytes [44:51] – double number 20.
5. Bytes [52:55] – integer 200.
6. Bytes [56:59] - integer 1.

The server performs the calculations and sends the response in the following format:

1. The first four bytes of a packet are for storing the packet length (i.e. the total number of bytes in packet). It is a regular C/C++ integer.
2. The following bytes present the regular Unicode C/C++ (*wchar_t*) string *"Curve"* with terminating two zero bytes at the end.
3. The main part of package consists of double numbers presenting the coordinates of points. Each point is presented with pair {x, y}. There are no separators between points.

If the calculation has failed, the server sends an error message:

1. The first four bytes of a packet are for storing the packet length (i.e. the total number of bytes in packet). It is a regular C/C++ integer.
2. The following bytes present the message text: a regular Unicode C/C++ (*wchar_t*) string with terminating two zero bytes at the end. The first word of message is never *"Curve"*.

The server waits for the client to establish connection. After that the client may start to send commands. There are no timeouts: the server answers as soon as possible.

To close the connection the client has to send the stop command:

1. The first four bytes of the packet are for storing the packet length (i.e. the total number of bytes in packet). It is a regular C/C++ integer.
2. The following bytes present string *"Stop"*. It is a regular Unicode C/C++ (*wchar_t*) string with terminating two zero bytes at the end.

There is no response to the stop command.

To close application *ICS0025SpecialFunPipeServer* type the command prompt command *exit*. To return application *ICS0025SpecialFunPipeServer* to its initial state type command *reset*.

Requirements

The GUI must have widgets for (see also the Appendix):

1. Selecting the curve type.
2. Specifying the curve parameters (range, number of points, order of Bessel function).
3. Establishing and breaking of the connection with server.
4. Sending commands to the server.
5. Viewing the curve.
6. Logging the user actions (connecting, disconnecting) and local socket errors.
7. Showing the messages from *ICS0025SpecialFunPipeServer*.
8. Exiting the application.

The GUI must have a correct user-friendly look and feel and keep it when the user resizes the main window. The application should dynamically enable and disable the widgets giving some guidance to the

user and prohibiting actions which are inappropriate at the current moment. If the user resizes the window, the GUI must keep its normal look.

The server tries to check the correctness of input parameters but its capabilities to do it are rather restricted.. The student's application must not allow the human operator to input and send incorrect data.

The students are free to select between QtDesigner and QML.

Tips

1. The wizard-created Qt project file *.pro contains row:
   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
   Complete it:
   greaterThan(QT_MAJOR_VERSION, 4): QT += widgets network printsupport
2. Qt has its own standard class for named pipe: *QLocalSocket*. See:
   a. https://doc.qt.io/qt-5/qlocalsocket.html#details
   b. https://stackoverflow.com/questions/18945407/is-qlocalsocket-really-used-for-namedpipes (this webpage presents a very useful example).
3. The *QLocalSocket::waitForConnected* and *QLocalSocket::waitForDisconnected* may not work properly. If so, simply do not use them. Even more: as the server and client are running in the same computer, the calculations are fast and there are no timeouts. Consider are the threads in this application needed or not.
4. To get the *QLocalSocket* errors, connect the signal and slot in the following way:
   connect(pointer_to_qlocalsocket,
   QOverload<QLocalSocket::LocalSocketError>::of(&QLocalSocket::error),
   pointer_to_slot, slot_method);
5. For drawing the curves use freeware from http://www.qcustomplot.com/. Download the widget files (*QCustomPlot.cpp* and *QCustomPlot.h*) and insert them into your project. All the material you need is on pages:
   a. https://www.qcustomplot.com/index.php/tutorials/settingup
   b. https://www.qcustomplot.com/index.php/tutorials/basicplotting
   c. Would you want to make you application more interesting, read also https://www.qcustomplot.com/index.php/tutorials/userinteractions
6. **IMPORTANT**: at this moment (February 2022) *QCustomPlot* does not support Qt version 6.2. Use Qt versions 5.15 or 5.12, see https://download.qt.io/official_releases/qt/.
7. You cannot manage without Qt classes *QString* and *QByteArray*. Study them and only after that start to write code.
8. Do not forget that C and C++ standard functions are all declared in namespace *std*.
9. Do not use C-style casts: the QtCreator will send warnings.
10. In testing it is advisable to calculate 200 points from range 0…20. However, for cosine integral do not apply arguments close to 0.

Evaluation

The deadline is the end of examinations in June. However, it is strongly advised to present the results of coursework during the semester. The students can do it after each lecture.

The GUI application works perfectly if all the requirements specified above are met.

A student can get the assessment only if he / she attends personally. Electronically (e-mail, cloud, GitHub, etc.) sent courseworks are neither accepted nor reviewed. Presenting the final release is not necessary. It is OK to demonstrate the work of application in QtCreator environment.

Appendix: GUI example