Lab session 2

Working with Visual Studio

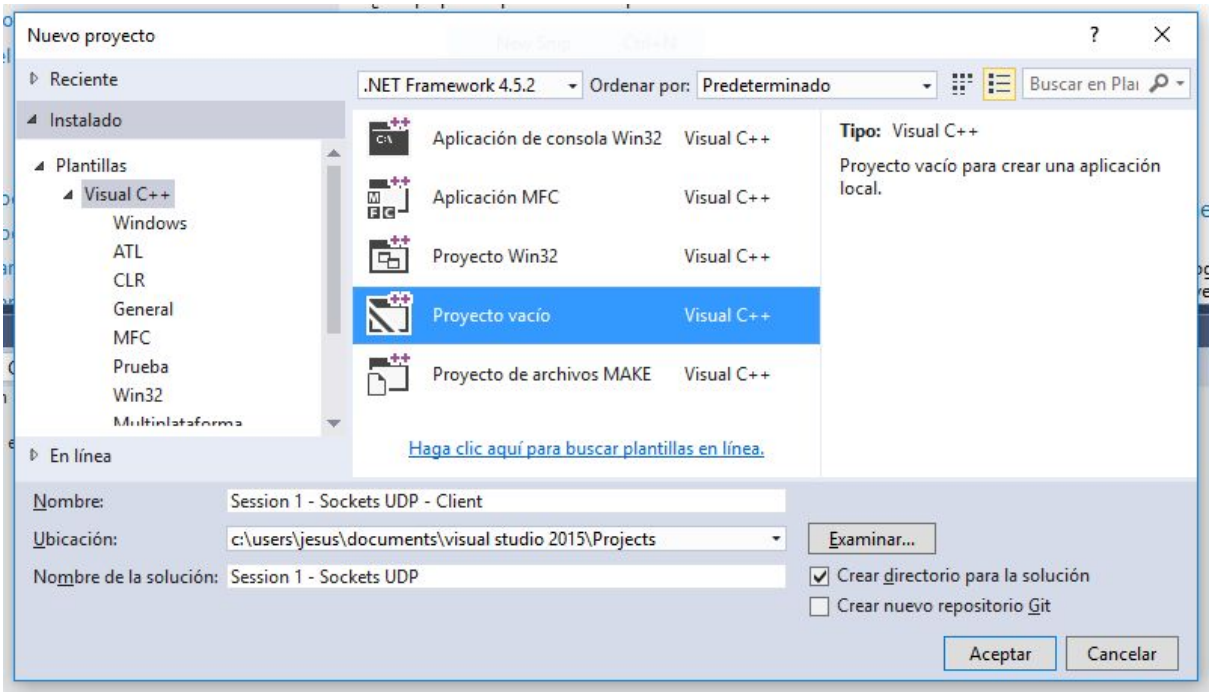
This document focuses on the preparation of the IDE Visual C++. It describes the steps to follow to have the first exercise working. First, we will have to specify which libraries are needed in order to work with the sockets API in Windows. It will also be interesting knowing a few tricks that will allow us to debug our applications.

Preparing a solution

The first exercise will consist of two parts, which will be implemented in two separate projects: client and server. To that aim, we will create a solution containing two projects. We will also learn how to execute and debug both of them when necessary.

Creating the solution and the client project

We will start working from an empty project in Visual Studio (File Menu > New > Project > Empty C++ project). We will give the solution and the project different names. Let's call the project *Client*, and the solution *UDP Sockets*.

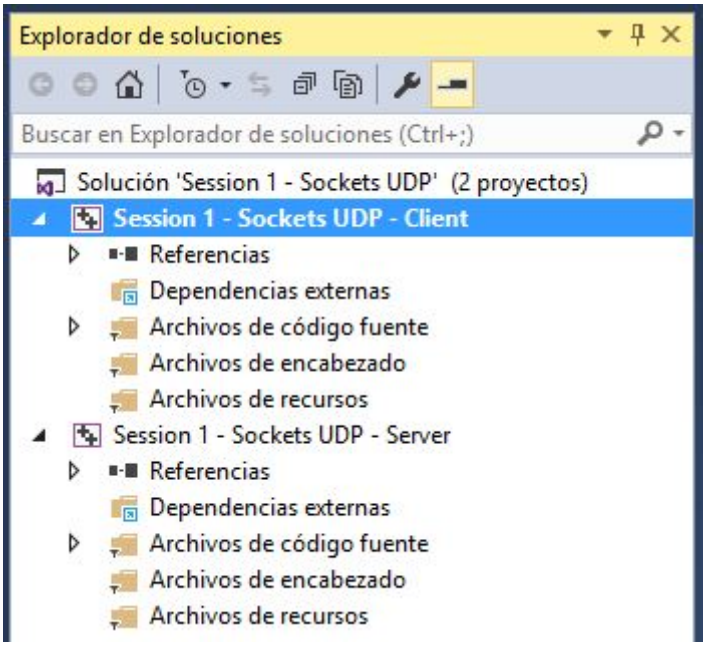


Creating the server project

The server project will be within our solution, along with the client project. To add a new project to the existing solution, right click on the solution in the solution explorer and then Add > New Project. Let's give this project the name *Server*.

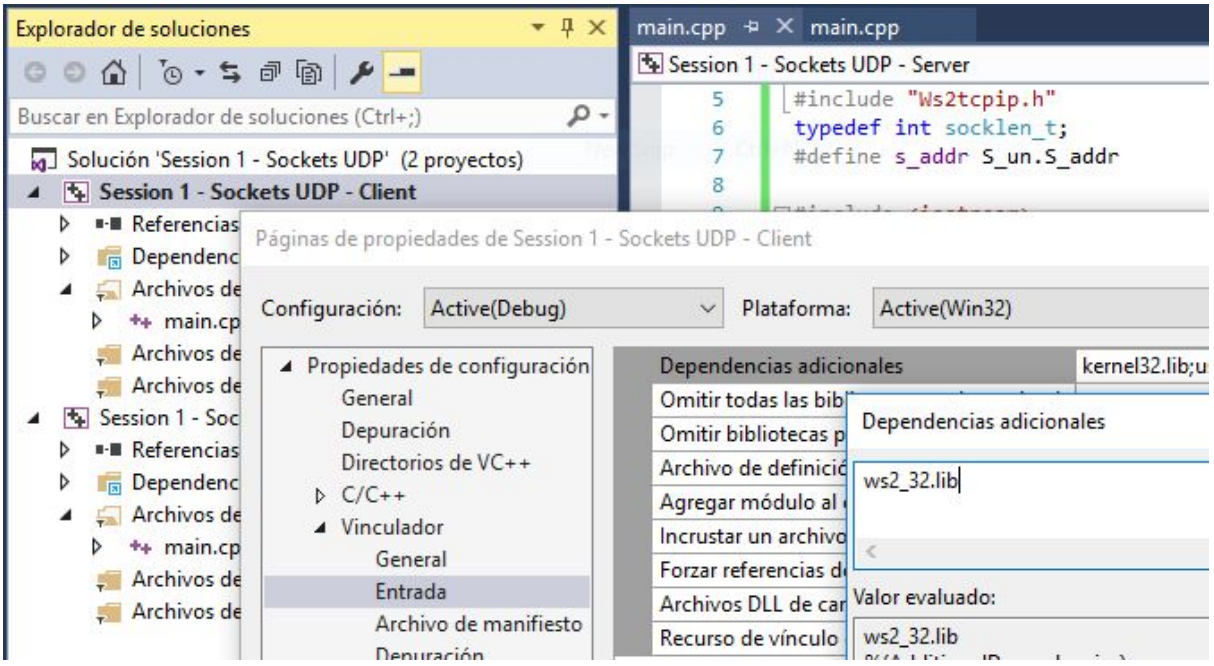
In both projects, we need to create a new C++ file with the application entry point: *int main(int argc, char **argv)*.

At this point, the solution explorer should look similar to the following image:



Linking to the sockets library

The native Windows sockets library is called Winsock (Windows Sockets 2). Whenever we need to use the sockets API, we will need to configure each project in Visual Studio so that the sockets library is linked to the output binary. To do that, we will head to the project properties (right click on the project in the *solution explorer* > *properties*) and will introduce the library ws2_32.lib as an additional dependency in the section *Properties* > *Linker* > *Input*, as shown in the following picture:



We will repeat this operation for the other project in the solution.

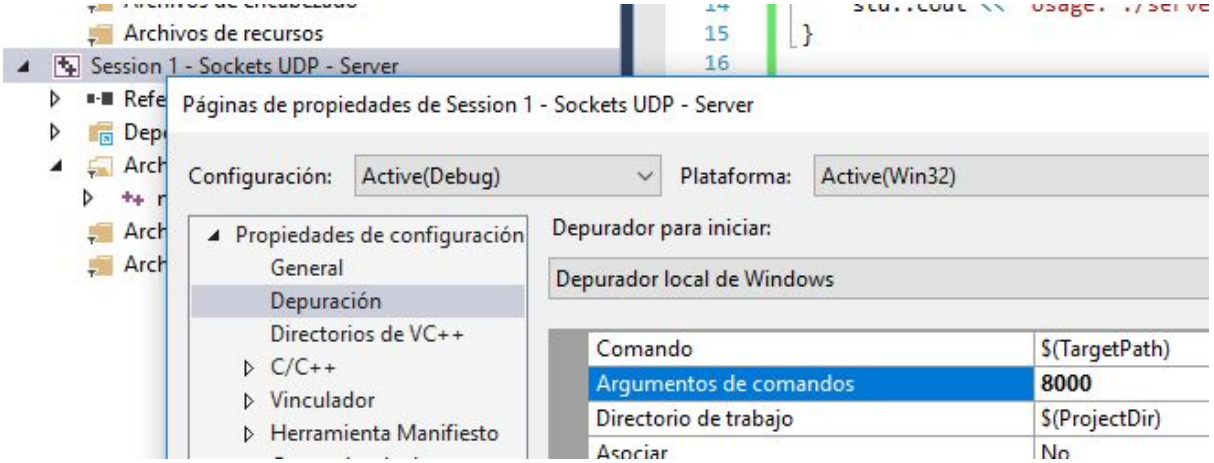
Instead of configuring the project as shown above, we can alternatively indicate libraries we want to link to in a source file of each project this way:

```
#pragma comment(lib, "ws2_32.lib")
```

Execution

Command line arguments (from Visual Studio)

In most cases, it will be interesting passing input arguments to our program. An easy way to do this is through the command line. We can set input command line arguments in the project properties in Visual Studio. To do that, open one of the project properties panel and go to *Debug* > *Command line arguments*, as shown in the following image, where we are passing the number 8000 (that will be the port used by the server to receive and send information through a socket).



Each project in the solution can have its own input command line arguments. For instance, in this first example it could be interesting that the only argument provided to the server application was this given port for incoming data, but in the client project, it is interesting knowing both the IP address of the machine where the server is executed and the port the server will be attending (8000).

Passing command line arguments with batch files (.bat)

Another way to pass command line arguments to our application is through .bat files. This method will also allow us to launch several instances of the application more easily. Batch files can be placed in the same directory where Visual Studio puts the output executable, and their content would be similar to this (for the server):

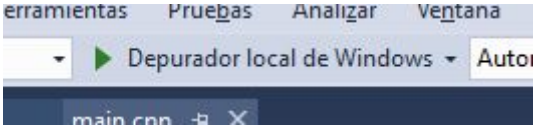
```
Server.exe 8000
```

And for the client:

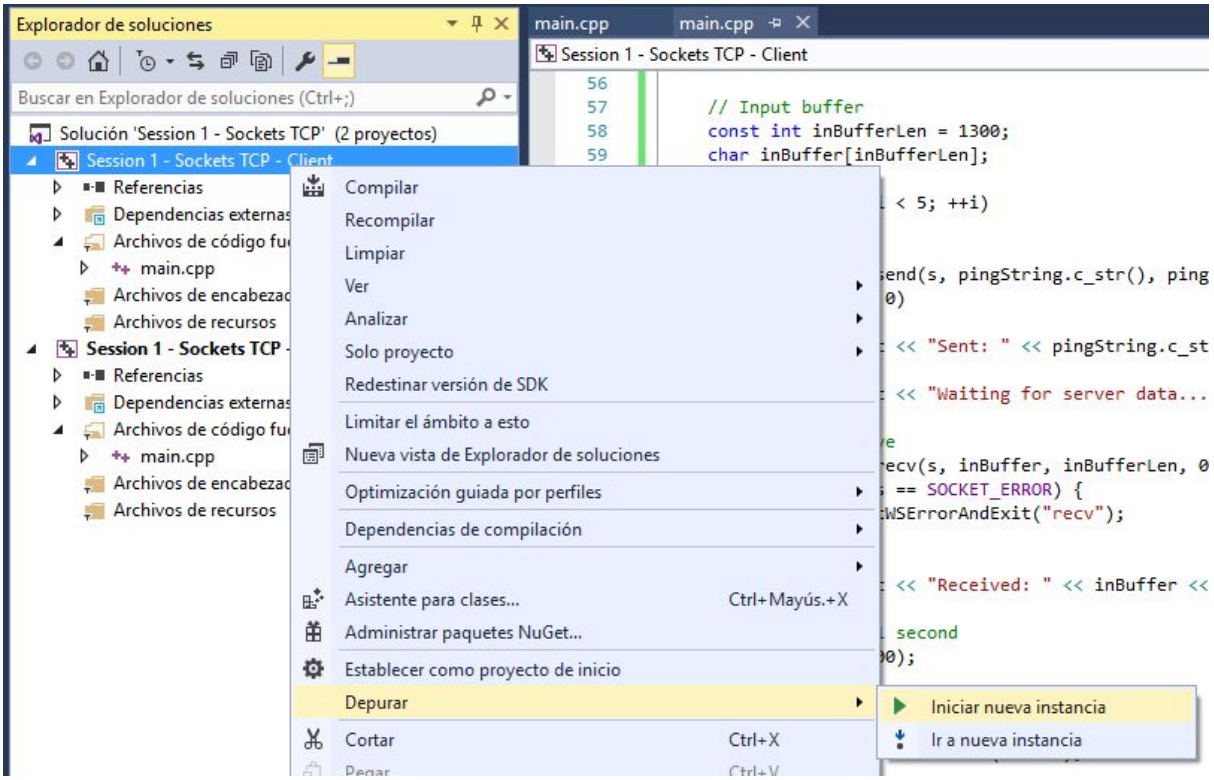
```
Client.exe 127.0.0.1 8000
```

Executing several instances

The *play* button in the toolbar allows us executing and debugging our code:



However, the *play* button only allows executing and debugging the startup project (the project in **bold** in the solution explorer). Luckily, we can execute any other project simultaneously by clicking the *Start new instance* option, as shown in the following image:



This way, we can execute both projects at the same time from Visual Studio (we will have both instances running simultaneously) so that the *Client* and *Server* projects can communicate through the sockets API.

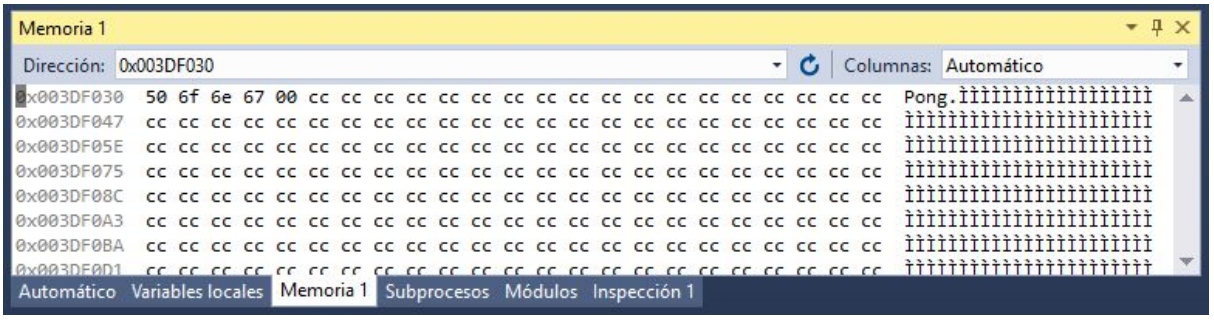
However, in other exercises that require an environment to perform tests with several clients and a server, it will be necessary to execute some more client instances from outside the Visual Studio IDE (for instance, using batch files as described before).

Debugging

Memory windows

Sometimes it will be useful inspecting the contents of some memory regions to check that everything is right. For instance, we could revise the data received in a buffer after reading input data from a socket, and check that the data is correct.

To check this, once we are debugging an application, we will need to activate a memory window in *Debug* > *Windows* > *Memory* > *Memory 1*. Below the text editor, we should see a memory window like the following:



Here we can see the contents of any memory address corresponding to the process being executed. To know the address of any variable (a data buffer for instance) we only need to move the mouse pointer and hover the variable, and then copy its memory address, as shown in this picture: