ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

“PROGRAMACIÓN DE UN SERVIDOR WEB EN HTML”

AUTORÍA
JOSÉ RUIZ DÍAZ
TEMÁTICA
TECNOLOGÍA
ETAPA
2º BACHILLERATO

RESUMEN

En el presente libro se pretende crear un servidor HTML que sea capaz de entregar datos a un ordenador cliente que se encuentra en un lugar remoto.

El lenguaje de programación elegido es el **Visual Basic** sobre todo por la sencillez con la que se consiguen resolver aplicaciones complejas aún a costa de perder velocidad y eficiencia en las aplicaciones con respecto a otros lenguajes.

PALABRAS CLAVES

Visual Basic, HTML, Winsock, Servidor, Aplicación Cliente-Servidor, Formulario, dirección IP, TCP, UDP, envío de datos, recepción de datos, módulos, subsanación de errores.

INTRODUCCIÓN

El lenguaje Visual Basic es un lenguaje de programación visual, es decir, un gran número de tareas se realizan sin escribir código, simplemente con operaciones gráficas realizadas con el ratón sobre la pantalla.

Un vistazo al lenguaje Visual Basic y a los controles que tiene preparados para explotar los recursos de la red, nos hace deducir que existe uno específico para comunicarse con otro ordenador mediante TCP y UDP. El control **Winsock**.

El control WinSock permitirá conectarse a un equipo e intercambiar datos con el protocolo TCP.

A partir de lo dicho, se podrá crear una aplicación que resida en un servidor y al que se pueda acceder desde varios clientes. Es lo que se conoce como una **aplicación Cliente/Servidor**.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

1. CONTROL WINSOCK

El control Winsock es el más elemental de los controles que Visual Basic dedica a las redes IP, ya que lo único que hace en sí es efectuar la conexión.

No implementa ningún servicio, eso se tiene que hacer con un programa pero en contraposición nos permite tener muchos datos y mucho control sobre ella. Es decir, una vez establecida la conexión, se establecerá un servicio mediante código.

El control Winsock habrá que agregarlo en la caja de herramientas (Proyecto >Componentes >). Su nombre completo es Microsoft Winsock Control 6.0. No es visible en tiempo de ejecución, sólo el programador sabe que el control se encuentra en nuestra aplicación y cuales son sus propiedades. Tiene esta forma:

Fig. Icono del control Winsock

La conexión con la red se establecerá a través de una conexión de acceso telefónico a redes y a partir de aquí el control Winsock comenzará a trabajar. Para poder comunicarse con el otro equipo, se debe conocer una serie de parámetros del PC, parámetros que se deberán pasar al otro equipo para que le reconozca y le permita establecer el diálogo. El control Winsock realiza esas tareas que permitirán conocer los parámetros del equipo.

2. ENTORNO DE PROGRAMACIÓN VISUAL BASIC 6.0

Antes de empezar a crear el formulario, en el presente apartado se pretende dar una descripción sobre como funciona el entorno en el que se ha programado la aplicación que servirá para clarificar los pasos dados hasta la aplicación final.

Cuando se arranca Visual Basic 6.0 aparece en pantalla una configuración como la que se ve en la siguiente figura:

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

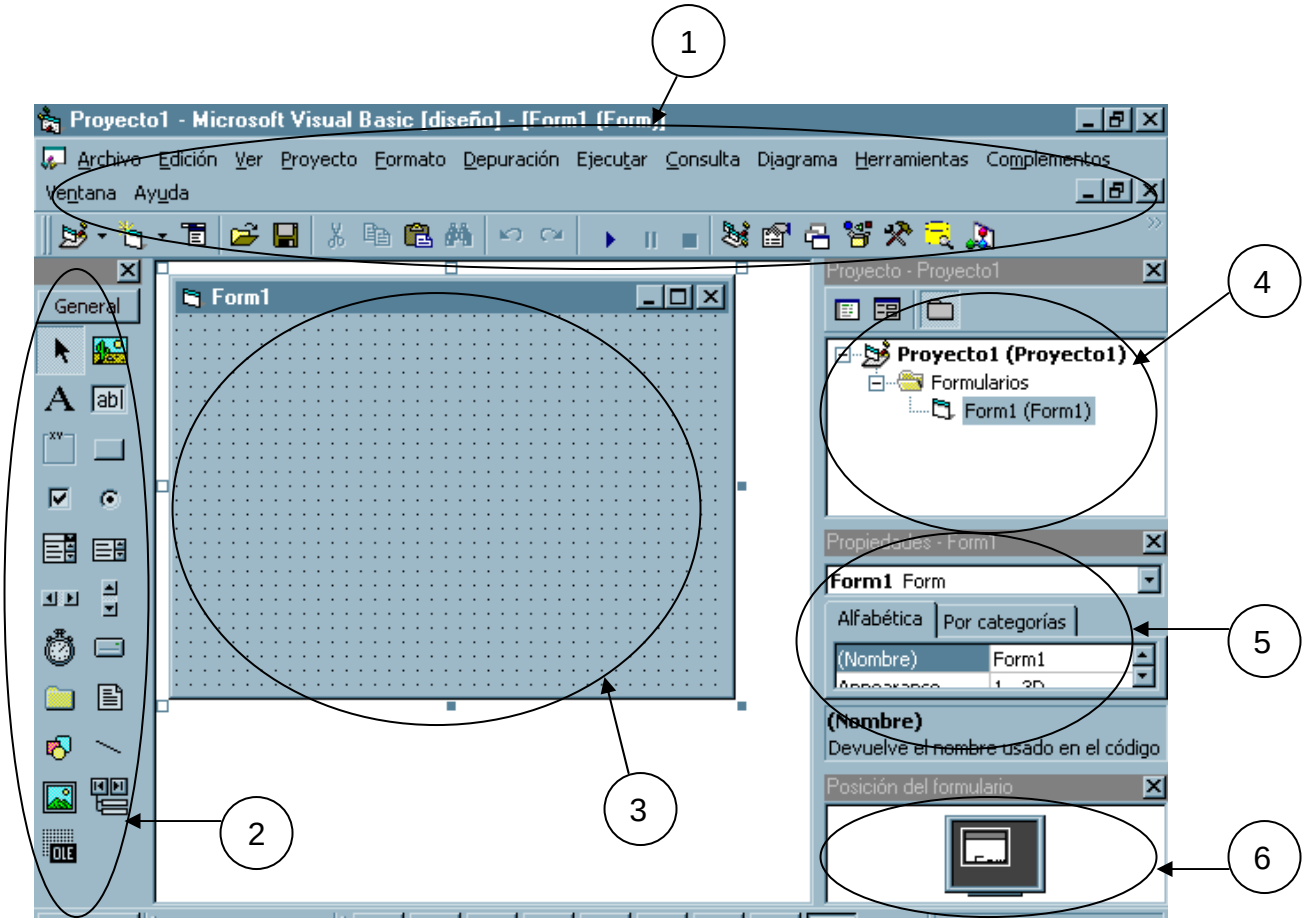

Fig. Entorno de programación de Visual Basic 6.0

En la figura se distinguen los siguientes elementos:

1) La barra de título, la barra de menús y la barra de herramientas.
2) Caja de herramientas: Se localizan los controles disponibles.
3) Formulario (form): en gris, en el que irán situando los controles. Esta dotado de una rejilla (gris) para facilitar la alineación de los controles.
4) Ventana de proyecto: muestra los formularios y otros módulos de programa que forman parte de la aplicación.
5) Ventana de propiedades: en la que se pueden ver las propiedades del objeto seleccionado o del propio formulario
6) Ventana de posición de formulario: determina la posición en que se abrirá la aplicación cuando comience a ejecutarse.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

3.- CREACIÓN DEL FORMULARIO

Visual Basic es una herramienta de diseño de aplicaciones para Windows, en la que estas se desarrollan en una gran parte a partir del diseño de una *interfaz gráfica*. En una aplicación de Visual Basic, el programa está formado por una parte de código puro, y otras partes asociadas a los objetos que forman la interfaz gráfica.

Es por tanto un término medio entre la programación tradicional, formada por una sucesión lineal de código estructurado y la programación orientada a objetos.

Combina ambas tendencias, es por tanto que Visual Basic es una *programación visual.*

Lo primero que se debe hacer es, crear una interfaz de usuario que será la principal vía de comunicación entre hombre y máquina tanto para salida como para entrada de datos.

Se parte de una ventana (formulario) a la que le iremos añadiendo los controles necesarios. La ubicación de dichos controles es a gusto del programador, siempre tratando que el usuario final este a gusto con el producto y que se pueda manejar libremente sin problemas por el entorno del mismo.

Los controles utilizados son:

- Winsock Control
- 1 botón
- 1 Lista (List Box)
- 1 caja de texto (TextBox)

De estos controles se establecerán a continuación sus propiedades más interesantes:

Control	Propiedad
Winsock	WS
Command1	Caption = “Inicio”
ListBox	Text =
TextBox	Text =

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

La ubicación de los controles es la que se indica en la siguiente figura:

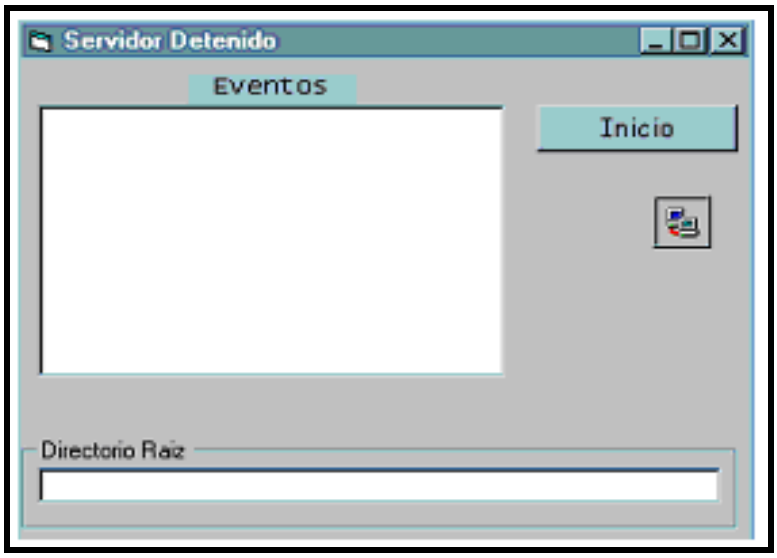

Fig. Ubicación de controles en el formulario

4. CÓDIGO DE LA APLICACIÓN

A partir del formulario ya obtenido, se puede empezar a tipear código. El sangrado del programa es una cuestión meramente de entendimiento para el programador, algunos recurren a él y otros no. Es una cuestión del que programa.

Se aplicará con el fin de un mejor entendimiento de qué hace el programa.

4.1. CARGA DEL FORMULARIO

Una vez finalizada la fase de diseño empieza la fase de ejecución del programa.
El procedimiento que describe inicialmente al formulario es el siguiente:

```
Private Sub Form_Load()
DRaiz = App.PATH & "\"
End Sub
```

En la caja de texto (con nombre DRaiz), se introducirá el directorio donde se encuentra la página web que se quiere enviar.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

Para obtener el directorio Raíz donde se encuentra la página html se usa el objeto App (propiedad Path, con lo que en la caja de texto aparecerá el directorio donde se encuentra la página que se quiere enviar.

4.2 ACTIVACIÓN/DESACTIVACIÓN DEL SERVIDOR

El código para activar ó desactivar el servidor sería el siguiente:

```
Private Sub Estado_Click ()
     If Estado.Caption = “Inicio” Then
          PATH = DRaiz.Text
          Estado.Caption = “Detener”
          WS. LocalPort = 80
          WS. Listen
          ListaEventos.Clear
          Servidor.Caption = “Iniciando en http://” & WS.LocalIP
     Else
     Estado.Caption = “Inicio”
     Servidor.Caption = “Servidor Detenido”
     WS.Close
     EndIf
End Sub
```

Ante la ocurrencia del evento de “click” el botón que posee la propiedad Caption “Estado”, habrá que tener en cuenta si el servidor está detenido ó si por el contrario está en funcionamiento. Es por ello que se utiliza una estructura de selección (sentencia If) para discernir entre ambos casos.

Si el servidor estaba detenido, el nuevo título de la etiqueta (propiedad caption) pasará a ser “Detener” por si se quiere desactivar posteriormente. Además, el directorio raíz actual pasa a ser el que estaba en la caja de texto con nombre DRaiz.

Por otro lado, cuando un equipo quiere enviar información debe conocer la dirección completa del equipo al que va a llamar. La dirección completa es su dirección IP y el puerto al que va dirigida la información. Pero si el equipo va a recibir, lo único que debemos decirle es el puerto por el que debe estar escuchando.

No necesita conocer más datos pues su dirección IP propia ya la conoce y no necesita ningún dato del equipo que le va a enviar información.

El servidor que se va a desarrollar será una aplicación que está escuchando a otra aplicación cliente que está enviando información. Es por tanto que debemos generar un socket y establecerlo a modo de escucha. Esto se consigue a través de la sentencia WS.Listen (NombredelWinsock.Listen). Resaltar que esta instrucción se usa sólo para conexiones TCP.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

Por otra parte, debemos decirle el servicio al que va destinado la información (puerto). En concreto, el servicio para recibir páginas web que es lo que se va a transmitir tiene reservado el puerto número 80 como se puede observar en la figura siguiente.

Ahora el puerto 80 está siendo vigilado para aceptar conexiones remotas.

Nº de puerto	Descripción
0	Reservado
1	**TCP** Servicio de multiplexado de puertos (TCPMUX)
4	No asignado
…..	…….
…..	…….
80	**HTTP** ("Hyper Text Transfer Protocol")

Fig. Tabla con los puertos más importantes que usan los servicios

A modo ilustrativo, en el título del servidor ya activo se puede resaltar la dirección IP de la máquina local en el formato xxx.xxx.xxx.xxx. Esto se consigue mediante la propiedad LocalIP. Es sólo de lectura y no está disponible en tiempo de diseño.

Si la dirección del equipo donde se encuentra el servidor es la 127.0.0.1 significará que no está conectado.

Finalmente, en el control Lista se irán mostrando los distintos eventos que ocurren durante el proceso de la comunicación. Se vaciará completamente el contenido de la lista (hay que pensar que puede tener eventos sucedidos en conexiones anteriores) utilizando el método Clear.

En la siguiente figura puede verse como quedaría el formulario tras la activación del servidor.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

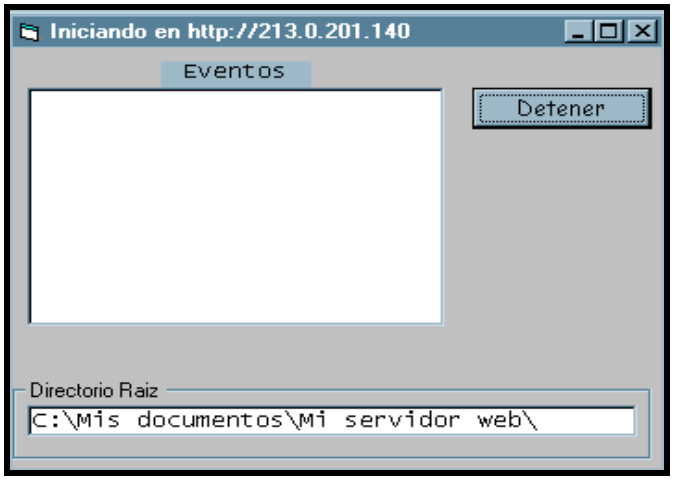

Fig. Activación del servidor. Arriba en la barra se puede observar la dirección IP del servidor (213.0.201.140). Abajo, el directorio raíz donde se encuentra la página.

Si por el contrario el servidor ya estaba activo y se pulsa el botón (ahora la propiedad caption del botón será “Detener”) su título será “Inicio” y en la barra del formulario se indicará que el servidor está detenido.

Además, al desactivarse es muy importante que el servidor cierre correctamente la conexión TCP. Para ello, debe usar el método Close (WS.Close).

Ahora el formulario quedaría del siguiente modo:

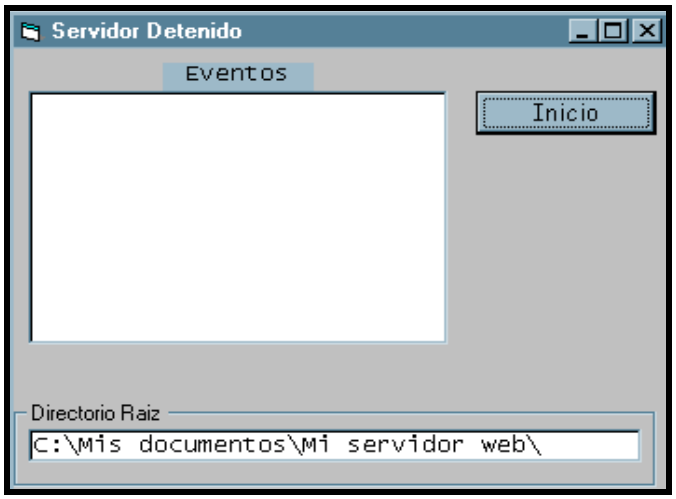

Fig. Servidor detenido

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

4.3. LISTA DE EVENTOS

El código para añadir eventos al control lista es el siguiente:

```
Private Sub Eventos (Texto As String)
     ListaEventos.AddItem Texto
     ListaEventos.ListIndex = ListaEventos.NewIndex
End Sub
```

Cada vez que ocurra un evento relacionado con la conexión (Solicitud de conexión, Conexión aceptada,…) se pasará como parámetro al procedimiento (Texto).

Para ir añadiendo eventos (registros) en tiempo de ejecución conforme vayan sucediendo se utilizará el método AddItem.

Además, con el fin de llevar un orden en la lista, utilizaremos la propiedad NewIndex que almacena el número de posición siguiente a la posición del último evento ocurrido. Cuando un nuevo evento sucede, su posición en la lista se la dará la propiedad NewIndex. Resaltar que los elementos de la lista se empiezan a numerar por cero y que si no hay eventos el valor de la propiedad ListIndex es -1.

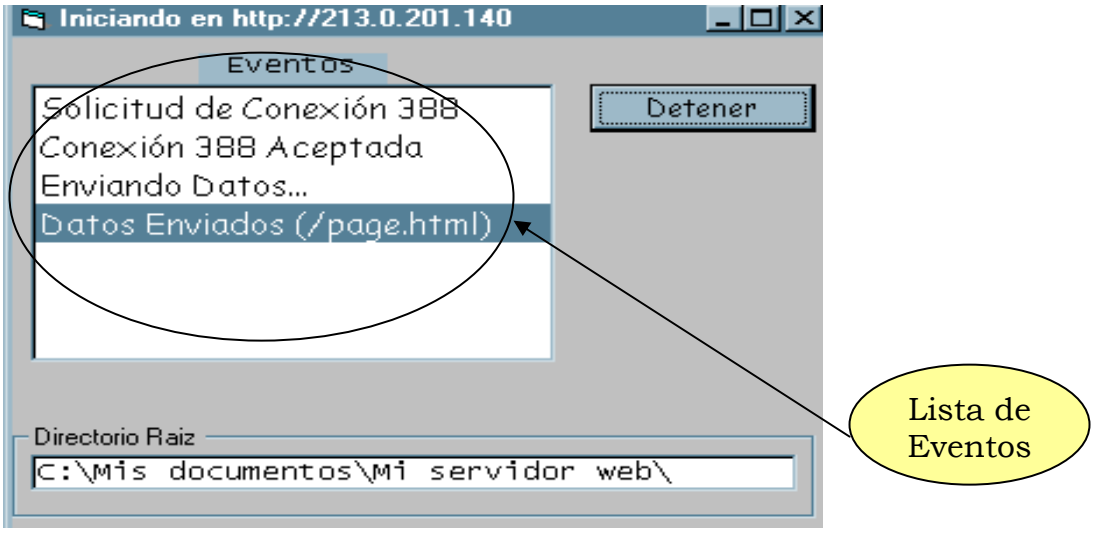

Fig. Lista de eventos de la comunicación

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

4.4. SOLICITUD DE CONEXIÓN

El procedimiento de solicitud de conexión es el siguiente:

```
Private Sub WS_ConnectionRequest (ByVal requestID As Long)
Eventos (“Solicitud de Conexión “ & requestID)
      If WS.State <> sckClosed Then Ws.Close
            WS.LocalPort = 0
            WS.Accept requestID
            Eventos “Conexión “ & requestID & “Aceptada”
            Eventos “Enviando datos...”
Do Events
```

El evento de Solicitud de conexión se produce en el equipo local cuando el cliente pide una conexión (nota: Sólo es válido para aplicaciones de servidor TCP).

Como se observa, este evento pasa como parámetro un número Long (requestID) que es un identificador de la petición de conexión. Este identificador lo genera el servidor y es necesario para aceptar la conexión posteriormente (método Accept)

El parámetro requestID identificará esa conexión hasta que se cierre. La identificación la hace el winsock estudiando en el datagrama entrante la dirección IP del origen, puerto origen y puerto destino, parámetros que no pueden coincidir al mismo tiempo con los de otra comunicación. Este mecanismo permite que winsock reciba varias comunicaciones simultáneas y sea capaz de diferenciar una de otra.

Consecuentemente, se añade un nuevo evento a la lista en el que se indica que se solicita una conexión.

Es fundamental que el winsock se encuentre cerrado (recordar que es necesario cerrar de forma adecuada una conexión TCP) si no lo estaba de una conexión anterior. Para ello se chequea el estado (propiedad State) del winsock y se cierra si la conexión que está realizando el winsock no está cerrada.

Ahora ya se está preparado para transmitir datos desde el servidor. Aunque se puede asignar un puerto determinado para enviar los datos es mucho más prudente dejar al winsock que utilice el que estime oportuno. De esta forma se asegura que el winsock siempre emitirá a través de un puerto que se encuentre libre. Además, la experiencia indica que si se asigna un puerto en concreto para transmitir lo más probable que pase es que el terminal se quede colgado.

Para asegurarse que no se ha asignado ningún puerto, bastará con poner la propiedad LocalPort del Winsock a 0 (WS.LocalPort = 0).

A partir de aquí, se utiliza el método Accept para aceptar la conexión entrante (sólo se usa con el protocolo TCP). Además, se indicará el número identificador de la solicitud de conexión que pasa el evento ConnectionRequest como parámetro, es decir, el parámetro requestID.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

El aceptar la conexión se visualiza en otro evento en donde a modo informativo se incluirá el citado identificador de la conexión.

Ahora la conexión está establecida, es entonces cuando empieza el envío de datos que se incluye como nuevo evento. Ver fig.

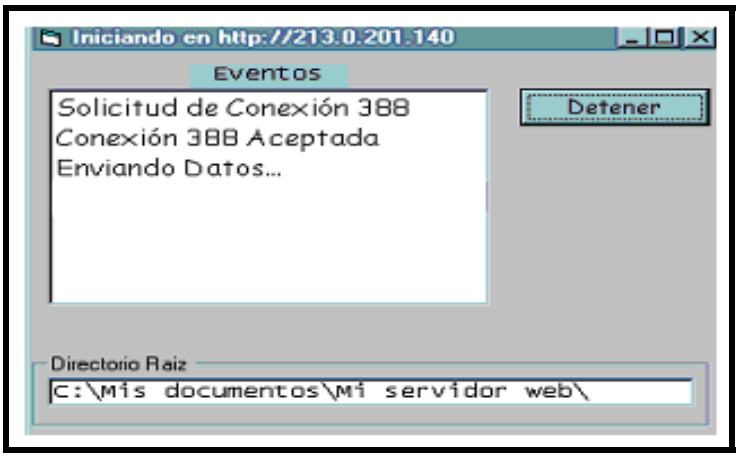

Fig. Se efectúa la solicitud de conexión (número identificador de solicitud de conexión 388). Se acepta y se procede al envío de datos.

Se está entonces a la espera de que lleguen nuevos datos (evento DataArrival) que se explicará en un apartado posterior.

Como no se hace nada especial mientras se está procesando este código, el formulario dará la impresión de estar “congelado”. Es por tanto conveniente dar paso a otros eventos, lo cual se hace a través de la instrucción DoEvents.

4.5. LLEGADA DE DATOS

Se debe hacer algo con la información que llega del cliente como es lógico. Se genera el evento DataArrival. El código de este evento sería:

```
Private Sub WS_DataArrival (ByVal bytesTotal As Long)
      Dim DATA As String
      Dim STRAUX As String
      Dim BUSCAR_ENVIAR As String
      Dim BUSCAR_RECIBIR As String
      WS.GetData DATA
      If Mid (DATA, 1, 3) = “GET” Then
            BUSCAR_ENVIAR = InStr (DATA, “GET”)
            STRAUX = InStr (BUSCAR_ENVIAR + 5, DATA, “ “) pagina = Mid (DATA,
            BUSCAR_ENVIAR + 4, STRAUX – (BUSCAR_ENVIAR +4))
      End If
```

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

Como se ha comentado, el evento DataArrival se produce cuando llegan nuevos datos. Además, en este evento se pasará como parámetro el número de bytes recibidos (bytesTotal).

Este evento no se producirá si no se recupera todos los datos. Esto se hace a través de la llamada GetData (Sólo se activa cuando hay datos nuevos).

Con el método GetData, se recuperan los datos existentes en el buffer de recepción del winsock, se almacenan en una variable (DATA en este caso) y se borra el contenido del buffer.

Llegados a este punto, resulta imprescindible definir el método GET. Con el método GET el cliente solicita información al servidor web, información que es concretamente la página donde se encuentran los datos.

Para ello, si el cliente solicita obtener GET http://213.0.201.140/page.html el servidor entenderá que la cadena de caracteres que conforma la página web solicitada es http://213.0.201.140/page.html.

Una vez que deduce la página que desea visionar el cliente, el servidor procede al envío.

4.6. ENVÍO DE DATOS

El código para el envío de datos es:

```
If ExistArchivo (PATH & “/” & pagina) Then
     WS.SendData LeerArchivo (PATH & “/” & pagina)
Else
     WS.SendData “No se encuentra la página”
End If
```

A través del método SendData se consigue enviar los datos al cliente que se encuentra en un lugar remoto. Este método se emplea tanto en UDP como en TCP. No devuelve ningún valor.

Lógicamente, habrá que chequear si la página html solicitada existe ó no. Para tal fin se programa un módulo estándar (fichero pagina.bas) en el cual se incluye un procedimiento (ExisteArchivo) que se explicará posteriormente y qué, simplemente, sirve para verificar si la página existe ó no. Si la página no existe, se visualizará una página de error.

El error visualizado se produce cuando el servidor web encuentra un fallo al intentar servir la petición ya que la página no existe. Como se observa, el código de error se presenta al usuario con una breve explicación.

Es interesante hacer que la página web de error esté un poco personalizada y con una breve descripción del fallo para que la página no le resulte tan fría al cliente que visita el enlace.

En cambio, si la página existe se puede proceder al envío de ésta. Se necesita otro programa que sea capaz de abrir la página y envíar ésta en cadenas de caracteres. Este programa se va a introducir en el módulo estándar (archivo pagina.bas), es una nueva función llamada LeerArchivo que se detallará en apartados siguientes.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

4.7. FIN DEL ENVÍO

El código asociado a dicho evento es el siguiente:

```
Private Sub WS_SendComplete ()
      Eventos “Datos Enviados (“ & pagina & “)”
      WS.Close
      WS.LocalPort = 80
      WS.Listen
End Sub
```

Cuando se finaliza la operación de envío se produce el evento SendComplete. Resaltar que no se pasan argumentos.

Se indicará a través de un nuevo evento en la lista que los datos (en concreto la página de prueba en html mostrada posteriormente a la que se accede desde el cliente) han sido enviados de modo satisfactorio. Como se expuso anteriormente, se deberá cerrar correctamente la conexión (evento Close).

Finalmente, es muy importante que no se olvide dejar el servidor escuchando a nuevas conexiones que puedan solicitarlo y, claro está, el puerto al que va la información (recordar que era el puerto nº80 para el servicio de páginas web).

Fig. Finalmente, el servidor envía la página (page.html). Si se introduce http:///(dirección IP)/nombre de página.html en el browser delnavegador aparecen los datos requeridos por el cliente al servidor. En este caso puede verse una preciosa imagen que el servidor ha captado.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

4.8. MÓDULOS

Un módulo estándar (ficheros *.bas) contiene sólo código que, en general, puede ser utilizado por distintos formularios y/o controles del proyecto e incluso por varios proyectos. Normalmente contienen siempre algunas declaraciones de variables globales o Public, que serán accesibles directamente desde todos los formularios.

Las dos funciones que a continuación se exponen pueden ser por tanto utilizadas por otros proyectos que trabajen con archivos.

4.8.1. Existe Archivo

La primera función que se llama ExisteArchivo tiene el siguiente código:

```
Public Function ExisteArchivo (ByVal Archivo As String) As Integer
      Dim L As Integer
      On Error Resume Next
      L = Len (Dir(Archivo))
      If Err Or L=0 Then
            ExisteArchivo = False
      Else
            ExisteArchivo = True
      EndIf
End Function
```

Lo que se hace simplemente es chequear si el archivo está. Para ello, se obtiene su longitud y se almacena en una variable (en este caso L). Si la longitud del archivo es la misma que la del archivo que se pasa como parámetro en la función se devuelve que el archivo se encuentra (TRUE), en caso contrario ó caso de que no se encuentre, se devuelve que no se encuentra (FALSE).

Si el archivo no está inicialmente, habrá que seguir la ejecución para que pueda devolver en este caso FALSE (no se encuentra). Es por ello que se introduce al principio para tal fin la instrucción On Error Resume Next para decir que continúe la ejecución de la función.

4.8.2. Leer Archivo

En este procedimiento, básicamente lo que se pretende es abrir un fichero pasado como parámetro leerlo carácter a carácter (se conoce como abrir un archivo de forma secuencial) hasta formar una cadena de caracteres que es la que finalmente se devuelve.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

El código de este módulo será:

```
Public Function LeerArchivo(Archivo As String) As String
      Dim cadena As String
      Dim numerocanal As Integer
      Dim car As String * 1
      On Error Resume Next
      numerocanal = FreeFile
      cadena = " "
      If ExisteArchivo(Archivo) Then
            If Len(Archivo) Then
                  Open Archivo For Input As #numerocanal
                  Do While Not EOF(numerocanal)
                        car = Input(1, #numerocanal)
                        cadena = " " & cadena & car
                  Loop
                  Close #numerocanal
            End If
            LeerArchivo = cadena
      Else
            LeerArchivo = " "
      End If
End Function
```

Se definirán tres variables: cadena que será una cadena de caracteres (inicialmente vacía), car que será de tipo carácter y en donde se almacenarán cada uno de los caracteres leídos y numerocanal que será un entero (comprendido entre 1 y 255) que representa el número del canal por donde se introducen los datos, se le conoce como *número de archivo.* No puede haber más de un archivo abierto con un número de canal determinado.

Inicialmente se parte de la cadena sin caracteres (“ “) y se provee un número de archivo que no esté en uso a través de la instrucción FreeFile.

En segundo lugar se chequea la existencia del archivo y su longitud, si es realmente el archivo que se desea mandar se abre mediante la instrucción Open en la que se le da un número de canal concreto.

Mientras no se llegue al fin del archivo, se extrae carácter a carácter el archivo abierto con ese número de canal, se almacena en la variable car y cada carácter se va añadiendo a la cadena conforme se extraen. Esta cadena es la que se devuelve una vez leída. Es realmente una lectura secuencial del fichero lo que se hace.

Por otra parte, habrá que tener en cuenta que si no existe el archivo habrá que devolver la cadena vacía (“ “).

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

Finalmente, cuando se termina de leer un fichero, es preciso cerrarlo con el fin de no tener muchos a la vez abiertos (instrucción Close).

4.9 SUBSANACIÓN DE ERRORES

Una vez elaborado el programa servidor, se procede a la detección y corrección de errores.

Para tal fin, la herramienta utilizada es el depurador ó Debugger. Con el Debbugger es posible ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables. De esta manera se facilita enormemente el descubrimiento de fuentes de errores.

Para ejecutar el programa paso a paso bastará con pulsar la tecla <F8> .

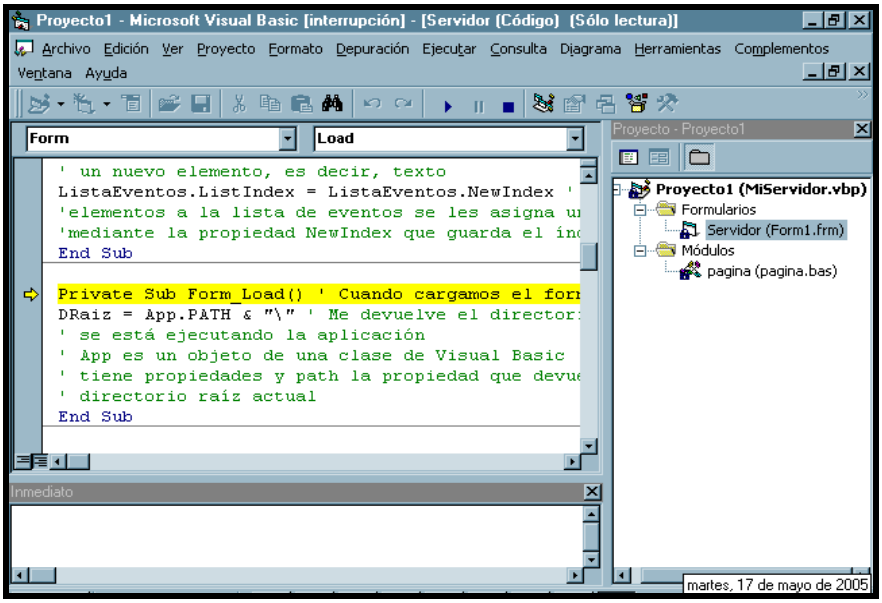

Fig. Ejecución paso a paso. Cada vez que se pulse <F8> se ejecuta una línea de código (marcada en amarillo).

4.9. FINALIZACIÓN

Una vez finalizada la programación de la nueva aplicación, la siguiente tarea consistirá en la creación de un programa ejecutable para su distribución e instalación en cuantos ordenadores se desee, incluso aunque en ellos no esté instalado Visual Basic 6.0.

Para crear un programa ejecutable se utiliza el comando Generar miservidor.exe dentro del menú Archivo (el nombre de la aplicación será MiServidor).

De este modo se generará un fichero cuya extensión será **.exe*.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 37– DICIEMBRE DE 2010

BIBLIOGRAFÍA

- Árboles S., Navarro L. (1999). **"Visual Basic 6 a Fondo"** Ed. Infor books S.L.: Interesante libro que puede servir como un completo manual para programar en Visual Basic. Resulta muy interesante el apartado que trata sobre aplicaciones cliente-servidor.
- VV.AA (1999) **"Visual Basic 6: Programación Cliente-Servidor".** Ed Thomson Paraninfo S.A.: Se centra en el apartado de las programaciones cliente-servidor con lo cual resulta de gran interés para el artículo.
- VV.AA. (1999) "**¿COMO SE HACE CON VISUAL BASIC 6 CLIENTE/SERVIDOR?"** Ed. Infor books: Otro completo libro que trata sobre la misma temática centrándose en el uso del comando Winsock para redes IP.

Autoría

- Nombre y Apellidos: José Ruiz Díaz
- Centro, localidad, provincia: I.E.S. Benjamín de Tudela. Tudela (Navarra)
- E-mail: jruizdia@pnte.cfnavarra.es