Worksheet 2 – Files, Sockets and ProtocolSI

.NET Classes required for practical classes

Covered topics:
- Binary files
- Sockets TCP
- ProtocolSI

©2021: {rui.ferreira, nuno.costa,vitor.fernandes,ricardo.p.gomes,nuno.reis, marisa.maximiano}@ipleiria.pt

1. Binary Files

The following exercises are intended to show how you can manipulate a file in binary mode.

Exercises

1. Download the file **security.jpg** from the class page and save to directory "c:\temp".

2. Create an application, *Windows Forms Application*, which copies the file "security.jpg" to "bak_security.jpg" partially, i.e., reads and writes N (20480) bytes before proceeding to the next iteration:

    a) Use the class *FileStream*, methods .Read() and .Write();

    b) and, if necessary: *FileInfo* and *File*.

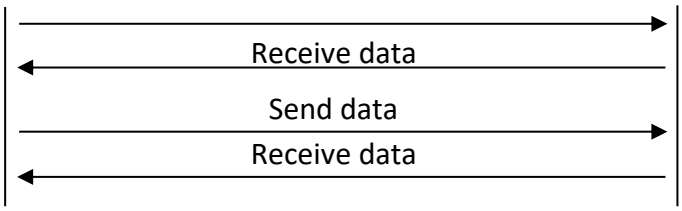
2. Sockets

Below are presented exercises that will allow the use of the TCP protocol for the exchange of information through network and to check some of the properties of this protocol.

To avoid problems in the use of the TCP protocol, the way of communication to be used during classes should be as follows: each packet sent to the receiver **must be** received, a packet coming from that receiver:
NOTE: all the applications to be elaborated in the practical classes will be 1 to 1, meaning that there will be only one client and one server.

A **B**

Send data



Exercises

1. Develop Client / Server scenario, *Console Application*, using 9999 as the port and that use the following classes:

   Server:

   - *TcpListener*, *TcpClient* and *NetworkStream* (.*Read* and .*Write* methods)

   Client:

   - *TcpClient* and *NetworkStream* (.*Read* and .*Write* methods)

   Both:

   - *Encoding.UTF8.GetString* (byte[]) and *Encoding.UTF8.GetBytes*(string)

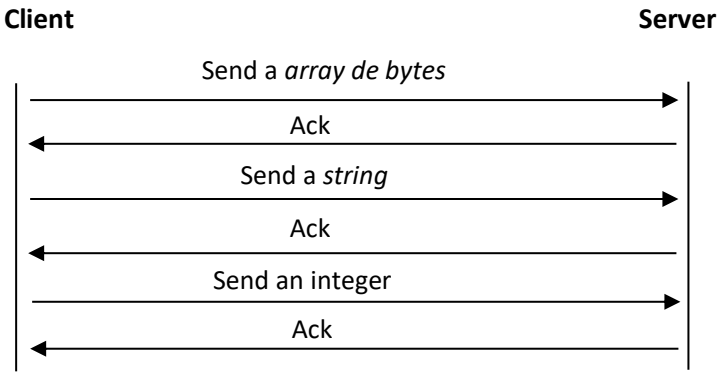
2. Implement the following exchange of information between client and server, so the server shows, in the terminal, information sent by the client:

   a) Sending an *array* de *bytes*;

   b) Sending a *string*;

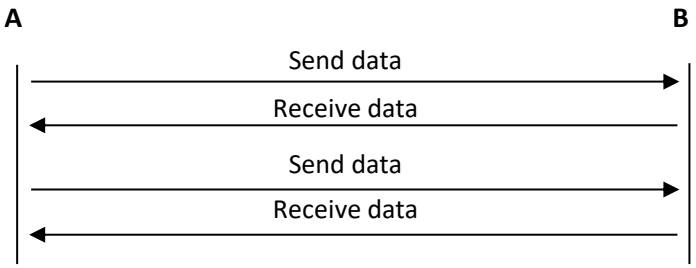
   c) Sending an integer.

   IMPORTANT: Ensure that applications take caution with synchronization of communication between client and server, so that the following protocol is implemented:



3. ProtocolSI

The *ProtocolSI* is structured in a very simple way and is intended only to optimize the way data is sent and received in the TCP data layer. All TCP attributes are still present, so it is necessary to keep the following particularity: each packet sent to the receiver **must be** received a packet coming from that receiver:



The following are the main characteristics of the protocol:

- The maximum size, in bytes, of each package is: 1403.

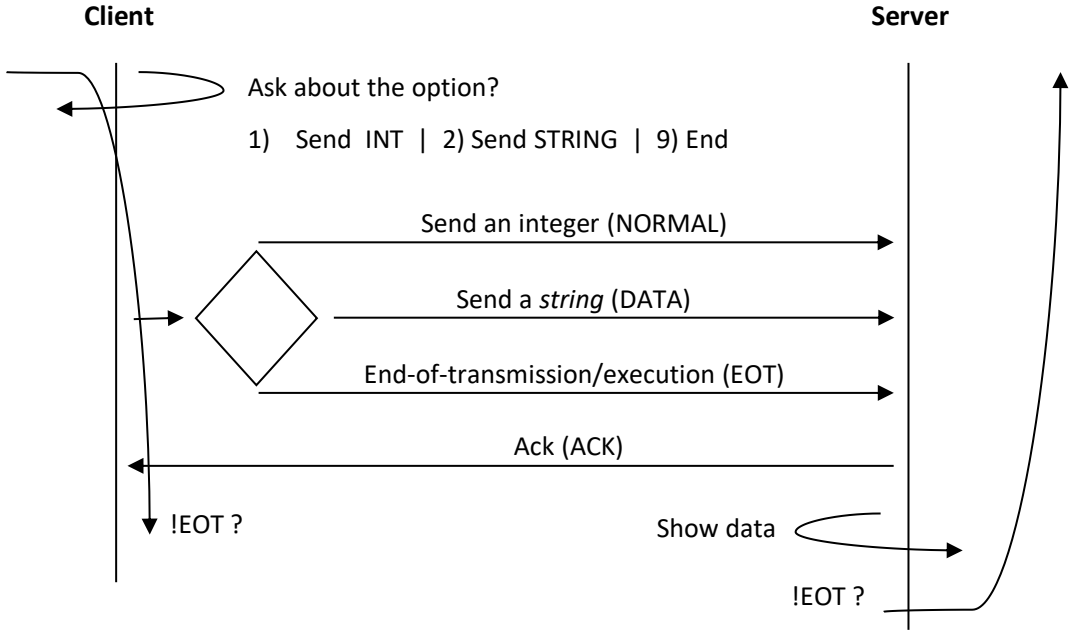
Command	Length of the data	Data
1 byte	2 bytes	until … 1400 bytes

- The first byte [**command**] indicates the command. The command meaning the purpose that we want to assign to the packet. There are several commands available: NORMAL, ACK, NACK, DATE, etc. (see documentation. CHM file);
- The second and third byte [**length of the data**] indicates how many bytes of data comes next;
- The remaining bytes [**data**] are the information you want to send / receive. In this implementation only can be sent up to 1400 bytes per packet.

The implementation of the protocol is provided in a DLL file, which is found in *moodle*. To use it, it is necessary to import the same for each project where it is needed.

Exercises

1. Develop a new client / server scenario for using the **ProtocolSI** as showed in the next protocol:



4. Extra Class

1. Using the project from exercise 1:
    a) Create a ProgressBar that shows the progress based on the bytes copied.
    b) Create a file named “log.txt” in the “c:\temp” directory and use it to register the copies made by the application, including the number of copied bytes (hint: use the StreamWriter class and the File.AppendText() method to create the log).

2. Using the project from exercise 2:
    a) Add the option of sending a *string* encrypted with the Caesar cipher using a shared key. (https://pt.wikipedia.org/wiki/Cifra_de_César or https://en.wikipedia.org/wiki/Caesar_cipher)

3. Using the project from exercise 3, simulate a Man-In-The-Middle (MITM) scenario as follows:
    a) Create a new *Console Application* named MITM;
    b) Change the port used by the Client to 9998, in order to implement the following communication scheme:

        `Client [9998] ←→ [9998] MITM [9999] ←→ [9999] Server`

    c) Implement in MITM a simple forward of requests among the remaining elements, registering to the console what each one receives / sends.