TPC#9 : Client & Server !

Table des matières

1 Rendering architecture

The rendering architecture does not change much from the one you usual see. Once again, `login_x` is to be replaced by your own login.

```
rendu-tpcs9-login_x.zip
    - login_x/
        - AUTHORS
        - Client/
           - Client.sln
           - Client/
        - Server/
           - Server.sln
           - Server/
```

2 Introduction

In this practical work, we are going to introduce you sockets, threads and other general network notions. You are also going to review notions previously seen (Stream, Object). Therefore, you are going to achieve a little chat and also a sever for this chat.

"The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.[1] Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests." - Wikipedia

3 Sockets and Threads

3.1 Sockets

3.1.1 Course et examples

Before we begin our chat, you need to learn new notions. You are going to see few examples to learn how to use these tools, then you will be able to begin the chat.
A socket is a software interface between the operating system services. But who says "socket" does not mean network : we can use sockets to communicate between the applications of a computer. To establish a connexion in C#, we use the Socket class. We will then be able to get streams from this object to communicate. To establish this connexion, you have to know what the remote address (the server) and the port are. The port is a simple number representing a service. We often match a port with a protocol.

```
using System.Net.Sockets;
```

The constructor of a socket takes 3 parameters :

- The address type, here we will use IPv4 : AddressFamily.InterNetwork
- The socket type : SocketType.Stream
- The connexion protocol used between the two computers. We will use the TCP protocol, it is the most used protocol for this kind of application, it allows to keep a connexion between two computers, to check that the recipient did receive the message, and lots of other things too. Use then ProtocolType.Tcp.

Once you socket is initialized, you can establish a connexion thanks to the Connect method of your socket.
Here is an example of code using sockets to send "Hello" to Google (take note of the use of the Stream class, that you will have to perform for the next part).
Do not forget to flush streams and to close the sockets.

```
Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
s.Connect("www.google.fr", 80);
Stream stream = new NetworkStream(s);
stream.WriteByte((byte)'H');
stream.WriteByte((byte)'e');
stream.WriteByte((byte)'l');
stream.WriteByte((byte)'l');
stream.WriteByte((byte)'o');
stream.WriteByte((byte)'\n');
stream.Flush();
s.Close();
```

3.1.2 Exercise

Create a new Console project named MyBrowser. In this project, you are going to connect to a HTTP server and to display the answer of the server. The goal is to get a web page.

- The website's name has to be retrieved as a parameter on the command line (use args).
- The tested website will be of this form : "www.example.com".
- Use the 80 port, which is the HTTP port. It is on this port that the browser connects to the websites to get webpages.
- To get a webpage, you will have to send the following message :
  "GET /[pathtoressources] HTTP/1.1\nHost : [host]\nConnection : close\n\n" You can content yourself with copy/pasting this line, but you will have to replace [host] by the name of the website without the "www." prefix.
  For example, for http ://www.google.com, you will have to put :
  "GET / HTTP/1.1\nHost : google.fr\nConnection : close\n\n"
  You can consult the Wikipedia webpage about the HTTP protocol to learn more about it because it won't be the subject of today's practical work.
- Use a Console.Read() at the end of your main method in order to be able to see what is displayed.
- Use the StreamReader and StreamWriter classes.
- As an example for this practical, you can try to get the file browser_test : perso.epita.fr/deabre_p/browser_test

3.2 Threads

You will need to use threads for your chat.

3.2.1 Course and examples

What is a thread ? A thread is a task that can be achieved in parallel with other tasks. The execution of your program will be splitted in many subtasks. For now, we only have one thread (only one task). With an other thread, we could do more. We will handle the reception of the messages of the chat on another thread. Threads are especially useful for graphical interfaces, it allows to compute heavy calculation on one (or more) thread and keep making working the graphical interface on another thread. The constructor takes as parameter a ThreadStart object, that takes itself a delegate (this notion will be discussed later, simply pass the name of your method without parenthesis). Now just call the Start method of our object in order to launch the thread.
Code example :

```
static void Main(string[] args)
{
Thread t = new Thread(new ThreadStart(MyThread)); // Creation of the Thread object.
t.Start(); // Call of this method calls the "MyThread" method.
Thread.Sleep(500); // Pause the main thread for 500 ms.
t.Abort();\\
}
\\
static void MyThread()
{
// Function that will be launched for each thread.
}
}
```

3.2.2 Exercise

Create a new Console project named MyFirstThread. Create a static method that displays 100 times "Thread 1" and another displaying 100 times "Thread 2".
In the main method, create two threads each one executing one of the two methods.
This simple exercise will make you understand how threads execute themselves.

4 The client

We are now going to achieve the client part of this practical work. The client part of your application is the software that will connect to the chat's server. In order to test this part, ask your assistants to launch their server and that they give you the IP address of their computer.

First begin with making a new console project that we will name Client. Create a new class named Client too. This class has to contain the following attributes :
- The name of the user trying to get connected
- A socket
- A StreamWriter
- A StreamReader

You can add other attributes if it seems useful to you.

First, you will have to provide a constructor without parameters that initializes the socket and the following methods :

```
public void Connect(string host, int port, string name);
public bool IsConnected();
public void Run();
public void Read();
public void Write();
public void Close();
```

- Connect : connects to the server asked thanks to the host and the port passed as parameters. You can handle the case where the connexion won't succeed thanks to a SocketException and display an error message. During the connexion, you first have to send your name to the server. Tehn, if the received message on the Stream is "Welcome", then we will display the following message : "Confirmation : Welcome", else we will display "You're blacklisted, get lost".
- IsConnected : returns true if the socket is connected to a remote host. There is a method of the Socket class allowing to check this information.
- Run : creates two threads (you will pass as parameter of the ThreadStart object the Read method to one and Write to the other, defined below), launches them and then makes sure that threads block/wait until the thread that the object represents ends, for this we will use the Join method. It will then call the Close method (see below).
- Read : if there is data to read, displays this data (that are on the stream) on the console. For this, take a look at the Poll method of a Socket.
- Write : reads on the console the message that will be sent to the server, puts it on the stream. Do not forget to flush this last one at the end.
- Close : flushes the StreamWriter, close this one, the StreamReader and the socket.

5 The server

Now that the client is done, we will create the server. For the server, you will create a new Console project because the client and the server are two different parts (there will be two different executables, you will launch the server on one side and the client on the other !). Name this project Server. We will split this part en two subsections : users handling and the treatment and transmission of messages to the users.

5.1 The Client class

You are going to create a Client class. This one should have the following attributes :
- A name, that we will be able to get but not set (use getters and setters).
- A host name
- A port number
- An associated socket
- A StreamReader
- A StreamWriter

This class will have a constructor taking a name (for the user), a host name, a port number and a socket. It will have the following methods too :

```
public void Send(string message);
public string Receive();
```

Send writes the message given as parameter on the stream (do not forget to flush).
Receive reads a message on the stream and returns it. If an error occurs, the method returns null.

5.2 The Server class

You will now create a Server class. Our server will have two attributes : a socket that is associated to it and a LinkedList to stock the different connected clients in order to handle several clients.
The constructor will take as parameter the port. In this constructor, we initialize the socket. We want to accept a connexion. In order to do this, we will use the Bind and Listen methods. The Bind method takes as parameter an IPEndPoint that takes itself an IPAddress.Any as parameter. Take a look at how to use the Listen method, it will be useful. We also initialize the client list. You have to do the following methods :

```
public void Run();
public void AcceptClient();
public void Chat();
```

- Run : creates two threads : one to launch AcceptClient (see below) and another one to launch the Chat method (see below).
- AcceptClient : it waits for a client connexion thanks to the Accept method that returns a Socket type object and that represents the link between the client and the server. We will create here our client (we access to the necessary informations thanks to returned socket), add it to our list and send "Welcome". We will also write on the console "Connexion from [client name]". "Client name" is to be replaced by the name of the client of course.
- Chat : This method will write the messages that will be sent by the different clients (those who are connected to the server obviously) on the console the following way : "[client name] : [message]". Each client will also receive this message (this is quite the purpose of a chat !).