TP C#11 : MyTinyIrc

Assignment

Archive

You must submit a zip file with the following architecture :

```
rendu-tp-firstname.lastname.zip
|-- firstname.lastname/
    |-- AUTHORS
    |-- README
    |-- MyTinyIrc/
        |-- MyTinyIrc.sln
        |-- Client/
            |-- Everything except bin/ and obj/
        |-- Server/
            |-- Everything except bin/ and obj/
```

- You shall obviously replace *firstname.lastname* with your login (which is *firstname.lastname*).
- The code must compile.
- In this practical, you are allowed to implement methods other than those specified, they will be considered as bonus. However, you must keep the archive's size as light as possible.
- Do not submit any test image ! We will test your code with our own pictures.

AUTHORS

This file must contain the following : a star (*), a space, your login and a newline.
Here is an example (where `$` represents the newline and ␣ a blank space) :

```
* firstname.lastname$
```

Please note that the filename is AUTHORS with NO extension. To create your `AUTHORS` file simply, you can type the following command in a terminal :

```
echo "* firstname.lastname" > AUTHORS
```

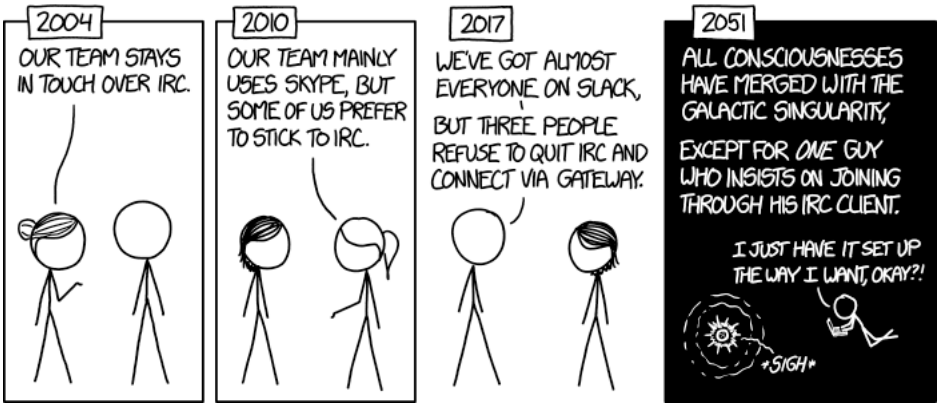
README

You must write in this file any comments on the practical, your work, or more generally on your strengths / weaknesses. You must list and explain all the boni you implemented. An empty `README` file will be considered as an invalid archive (malus).

1 Introduction

1.1 Objectives

Many of you know Messenger, Skype, Discord or Slack. But the real ones know the existence of the precursor of the Internet chat, the IRC protocol. IRC, or Internet Relay Chat, was designed in 1988 by Jarkko 'WiZ' Oikarinen. The messages were relayed by an IRC server from a point A to a point B. You can connect to the EPITA IRC network 'irc.rezosup.org' using a client like X-Chat or WeChat.



2 Courses

2.1 It's dangerous to go alone, take this

This practical work is known as being quite hard because of the network concepts which can be tricky to understand. We tried to make it as easy as possible but in order to make it to the end, you will have to read the documentation. Do no hesitate to Google your questions and search for the MDSN documentation.

A lesson without pain is meaningless – Edward Elric, Full Metal Alchemist

2.2 Sockets

The whole Internet is based on the IP protocol. It allows you to visit websites, send mails, play online, ... However, as IP protocol is hard to use, sockets were created for us, developers, in order to make our job easier.

A socket is an connection interface that's going to handle the communication between two processes in a computer or between two computers on a network. As the communication is possible in both ways on the same socket, it's possible to send and to receive data at the same time.

In order to use Internet sockets, you have to understand the basics of the IP protocol. On Internet each machine has to be identified to communicate with others. It's like a postal address. The IP address identifies the machine and the port identifies the application using the socket. If you prefer an IP address would be the postal address of a building and the port would be the apartment number. `<ip_address>:<port_number>`, for example `127.0.0.1:4242` or `192.168.0.1:1337`. The IP address 127.0.0.1 is special - it represents the machine itself. Use it to do your local tests.

When a developer needs to send data to another machine, he creates a socket with three main characteristics.

The local address of the machine sending the information and the port number of the application using the socket.

The distant address of the receiving machine and its port number.

The transport mode which affects the integrity and the speed of the transmitted data. There are two main transport modes.

— **TCP** (*Transmission Control Protocol*) which gives priority to the integrity of the transmission and assures that they will be delivered to the right machine, in the right order and intact.

— **UDP** (*User Datagram Protocol*) which gives priority to the speed of the transmission. This mode is mainly used in online video games because some data might be lost but the transmission has to be fast. This is the main cause of the lag.

2.3 The `Socket` class

The framework `.NET` does most of the work for you and gives you a group of class to send data through the network. The one that we are really interested in is the `System.Net.Sockets` class. [1].

You will have to instantiate the socket class first with the following constructor.

```
Socket (AddressFamily, SocketType, ProtocolType);
```

— **AddressFamily** represents the IPv4 address of the machine (IPv6 is an interesting alternative, but we're not going to use it here). To get it, use `AddressFamily.InterNetwork`.

— **SocketType**, which is the type of the socket. We are going to use `SocketType.Stream`.

— **ProtocolType** which is the kind of protocol to use. We are going to use `ProtocolType.Tcp`.

2.3.1 The client socket

Once your socket is declared, you need to connect to the server. To do so, you have to use the `Connect` method of `Socket`. Once the socket is connected, you have only to send the data to the server.

```
void Connect (IPAddress address, int port);
```

2.3.2 The server socket

On the server side, you have to accept connections. To do sp, you will have to use the `Bind`, `Listen` and `Accept` methods.

```
void Bin (IPEndPoint endPoint);
void Listen (int maxWaitList);
Socket Accept();
```

`Bind` is going to bind the socket you declared to an entry point (or endpoint, it's going both ways remember) on your computer. `Listen` is going to listen the endpoint of the socket and wait for any connection. When a connection is established, the connection will go in the socket waiting queue. `Accept` is going to accept a pending connection in the waiting queue of `Listen` in order to handle it. It returns a socket directly connected to the client. Once the connection is established you just have to wait for the data.

1. `https://msdn.microsoft.com/fr-fr/library/system.net.sockets.socket.aspx`

2.3.3 Send and receive data

To send and receive data with the socket, once the connection is established, you have to use `Send` and `Receive` methods.

`Send` takes an array of bytes as argument, which contains the data to send and returns the number of bytes that were sent.

`Receive` takes an array of bytes as argument which will be filled with the received data. The array has to be big enough to contain all the data. This function returns the number of bytes received.

2.3.4 Bytes array

Of course, you never used the `byte` type until now. Indeed, you know how to use the `char` and `int` types and a lot more but to send data with `Send` or read it once received you have to convert these types.

To convert a `string` to a byte array, you can use the `GetBytes` method. The operation is reversible with `GetString`. Those methods are in `System.Text`.

```
byte[] bt = System.Text.Encoding.UTF8.GetBytes("Shepard, a platinum ACDC");
string str = System.Text.Encoding.UTF8.GetString(bt);
```

For the other types, you can use `BitConverter`.

```
byte[] btInt = BitConverter.GetBytes(1337);
byte[] btFloat = BitConverter.GetBytes(13.37);
byte[] btChar = BitConverter.GetBytes('a');
byte[] btBool = BitConverter.GetBytes(true);

int c = BitConverter.ToInt32(btInt, 0);
float a = BitConverter.ToSingle(btFloat, 0);
char f = BitConverter.ToChar(btChar, 0);
bool e = BitConverter.ToBoolean(btBool, 0);
```

2.4 Matroska

It's possible to create multiple projects in only one solution. When you create a new solution with MonoDevelop, it automatically creates a project. To add a new project click right on the solution in the solution tree and select `Add`, then `Add New Project`.

As there is two projects in the same solution, MonoDevelop compiles the active one in bold in the solution tree. To change the active project, right click on it and select `Set As Startup Project`.

Of course, your solution will have to contain two projects, `Client` and `Server`.

2.4.1 Keep Calm and RTFM

We insist that you must read the documentation before asking any question. You should probably use `try` and `catch` for your network operations.

> *Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time.* – Thomas A. Edisson

3 Exercises

3.1 MyTinyIrc

We are now going to write our own chat client. You will create a client able to send a message or a file to a server. Look at the examples bellow to understand the concept.

I promise I will tell you when it's time to panic – John Reese, Person of Interest

```
> ./Client.exe 127.0.0.1 4242 data.txt
Connected to 127.0.0.1:4242
File sent successfully !
> ./Client.exe 127.0.0.1 4242
Connected to 127.0.0.1:4242
>> I am mad scentist !
>> It's so cool !
```

```
> ./Server.exe 4242 received.txt
Incoming connection from 127.0.0.1:4242
File received successfully !
> ./Server.exe 4242
Incoming connection from 127.0.0.1
From Client : I am mad scientist !
From Client : It's so cool !
```

3.1.1 Threshold 0 : Arguments

In this threshold your program should be able to take arguments. You can also customize your own error and log messages as long as you stay within the subject.

The client has to accept two arguments, the IP address of the server and its port number. A third optional argument is the name of the file to send. If the arguments are not correct, you have to display the correct usage of the binary. Of course you have to check the validity of the IP address before trying to connect. If any problems occurs you have to display an error message

```
> ./Client.exe 127.0.1 4242
/*Send the message to the server*/
> ./Client.exe 37.187.3.40 4242 data_send.txt
/*Send the content of data_send.txt to the server*/
> ./Client.exe
/*Bad usage of the binary*/
Usage : Client.exe ip_address port_number [file]
```

The server need one argument which is the port number to listen to. It can eventually accept a second argument which is the name of the file where the received data will be written (it could be helpful when someone sends you a file). Again, you have to display an error message in case of any problem.

```
> ./Server.exe 4242
/*Wait to receive data on 4242 port*/
> ./Server.exe 4242 data_received.txt
/*Wait for data on 4242 port and write them to a file*/
> ./Server.exe
/*Bad usage of the binary*/
Usage : Server.exe port_number [file]
```

3.1.2 Threshold 1 : Server.exe

You will code a server able to initiate a socket, to bind it on the address and port given in parameter. It has to wait for an incoming connection. Don't forget to display an error message in case there is a problem [2]. You will also display a message when a client is successfully connected.

```
> ./Server.exe 80
Could not bind 127.0.0.1:80 : Port already in use
> ./Server.exe 4242
Incoming connection from 127.0.0.1
```

We advise you to implement the following methods :

```
Server (int port, string filename = null);
/*Initialize the socket*/
Run ();
/*Bind the socket and listen for connections*/
```

3.1.3 Threshold 2 : Client.exe

In this threshold, your client must be able to send a message to the server. It must show the failure or success of the operation.

```
> ./Client.exe 127.0.0.1 666
Could note connect to 127.0.0.1:666 : Is the server running ?
> ./Client.exe 127.0.0.1 666
Connected to 127.0.0.1:666
```

We advise you to implement the following methods :

```
Client (IPAddress address, int port, string filename = null);
/*Initialize the socket*/
Run ();
/* Connect the socket */
```

3.1.4 Threshold 3 : Exchange

In this threshold you must be able to send or receive data. For the client, when no file is specified in argument, you have to read on standard input and to send it to the server. For the server, when no file is specified in argument, you have to write the received data on the standard output.

2. Protip : `try { ... } catch { ... }`

```
> ./Server.exe 1337
Incoming connection from 127.0.0.1
From Client : Chaussette
```

```
> ./Client.exe 127.0.0.1 1337
Connected to 127.0.0.1:1337
>> Chaussette
```

3.1.5 Threshold 4 : File

For this threshold you have to be able to send a file from the client to the server. When a file name is specified to the client, it must send its content to the server instead of reading the standard input. When a file name is specified to the server, it has to write the received data in the file instead of writing it on the standard output. Of course you should display an error message if the file does not exist.

```
> ./Server.exe 1337 output.txt
Incoming connection from 127.0.0.1
File received successfully !
> cat output.txt
El Psy Congroo
```

```
> cat file.txt
El Psy Congroo
> ./Client.exe 127.0.0.1 1337 file.txt
File sent successfully !
```

3.1.6 Threshold 5 : Ghost Protocol

In this threshold, you must send to the server a timestamp and the pseudo of the client. We let you include a pseudo argument (with `-p` as argument of your client). is this pseudo is not defined, you must use the default pseudo for your client, which is `root`. You are free to implement this threshold as you want but we suggest you to create a tiny protocol in order to send to the server the pseudo and timestamp of each message.

3.1.7 Threshold 6 : Bonus

You are free to implement any bonuses you want. Do not forget to ask your ACDC to validate your ideas. Here is a non-exhaustive list of bonuses...

— IRC-like options to change pseudo and colors on-the-fly.
— A server able to recycle connections from the Client.
— A server capable of being connected to multiple clients at the same time.
— A P2P chat.

The code is the Law