2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference (TAEE) | 978-1-6654-2161-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/TAEE54169.2022.9840584

TCPConex: library to create TCP/IP communication applications

Juan Cerezo Sánchez
Inst. Univ. de Microelectrónica Aplicada (IUMA). Univ. Las Palmas de G.C. (ULPGC).
L.P. de Gran Canaria, Spain.
ORCID: 0000-0002-2914-170X

Sonia León del Rosario
Inst. Univ. de Microelectrónica Aplicada (IUMA). Univ. Las Palmas de G.C. (ULPGC).
L.P. de Gran Canaria, Spain.
ORCID: 0000-0001-8998-455X

Carlos Vega García
Inst. Univ. de Microelectrónica Aplicada (IUMA). Univ. Las Palmas de G.C. (ULPGC).
L.P. de Gran Canaria, Spain.
ORCID: 0000-0002-5629-1471

Aurelio Vega Martínez
Inst. Univ. de Microelectrónica Aplicada (IUMA). Univ. Las Palmas de G.C. (ULPGC).
L.P. de Gran Canaria, Spain.
ORCID: 0000-0002-4154-8799

***Abstract*—A library is presented that makes it easier for students to create TCP/IP communications applications. Even though the library offers elementary and high-level usage functions, additionally, resolves all asynchronous call sequences with low-level sockets.**

Keywords— Library, Communication Application, Sockets, TCP/IP

I. INTRODUCTION

During the development of communication applications over TCP/IP, the problem of managing the sequence of calls to low-level functions that work with sockets arises. This issue may be solved by using blocking functions or asynchronous functions. The library presented in this article handle this problem by using an asynchronous approach for managing the sockets [1] [2].

The developed library, *TCPConex*, release the programmer from managing calls to the socket functions. With this library, the student or programmer who wants to create a TCP communication application may leave out the low-level functions and only needs to focus on the high-level functionality of the application, that is, the exchange of messages.

II. THE TROUBLE

A. Creation of communications applications

In the Industrial Informatics subject of the Electronic and Automatic Engineering Degree, communications applications are work focus. One of the laboratory tasks consists of the development of an integration application that needs to establish TCP connections between client and server (Fig. 1) The development has to be done in C# language with the .Net Framework library on Microsoft's Visual Studio environment.

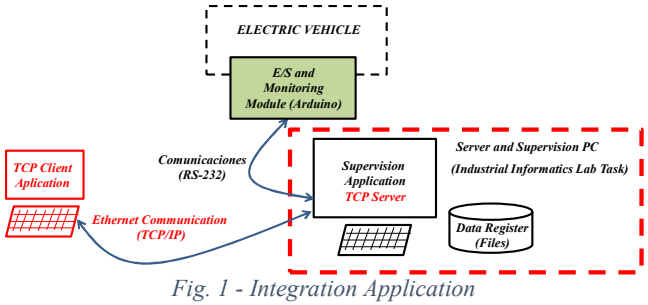

Fig. 1 - Integration Application

a) Students face a double problem during its development:

b) Manage client and server connections to connect they both.

c) Manage transferred messages.

The problem of message managing is unavoidable and depends on the functional specifications of the application to be developed. However, the problem of connections managing is also a very common situation and can be faced in many ways.

There are multiple options to manage TCP connections in C#. The available libraries offer classes with methods that allow the user to solve this problem. In all cases, to use these libraries, it is necessary to handle concepts that touch the limit of the content scope worked on a subject based on communication concepts.

Due to the aforementioned idea, it was decided to create a DLL, that is, a library of methods supporting the necessary functionality to make it easier for students to manage TCP connections in their communications applications.

III. DEVELOPMENT ALTERNATIVES

A. Libraries

There are multiple options for using libraries to manage TCP connections in C# on Visual Studio The first thing is to decide which library you are going to work with. .Net Framework is used in the subject. However, other options exists: .NET, .NET Core, and .NET Standard. Recently it seems that even Microsoft has realized the mess of options and versions it offers, so the trend is to unify everything and use a single library [3].

B. Using the functions

Even so, after choosing a library, there are also multiple ways to solve the problem. In the case of the .Net Framework, there are also multiple classes that allow finding a solution.

The underlying problem to be solved is that in communications applications it is necessary to attend to the different situations that may occur at any time during the connections.

C. TCP connection

In a TCP connection, there are two types of applications involved:

a) Client application.

It is the application that initiates the communication and sends a connection request to the server. Once sent, it waits to receive a response from the server.

b) Server Application.

It is an application that waits to receive a connection request from a client, so it must be started before receiving this request. Once the request is received, it will send an affirmative response to establish the connection, or a negative response to reject it.

978-1-6654-2161-4/22/$31.00 ©2022 IEEE

Once the TCP connection is established, both applications can send and receive messages independently of each other. At any time, any application can close the connection (Fig. 2).

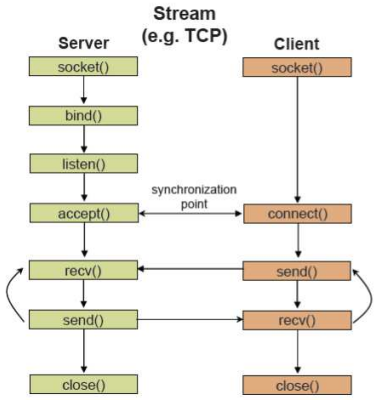

Fig. 2 - TCP connection

D. Synchronous vs Asynchronous

The pint is to decide which classes to use, which methods to use and how to do it, which ones are better, why and for which cases.

There are two large groups of methods:

a) Synchronous methods.

They perform their functionality, and it is assumed that the conditions for the execution to be carried out are met. If the conditions are not met, the method does not return until finished, so it becomes a blocking function, that is, the execution remains blocked waiting to finish the function, and this can cause the program and the application to block. To solve these situations, these calls to blocking functions are usually included into independent processes or threads, so that if the thread is blocked it does not affect the rest of the program. This type of method fits for simpler applications.

b) Asynchronous methods.

They are based on launching an execution indicating that it must notify when finished by means of an event. On the other hand, this event must be attended when it occurs. In this type of methods, there are no blocking situations, but events must be handled. Methods called 'callback' are usually used to manage the events. Callback methods are executed when the expected event occurs.

Both types of methods are useful for creating the aforementioned DLL, since it does not need high requirements.

Synchronous methods are easier to handle, but threads need to be well synchronized. On the other hand, asynchronous methods are more complex to control due to the handling of events with callbacks. However, they adapt better to be used in the DLL to be created, since it provides more flexibility and is more interesting from the didactic point of view.

E. Classes and Methods

C# is a language that works with object-oriented programming, and with classes. Classes are functional blocks with a predefined behaviour, with methods to perform operations and use properties to store their own information. Additionally, classes are grouped into what is called a NameSpace. Table 1 shows the main classes to create applications that work with TCP communications.

Table 1 - Clases del Namespace System.Net.Sockets

Name	Description
Socket	The Socket class provides a rich set of methods and properties for network communications. The Socket class allows to perform both synchronous and asynchronous data transfer.
TcpClient	The TcpClient class provides simple methods for connecting, sending, and receiving stream data over a network in synchronous blocking mode.
TcpListener	The TcpListener class provides simple methods that listen for and accept incoming connection requests in blocking synchronous mode. Either a TcpClient or a Socket to be connected with a TcpListener.

IV. TcpConex

A. Objectives

The main objective of this *TcpConex library* is to provide students with a utility so that they can easily create communications applications without having to manage TCP connections.

B. DLL outline

A DLL has been created that offers TCP connection management methods to users. It allows distinguishing the use depending on whether it is a server or client application. The library has been developed in C# on Microsoft's Visual Studio environment.

It makes use of the Socket class and works with asynchronous communications. This way, non-blocking methods are created that notify about the completion of their execution by firing events. For each event there is a callback function associated that is called when the event is activated.

The library offers the user *methods* and *properties* to be invoked and queried respectively. It also offers *Structures*, *Delegates* and *enumerations* that are needed for using the methods.

C. Structures

Las estructuras son agrupaciones de datos (Table 2 - StructuresTable 2).

Table 2 - Structures

Name	Description
tcpEvArg	It mainly contains the following data fields: • string **message**. Contains Text generated by the DLL • byte[] **buffer**. Contains the transferred bytes • int **BytesTransfered**. Indicates the amount of data in buffer.

D. Delegates

Delegates are C# types that encapsulate methods, like pointers to functions (Table 3). They are mechanisms that allow methods to be executed indirectly.

Table 3 - Delegates

Name	Description
successHdler	public delegate void **successHdler** (object sender, tcpEvtArg e); It allows creating callbacks that warn of an event in the DLL. The information of the occurred event is included in the object **e** of type **tcpEvtArg** .
loggerMsgConex	public delegate void **loggerMsgConex** (string msg , eMessageType type = eMessageType.FORM); Allows to create a callback function that notifies that a message has been generated. The message is in the first argument. This kind of messages are offered to the user in case he/she wants to record the sequence of operations carried out by the DLL.

E. Enumerations

They are C# types that contain sets of constants (Table 4).

Table 4 - Enumerations

Name	Description
connectionState	Indicates connection status
eConexType	Indicates the type of connection: client or server.
eConexOp	Indicates the type of operation to be executed
eTipoMensaje	Indicates the type of message
eTcpConexError	Indicates the type of error occurred

F. Methods

These are the functions offered by the DLL to perform operations with TCP connections (Table 5).

Table 5 - Methods

Name	Description
tcpConex	**tcpConex**(successHdler cbOperation, loggerMsgConex cbLoggerMsg = null, string name = "") It is the constructor method that creates the object to be used with the rest of the methods.
connect	public eTcpConexError **connect** (string ipAddr, int port) It is the method to create a client and connect it to the server at the indicated address and port
listen	public eTcpConexError **listen** (string ipAddr, int port) It is the method to create a server and stay listening to a client request at the indicated address and port
read	public int **read** (int RXBufferSize, int MaxBytesToReceive) It is the method used to read the received data. Buffer size and max size of the receiving buffer are indicated.
close	public int **close**() It is the method to close the connection

G. Properties

These are the values that can be queried (Table 6).

Table 6 - Properties

Name	Description
version	Indicates the version of the library
Credis	It is the text that indicates the credits of the library

V. Aplicación Demo

A. Open application

Along with the library, the students are provided with a demonstrative application of the use of the library (Fig. 3). This application is offered with all the source code so that its structure and operation can be studied.

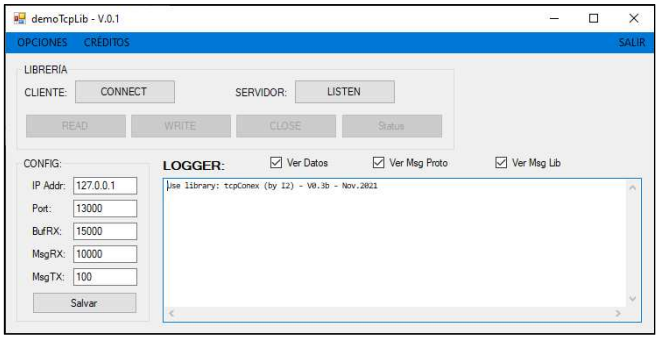

Fig. 3 – Demo Application

B. Documentation

Students are also provided with a manual that describes the steps to follow to use the library along with the demo application example. It is summarized in the following table.

Table 7 - Steps to execute

Step 1.- Initialize → TcpInit()
• Assign the variables that register the callBacks functions
 	
m_cbSuccessOp = new successHdler(successOperationTcpLib);
m_cbLoggerMsg = new loggerMsgFunction(mostrarMensajePorTipo);

SERVER:
 	
Step 2.- Connect → tcpListen_Server ()
• Create the tcpConex object and pass it the callbacks functions
• Execute the listen function. The listening IP and PORT are passed to it. Waits for a connection request from a client.
 	
m_tcpConex_Server = new tcpConex(m_cbSuccessOp, m_cbLoggerMsg, "DemoServidor");
err = m_tcpConex_Server.listen(Param.sIpAddr, Param.iPort);
 	
Step 3.- Read → tcpRead_Server ()
• Execute the read function. Size of the receive buffer, and the maximum size of the message to receive are passed.
 	
m_tcpConex_Server.read(Param.iTamBufRX, Param.iTamMaxMsgRX);
 	
Step 4.- Write → tcpWrite_Server ()
• Create the message to send
• Execute the write function. The message is passed.
 	
byte[] buf = u.creaMensajeGenerico(Param.iTamMsgTX);
if (m_tcpConex_Server != null) m_tcpConex_Server.write(buf);
 	
Step 5.- Status → tcpStatus_Server ()
• Executes the status function. Returns the connection status. Indicates bytes left to read (Available)
 	
st = m_tcpConex_Server.status();
 	
Step 6.- Close → tcpClosed_Server ()
• Execute the close function. Close the connection.
 	
st = m_tcpConex_Server.close();

CLIENT: **Step 2.- Connect → tcpConnect_Client()** • Create the tcpConex object. Callbacks functions are passed.

- Execute the connect function. IP and connection PORT are passed. Try to connect to the server.

```
m_tcpConex_Client = new tcpConex(m_cbSuccessOp, m_cbLoggerMsg, "DemoCliente");
err = m_tcpConex_Client.connect(Param.sIpAddr, Param.iPort);
```

Step 3.- Write → tcpWrite_Client ()

- Create the message to send
- Execute the write function. The message is passed.

```
byte[] buf = u.creaMensajeGenerico(Param.iTamMsgTX);
if (m_tcpConex_Client != null) m_tcpConex_Client.write(buf);
```

Step 4.- Read → tcpRead_Client ()

- Execute the read function. The size of the receive buffer, and the maximum size of the message to receive are passed.

```
m_tcpConex_Client.read(Param.iTamBufRX, Param.iTamMaxMsgRX);
```

Step 5.- Status → tcpStatus_Client ()

- Executes the status function. Returns the connection status. Indicates bytes left to read (Available)

```
st = m_tcpConex_Client.status();
```

Step 6.- Close → tcpClosed_Client ()

Execute the close function. Close the connection.

```
st = m_tcpConex_Client.close();
```

VI. Results

The results obtained after delivering this library to the students have been quite satisfactory. This tool was optionally provided to them. Most of the students used it in their development (78%). However, there was a small part of them that decided not to use it (12%). Maybe it because the repeat students had already developed it with the alternatives studied in previous courses.

Perhaps more efforts are required to clarify and demonstrate the advantages of using it over another alternatives.

VII. Conclusions

The library that has been presented in this article is the result of an extra work dedicated to creating tools to assist the work of students. In the near future it is planned to modify this library so that it can be used with .NET Core.

References

[1] M. J. D. K. L. C. David Makofske, TCP/IP Sockets in C#, Morgan Kaufmann, 2004.

[2] A. Davies, Async in C# 5.0, O'Reilly Media, Inc., 2012.

[3] «¿Confusión de términos? .NET vs .NET Core vs .NET Framework vs .NET Standard ¡Te lo explicamos!.,» [En línea]. Available: https://www.campusmvp.es/recursos/post/confusion-de-terminos-net-vs-net-core-vs-net-framework-vs-net-standard-te-lo-explicamos.aspx.