Task

Update the software implemented in Coursework 1 with the possibility to receive new items through a named pipe. Tip:

- First study https://docs.microsoft.com/en-us/windows/desktop/ipc/pipes. Everything you need to know about pipes you can find on this website.

Application ICS0025PipeServer

Console application *ICS0025PipeServer* is implemented by the instructor. File *ICS0025PipeServer.exe* is in folder *Coursework 2.* The server uses also file *Birds.txt* from folder *General*. Files *ICS0025PipeServer.exe* and *Birds.txt* must be in the same folder.

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

The server sends the description of items as regular C strings in the following format:

group_name subgroup_number <item_name> day month year0

for example:

A 2 <Great Crested Grebe> 29 Nov 2018

There is just one space between components. The item name enclosed into angle brackets may include any number of spaces and any characters except < and >.

The server waits for the client to establish connection (i.e to open file \\.*pipe**ICS0025*). After that the client must send to the server (i.e. write into file) message *ready* (as a regular C string). The server responds with description of an item. The client reads the response string from the pipe file, parses it, creates the new item and inserts it into data structure implemented in courswork 1. After that the client sends message *ready.* To disconnect the connection the client must instead of *ready* send message *stop*.

The server may not respond to client's *ready* message immediately. The pause may be up to 10 seconds. However, the client should respond as soon as possible. If the client has been silent more that 10 seconds, the server closes the connection.

To close the server type command *exit*. To return the server to its initial state type command *reset*.

Requirements

1. The client application is controlled by commands typed by user:
    a. *connect* opens the pipe file and sends the first *ready* message.
    b. *stop* sends the *stop* message (see above) and closes the pipe file. After this command the client application must stay active and the user must be able to type command *connect* once more.

    c. *exit* forces the client to print the contents of data structure and exit the application. The client application must obey this command at any moment.
2. The *ready* messages except the first one must be sent automatically.
3. The client's software must all time be under the user's control. It means that:
    a. The keyboard must never be blocked: the user should be able to type commands at any moment. Consequently there must be at least three threads: one for listening the keyboard, one for reading data from the server and one for sending messages to the server. Reading from and writing to server must be asynchronous.
    b. If the server exits or closes the connection, the client must inform the user but continue to run.

Tips

- Print a lot of debugging info: when the threads are starting and stoping, how many bytes of data has arrived or sent, text of messages that were read or written, etc.
- Follow the server's command prompt window – there is also a lot of debugging info.

Test cases

The client application works perfectly if the following sequences of commands are performed correctly and according to the specification:

1. Switch on the server → launch the client → connect → let to work a bit → stop → connect → let to work a bit → exit.
2. Launch the client → exit.
3. Launch the client → connect → let to work a bit → switch off the server → *connect* → exit.
4. Switch on the server → launch the client → *connec* → connect → let to work a bit → *connect* → let to work a bit → stop → *stop* → exit.

Remark: the client application must respond with error messages to commands printed in *red*.

Evaluation

The deadline is the last lecture on week 16. However, the students can present his / her results earlier.

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 Visual Studio environment.

The coursework is approved if the test cases presented above run successfully.