SISTEMAS OPERATIVOS
SOCKETS

Objetivo

El objetivo de esta guía es aprender en C# a comunicar 2 aplicaciones que pueden estar ejecutándose en la misma máquina o en 2 máquinas diferentes.

Tareas Principales

Las tareas principales que se deben realizar son las siguientes:

1. **Implementar un servidor C# utilizando las clases Socket.**

2. **Implementar un cliente C# utilizando las clases Socket.**

Introducción

En una red de ordenadores hay varios ordenadores que están conectados entre sí por un cable o vía wifi y pueden, por tanto, transmitirse información. Es claro que deben estar de acuerdo en cómo transmitir esa información, de forma que cualquiera de ellos pueda entender lo que están transmitiendo los otros, de la misma forma que nosotros nos ponemos de acuerdo para hablar en inglés cuando uno es italiano, el otro francés, el otro español y el otro alemán.

Al "idioma" que utilizan los ordenadores para comunicarse cuando están en red se le denomina protocolo. Hay muchísimos protocolos de comunicación, entre los cuales el más extendido es el TCP/IP. Es el más extendido porque es el que se utiliza en Internet.

Aunque todo esto pueda parecer complicado y que no podemos hacer mucho con ello, lo cierto es que podemos aprovecharlo para comunicar dos programas nuestros que estén corriendo en ordenadores distintos. En C# tenemos una serie de clases que nos permiten enviar y recibir datos a/de otros programas, en C o en otros lenguajes de programación, que estén corriendo en otros ordenadores de la misma red.

Sockets

Una forma de conseguir que dos programas se transmitan datos es la programación de sockets. Un socket no es más que un "canal de comunicación" entre dos programas que corren sobre ordenadores distintos o incluso en el mismo ordenador.

Existen básicamente dos protocolos de comunicación con sockets internet (PF_INET/AF_INET), los que utilizan el protocolo UDP y los que utilizan el protocolo TCP.

Cuando se utiliza el protocolo UDP hay que tener en cuenta que dicho protocolo es:

- *No orientado a conexión*. No se establece una conexión previa con el otro extremo para transmitir un mensaje UDP. Los mensajes se envían sin más y éstos pueden duplicarse o llegar desordenados al destino.

- *No fiable*. Los mensajes UDP se pueden perder o llegar dañados.

Contrariamente a UDP, el protocolo TCP es:

- *Orientado a conexión*. Es necesario establecer una conexión previa entre las dos máquinas antes de poder transmitir ningún dato. A través de esta conexión los datos llegarán siempre a la aplicación destino de forma ordenada y sin duplicados. Finalmente, es necesario cerrar la conexión.

- *Fiable*. La información que envía el emisor llega de forma correcta al destino.

La mayoría de aplicaciones utilizan TCP en lugar de UDP – pese a ser más lento - para no tener que realizar a nivel de aplicación el control de errores, ya que TCP garantiza que cuando un paquete no llega pasado un cierto tiempo o llega dañado, se pide la retransmisión del paquete al emisor. Sin embargo, hay aplicaciones en que esa retransmisión de paquetes, no es necesaria, por ejemplo, en aplicaciones de streaming (audio, video, etc..) donde la llegada a tiempo de los paquetes es más importante que la fiabilidad.

A partir de este momento nos referimos únicamente a sockets TCP. El uso de UDP es muy parecido, aunque hay pequeñas diferencias en la forma de abrirlos y de enviar los mensajes.

Arquitectura Cliente/Servidor

A la hora de comunicar dos programas, existen varias posibilidades para establecer la conexión inicialmente. Una de ellas es la utilizada aquí. Uno de los programas debe estar arrancado y en espera de que otro quiera conectarse a él. Nunca da "el primer paso" en la conexión. Al programa que actúa de esta forma se le conoce como servidor. Su nombre se debe a que normalmente es el que tiene la información y la "sirve" al que se la pida. Por ejemplo, el servidor de páginas web tiene las páginas web y se las envía al navegador que se lo solicite.

El otro programa es el que da el primer paso, realiza una petición de servicio al servidor. En el momento de arrancarlo o cuando lo necesite, intenta conectarse al servidor. Este programa se denomina cliente. Su nombre se debe a que es el que solicita información al servidor. El navegador de Internet pide la página web al servidor de Internet.

En este ejemplo, el servidor de páginas web se llama servidor porque está (o debería de estar) siempre encendido y pendiente de que alguien se conecte a él y le pida una página. El navegador de Internet es el cliente, puesto que se arranca cuando nosotros lo arrancamos y solicita conexión con el servidor cuando nosotros escribimos, por ejemplo, www.chuidiang.com

En el juego del Quake, debe haber un servidor que es el que tiene el escenario del juego y la situación de todos los jugadores en él. Cuando un nuevo jugador arranca el juego en su ordenador, se conecta al servidor y le pide el escenario del juego para presentarlo en la pantalla. Los movimientos que realiza el jugador se transmiten al servidor y este actualiza escenarios a todos los jugadores.

Resumiendo, servidor es el programa que permanece pasivo a la espera de que alguien solicite conexión con él, normalmente, para pedirle que realice alguna acción o servicio. Cliente es el programa que solicita la conexión para, normalmente, pedir un servicio al servidor.

La Conexión

Para poder realizar la conexión entre ambos programas son necesarias varias cosas:

- **Dirección IP del servidor**.
  Cada ordenador de una red tiene asignado un número único, que sirve para identificarle y distinguirle de los demás dentro de la red, de forma que cuando un ordenador quiere hablar con otro, manda la información a dicho número. A este número único se le conoce como dirección IP. Es similar a nuestros números de teléfono. Si quiero hablar con mi amigo "Josechu", primero marco su número de teléfono y luego hablo con él. La dirección IP es un número del estilo 192.100.23.4.

  Volviendo a nuestra arquitectura cliente/servidor, el servidor no necesita conocer la dirección de los clientes, al igual que nosotros, para recibir una llamada por teléfono, no necesitamos saber el número de nadie, ni siquiera el nuestro. El cliente sí necesita saber el número del servidor, al igual que nosotros para llamar a alguien necesitamos saber su número de teléfono.

  En resumidas cuentas, el cliente debe conocer a qué ordenador desea conectarse. En nuestro navegador de Internet facilitamos la dirección IP del servidor al que queremos conectarnos a través de su nombre (www.chuidiang.com) y se produce una resolución de nombre a IP, que traduce ese nombre en una dirección IP.

- **Servicio/Puerto** que queremos crear / utilizar.

  Si llamamos a una empresa, puede haber en ella muchas personas, cada una con su extensión de teléfono propia. Normalmente la persona en concreto con la que hablemos nos da igual, lo que queremos es alguien que nos atienda y nos de un determinado "servicio", como recoger una queja, darnos una información, tomarnos nota de un pedido, etc.

  De la misma forma, en un mismo ordenador pueden estar corriendo varios programas servidores, cada uno de ellos dando un servicio distinto. Por ejemplo, un ordenador puede tener un servidor de Quake y un servidor de páginas web corriendo a la vez. Cuando un cliente desea conectarse, debe indicar qué servicio quiere, igual que al llamar a la empresa necesitamos decir la extensión de la persona con la que queremos hablar o, al menos, decir su nombre o el departamento al que pertenece para que la telefonista nos ponga con la persona adecuada.

  Por ello, cada servicio dentro del ordenador debe tener un número único que lo identifique (como la extensión de teléfono). Estos números son enteros normales y van de 1 a 65535. Los números bajos, desde 1 a 1023 están reservados para determinados servicios estandarizados (www, ftp, mail, ping, etc). El resto están a disposición del programador y sirven para cosas como Quake.

  Tanto el servidor como el cliente deben conocer el número del servicio al que atienden o se conectan. El servidor le indica al sistema operativo qué servicio quiere atender, al igual que en una empresa el empleado recién contratado (o alguien en su lugar) debe informar a la telefonista en qué extensión se encuentra.

  El cliente, cuando llame a la empresa, debe dar el número de extensión (o nombre de empleado), de forma que la telefonista le ponga con la persona adecuada. En el caso del navegador de Internet, estamos indicando el servicio con la www o con http:// en http://www.chuidiang.com, servicio de páginas web, que siguiendo el estándar significa servicio/puerto 80. También es posible, por ejemplo "ftp.chuidiang.com", si chuidiang.com admite clientes ftp. El estándar que debe seguir el sistema operativo en este caso indicaría que se utilizan los puertos 20 y 21.

El Servidor

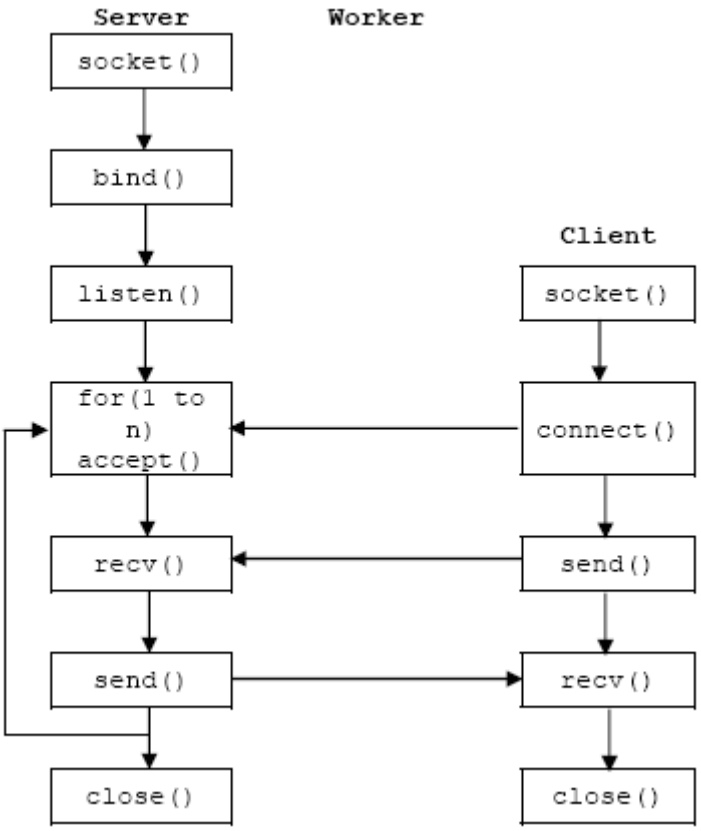
Para poder realizar la conexión entre ambos programas son necesarias varias cosas:

- Crear un objeto de la clase **Socket.** Se crea un canal de comunicación sin nombre, es decir, no tiene asignado ni dirección IP ni puerto, este canal o socket es el que denominamos socket de conexión, pues nos permite recibir conexiones de los clientes.
- **Avisar al sistema operativo** de que hemos abierto un socket, queremos que asocie nuestro programa a dicho socket y en este momento se le asigna una IP y un puerto. Se consigue mediante el método **Bind()**. El sistema todavía no atenderá a las conexiones de clientes, simplemente anota que cuando empiece a hacerlo, tendrá que avisarnos a nosotros. Es en esta llamada cuando se debe indicar el número de servicio al que se quiere atender.
- Avisar al sistema de que **comience a atender dicha conexión** de red. Se consigue mediante el método **Listen()**. A partir de este momento el sistema operativo anotará la conexión de cualquier cliente para pasárnosla cuando se lo pidamos. Si llegan clientes más rápido de lo que somos capaces de atenderlos, el sistema operativo hace una "cola de espera" con ellos y nos los irá pasando según vayamos pidiéndolo.
- **Aceptar las conexiones** de clientes al sistema operativo. Para ello hacemos una llamada al método **Accept()**. Esta función le indica al sistema operativo que nos dé al siguiente cliente de la cola. Si no hay clientes se quedará bloqueada hasta que algún cliente se conecte. Esta llamada, además nos retornarà un nuevo socket, con el que nos comunicacremos –intercambiando datos- con el cliente.
- **Escribir y recibir datos** del cliente
- **Cierre de la comunicación y del socket de comunicación**, por medio del método **Close().**

El Cliente

Los pasos que debe seguir un programa cliente son los siguientes:

- Crear un objeto de la clase **Socket**, como el servidor.
- **Solicitar conexión** con el servidor por medio del método **Connect()**. Dicha función quedará bloqueada hasta que el servidor acepte nuestra conexión, se rechace la connexión o bien si no hay servidor en el sitio indicado, saldrá dando un error. En esta llamada se debe facilitar la dirección IP del servidor y el número de servicio(puerto) que se desea.
- **Escribir y recibir datos** del servidor.
- **Cerrar la comunicación** por medio de **Close()**.



Programando en C#

Clases IPAddress e IPEndPoint

Clase IPAddress

Esta clase se usa para representar una dirección IP como un objeto.

Una de las formas más habituales de crear un objeto de la clase IPAddress es pasar como un argumento de la clase String la IP de la máquina:

```
IPAddress newaddress = IPAddress.Parse("192.168.1.1");
```

Otra de las formas es obtener la o las direcciones IPs a través de una resolución de nombre a IP (servicio de DNS), es decir, pasando el nombre de la máquina, tal y como hemos comentado anteriormente:

```
string howtogeek = "www.howtogeek.com";
IPAddress[] addresslist = Dns.GetHostAddresses(howtogeek);
```

En una resolución de nombre a IP, dependiendo del nombre, es posible que un mismo nombre tenga varias IPs asignadas, por lo que la resolución devuelve un array de objetos de la clase IPAddress. Podemos utilizar cualquiera de esos objetos IPAddress. Por lo general, en nuestros programas, utilizaremos la primera de ellas.

Por otro lado, una máquina puede tener varias tarjetas de red y cada una de ellas tiene asignada una IP (si estamos conectados a Internet a través de la tarjeta de red). Para indicar en el programa servidor que éste debe escuchar la actividad del cliente en todas las interfaces de red de la máquina se utiliza:

```
IPAddress.Any
```

Finalmente, para probar localmente una aplicación cliente/servidor en una única máquina, independientemente de que disponga de tarjeta de red o no, podemos utilizar el nombre reservado localhost que es traducido como la dirección IP 127.0.0.1.

Clase IPEndPoint

Esta clase se usa para representar la combinación de una dirección IP y un puerto como un objeto, que son dos datos que necesitamos para poder crear el socket tanto en la parte de servidor como en la parte de cliente.El constructor más común para crear un objeto IPEndPoint es:

```
public IPEndPoint(IPAddress address, int port)
```

y por tanto, crearíamos un objeto IPEndPoint en la parte de servidor con el siguiente código:

```
IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);
```

La clase Socket

El constructor de la clase Socket sigue el formato original de Unix y es el siguiente:

```
Socket(AddressFamily af, SocketType st, ProtocolType pt)
```

Usa tres parámetros para definir el tipo de socket a crear:

- Un AddressFamily que define el tipo de red
- Un SocketType para definir el tipo de datos de la conexión
- Un ProtocolType para definir un protocolo específico de red

Para utilizar el protocolo a nivel de red IP (Internet Protocol) se especifica el valor AddressFamily.InterNetwork en el primer parámetro del constructor del socket. Para los parámetros SocketType y ProtocolType la siguiente tabla muestra algunos valores que se pueden utilizar, una vez establecido el protocolo IP como protocolo a nivel de red

SocketType	ProtocolType	Descripción
Dgram	Udp	Comunicación no orientada a conexión
Stream	Tcp	Comunicación orientada a conexión

La siguientes líneas de código muestran cómo crear un objeto de la clase Socket que utiliza protocolo TCP/IP. Así para crear un socket IP con protocolo TCP utilizaríamos las siguientes líneas de código:

```
Socket server = new Socket(AddressFamily.InterNetwork,  SocketType.Stream,
ProtocolType.Tcp);
```

Métodos:

Los siguientes son algunos de los métodos de la clase Socket:

Method	Description	Applicable
void Bind(IPEndPoint ep)	binds a server socket to a local end-point	Tcp Server
void Listen(int max)	listen for clients; max is the maximum number of clients to enqueue, while waiting for connection	Tcp Server
Socket Accept()	connects to a client and returns a reference to the client's socket	Tcp Server
void Connect(IPEndPoint)	connects to a remote end-point	Tcp Client
int Receive(byte[] data) int Receive(byte[] data, int size, SocketFlags sf) int Receive(byte[] data, int offset, int size, SocketFlags sf)	overloaded,: reads bytes from a socket	Tcp server/client
int Send(byte[]) int Send(byte[] data, int size, SocketFlags sf) int Send(byte[] data, int offset, int size, SocketFlags sf)	overloaded, sends bytes to a socket	Tcp server/client
void ReceiveFrom(byte[], ref EndPoint)	overloaded, receives from a client at EndPoint	Udp client
void SentTo(byte[] ref EndPoint)	overloaded, sends to a client at EndPoint	Udp client

void Shutdown(SocketShutdown how)	Disables sends and receives on a socket	All
void Close()	close a socket.	All

Crear un servidor TCP utilizando la clase Socket

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

class SimpleTcpSocketServer
{
    public static void Main()
    {
        Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);
        server.Bind(localEP);
        server.Listen(10);

        while (true)
        {
            Console.WriteLine("Waiting for Client...");
            Socket client = server.Accept();
            IPAddress clientAddress = ((IPEndPoint)client.RemoteEndPoint).Address;
            int clientPort = ((IPEndPoint)client.RemoteEndPoint).Port;
            Console.WriteLine("Got connection from " + clientAddress + " port " +
clientPort);

            byte[] data = Encoding.ASCII.GetBytes("Welcome to my test server");
            client.Send(data);

            int dataSize = 0;
            data = new byte[1024];
            dataSize = client.Receive(data);

            Console.WriteLine(Encoding.ASCII.GetString(data, 0, dataSize));

            client.Close();
        }
        server.Close();
    }
}
```

Similar a los servidores de Unix, una vez se ha creado el objeto Socket en el servidor, éste debe ser enlazado a una dirección local de red. El método Bind() es utilizado para realizar esta función.

`Bind(localEP)`

El parámetro localEP debe apuntar a un objeto de la clase IPEndPoint, el cual incluye una dirección IP local y un número de puerto. En el caso que mostramos en el ejemplo el objeto IPEndPoint representa cualquiera de las IPs locales de la máquina servidor, con lo que este servidor atendería a un cliente que se conectara utilizando localhost, o utilizando la IP pública de su tarjeta de red Ethernet o utilizando la IP pública de su tarjeta de red inalámbrica, ect… Si se desea una IP local específica de la máquina servidor la forma de hacerlo sería:

```
IPEndPoint localEP = new IPEndPoint(IPAddress.Parse("192.168.1.1"), 9050);
```

Una vez el socket es enlazado a una dirección local, se utiliza el método Listen() para esperar conexiones de clientes.

`Listen(int backlog)`

El parámetro backlog define el número de conexiones que el sistema mantendrá en la cola esperando a que la aplicación servidor las sirva. Cualquier intento de conexión por encima de este número de conexiones en espera será rechazado. Las conexiones pendientes de ser atendidas

utilizan espacio en el buffer TCP, lo que implica menos espacio disponible para enviar y recibir paquetes.

Una vez se ejecuta el método Listen, el servidor está preparado para aceptar peticiones de clientes. Esta acción se realiza con el método Accept(). El método Accept() devuelve un nuevo socket para cada cliente que se conecta liberando así el socket que espera peticiones de los clientes.

```
Socket client = server.Accept();
```

Este programa se bloqueará en la llamada al método Accept(), esperando conexiones de clientes. Cuando un cliente se conecte el objeto de la clase Socket client contendrá toda la información necesaria para la comunicación entre el servidor y el cliente aceptado. El objeto de la clase Socket server puede seguir siendo usado para esperar conexiones de otros clientes.

Una vez la conexión del cliente ha sido aceptada, servidor y cliente pueden empezar a transferir datos. Para realizar la transmisión de datos se utilizan los métodos Receive() y Send(). Los métodos Receive() y Send() envían y reciben bytes. Por esta razón, por ejemplo el mensaje de bienvenida que el servidor envía al cliente debe ser transformado de secuencia de caracteres a secuencia de bytes:

```
byte[] data = Encoding.ASCII.GetBytes("Welcome to my test server");
client.Send(data);
```

Y el mensaje que recibe el servidor del cliente debe transformarse de secuencia de bytes a secuencia de caracteres:

```
dataSize = client.Receive(data);
Console.WriteLine(Encoding.ASCII.GetString(data, 0, dataSize));
```

Utilizando telnet para testear una aplicación servidor

Podemos testear el servidor anterior utilizando como cliente el programa Telnet de Microsoft, presente en todos los sistemas operativos de Windows. Si tienes un Windows 7 busca en google como activar el cliente de telnet en tu sistema operativo.

Para ello:

1. Pon en marcha el servidor (crea una aplicación de consola en el Visual C#, copia el código del servidor y ejecuta la aplicación)
2. Pon en marcha el cliente (abre una consola de comandos y teclea el siguiente comando: `telnet direccionIP puerto`" – donde `direccionIP` puede ser `localhost` , `127.0.0.1` o cualquiera de las IPs de la máquina en la que estás haciendo la prueba y `puerto` es 9050

Una vez realizados los pasos anteriores tendrás dos ventadas de MSDOS con el siguiente aspecto:

Ahora puedes teclear en la ventana del cliente una tecla. La tecla se enviará al servidor y el servidor te lo devolverá mostrándose por la pantalla del cliente. El mensaje enviado al servidor también se mostrará en la ventana del servidor. La aplicación de telnet se cerrará porque el servidor cierra la conexión y se queda esperando a nuevos clientes.

Crear un cliente TCP utilizando la clase Socket

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
```

```
class SimpleTcpSocketClient
{
    public static void Main()
    {
        Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
        try
        {
            socket.Connect(remoteEP);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Unable to connect to server. ");
            Console.WriteLine(e);
            return;
        }

        byte[] data = new byte[1024];
        int dataSize = socket.Receive(data);
        Console.WriteLine(Encoding.ASCII.GetString(data, 0, dataSize));

        String input = null;
        Console.Write("Enter Message for Server <Enter to Stop>: ");
        input = Console.ReadLine();
        socket.Send(Encoding.ASCII.GetBytes(input));

        Console.WriteLine("Disconnecting from Server..");
        socket.Shutdown(SocketShutdown.Both);
        socket.Close();
        input = Console.ReadLine();
    }
}
```

El programa cliente debe enlazar también una dirección al objeto Socket creado, pero utiliza el método Connect() en vez del método Bind(). Como con Bind(), Connect() requiere un objeto IPEndPoint del dispositivo remoto con el que el cliente quiere conectar:

```
IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
```

El método Connect() se bloqueará hasta que la conexión se establezca. Si la conexión no puede establecerse, se producirá una excepción.

Una vez la conexión se haya establecido, el cliente puede usar los métodos Send() y Receive() para enviar y recibir datos.

Probar la aplicación Cliente / Servidor

Para ello:

1. Pon en marcha el servidor (crea una aplicación de consola en el Visual C#, copia el código del ejemplo y ejecuta la aplicación)
2. Pon en marcha el cliente (crea una aplicación de consola en el Visual C#, copia el código del ejemplo y ejecuta la aplicación)

Ahora puedes teclear en la ventana del cliente cualquier mensaje y pulsar enter. El mensaje introducido se enviará al servidor y el servidor te lo devolverá mostrándose por la pantalla del cliente. El mensaje enviado al servidor también se mostrará en la ventana del servidor. Para finalizar la aplicación cliente y el servidor envía un mensaje con sólo un enter.