Práctica

9

GESTIÓN Y UTILIZACIÓN DE REDES LOCALES

Curso 2001/2002

TCP/IP: protocolo TCP

Introducción

Como se ha comentado en la práctica anterior, el protocolo UDP es muy sencillo de implementar, pero presenta el inconveniente de un control de errores muy pobre, pues únicamente incorpora un *checksum* opcional. Además, en caso de error no avisa al transmisor del datagrama. Esto supone que cuando enviamos un datagrama UDP no podemos suponer que llega a su destino ni que se recibe sin error.

Por tanto, en aquellos casos en que haya que enviar mucha información, o en los que deseemos una mayor seguridad en la comunicación, es necesario emplear el protocolo TCP. Este protocolo proporciona una comunicación totalmente fiable, pero al mismo tiempo consume un mayor número de recursos, tanto de ancho de banda de la red (cabeceras mayores que en UDP), como de procesador (requiere un mayor número de comprobaciones).

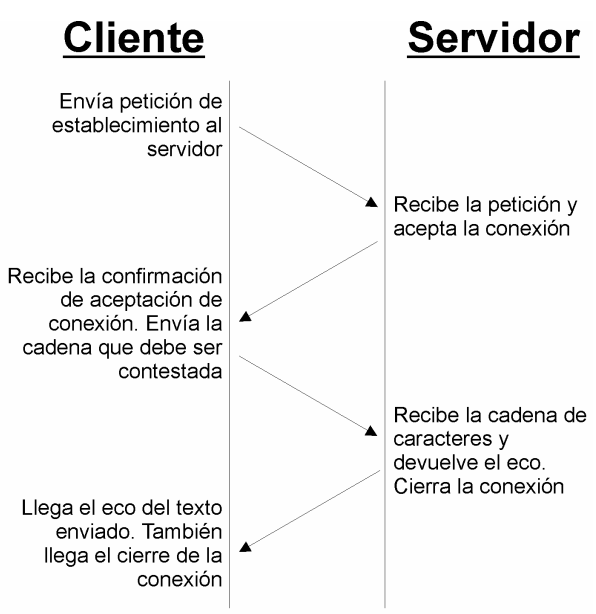
Por otra parte, a diferencia de UDP, TCP requiere un establecimiento previo de la conexión. Con TCP no es posible enviar directamente un datagrama a la red, como hacíamos en UDP. Con TCP es necesario establecer la conexión antes de enviar o recibir algún dato. Además, una vez se ha finalizado el intercambio de datos es necesario cerrar la conexión previamente establecida.

Con el objetivo de afianzar estos conceptos, en esta práctica vamos a implementar en Visual Basic un cliente y un servidor de eco utilizando los servicios proporcionados por el protocolo TCP. El esquema cliente-servidor es muy común en el mundo de las redes. Por ejemplo, cuando hacemos un ftp a una máquina remota, nuestro ordenador actúa como cliente y la máquina remota actúa como servidor. Otro ejemplo son los servidores web. Cada vez que navegamos por Internet nuestra máquina se convierte en el cliente, mientras que la máquina que alberga las páginas web que estamos visitando actúa como servidor.

Volviendo a nuestro caso concreto del cliente y servidor de eco, para realizar la práctica es necesario utilizar dos ordenadores (por tanto un grupo creará el cliente y el otro grupo creará el servidor). El papel que juega cada uno de los sistemas es:

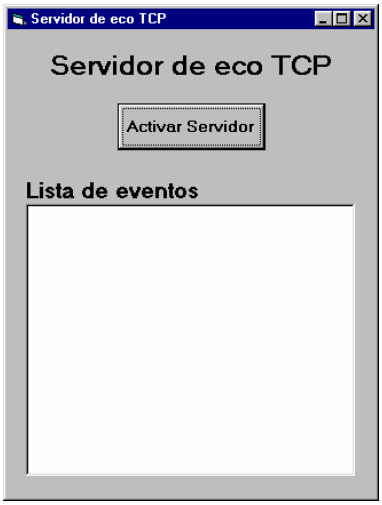
- **Servidor**: un servidor de eco se limita a responder con la misma información que le llega (para algo se llama "eco"). De esta forma, permanece inactivo escuchando en el puerto 7 hasta que recibe una petición de conexión. Dado que es un servidor de eco, aceptará la conexión. Una vez aceptada, el cliente le enviará el texto que el servidor debe responder. Tras recibir el texto, y devolverlo al cliente, el servidor cerrará la conexión.
- **Cliente**: el cliente inicia todo el proceso enviando al servidor una petición de establecimiento de conexión. Dado que quiere que se realice un eco, dicha petición la enviará al puerto 7 del servidor. El puerto desde el cual el cliente envía la petición no es importante. Una vez el servidor acepta la conexión, el cliente le envía la cadena de caracteres con la cual se debe hacer el eco, y espera la respuesta del servidor.

En la figura de la derecha se muestra la secuencia en la que se producen los diferentes eventos.

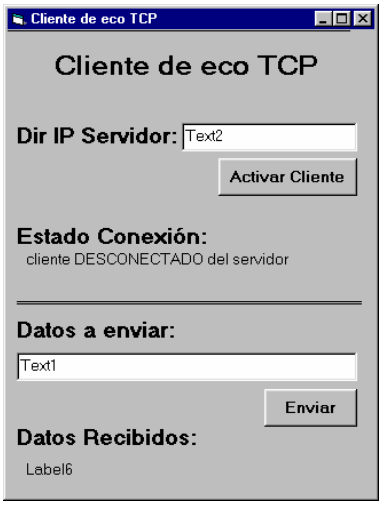


Realización de la práctica

Para simplificar el trabajo a realizar en esta práctica, se proporciona una versión incompleta de los programas a desarrollar en Visual Basic. Estos programas (el cliente y el servidor de eco) se encuentran en el directorio "**`\\herodes\practicas\Gyurl\Practica9`**". El funcionamiento de estos programas se detalla a continuación. En la explicación que sigue se asume que tanto el código del cliente como el del servidor ya se han acabado completamente.

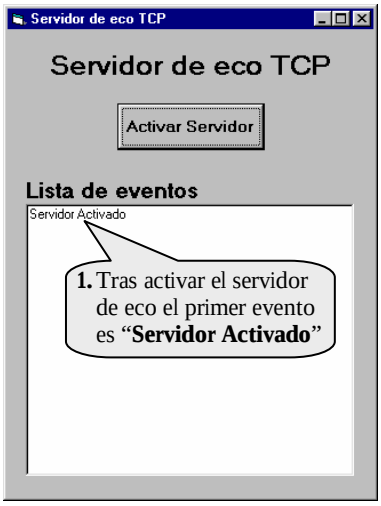


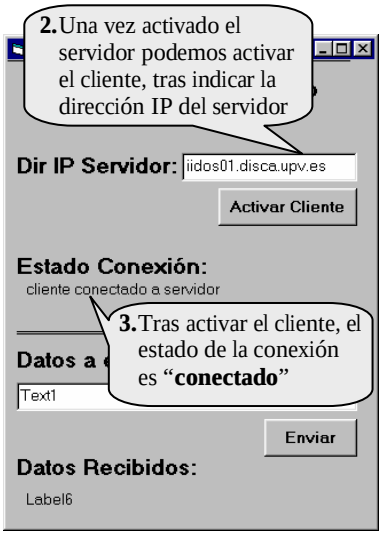
El servidor de eco se muestra en la figura de la derecha. Una vez en ejecución, lo primero que hay que hacer es activarlo, pulsando el botón "**`Activar Servidor`**". De esta forma el servidor se pone a la escucha en el puerto 7, a la espera de recibir alguna petición de establecimiento de conexión. El área en blanco que aparece en la ventana del servidor es una **ListBox**, donde irán apareciendo los diferentes eventos que vayan ocurriendo en el servidor.

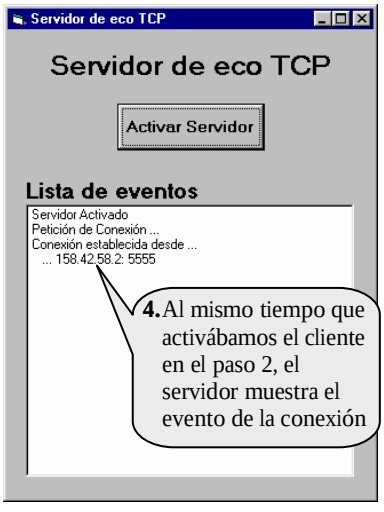


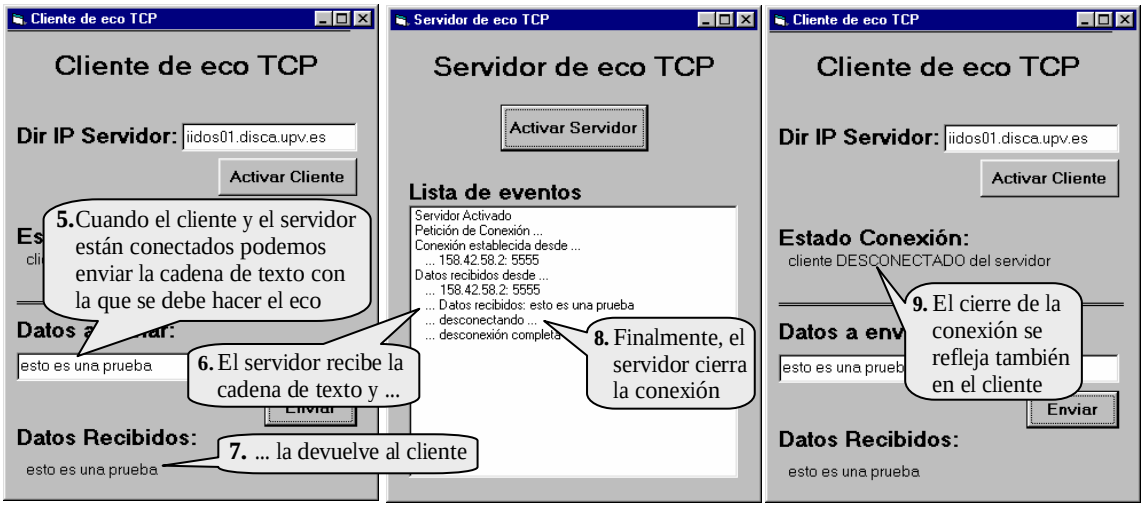
El cliente de eco se muestra en la siguiente figura. En la casilla "**`Dir IP Servidor`**" debemos especificar la dirección IP del servidor de eco, o bien el nombre de la máquina donde se está ejecutando dicho servidor. El estado de la conexión se muestra más abajo. En la figura se puede ver que inicialmente el cliente está desconectado del servidor. Una vez introducida la dirección del servidor, debemos pinchar en el botón "**`Activar Cliente`**" para iniciar el proceso de conexión.

En las siguientes figuras se muestra la evolución del cliente y del servidor conforme avanza la conexión.



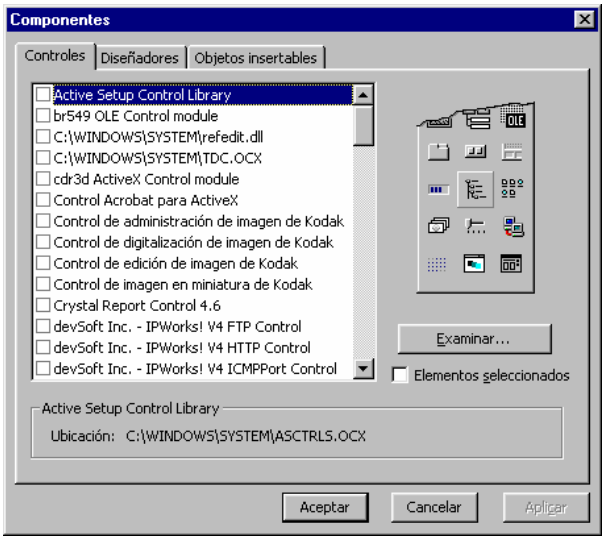






Programación con TCP en Visual Basic

Como en prácticas anteriores, necesitamos el control correspondiente para poder utilizar el protocolo TCP en Visual Basic. Para poder incluir en nuestra aplicación el control debemos primero incluirlo en la barra de herramientas del entorno Visual Basic. Para ello iremos al menú de "**Proyecto**" y escogeremos la opción "**Componentes**". Obtendremos la ventana que aparece a la derecha. Sin embargo, dado que el control a utilizar dependerá de si estamos implementando el cliente de eco o el servidor de eco, vamos a tratar cada uno de los casos por separado.



El cliente de eco TCP

El control a insertar en nuestro cliente de eco es el que figura como "**devSoft Inc. - IPWorks! V4 IPPort Control**". Entre las propiedades de este control, las que tienen que ver estrictamente con la comunicación son las siguientes:

- **RemoteHost:** nos sirve para especificar la dirección IP (o el nombre) del ordenador con el que deseamos conectar.
- **RemotePort:** sirve para especificar el número de puerto en el destino.
- **LocalPort:** nos permite especificar el puerto local (en nuestro ordenador) en el que escuchamos por si se recibe algo.
- **LocalHost:** es nuestro ordenador (su nombre o dirección IP).

Por otro lado, este control dispone de una serie de propiedades sólo utilizables durante la ejecución del programa, como **DataToSend**, que nos permite enviar una cadena de caracteres, o **Connected** que sirve para iniciar y finalizar una conexión (poniéndola a **True** o a

False). Además hay una serie de procedimientos asociados a eventos. Los más importantes los detallamos a continuación:

- **Connected**: Cuando la propiedad **Connected** se pone a **True**, la conexión no se realiza inmediatamente, sino que debe ser aceptada por el servidor. Por este motivo, cuando la propiedad **Connected** se pone a **True** no tenemos la certeza de que la conexión se haya establecido. Por eso es necesario el evento **Connected**. Este evento nos informa del resultado de la fase de establecimiento de conexión, a través de su parámetro **StatusCode**. Si la conexión se ha estableido con éxito, entonces StatusCode es 0 (cero). Si ha habido algun error (por ejemplo, que el servidor ha rechazado la conexión), entonces StatusCode es diferente de 0.
- **DataIn**: Este evento se activa cada vez que se reciben datos por la conexión establecida.
- **Disconnected**: Se produce al detectarse el fin de una conexión. Recordad que en TCP el fin de la conexión puede provocarlo cualquiera de los dos sistemas que están conectados.

De esta forma, el código mínimo para enviar una cadena de caracteres al servidor de eco sería el siguiente (asumiendo que la conexión se ha realizado sin problemas):

```
Private Sub Procedimiento_1()
    'iniciar una conexión
  IPPort1.RemoteHost = "iidos01.disca.upv.es"    'servidor de eco TCP
  IPPort1.RemotePort = 7    'puerto del servidor de eco TCP
  IPPort1.Connected = True    'inicia la conexión TCP


Private Sub Procedimiento_2()
IPPort1.DataToSend = "Datos a enviar"    'transmite datos
```

El servidor de eco TCP

El control a insertar para implementar el servidor de eco es el que figura como "**devSoft Inc. - IPWorks! V4 IPDaemon Control**". Este control se parece bastante al que hemos utilizado en el caso del cliente. La principal diferencia estriba en que aquel primer control se utiliza para iniciar el proceso de la conexión y este segundo control es el utilizado para recibir las peticiones de conexión.

Entre las propiedades de este control, algunas son idénticas a las del control anterior, como **LocalPort**, **LocalHost**, **RemotePort** y **RemoteHost**. Con respecto a las propiedades sólo utilizables durante la ejecución del programa, la propiedad **Connected** se mantiene, aunque en este control ya no sirve para iniciar una conexión, sino únicamente para cancelar conexiones previamente establecidas. Para activar el control se debe usar la propiedad **Listening** (poniéndola a **True**). Respecto a **DataToSend** no se produce ningún cambio. Por otra parte, de los procedimientos asociados a eventos, los más importantes son:

- **ConnectionRequest**: este evento se produce cuando otro equipo quiere establecer una conexión con nuestro servidor. Cuando se dispara este evento tenemos la posibilidad de aceptar o rechazar la conexión. Por defecto se aceptan todas las conexiones.
- **Connected**: Al igual que en el caso del control descrito en la sección anterior, este evento se produce cuando finalmente se establece la conexión.
- **DataIn**: Este evento se activa cada vez que se reciben datos.
- **Disconnected**: Se produce al finalizar una conexión.

Es conveniente destacar que un servidor puede tener en marcha varias conexiones al mismo tiempo. Este sería el caso en que varios ordenadores tratan de conectarse al servidor simultáneamente. En este caso, cuando se active el evento **DataIn**, indicando que llegan nuevos datos, ¿a cuál de todas las conexiones que están en marcha pertenecen esos datos?. Para solucionar esta ambigüedad los eventos del control **IPDaemon** tienen un parámetro adicional, llamado **ConnectionId**, que sirve para identificar a cual de todas las conexiones abiertas se refiere el evento en cuestión. Este parámetro adicional se debe utilizar como el índice de un array. Por ejemplo, si queremos enviar datos utilizando la propiedad **DataToSend**, el código a utilizar sería: **`IPDaemon1.DataToSend(ConnectionId) = "Datos a enviar"`**. Por otra parte, si queremos cerrar una conexión también tenemos que usar **ConnectionId**: **`IPDaemon1.Connected(ConnectionId) = False.`**

Trabajo a desarrollar

Después de revisar el concepto de cliente y de servidor, y de ver cómo se comportan cada uno de ellos y cuales son las propiedades y eventos asociados a los controles necesarios para desarrollar ambos programas en Visual Basic, el trabajo a desarrollar en esta práctica consiste en completar el código de ambos programas. En el propio código de cada programa se detallan las zonas en las que es necesario completar el desarrollo del mismo.