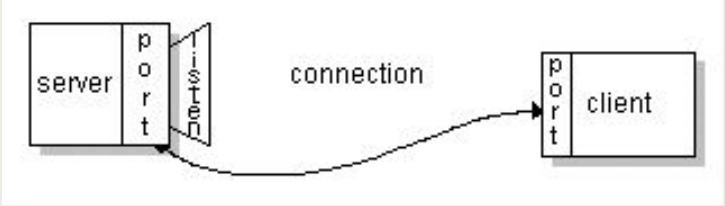
REDES

Sockets en C#

Introducción

- Para programar conexiones entre programas, utilizaremos Sockets
- Generalmente un programa servidor se ejecuta en un ordenador y se enlaza a un puerto específico. El servidor “escucha” al socket hasta que un cliente pide una conexión

- Si todo va bien, el servidor acepta la conexión. Una vez aceptada el servidor tendrá un nuevo socket enlazado al mismo puerto local en un extremo, y a la dirección y puerto del cliente en el otro. Esto se hace para que el servidor pueda seguir escuchando por nuevos clientes con el socket de escucha



Introducción

- El cliente a su vez es otro programa que intentará conectarse al puerto en el servidor está escuchando.
- Si la conexión es aceptada, se crea un socket que el cliente podrá usar para comunicarse con el servidor.
- De esta manera, cuando un cliente se conecta satisfactoriamente, el cliente tiene un socket y el servidor otro socket que están conectados.
- La comunicación entre cliente y servidor se realizará leyendo y escribiendo en sus sockets
  - *El cliente lee del socket para recibir información del servidor y escribe en el socket para enviar información al servidor*
  - *El servidor lee del socket para recibir información del cliente y escribe en el socket para enviar información al cliente*
- Un **socket** es un extremo (endpoint) de una comunicación bidireccional entre dos programas que se ejecutan en una red. Un socket está enlazado a un puerto de manera que la capa TCP puede identificar la aplicación a la que tiene que enviar los datos que reciba.

Introducción

- Un **endpoint** es la combinación de una IP y un número de puerto. Cada conexión TCP puede ser identificada inequívocamente por sus dos endpoints. De esta manera podremos tener varias conexiones entre la máquina cliente y el servidor.
- Los paquetes System.Net y System.Net.Sockets provee las clases que necesitamos para trabajar con sockets

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

Servidor básico: https://www.dropbox.com/s/5di0p9f7d2wnfaf/Sockets1.zip?dl=0

Servidor básico

- El primer ejemplo que veremos es un servidor y un cliente que se conectan, y terminan la ejecución, sin envío de datos.

Los pasos clave en este programa serán la inicialización del socket, donde le daremos una direccción:

```
Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
IPEndPoint direccion = new IPEndPoint(IPAddress.Any, 1234);
mySocket.Bind(direccion);
```

Y posteriormente la escucha, donde el servidor esperará hasta que alguien se conecte:

```
mySocket.Listen(1);
Socket clientSocket = mySocket.Accept();
```

Para acabar, los sockets se tienen que cerrar

```
clientSocket.Close();
mySocket.Close();
```

Cliente básico

Muy similar al servidor, los pasos a dar son los mismos, aunque el proceso cambia ligeramente. Primero inicializamos:

```
Socket mySocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
IPEndPoint direccion = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 1234);
```

Luego conectamos, y a diferencia del servidor, aquí es donde enlazamos la dirección con el socket:

```
mySocket.Connect(direccion);
```

Ojo! Esta línea anterior solo funcionará si tenemos en esa dirección un socket esperando datos (si el servidor ha llegado a la línea de Accept()

Para terminar también se cierra el socket en el extremo del cliente:

```
mySocket.Close();
```

Servidor y clientes v2: Enviando datos

El siguiente cambio que haremos será el envío de datos del cliente al servidor. No será un cambio grande, sólamente haremos que el cliente mande datos y el servidor los lea, sin comunicación de vuelta.

Cliente:

```
//leemos texto de consola y lo convertimos a byte[]
string textToSend = Console.ReadLine();
byte[] textAsBytes = Encoding.Default.GetBytes(textToSend);

//enviamos el texto. Primero el array, luego la posición de inicio de envío,
//después el tamaño del array y luego los flags de envío
mySocket.Send(textAsBytes, 0, textAsBytes.Length, SocketFlags.None);
```

Para que quede mejor, antes del readline() podrías escribir por pantalla algo como “introduzca texto a enviar”

Servidor:

```
//mismos parámetros que en el send del cliente
int numBytesReceived = clientSocket.Receive(bytesToReceive, 0, bytesToReceive.Length, SocketFlags.None);

//cambiamos el tamaño del array a lo recibido ("cortamos" lo que sobra)
Array.Resize(ref bytesToReceive, numBytesReceived);

string text = Encoding.Default.GetString(bytesToReceive);
Console.WriteLine("Recibido: "+ text);
```

Servidor y clientes v3: Bucle y control de cierre

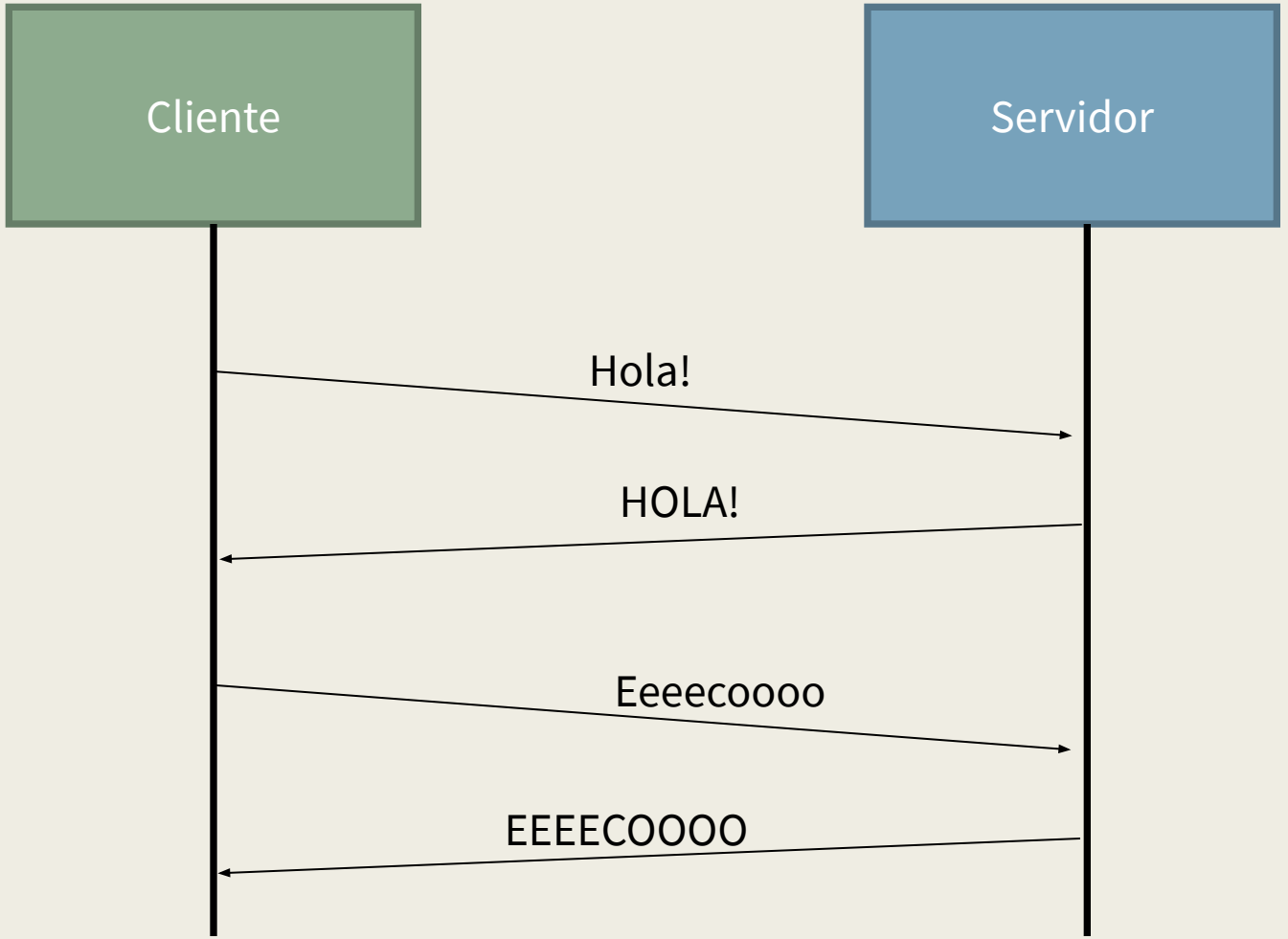
Hasta ahora nuestro servidor y cliente cierran la conexión después de un envío.

Vas a hacer un cambio para que los dos se mantengan conectados y mandando información hasta que el cliente envíe la palabra “exit”.

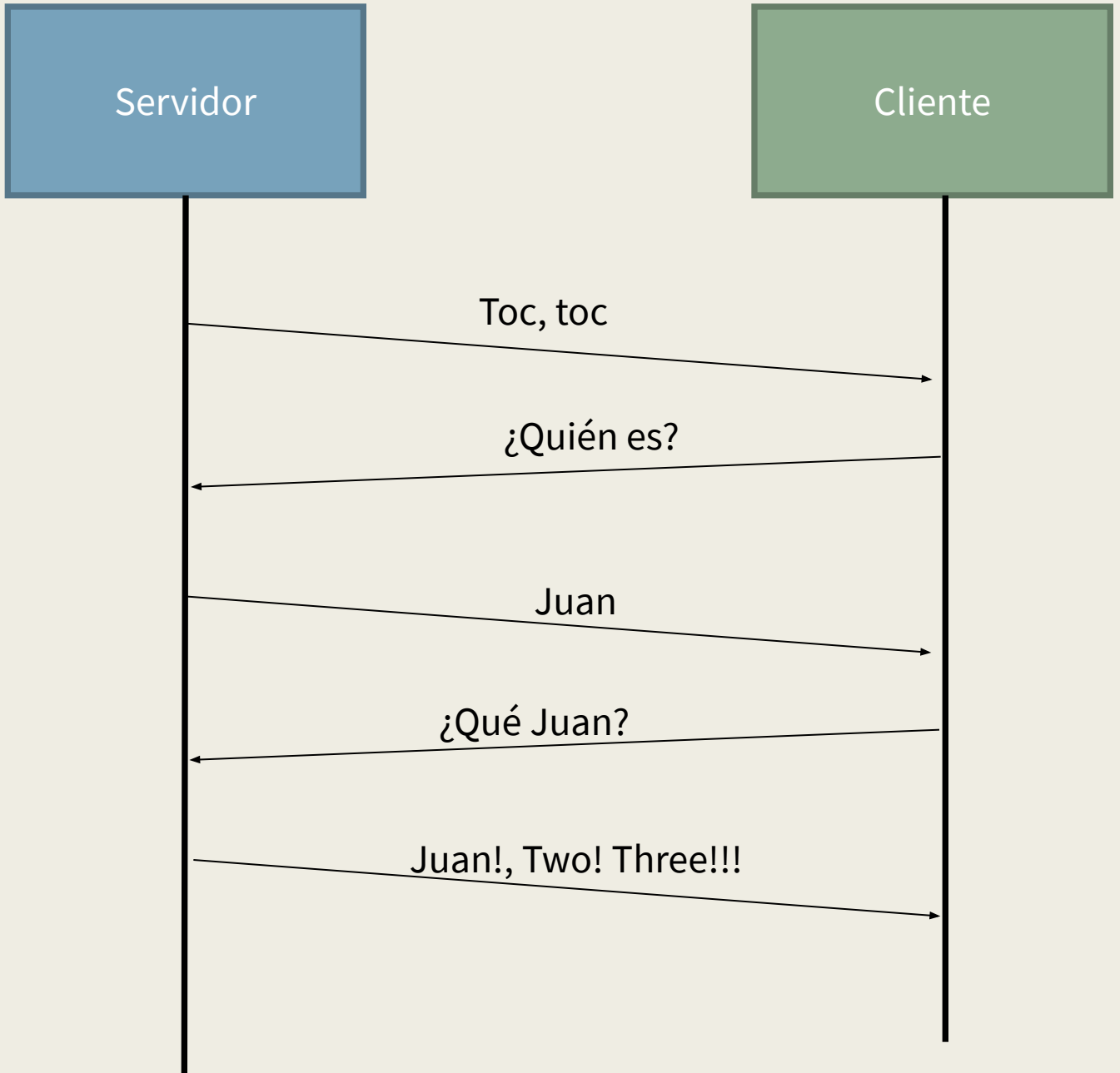
- El código de red va a seguir siendo el mismo
- Cambia la lógica alrededor de ello
  - *Tendrás que meter parte del código de cliente y servidor en un bucle para que estén enviando y recibiendo datos hasta la condición de parada*

Ejercicio 1: Servidor de echo

- El primer ejemplo que veremos es un servidor que devuelve lo que dice el cliente, pero en mayúsculas. La ejecución sigue hasta que el cliente escriba “exit”

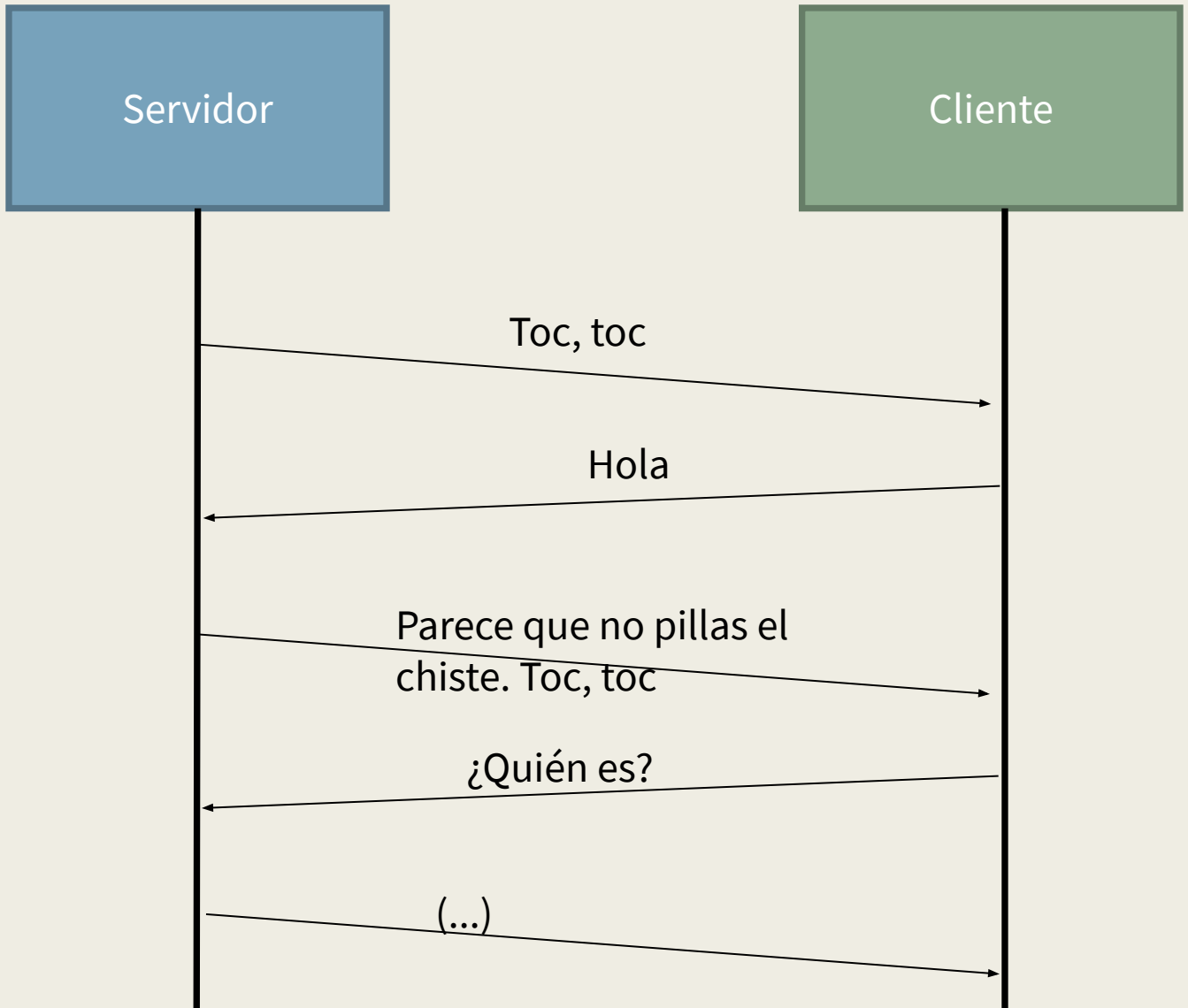


Ejercicio 2: Servidor Toc,toc



Ejercicio 2: Servidor Toc,toc

Ejemplo de error: Si el cliente no envía el texto correcto, el servidor tiene que enviarle un error.



La clase TocTocProtocol

Para que no tengas que montar el sistema de bromas, tienes la clase ya hecha.

https://www.dropbox.com/s/jkjnbnxb8qh6s6k/TocTocProtocol.cs?dl=0

Funcionamento:

1. Inicializar un objeto de tipo TocTocProtocol
2. Llamar al método processInput para obtener la respuesta del servidor
   a. *Si el método devuelve "Adiós", es que la broma ha terminado*
   b. *Si no, la broma sigue*

Consideraciones:

- Tienes que controlar que si TocTocProtocol dice "Adiós", hay que enviarlo al cliente, y cerrar el socket servidor.
  - *En la parte de cliente, cuando recibes "Adiós", también tienes que cerrar el socket y salir.*